pgmoneta

Backup/Restore for PostgreSQL

Contents

1 Introduction

pgmoneta is a backup / restore solution for PostgreSQL.

Ideally, you would not need to do backups and disaster recovery, but that isn't how the real World works.

Possible scenarios that could happen

- Data corruption
- System failure
- Human error
- Natural disaster

and then it is up to the database administrator to get the database system back on-line, and to the correct recovery point.

Two key factors are

- Recovery Point Objective (RPO): Maximum targeted period in which data might be lost from an IT service due to a major incident
- Recovery Time Objective (RTO): The targeted duration of time and a service level within which a business process must be restored after a disaster (or disruption) in order to avoid unacceptable consequences associated with a break in business continuity

You would like to have both of these as close to zero as possible, since RPO of 0 means that you won't lose data, and RTO of 0 means that your system recovers at once. However, that is easier said than done.

pgmoneta is focused on having features that will allow database systems to get as close to these goals as possible such that high availability of 99.99% or more can be implemented, and monitored through standard tools.

pgmoneta is named after the Roman Goddess of Memory.

1.1 Features

- Full backup
- Restore
- Compression (gzip, zstd, lz4, bzip2)
- AES encryption support
- Symlink support
- WAL shipping support

- Hot standby
- Prometheus support
- Remote management
- Offline detection
- Transport Layer Security (TLS) v1.2+ support
- Daemon mode
- User vault

1.2 Platforms

The supported platforms are

- Fedora 39+
- RHEL 9
- RockyLinux 9
- FreeBSD
- OpenBSD

2 Installation

2.1 Rocky Linux 9.x / 10.x

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 **pgmoneta**

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

Add a configuration directory for **pgmoneta**

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

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

```
firewall-cmd --permanent --zone=public --add-port=5001/tcp
firewall-cmd --permanent --zone=public --add-port=5002/tcp
```

2.2 PostgreSQL 17

We will install PostgreSQL 17 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
   -9-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
   -9-aarch64/pgdg-redhat-repo-latest.noarch.rpm
```

and do the install via

```
dnf install -y postgresql17 postgresql17-server postgresql17-contrib
```

First, we will update ~ / `.bashrc` with

```
cat >> ~/.bashrc
export PGHOST=/tmp
export PATH=/usr/pgsql-17/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'
```

pg_hba.conf

```
local   all             all                     trust
host    postgres        repl    127.0.0.1/32    scram-sha-256
host    postgres        repl    ::1/128         scram-sha-256
host    replication     repl    127.0.0.1/32    scram-sha-256
host    replication     repl    ::1/128         scram-sha-256
```

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
```

Lets connect, add the replication user, and create the Write-Ahead Log (WAL) slot that we need for **pgmoneta**

```
psql postgres
CREATE ROLE repl WITH LOGIN REPLICATION PASSWORD 'repl';
SELECT pg_create_physical_replication_slot('repl', true, false);
\q
```

2.2.1 Transport Level Security support

Create the client key

```
openssl ecparam -name prime256v1 -genkey -noout -out client.key
```

Create the client request - remember that the `CN` has to have the name of the replication user

```
openssl req -new -sha256 -key client.key -out client.csr -subj "/CN=repl"
```

Generate the client certificate

```
openssl x509 -req -in client.csr -CA root.crt -CAkey server.key -
   CAcreateserial -out client.crt -days 3650 -sha256
```

You can test your setup by copying the files into the default PostgreSQL client directory, like

```
mkdir ~/.postgresql
cp client.crt ~/.postgresql/postgresql.crt
cp client.key ~/.postgresql/postgresql.key
cp root.crt ~/.postgresql/ca.crt
chmod 0600 ~/.postgresql/postgresql.crt ~/.postgresql/postgresql.key ~/.
   postgresql/ca.crt
```

and then test with the `psql` command.

In `pg_hba.conf` change

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

to

```
hostssl    all           all           0.0.0.0/0          scram-sha-256
   clientcert=verify-ca
```

More information

- Secure TCP/IP Connections with SSL
- The pg_hba.conf File

2.3 pgmoneta

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

```
dnf install -y pgmoneta
```

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

```
pgmoneta-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 `PGMONETA_PASSWORD` environment variable. Note that passing the key using the command line might not be secure.

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

```
cat > /etc/pgmoneta/pgmoneta.conf
[pgmoneta]
host = *
metrics = 5001

base_dir = /home/pgmoneta/backup

compression = zstd

retention = 7

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

unix_socket_dir = /tmp/

[primary]
host = localhost
```

```
port = 5432
user = repl
wal_slot = repl
```

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

Then, we will create the user configuration,

```
pgmoneta-admin -f /etc/pgmoneta/pgmoneta_users.conf -U repl -P repl user
   add
```

Lets create the base directory, and start **pgmoneta** now, by

```
mkdir backup
pgmoneta -d
```

3 Quick start

Make sure that **pgmoneta** is installed and in your path by using `pgmoneta -?`. You should see

```
pgmoneta 0.19.0
  Backup / restore solution for PostgreSQL

Usage:
  pgmoneta [ -c CONFIG_FILE ] [ -u USERS_FILE ] [ -A ADMINS_FILE ] [ -D
     DIRECTORY ] [ -d ]

Options:
  -c, --config CONFIG_FILE  Set the path to the pgmoneta.conf file
  -u, --users USERS_FILE    Set the path to the pgmoneta_users.conf file
  -A, --admins ADMINS_FILE  Set the path to the pgmoneta_admins.conf file
  -D, --directory DIRECTORY Set the directory containing all configuration
      files
                            Can also be set via
                               PGMONETA_CONFIGURATION_PATH environment
                               variable
  -d, --daemon              Run as a daemon
  -V, --version             Display version information
  -?, --help                Display help

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

If you encounter any issues following the above steps, you can refer to the **Installation** chapter to see how to install or compile pgmoneta on your system.

3.1 Configuration

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

```
[pgmoneta]
host = *
metrics = 5001

base_dir = /home/pgmoneta

compression = zstd

retention = 7

log_type = file
log_level = info
log_path = /tmp/pgmoneta.log
```

```
unix_socket_dir = /tmp/

[primary]
host = localhost
port = 5432
user = repl
wal_slot = repl
```

In our main section called `[pgmoneta]` we setup **pgmoneta** to listen on all network addresses. We will enable Prometheus metrics on port 5001 and have the backups live in the `/home/pgmoneta` directory. All backups are being compressed with zstd and kept for 7 days. Logging will be performed at `info` level and put in a file called `/tmp/pgmoneta.log`. Last we specify the location of the `unix_socket_dir` used for management operations and the path for the PostgreSQL command line tools.

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 and we will use the `repl` user account to connect, and the Write+Ahead slot will be named `repl` as well.

The `repl` user must have the `REPLICATION` role and have access to the `postgres` database, so for example

```
CREATE ROLE repl WITH LOGIN REPLICATION PASSWORD 'secretpassword';
```

and in `pg_hba.conf`

```
local   postgres        repl                    scram-sha-256
host    postgres        repl    127.0.0.1/32    scram-sha-256
host    postgres        repl    ::1/128         scram-sha-256
host    replication     repl    127.0.0.1/32    scram-sha-256
host    replication     repl    ::1/128         scram-sha-256
```

The authentication type should be based on `postgresql.conf`'s `password_encryption` value.

Then, create a physical replication slot that will be used for Write-Ahead Log streaming, like

```
SELECT pg_create_physical_replication_slot('repl', true, false);
```

Alternatively, configure automatically slot creation by adding `create_slot = yes` to `[pgmoneta]` or corresponding server section.

We will need a user vault for the `repl` account, so the following commands will add a master key, and the `repl` password. The master key should be longer than 8 characters.

```
pgmoneta-admin master-key
pgmoneta-admin -f pgmoneta_users.conf user add
```

For scripted use, the master key and user password can be provided using the `PGMONETA_PASSWORD` environment variable.

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

See the **Configuration** charpter for all configuration options.

3.2 Running

We will run **pgmoneta** using the command

```
pgmoneta -c pgmoneta.conf -u pgmoneta_users.conf
```

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

pgmoneta 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

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

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

```
pgmoneta-cli 0.19.0
  Command line utility for pgmoneta

Usage:
  pgmoneta-cli [ -c CONFIG_FILE ] [ COMMAND ]

Options:
  -c, --config CONFIG_FILE                      Set the path to the
     pgmoneta.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
  -v, --verbose                                 Output text string of
     result
  -V, --version                                 Display version
     information
  -F, --format text|json|raw                    Set the output format
  -C, --compress none|gz|zstd|lz4|bz2           Compress the wire
     protocol
  -E, --encrypt none|aes|aes256|aes192|aes128   Encrypt the wire protocol
  -s, --sort asc|desc                           Sort result (for list-
     backup)
```

```
      --cascade                                  Cascade a retain/expunge
         backup
  -?, --help                                     Display help

Commands:
  annotate                 Annotate a backup with comments
  archive                  Archive a backup from a server
  backup                   Backup a server
  clear <what>             Clear data, with:
                           - 'prometheus' to reset the Prometheus
                              statistics
  compress                 Compress a file using configured method
  conf <action>            Manage the configuration, with one of
     subcommands:
                           - 'get' to obtain information about a runtime
                              configuration value
                             conf get <parameter_name>
                           - 'ls' to print the configurations used
                           - 'reload' to reload the configuration
                           - 'set' to modify a configuration value;
                             conf set <parameter_name> <parameter_value>;
  decompress               Decompress a file using configured method
  decrypt                  Decrypt a file using master-key
  delete                   Delete a backup from a server
  encrypt                  Encrypt a file using master-key
  expunge                  Expunge a backup from a server
  info                     Information about a backup
  list-backup              List the backups for a server
  mode                     Switch the mode for a server
  ping                     Check if pgmoneta is alive
  restore                  Restore a backup from a server
  retain                   Retain a backup from a server
  shutdown                 Shutdown pgmoneta
  status [details]         Status of pgmoneta, with optional details
  verify                   Verify a backup from a server

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

This tool can be used on the machine running **pgmoneta** to do a backup like

```
pgmoneta-cli -c pgmoneta.conf backup primary
```

A restore would be

```
pgmoneta-cli -c pgmoneta.conf restore primary <timestamp> /path/to/restore
```

To shutdown pgmoneta you would use

```
pgmoneta-cli -c pgmoneta.conf shutdown
```

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

```
echo $?
```

or by using the `-v` flag.

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

3.4 Administration

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

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

```
pgmoneta-admin 0.19.0
  Administration utility for pgmoneta

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

Options:
  -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

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
```

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

```
pgmoneta-admin -g master-key
```

The master key must be at least 8 characters.

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

```
pgmoneta-admin -f pgmoneta_users.conf user add
```

3.5 Next Steps

Next steps in improving pgmoneta's configuration could be

- Read the manual
- Update `pgmoneta.conf` with the required settings for your system
- Enable Transport Layer Security v1.2+ (TLS) for administrator access

See Configuration for more information on these subjects.

3.6 Closing

The pgmoneta 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

4.1 pgmoneta.conf

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

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

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

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 `pgmoneta` on `localhost`.

Note, that PostgreSQL 13+ is required, as well as having `wal_level` at `replica` or `logical` level.

4.1.1 pgmoneta

General

Property	Default	Unit	Required	Description
host		String	Yes	The bind address for pgmoneta
unix_socket_dir		String	Yes	The Unix Domain Socket location
base_dir		String	Yes	The base directory for the backup

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

Monitoring

Property	Default	Unit	Required	Description
metrics	0	Int	No	The metrics port (disable = 0)
metrics_cache_max_	0	String	No	The time to keep a Prometheus (metrics) response in cache. If this value is specified without units, it is taken as seconds. Setting this parameter to 0 disables caching. It supports the following units as suffixes: ‘S’ for seconds (default), ‘M’ for minutes, ‘H’ for hours, ‘D’ for days, and ‘W’ for weeks.
metrics_cache_max_size	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
metrics_cert_file		String	No	Certificate file for TLS for Prometheus metrics. This file must be owned by either the user running pgmoneta 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 pgmoneta or root. Additionally permissions must be at least `0640` when owned by root or `0600` otherwise.
metrics_ca_file		String	No	Certificate Authority (CA) file for TLS for Prometheus metrics. This file must be owned by either the user running pgmoneta or root.

Management

Property	Default	Unit	Required	Description
management	0	Int	No	The remote management port (disable = 0)

Compression

Property	Default	Unit	Required	Description
compression	zstd	String	No	The compression type (none, gzip, client-gzip, server-gzip, zstd, client-zstd, server-zstd, lz4, client-lz4, server-lz4, bzip2, client-bzip2)
compression_level	3	Int	No	The compression level

Workers

Property	Default	Unit	Required	Description
workers	0	Int	No	The number of workers that each process can use for its work. Use 0 to disable. Maximum is CPU count

Workspace

Property	Default	Unit	Required	Description
workspace	/tmp/pgmoneta-workspace/	String	No	The directory for the workspace that incremental backup can use for its work

Storage

Property	Default	Unit	Required	Description
storage_engine	local	String	No	The storage engine type (local, ssh, s3, azure)

Encryption

Property	Default	Unit	Required	Description
encryption	none	String	No	The encryption mode for encrypt wal and data `none`: No encryption `aes | aes-256 | aes-256-cbc`: AES CBC (Cipher Block Chaining) mode with 256 bit key length `aes-192 | aes-192-cbc`: AES CBC mode with 192 bit key length `aes-128 | aes-128-cbc`: AES CBC mode with 128 bit key length `aes-256-ctr`: AES CTR (Counter) mode with 256 bit key length `aes-192-ctr`: AES CTR mode with 192 bit key length `aes-128-ctr`: AES CTR mode with 128 bit key length

Slot management

Property	Default	Unit	Required	Description
create_slot	no	Bool	No	Create a replication slot for all server. Valid values are: yes, no

SSH

Property	Default	Unit	Required	Description
ssh_hostname		String	Yes	Defines the hostname of the remote system for connection
ssh_username		String	Yes	Defines the username of the remote system for connection
ssh_base_dir		String	Yes	The base directory for the remote backup
ssh_ciphers	aes-256-ctr, aes-192-ctr, aes-128-ctr	String	No	The supported ciphers for communication. `aes | aes-256 | aes-256-cbc`: AES CBC (Cipher Block Chaining) mode with 256 bit key length `aes-192 | aes-192-cbc`: AES CBC mode with 192 bit key length `aes-128 | aes-128-cbc`: AES CBC mode with 128 bit key length `aes-256-ctr`: AES CTR (Counter) mode with 256 bit key length `aes-192-ctr`: AES CTR mode with 192 bit key length `aes-128-ctr`: AES CTR mode with 128 bit key length. Otherwise verbatim

S3

Property	Default	Unit	Required	Description
s3_aws_region		String	Yes	The AWS region
s3_access_key_id		String	Yes	The IAM access key ID
s3_secret_access_key		String	Yes	The IAM secret access key
s3_bucket		String	Yes	The AWS S3 bucket name
s3_base_dir		String	Yes	The base directory for the S3 bucket

Azure

Property	Default	Unit	Required	Description
azure_storage_account		String	Yes	The Azure storage account name
azure_container		String	Yes	The Azure container name
azure_shared_key		String	Yes	The Azure storage account key
azure_base_dir		String	Yes	The base directory for the Azure container

Retention

Property	Default	Unit	Required	Description
retention	7, - , - , -	Array	No	The retention time in days, weeks, months, years

Verification

Property	Default	Unit	Required	Description
verification	0	String	No	The time between verification of a backup. If this value is specified without units,

it is taken as seconds. Setting this parameter to 0 disables verification. It supports the following units as suffixes: ‘S’ for seconds (default), ‘M’ for minutes, ‘H’ for hours, ‘D’ for days, and ‘W’ for weeks. Default is 0 (disabled) |

Logging

Property	Default	Unit	Required	Description
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	pgmoneta.log	String	No	The log file location. Can be a strftime(3) compatible string.

Property	Default	Unit	Required	Description
log_rotation_age	0	String	No	The time after which log file rotation is triggered. If this value is specified without units, it is taken as seconds. Setting this parameter to 0 disables log rotation based on time. It supports the following units as suffixes: ‘S’ for seconds (default), ‘M’ for minutes, ‘H’ for hours, ‘D’ for days, and ‘W’ for weeks.
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)

Transport Level Security

Property	Default	Unit	Required	Description
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 pgmoneta or root.
tls_key_file		String	No	Private key file for TLS. This file must be owned by either the user running pgmoneta 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 pgmoneta or root.
libev	`auto`	String	No	Select the libev backend to use. Valid options: `auto`, `select`, `poll`, `epoll`, `iouring`, `devpoll` and `port`

Miscellaneous

Property	Default	Unit	Required	Description
backup_max_rate	0	Int	No	The number of bytes of tokens added every one second to limit the backup rate

Property	Default	Unit	Required	Description
network_max_rate	0	Int	No	The number of bytes of tokens added every one second to limit the netowrk backup rate
blocking_timeout	30	String	No	The number of seconds the process will be blocking for a connection. If this value is specified without units, it is taken as seconds. Setting this parameter to 0 disables it. It supports the following units as suffixes: 'S' for seconds (default), 'M' for minutes, 'H' for hours, 'D' for days, and 'W' for weeks.
keep_alive	on	Bool	No	Have `SO_KEEPALIVE` on sockets
nodelay	on	Bool	No	Have `TCP_NODELAY` on sockets
non_blocking	on	Bool	No	Have `O_NONBLOCK` on sockets
backlog	16	Int	No	The backlog for `listen()`. Minimum 16
hugepage	**`try`**	String	No	Huge page support (`off`, **`try`**, `on`)

Property	Default	Unit	Required	Description
pidfile		String	No	Path to the PID file. If not specified, it will be automatically set to `unix_socket_dir`/`pgmoneta.<host>.pid` where <`host`> is the value of the `host` parameter or `all` if `host = *`.

Property	Default	Unit	Required	Description
update_process_titl	`verbose`	String	No	The behavior for updating the operating system process title. 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 the base description; `verbose` (or `full`) to set the process title to the full description. 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.

4.1.2 Server section

Server

Property	Default	Unit	Required	Description
host		String	Yes	The address of the PostgreSQL instance
port		Int	Yes	The port of the PostgreSQL instance
user		String	Yes	The replication user name
wal_slot		String	Yes	The replication slot for WAL

The `user` specified must have the `REPLICATION` option in order to stream the Write-Ahead Log (WAL), and must have access to the `postgres` database in order to get the necessary configuration parameters.

Slot management

Property	Default	Unit	Required	Description
create_slot	no	Bool	No	Create a replication slot for this server. Valid values are: yes, no

Follow

Property	Default	Unit	Required	Description
follow		String	No	Failover to this server if follow server fails

Retention

Property	Default	Unit	Required	Description
retention		Array	No	The retention for the server in days, weeks, months, years

WAL shipping

Property	Default	Unit	Required	Description
wal_shipping		String	No	The WAL shipping directory

Hot standby

Property	Default	Unit	Required	Description
hot_standby		String	No	Hot standby directory. Single directory or comma separated directories up to 8 (e.g., /path/to/hot/standby1,/path/to/hot/standby2)
hot_standby_overric		String	No	Files to override in the hot standby directory. If multiple hot standbys are specified then this setting is separated by a |
hot_standby_tablespaces		String	No	Tablespace mappings for the hot standby. Syntax is [from -> to,?]+. If multiple hot standbys are specified then this setting is separated by a |

Workers

Property	Default	Unit	Required	Description
workers	-1	Int	No	The number of workers that each process can use for its work. Use 0 to disable, -1 means use the global settting. Maximum is CPU count

Transport Level Security

Property	Default	Unit	Required	Description
tls_cert_file		String	No	Certificate file for TLS. This file must be owned by either the user running pgmoneta or root.
tls_key_file		String	No	Private key file for TLS. This file must be owned by either the user running pgmoneta 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 pgmoneta or root.

Miscellaneous

Property	Default	Unit	Required	Description
backup_max_rate	-1	Int	No	The number of bytes of tokens added every one second to limit the backup rate. Use 0 to disable, -1 means use the global settting
network_max_rate	-1	Int	No	The number of bytes of tokens added every one second to limit the netowrk backup rate. Use 0 to disable, -1 means use the global settting

Extra

Property	Default	Unit	Required	Description
extra		String	No	The source directory for retrieval on the server side (details are in the extra section)

The `extra` configuration is set in the server section. It is not required, but if you configure this parameter, when you perform a backup using the CLI `pgmoneta-cli -c pgmoneta.conf backup primary`, it will also copy all specified files on the server side and send them back to the client side.

This `extra` feature requires the server side to install the pgmoneta_ext extension and also make the user `repl` a `SUPERUSER` (this will be improved in the future). Currently, this feature is only available to the `SUPERUSER` role.

You can set up `pgmoneta_ext` by following the README to easily install the extension. There are also more detailed instructions available in the DEVELOPERS documentation.

The format for the `extra` parameter is a path to a file or directory. You can list more than one file or directory separated by commas. The format is as follows:

```
extra = /tmp/myfile1, /tmp/myfile2, /tmp/mydir1, /tmp/mydir2
```

4.2 pgmoneta_users.conf

The pgmoneta_users configuration defines the users known to the system. This file is created and managed through the pgmoneta-admin tool.

The configuration is loaded from either the path specified by the -u flag or /etc/pgmoneta/pgmoneta_users.conf.

4.3 pgmoneta_admins.conf

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

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

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

4.4 Configuration Directory

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

Behavior: - When the directory flag (-D) is set, pgmoneta will look for all configuration files in the specified directory. - If a required file is not found in the specified directory, pgmoneta will look for it in its default location (e.g., /etc/pgmoneta/pgmoneta.conf). - If the file is not found in either location: - If the file is mandatory, pgmoneta will log an error and fail to start. - If the file is optional, pgmoneta 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, -u, -A, 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 (PGMONETA_CONFIG_DIR). - If neither the directory flag nor individual file flags are set, pgmoneta uses the default locations for all configuration files.

Using the Environment Variable: 1. Set the environment variable before starting pgmoneta:

```
export PGMONETA_CONFIG_DIR=/path/to/config_dir
pgmoneta -d
```

2. If both the environment variable and the `-D` flag are set, the flag takes precedence.

Example:

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

or

```
export PGMONETA_CONFIG_DIR=/custom/config/dir
pgmoneta -d
```

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

5 Command line interface

The **pgmoneta-cli** command line interface controls your interaction with **pgmoneta**.

It is important that you only use the pgmoneta-cli command line interface to operate on your backup directory

Using other commands on the backup directory could cause problems.

```
pgmoneta-cli 0.19.0
  Command line utility for pgmoneta

Usage:
  pgmoneta-cli [ -c CONFIG_FILE ] [ COMMAND ]

Options:
  -c, --config CONFIG_FILE                          Set the path to the
     pgmoneta.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
  -v, --verbose                                     Output text string of
     result
  -V, --version                                     Display version
     information
  -F, --format text|json|raw                        Set the output format
  -C, --compress none|gz|zstd|lz4|bz2               Compress the wire
     protocol
  -E, --encrypt none|aes|aes256|aes192|aes128       Encrypt the wire
     protocol
  -s, --sort asc|desc                               Sort result (for list-
     backup)
      --cascade                                     Cascade a retain/expunge
          backup
  -?, --help                                        Display help

Commands:
  annotate                  Annotate a backup with comments
  archive                   Archive a backup from a server
  backup                    Backup a server
  clear <what>              Clear data, with:
                            - 'prometheus' to reset the Prometheus
                               statistics
  compress                  Compress a file using configured method
  conf <action>             Manage the configuration, with one of
     subcommands:
                            - 'get' to obtain information about a runtime
                               configuration value
```

```
                              conf get <parameter_name>
                            - 'ls' to print the configurations used
                            - 'reload' to reload the configuration
                            - 'set' to modify a configuration value;
                              conf set <parameter_name> <parameter_value>;
  decompress                Decompress a file using configured method
  decrypt                   Decrypt a file using master-key
  delete                    Delete a backup from a server
  encrypt                   Encrypt a file using master-key
  expunge                   Expunge a backup from a server
  info                      Information about a backup
  list-backup               List the backups for a server
  mode                      Switch the mode for a server
  ping                      Check if pgmoneta is alive
  restore                   Restore a backup from a server
  retain                    Retain a backup from a server
  shutdown                  Shutdown pgmoneta
  status [details]          Status of pgmoneta, with optional details
  verify                    Verify a backup from a server

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

5.1 backup

Backup a server

The command for a full backup is

```
pgmoneta-cli backup <server>
```

Example

```
pgmoneta-cli backup primary
```

The command for an incremental backup is

```
pgmoneta-cli backup <server> <identifier>
```

where the `identifier` is the identifier for a backup.

Example

```
pgmoneta-cli backup primary 20250101120000
```

5.2 list-backup

List the backups for a server

Command

```
pgmoneta-cli list-backup <server> [--sort asc|desc]
```

The `--sort` option allows sorting backups by timestamp: - `asc` for ascending order (oldest first) - `desc` for descending order (newest first)

Example

```
pgmoneta-cli list-backup primary
```

Example with sorting

```
pgmoneta-cli list-backup primary --sort desc
```

5.3 restore

Restore a backup from a server

Command

```
pgmoneta-cli restore <server> [<timestamp>|oldest|newest] [[current|name=X
    |xid=X|lsn=X|time=X|inclusive=X|timeline=X|action=X|primary|replica],*]
     <directory>
```

where

- `current` means copy the Write-Ahead Log (WAL), and restore to first stable checkpoint
- `name=X` means copy the Write-Ahead Log (WAL), and restore to the label specified
- `xid=X` means copy the Write-Ahead Log (WAL), and restore to the XID specified
- `time=X` means copy the Write-Ahead Log (WAL), and restore to the timestamp specified
- `lsn=X` means copy the Write-Ahead Log (WAL), and restore to the Log Sequence Number (LSN) specified
- `inclusive=X` means that the restore is inclusive of the specified information
- `timeline=X` means that the restore is done to the specified information timeline
- `action=X` means which action should be executed after the restore (pause, shutdown)

More information

Example

```
pgmoneta-cli restore primary newest name=MyLabel,primary /tmp
```

5.4 verify

Verify a backup from a server

Command

```
pgmoneta-cli verify <server> <directory> [failed|all]
```

Example

```
pgmoneta-cli verify primary oldest /tmp
```

5.5 archive

Archive a backup from a server

Command

```
pgmoneta-cli archive <server> [<timestamp>|oldest|newest] [[current|name=X
   |xid=X|lsn=X|time=X|inclusive=X|timeline=X|action=X|primary|replica],*]
    <directory>
```

Example

```
pgmoneta-cli archive primary newest current /tmp
```

5.6 delete

Delete a backup from a server

Command

```
pgmoneta-cli delete <server> [<timestamp>|oldest|newest]
```

Example

```
pgmoneta-cli delete primary oldest
```

5.7 retain

Retain a backup from a server. The backup will not be deleted by the retention policy

Command

```
pgmoneta-cli retain [--cascade] <server> [<timestamp>|oldest|newest]
```

Example

```
pgmoneta-cli retain primary oldest
```

5.8 expunge

Expunge a backup from a server. The backup will be deleted by the retention policy

Command

```
pgmoneta-cli expunge [--cascade] <server> [<timestamp>|oldest|newest]
```

Example

```
pgmoneta-cli expunge primary oldest
```

5.9 encrypt

Encrypt the file in place, remove unencrypted file after successful encryption.

Command

```
pgmoneta-cli encrypt <file>
```

5.10 decrypt

Decrypt the file in place, remove encrypted file after successful decryption.

Command

```
pgmoneta-cli decrypt <file>
```

5.11 compress

Compress the file in place, remove uncompressed file after successful compression.

Command

```
pgmoneta-cli compress <file>
```

5.12 decompress

Decompress the file in place, remove compressed file after successful decompression.

Command

```
pgmoneta-cli decompress <file>
```

5.13 info

Information about a backup.

Command

```
pgmoneta-cli info <server> <timestamp|oldest|newest>
```

5.14 ping

Verify if **pgmoneta** is alive

Command

```
pgmoneta-cli ping
```

Example

```
pgmoneta-cli ping
```

5.15 mode

pgmoneta detects when a server is down. You can bring a server online or offline using the mode command.

Command

```
pgmoneta-cli mode <server> <online|offline>
```

Example

```
pgmoneta-cli mode primary offline
```

or

```
pgmoneta-cli mode primary online
```

pgmoneta will keep basic services running for an offline server such that you can verify a backup or do a restore.

5.16 shutdown

Shutdown **pgmoneta**

Command

```
pgmoneta-cli shutdown
```

Example

```
pgmoneta-cli shutdown
```

5.17 status

Status of **pgmoneta**, with a `details` option

Command

```
pgmoneta-cli status [details]
```

Example

```
pgmoneta-cli status details
```

5.18 conf

Manage the configuration

Command

```
pgmoneta-cli conf [reload | ls | get | set]
```

Subcommand

- `reload`: Reload configuration
- `ls` : To print the configurations used
- `get <config_key>` : To obtain information about a runtime configuration value
- `set <config_key> <config_value>` : To modify the runtime configuration value

Example

```
pgmoneta-cli conf reload
pgmoneta-cli conf ls
pgmoneta-cli conf get server.primary.host
pgmoneta-cli conf set encryption aes-256-cbc
```

conf get

Get the value of a runtime configuration key, or the entire configuration.

- If you provide a <config_key>, you get the value for that key.
  - For main section keys, you can use either just the key (e.g., host) or with the section (e.g., pgmoneta.host).
  - For server section keys, use the server name as the section (e.g., server.primary.host, server.myserver.port).
- If you run pgmoneta-cli conf get without any key, the complete configuration will be output.

Examples

```
pgmoneta-cli conf get
pgmoneta-cli conf get host
pgmoneta-cli conf get pgmoneta.host
pgmoneta-cli conf get server.primary.host
pgmoneta-cli conf get server.myserver.port
```

‘ **conf set**

Set the value of a runtime configuration parameter.

Syntax:

```
pgmoneta-cli conf set <config_key> <config_value>
```

Examples

```
# Logging and monitoring
pgmoneta-cli conf set log_level debug5
pgmoneta-cli conf set metrics 5001
pgmoneta-cli conf set management 5002

# Performance tuning
pgmoneta-cli conf set workers 4
pgmoneta-cli conf set backup_max_rate 1000000
pgmoneta-cli conf set compression zstd

# Retention policies
pgmoneta-cli conf set retention "14,2,6,1"
```

Key Formats: - **Main section parameters**: `key` or `pgmoneta.key` - Examples: `log_level`, `pgmoneta.metrics` - **Server section parameters**: `server.server_name.key` only - Examples: `server.primary.port`, `server.primary.host`

Important Notes: - Setting `metrics=0` or `management=0` disables those services - Invalid port numbers may show success but cause service failures (check server logs) - Server configuration uses format `server.name.parameter` (not `name.parameter`)

Response Types: - **Success (Applied)**: Configuration change applied to running instance immediately - **Success (Restart Required)**: Configuration change validated but requires manual update of configuration files AND restart - **Error**: Invalid key format, validation failure, or other errors

Important: Restart Required Changes When a configuration change requires restart, the change is only validated and stored temporarily in memory. To make the change permanent:

1. **Manually edit the configuration file** (e.g., `/etc/pgmoneta/pgmoneta.conf`)
2. **Restart pgmoneta** using `systemctl restart pgmoneta` or equivalent

Warning: Simply restarting pgmoneta without updating the configuration files will **revert** the change back to the file-based configuration.

Example of Restart Required Process:

```
# 1. Attempt to change host (requires restart)
pgmoneta-cli conf set host 192.168.1.100
# Output: Configuration change requires manual restart
#         Current value: localhost (unchanged in running instance)
#         Requested value: 192.168.1.100 (cannot be applied to live
   instance)

# 2. Manually edit /etc/pgmoneta/pgmoneta.conf
sudo nano /etc/pgmoneta/pgmoneta.conf
# Change: host = localhost
# To:     host = 192.168.1.100

# 3. Restart pgmoneta
sudo systemctl restart pgmoneta

# 4. Verify the change
pgmoneta-cli conf get host
# Output: 192.168.1.100
```

Why Manual File Editing is Required: - `pgmoneta-cli conf set` only validates and temporarily stores restart-required changes - Configuration files are **not automatically updated** by the command - On restart, pgmoneta always reads from the configuration files on disk - Without file updates, restart will revert to the original file-based values

5.19 clear

Clear data/statistics

Command

```
pgmoneta-cli clear [prometheus]
```

Subcommand

- `prometheus`: Reset the Prometheus statistics

Example

```
pgmoneta-cli clear prometheus
```

5.20 Shell completions

There is a minimal shell completion support for `pgmoneta-cli`.

Please refer to the manual for detailed information about how to enable and use shell completions.

6 Backup

6.1 Create a full backup

We can take a full backup from the primary with the following command

```
pgmoneta-cli backup primary
```

and you will get output like

```
Header:
  ClientVersion: 0.19.0
  Command: 1
  Output: 0
  Timestamp: 20240928065644
Outcome:
  Status: true
  Time: 00:00:20
Request:
  Server: primary
Response:
  Backup: 20240928065644
  BackupSize: 8531968
  Compression: 2
  Encryption: 0
  MajorVersion: 17
  MinorVersion: 0
  RestoreSize: 48799744
  Server: primary
  ServerVersion: 0.19.0
```

6.2 View backups

We can list all backups for a server with the following command

```
pgmoneta-cli list-backup primary
```

and you will get output like

```
Header:
  ClientVersion: 0.19.0
  Command: 2
  Output: 0
  Timestamp: 20240928065812
Outcome:
  Status: true
  Time: 00:00:00
Request:
```

```
  Server: primary
Response:
  Backups:
    - Backup: 20240928065644
      BackupSize: 8531968
      Comments: ''
      Compression: 2
      Encryption: 0
      Incremental: false
      Keep: false
      RestoreSize: 48799744
      Server: primary
      Valid: 1
      WAL: 0
  MajorVersion: 17
  MinorVersion: 0
  Server: primary
  ServerVersion: 0.19.0
```

6.3 Sorting backups

You can sort the backup list by timestamp using the `--sort` option:

```
pgmoneta-cli list-backup primary --sort asc
```

for ascending order (oldest first), or

```
pgmoneta-cli list-backup primary --sort desc
```

for descending order (newest first).

6.4 Create an incremental backup

We can take an incremental backup from the primary with the following command

```
pgmoneta-cli backup primary 20240928065644
```

and you will get output like

```
Header:
  ClientVersion: 0.19.0
  Command: 1
  Output: 0
  Timestamp: 20240928065730
Outcome:
  Status: true
  Time: 00:00:20
```

```
Request:
  Server: primary
Response:
  Backup: 20240928065750
  BackupSize: 124312
  Compression: 2
  Encryption: 0
  Incremental: true
  MajorVersion: 17
  MinorVersion: 0
  RestoreSize: 48799744
  Server: primary
  ServerVersion: 0.19.0
```

Incremental backups are supported when using PostgreSQL 17+. Note that currently branching is not allowed for incremental backup – a backup can have at most 1 incremental backup child.

6.5 Backup information

You can list the information about a backup

```
pgmoneta-cli -c pgmoneta.conf info primary newest
```

and you will get output like

```
Header:
  ClientVersion: 0.19.0
  Command: info
  Output: text
  Timestamp: 20241025163541
Outcome:
  Status: true
  Time: 00:00:00
Request:
  Backup: newest
  Server: primary
Response:
  Backup: 20241019163516
  BackupSize: 6.54MB
  CheckpointHiLSN: 0
  CheckpointLoLSN: 4F0000B8
  Comments: ''
  Compression: zstd
  Elapsed: 4
  Encryption: none
  EndHiLSN: 0
  EndLoLSN: 4F000158
  EndTimeline: 1
  Incremental: false
```

```
  Keep: true
  MajorVersion: 17
  MinorVersion: 0
  NumberOfTablespaces: 0
  RestoreSize: 45.82MB
  Server: primary
  ServerVersion: 0.19.0
  StartHiLSN: 0
  StartLoLSN: 4F000060
  StartTimeline: 1
  Tablespaces: {}
  Valid: yes
  WAL: 000000010000000000000004F
```

6.6 Verify a backup

You can use the command line interface to verify a backup by

```
pgmoneta-cli verify primary oldest /tmp
```

which will verify the oldest backup of the `[primary]` host.

pgmoneta creates a SHA512 checksum file(`backup.sha512`) for each backup at the backup root directory, which can be used to verify the integrity of the files.

Using `sha512sum`:

```
cd <path-to-specific-backup-directory>

sha512sum --check backup.sha512
```

The `verification` parameter can be use to control how frequently pgmoneta verifies the integrity of backup files. You can configure this in `pgmoneta.conf`:

```
[pgmoneta]
.
.
.
verification = 3600
```

For example, setting `verification = 3600` or `verification = 1H` will perform integrity checks every hour.

6.7 Encryption

By default, the encryption is disabled. To enable this feature, modify `pgmoneta.conf`:

```
encryption = aes-256-cbc
```

Many encryption modes are supported, see the documentation for the `encryption` property for details.

6.7.1 Encryption and Decryption Commands

pgmoneta use the same key created by `pgmoneta-admin master-key` to encrypt and decrypt files.

Encrypt a file with `pgmoneta-cli encrypt`, the file will be encrypted in place and remove unencrypted file on success.

```
pgmoneta-cli -c pgmoneta.conf encrypt '<path-to-your-file>/file.tar.zstd'
```

Decrypt a file with `pgmoneta-cli decrypt`, the file will be decrypted in place and remove encrypted file on success.

```
pgmoneta-cli -c pgmoneta.conf decrypt '<path-to-your-file>/file.tar.zstd.
   aes'
```

`pgmoneta-cli encrypt` and `pgmoneta-cli decrypt` are built to deal with files created by `pgmoneta-cli archive`. It can be used on other files though.

6.8 Annotate

Add a comment

You can add a comment by

```
pgmoneta-cli -c pgmoneta.conf annotate primary newest add mykey mycomment
```

Update a comment

You can update a comment by

```
pgmoneta-cli -c pgmoneta.conf annotate primary newest update mykey
   mynewcomment
```

Remove a comment

You can remove a comment by

```
pgmoneta-cli -c pgmoneta.conf annotate primary newest remove mykey
```

View comments

You can view the comments by

```
pgmoneta-cli -c pgmoneta.conf info primary newest
```

6.9 Archive

In order to create an archive of a backup use

```
pgmoneta-cli -c pgmoneta.conf archive primary newest current /tmp/
```

which will take the latest backup and all Write-Ahead Log (WAL) segments and create an archive named `/tmp/primary-<timestamp>.tar.zstd`. This archive will contain an up-to-date copy.

6.10 Crontab

Lets create a `crontab` such that a backup is made every day,

First, take a full backup if you are using PostgreSQL 17+,

```
pgmoneta-cli backup primary
```

then you can use incremental backup for your daily jobs,

```
crontab -e
```

and insert

```
0 6 * * * pgmoneta-cli backup primary latest
```

for taking an incremental backup every day at 6 am.

Otherwise use the full backup in the cron job.

7 Retention

The retention policy decide for how long a backup should be kept.

7.1 Retention configuration

The configuration is done in the main configuration section, or by a server basis with

Property	Default	Unit	Required	Description
retention	7, -, -, -	Array	No	The retention time in days, weeks, months, years

which means by default that backups are kept for 7 days.

Defining a retention policy is very important because it defines how you will be able to restore your system from the backups.

The key is to decide what your policy is, for example

```
7, 4, 12, 5
```

will keep backups for 7 days, one backup each Monday for 4 weeks, one backup for each month, and backups for 5 years.

There are a lot of ways to leave a parameter unspecified. For trailing parameters, you can simply omit them. And for parameters in between, you can use placeholders. Currently, placeholders we allow are: `-`, `X`, `x`, `0` or whitespaces (spaces or tabs).

If you want to restore from the latest backup plus the Write-Ahead Log (WAL) then the default **pgmoneta** policy maybe is enough.

Note, that if a backup has an incremental backup child that depends on it, its data will be rolled up to its child before getting deleted.

Current validation rule is:

1. Retention days >= 1
2. If retention months is specified, then 1 <= weeks <= 4, otherwise weeks >= 1
3. If retention years is specified, then 1 <= months <= 12, otherwise months >= 1
4. Retention years >= 1

Please note that the rule above only checks specified parameters, except for days, which should always be specified

The retention check runs every 5 minutes, and will delete one backup per run.

You can change this to every 30 minutes by

```
retention_interval = 1800
```

under the [pgmoneta] configuration.

7.2 Delete a backup

```
pgmoneta-cli -c pgmoneta.conf delete primary oldest
```

will delete the oldest backup on [primary].

Note, that if the backup has an incremental backup child that depends on it, its data will be rolled up to its child before getting deleted.

7.3 Write-Ahead Log shipping

In order to use WAL shipping, simply add

```
wal_shipping = your/local/wal/shipping/directory
```

to the corresponding server section of pgmoneta.conf, **pgmoneta** will create the directory if it doesn't exist, and ship a copy of WAL segments under the subdirectory your/local/wal/shipping/directory/server_name/wal.

8 Keeping backups

8.1 List backups

First, we can list our current backups using

```
pgmoneta-cli list-backup primary
```

you will get output like

```
Header:
  ClientVersion: 0.19.0
  Command: list-backup
  Output: text
  Timestamp: 20241018092853
Outcome:
  Status: true
  Time: 00:00:00
Request:
  Server: primary
Response:
  Backups:
    - Backup: 20241012091219
      BackupSize: 6.11MB
      Comments: ''
      Compression: zstd
      Encryption: none
      Keep: false
      RestoreSize: 39.13MB
      Server: primary
      Valid: yes
      WAL: 0
  MajorVersion: 17
  MinorVersion: 0
  Server: primary
  ServerVersion: 0.19.0
```

As you can see backup `20241012091219` has a `Keep` flag of **`false`**.

8.2 Keep a backup

Now, in order to keep the backup which means that it won't be deleted by the retention policy you can issue the following command,

```
pgmoneta-cli retain primary 20241012091219
```

and get output like,

```
Header:
  ClientVersion: 0.19.0
  Command: retain
  Output: text
  Timestamp: 20241018094129
Outcome:
  Status: true
  Time: 00:00:00
Request:
  Backup: 20241012091219
  Server: primary
Response:
  Backups:
    - 20241012091219
  Cascade: false
  Comments: ''
  Compression: none
  Encryption: none
  Keep: true
  MajorVersion: 17
  MinorVersion: 0
  Server: primary
  ServerVersion: 0.19.0
  Valid: yes
```

and you can see that the backup has a `Keep` flag of **`true`**.

8.3 Describe a backup

Now, you may want to add a description to your backup, and as you can see

```
Header:
  ClientVersion: 0.19.0
  Command: retain
  Output: text
  Timestamp: 20241018094129
Outcome:
  Status: true
  Time: 00:00:00
Request:
  Backup: 20241012091219
  Server: primary
Response:
  Backups:
    - 20241012091219
  Cascade: false
  Comments: ''
  Compression: none
  Encryption: none
```

```
  Keep: true
  MajorVersion: 17
  MinorVersion: 0
  Server: primary
  ServerVersion: 0.19.0
  Valid: yes
```

there is a `Comments` field to do that.

You can use the command,

```
pgmoneta-cli annotate primary 20241012091219 add Type "Main fall backup"
```

which will give

```
Header:
  ClientVersion: 0.19.0
  Command: annotate
  Output: text
  Timestamp: 20241018095906
Outcome:
  Status: true
  Time: 00:00:00
Request:
  Action: add
  Backup: 20241012091219
  Comment: Main fall backup
  Key: Type
  Server: primary
Response:
  Backup: 20241012091219
  BackupSize: 6.11MB
  CheckpointHiLSN: 0
  CheckpointLoLSN: 33554560
  Comments: Type|Main fall backup
  Compression: zstd
  Elapsed: 1
  Encryption: none
  EndHiLSN: 0
  EndLoLSN: 33554776
  EndTimeline: 1
  Keep: true
  MajorVersion: 17
  MinorVersion: 0
  NumberOfTablespaces: 0
  RestoreSize: 39.13MB
  Server: primary
  ServerVersion: 0.19.0
  StartHiLSN: 0
  StartLoLSN: 33554472
  StartTimeline: 1
```

```
  Tablespaces:

  Valid: yes
  WAL: 000000010000000000000002
```

As you can see the `Comments` field with the `Type` key.

The `annotate` command has `add`, `update` and `remove` commands to modify the `Comments` field.

8.4 Put a backup back into retention

When you don't need a backup anymore you can put into retention again by,

```
pgmoneta-cli expunge primary 20241012091219
```

will give,

```
Header:
  ClientVersion: 0.19.0
  Command: expunge
  Output: text
  Timestamp: 20241018101839
Outcome:
  Status: true
  Time: 00:00:00
Request:
  Backup: 20241012091219
  Server: primary
Response:
  Backup: 20241012091219
  Comments: Type|Main fall backup
  Compression: none
  Encryption: none
  Keep: false
  MajorVersion: 17
  MinorVersion: 0
  Server: primary
  ServerVersion: 0.19.0
  Valid: yes
```

and now, the `Keep` flag is back to **`false`**.

8.5 Cascade mode

You can retain/expunge the entire incremental backup chain using the `--cascade` option. This will retain/expunge all backups along the line until the root full backup.

Say you have an incremental backup chain: `20250625055547 (incremental) -> 20250625055528 (incremental) -> 20250625055517(full)`. Running `pgmoneta-cli retain --cascade primary 20250625055547` will also retain backup `20250625055528` and backup `20250625055517`.

This will give

```
Header:
  ClientVersion: 0.19.0
  Command: retain
  Compression: none
  Encryption: none
  Output: text
  Timestamp: 20250625055654
Outcome:
  Status: true
  Time: 00:00:0.1032
Request:
  Backup: newest
  Cascade: true
  Server: primary
Response:
  BackupSize: 0.00B
  Backups:
    - 20250625055547
    - 20250625055528
    - 20250625055517
  BiggestFileSize: 0.00B
  Cascade: true
  Comments: ''
  Compression: none
  Delta: 0.00B
  Encryption: none
  Keep: true
  MajorVersion: 17
  MinorVersion: 0
  RestoreSize: 0.00B
  Server: primary
  ServerVersion: 0.19.0
  Valid: yes
  WAL: 0.00B
```

9 Restore

9.1 Restore a backup

We can restore a backup from the primary with the following command

```
pgmoneta-cli restore primary newest current /tmp
```

where

- `current` means copy the Write-Ahead Log (WAL), and restore to first stable checkpoint
- `name=X` means copy the Write-Ahead Log (WAL), and restore to the label specified
- `xid=X` means copy the Write-Ahead Log (WAL), and restore to the XID specified
- `time=X` means copy the Write-Ahead Log (WAL), and restore to the timestamp specified
- `lsn=X` means copy the Write-Ahead Log (WAL), and restore to the Log Sequence Number (LSN) specified
- `inclusive=X` means that the restore is inclusive of the specified information
- `timeline=X` means that the restore is done to the specified information timeline
- `action=X` means which action should be executed after the restore (pause, shutdown)
- `primary` means that the cluster is setup as a primary
- `replica` means that the cluster is setup as a replica

More information

And, you will get output like

```
Header:
  ClientVersion: 0.19.0
  Command: 3
  Output: 0
  Timestamp: 20240928130406
Outcome:
  Status: true
  Time: 00:00:00
Request:
  Backup: newest
  Directory: /tmp
  Position: current
  Server: primary
Response:
  Backup: 20240928065644
  BackupSize: 8531968
  Comments: ''
  Compression: 2
  Encryption: 0
  MajorVersion: 17
```

```
  MinorVersion: 0
  RestoreSize: 48799744
  Server: primary
  ServerVersion: 0.19.0
```

This command take the latest backup and all Write-Ahead Log (WAL) segments and restore it into the `/tmp/primary-20240928065644` directory for an up-to-date copy.

9.2 Hot standby

In order to use hot standby, simply add

```
hot_standby = /your/local/hot/standby/directory
```

to the corresponding server section of `pgmoneta.conf`. **pgmoneta** will create the directory if it doesn't exist, and keep the latest backup in the defined directory.

You can also configure muptiple hot standby directories (up to 8) by providing comma-separated paths:

```
/path/to/hot/standby1,/path/to/hot/standby2,/path/to/hot/standby3
```

pgmoneta will maintain identical copies of the hot standby in all specified directories.

You can use

```
hot_standby_overrides = /your/local/hot/standby/overrides/
```

to override files in the `hot_standby` directories. The overrides will be applied to all hot_standby directories.

9.2.1 Tablespaces

By default tablespaces will be mapped to a similar path than the original one, for example `/tmp/mytblspc` becomes `/tmp/mytblspchs`.

However, you can use the directory name to map it to another directory, like

```
hot_standby_tablespaces = /tmp/mytblspc->/tmp/mybcktblspc
```

You can also use the `OID` for the key part, like

```
hot_standby_tablespaces = 16392->/tmp/mybcktblspc
```

Multiple tablespaces can be specified using a `,` between them.

10 Prometheus / Grafana

pgmoneta support Prometheus metrics.

We enabled the Prometheus metrics in the original configuration by setting

```
metrics = 5001
```

in `pgmoneta.conf`.

10.1 Access Prometheus metrics

You can now access the metrics via

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

10.2 Metrics

The following metrics are available.

pgmoneta_state

The state of pgmoneta

pgmoneta_version

The version of pgmoneta

Attribute	Description
version	The version of pgmoneta

pgmoneta_logging_info

The number of INFO logging statements

pgmoneta_logging_warn

The number of WARN logging statements

pgmoneta_logging_error

The number of ERROR logging statements

pgmoneta_logging_fatal

The number of FATAL logging statements

pgmoneta_retention_days

The retention days of pgmoneta

pgmoneta_retention_weeks

The retention weeks of pgmoneta

pgmoneta_retention_months

The retention months of pgmoneta

pgmoneta_retention_years

The retention years of pgmoneta

pgmoneta_retention_server

The retention of a server

Attribute	Description
name	The server identifier
parameter	The day, week, month or year

pgmoneta_compression

The compression used

pgmoneta_used_space

The disk space used for pgmoneta

pgmoneta_free_space

The free disk space for pgmoneta

pgmoneta_total_space

The total disk space for pgmoneta

pgmoneta_wal_shipping

The disk space used for WAL shipping for a server

Attribute	Description
name	The server identifier

pgmoneta_wal_shipping_used_space

The disk space used for WAL shipping of a server

Attribute	Description
name	The server identifier

pgmoneta_wal_shipping_free_space

The free disk space for WAL shipping of a server

Attribute	Description
name	The server identifier

pgmoneta_wal_shipping_total_space

The total disk space for WAL shipping of a server

Attribute	Description
name	The server identifier

pgmoneta_workspace

The disk space used for workspace for a server

Attribute	Description
name	The server identifier

pgmoneta_workspace_free_space

The free disk space for workspace of a server

Attribute	Description
name	The server identifier

pgmoneta_workspace_total_space

The total disk space for workspace of a server

Attribute	Description
name	The server identifier

pgmoneta_hot_standby

The disk space used for hot standby for a server

Attribute	Description
name	The server identifier

pgmoneta_hot_standby_free_space

The free disk space for hot standby of a server

Attribute	Description
name	The server identifier

pgmoneta_hot_standby_total_space

The total disk space for hot standby of a server

Attribute	Description
name	The server identifier

pgmoneta_server_timeline

The current timeline a server is on

Attribute	Description
name	The server identifier

pgmoneta_server_parent_tli

The parent timeline of a timeline on a server

Attribute	Description
name	The server identifier
tli	

pgmoneta_server_timeline_switchpos

The WAL switch position of a timeline on a server (showed in hex as a parameter)

Attribute	Description
name	The server identifier
tli	
walpos	

pgmoneta_server_workers

The numbeer of workers for a server

Attribute	Description
name	The server identifier

pgmoneta_server_online

Is the server online ?

Attribute	Description
name	The server identifier

pgmoneta_server_primary

Is the server a primary ?

Attribute	Description
name	The server identifier

pgmoneta_server_valid

Is the server in a valid state

Attribute	Description
name	The server identifier

pgmoneta_wal_streaming

The WAL streaming status of a server

Attribute	Description
name	The server identifier

pgmoneta_server_operation_count

The count of client operations of a server

Attribute	Description
name	The server identifier

pgmoneta_server_failed_operation_count

The count of failed client operations of a server

Attribute	Description
name	The server identifier

pgmoneta_server_last_operation_time

The time of the latest client operation of a server

Attribute	Description
name	The server identifier

pgmoneta_server_last_failed_operation_time

The time of the latest failed client operation of a server

Attribute	Description
name	The server identifier

pgmoneta_server_checksums

Are checksums enabled

Attribute	Description
name	The server identifier

pgmoneta_server_summarize_wal

Is summarize_wal enabled

Attribute	Description
name	The server identifier

pgmoneta_server_extensions_detected

The number of extensions detected on server

Attribute	Description
name	The server identifier

pgmoneta_server_extension

Information about installed extensions on server

Attribute	Description
name	The server identifier
extension	The name of the extension
version	The version of the extension
comment	Description of the extension's functionality

pgmoneta_extension_pgmoneta_ext

Status of the pgmoneta extension

Attribute	Description
name	The server identifier
version	The version of the pgmoneta extension (or "not_installed" if not present)

pgmoneta_backup_oldest

The oldest backup for a server

Attribute	Description
name	The server identifier

pgmoneta_backup_newest

The newest backup for a server

Attribute	Description
name	The server identifier

pgmoneta_backup_valid

The number of valid backups for a server

Attribute	Description
name	The server identifier

pgmoneta_backup_invalid

The number of invalid backups for a server

Attribute	Description
name	The server identifier

pgmoneta_backup

Is the backup valid for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_version

The version of postgresql for a backup

Attribute	Description
name	The server identifier
label	The backup label

Attribute	Description
major	The PostgreSQL major version
minor	The PostgreSQL minor version

pgmoneta_backup_total_elapsed_time

The backup in seconds for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_basebackup_elapsed_time

The duration for basebackup in seconds for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_manifest_elapsed_time

The duration for manifest in seconds for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_compression_zstd_elapsed_time

The duration for zstd compression in seconds for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_compression_gzip_elapsed_time

The duration for gzip compression in seconds for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_compression_bzip2_elapsed_time

The duration for bzip2 compression in seconds for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_compression_lz4_elapsed_time

The duration for lz4 compression in seconds for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_encryption_elapsed_time

The duration for encryption in seconds for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_linking_elapsed_time

The duration for linking in seconds for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_remote_ssh_elapsed_time

The duration for remote ssh in seconds for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_remote_s3_elapsed_time

The duration for remote_s3 in seconds for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_remote_azure_elapsed_time

The duration for remote_azure in seconds for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_start_timeline

The starting timeline of a backup for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_end_timeline

The ending timeline of a backup for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_start_walpos

The starting WAL position of a backup for a server

Attribute	Description
name	The server identifier
label	The backup label
walpos	The WAL position

pgmoneta_backup_checkpoint_walpos

The checkpoint WAL position of a backup for a server

Attribute	Description
name	The server identifier
label	The backup label
walpos	The WAL position

pgmoneta_backup_end_walpos

The ending WAL position of a backup for a server

Attribute	Description
name	The server identifier
label	The backup label
walpos	The WAL position

pgmoneta_restore_newest_size

The size of the newest restore for a server

Attribute	Description
name	The server identifier

pgmoneta_backup_newest_size

The size of the newest backup for a server

Attribute	Description
name	The server identifier

pgmoneta_restore_size

The size of a restore for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_restore_size_increment

The size increment of a restore for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_size

The size of a backup for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_compression_ratio

The ratio of backup size to restore size for each backup

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_throughput

The throughput of the backup for a server (MB/s)

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_basebackup_mbs

The throughput of the basebackup for a server (MB/s)

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_manifest_mbs

The throughput of the manifest for a server (MB/s)

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_compression_zstd_mbs

The throughput of the zstd compression for a server (MB/s)

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_compression_gzip_mbs

The throughput of the gzip compression for a server (MB/s)

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_compression_bzip2_mbs

The throughput of the bzip2 compression for a server (MB/s)

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_compression_lz4_mbs

The throughput of the lz4 compression for a server (MB/s)

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_encryption_mbs

The throughput of the encryption for a server (MB/s)

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_linking_mbs

The throughput of the linking for a server (MB/s)

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_remote_ssh_mbs

The throughput of the remote_ssh for a server (MB/s)

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_remote_s3_mbs

The throughput of the remote_s3 for a server (MB/s)

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_remote_azure_mbs

The throughput of the remote_azure for a server (MB/s)

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_retain

Retain backup for a server

Attribute	Description
name	The server identifier
label	The backup label

pgmoneta_backup_total_size

The total size of the backups for a server

Attribute	Description
name	The server identifier

pgmoneta_wal_total_size

The total size of the WAL for a server

Attribute	Description
name	The server identifier

pgmoneta_total_size

The total size for a server

Attribute	Description
name	The server identifier

pgmoneta_active_backup

Is there an active backup for a server

Attribute	Description
name	The server identifier

pgmoneta_active_restore

Is there an active restore for a server

Attribute	Description
name	The server identifier

pgmoneta_active_archive

Is there an active archiving for a server

Attribute	Description
name	The server identifier

pgmoneta_active_delete

Is there an active delete for a server

Attribute	Description
name	The server identifier

pgmoneta_active_retention

Is there an active archiving for a server

Attribute	Description
name	The server identifier

pgmoneta_current_wal_file

The current streaming WAL filename of a server

Attribute	Description
name	The server identifier
file	The WAL file name

pgmoneta_current_wal_lsn

The current WAL log sequence number

Attribute	Description
name	The server identifier
lsn	The Logical Sequence Number

10.3 Transport Level Security support

To add TLS support for Prometheus metrics, first we need a self-signed certificate. 1. Generate CA key and certificate

```
openssl genrsa -out ca.key 2048
openssl req -x509 -new -nodes -key ca.key -sha256 -days 3650 -out ca.crt -
    subj "/CN=My Local CA"
```

2. Generate server key and CSR

```
openssl genrsa -out server.key 2048
openssl req -new -key server.key -out server.csr -subj "/CN=localhost"
```

3. Create a config file for Subject Alternative Name

```
cat > server.ext << EOF
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment,
    dataEncipherment
subjectAltName = @alt_names

[alt_names]
DNS.1 = localhost
IP.1 = 127.0.0.1
EOF
```

4. Sign the server certificate with our CA

```
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial
    -out server.crt -days 3650 -sha256 -extfile server.ext
```

5. Generate client key and certificate

```
openssl genrsa -out client.key 2048
openssl req -new -key client.key -out client.csr -subj "/CN=Client
   Certificate"
openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -CAcreateserial
   -out client.crt -days 3650 -sha256
```

6. Create PKCS#12 file (Optional, needed for browser import)

```
openssl pkcs12 -export -out client.p12 -inkey client.key -in client.crt -
   certfile ca.crt -passout pass:<your_password>
```

Edit `pgmoneta.conf` to add the following keys under pgmoneta section:

```
[pgmoneta]
.
.
.
metrics_cert_file=<path_to_server_cert_file>
metrics_key_file=<path_to_server_key_file>
metrics_ca_file=<path_to_ca_file>
```

You can now access the metrics at `https://localhost:5001` using curl as follows:

```
curl -v -L "https://localhost:5001" --cacert <path_to_ca_file> --cert <
   path_to_client_cert_file> --key <path_to_client_key_file>
```

(Optional) If you want to access the page through the browser: - First install the certificates on your system - For Fedora: “‘ # Create directory if it doesn’t exist sudo mkdir -p /etc/pki/ca-trust/source/anchors/

```
# Copy CA cert to the trust store
sudo cp ca.crt /etc/pki/ca-trust/source/anchors/

# Update the CA trust store
sudo update-ca-trust extract
‘‘‘

- For Ubuntu:
‘‘‘
# Copy the CA certificate to the system certificate store
sudo cp ca.crt /usr/local/share/ca-certificates/

# Update the CA certificate store
sudo update-ca-certificates
‘‘‘

- For MacOS:
    - Open Keychain Access and import the certificate file
    - Set the certificate to "Always Trust"
```

- For browsers like Firefox
  - Go to Menu → Preferences → Privacy & Security
  - Scroll down to “Certificates” section and click “View Certificates”
  - Go to “Authorities” tab and click “Import”
  - Select your `ca.crt` file
  - Check “Trust this CA to identify websites” and click OK
  - Go to “Your Certificates” tab
  - Click “Import” and select the `client.p12` file
  - Enter the password you set when creating the PKCS#12 file
- For browsers like Chrome/Chromium
  - For client certificates, go to Settings → Privacy and security → Security → Manage certificates
  - Click on “Import” and select your `client.p12` file
  - Enter the password you set when creating it

You can now access metrics at `https://localhost:5001`

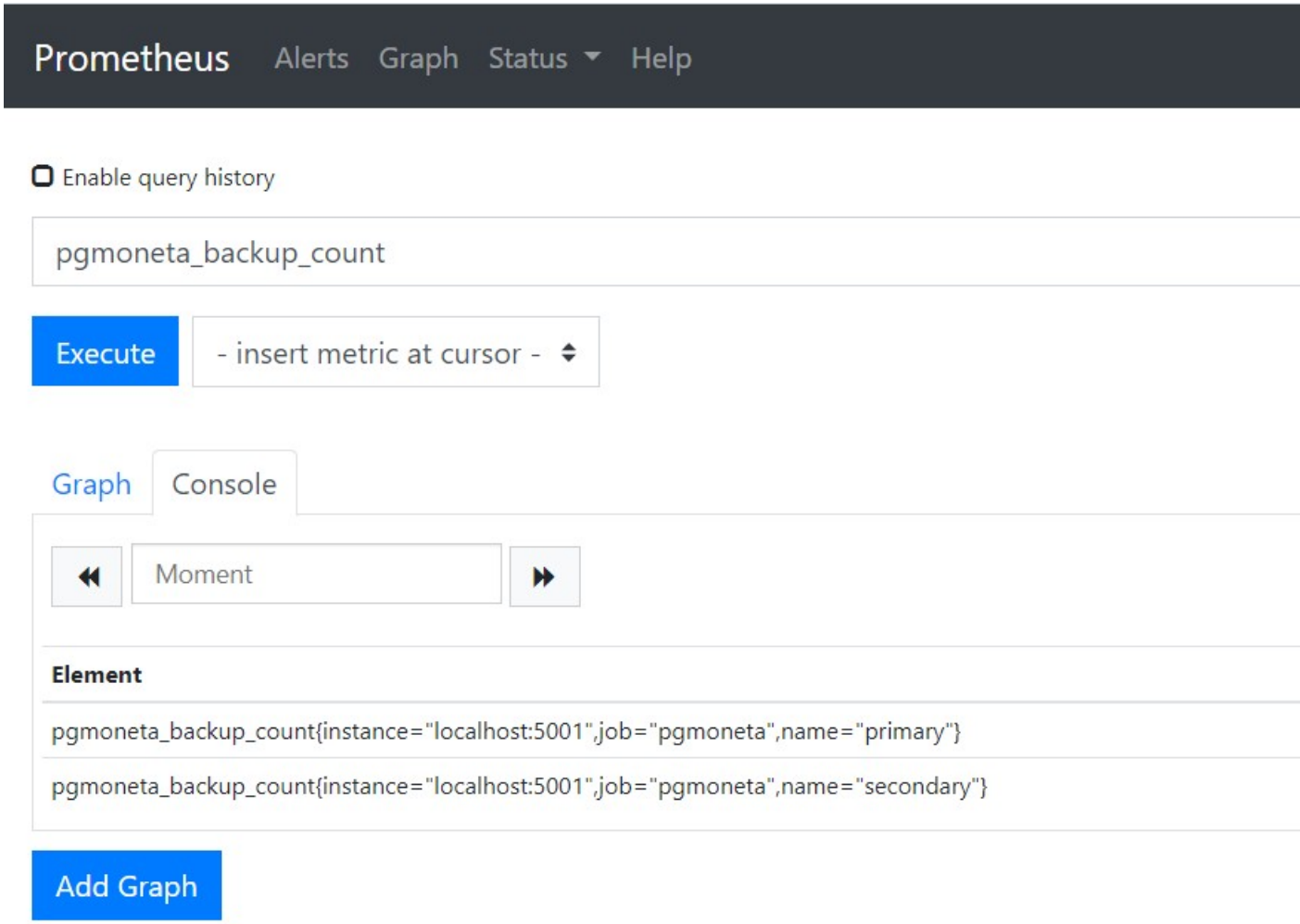
10.4 Grafana

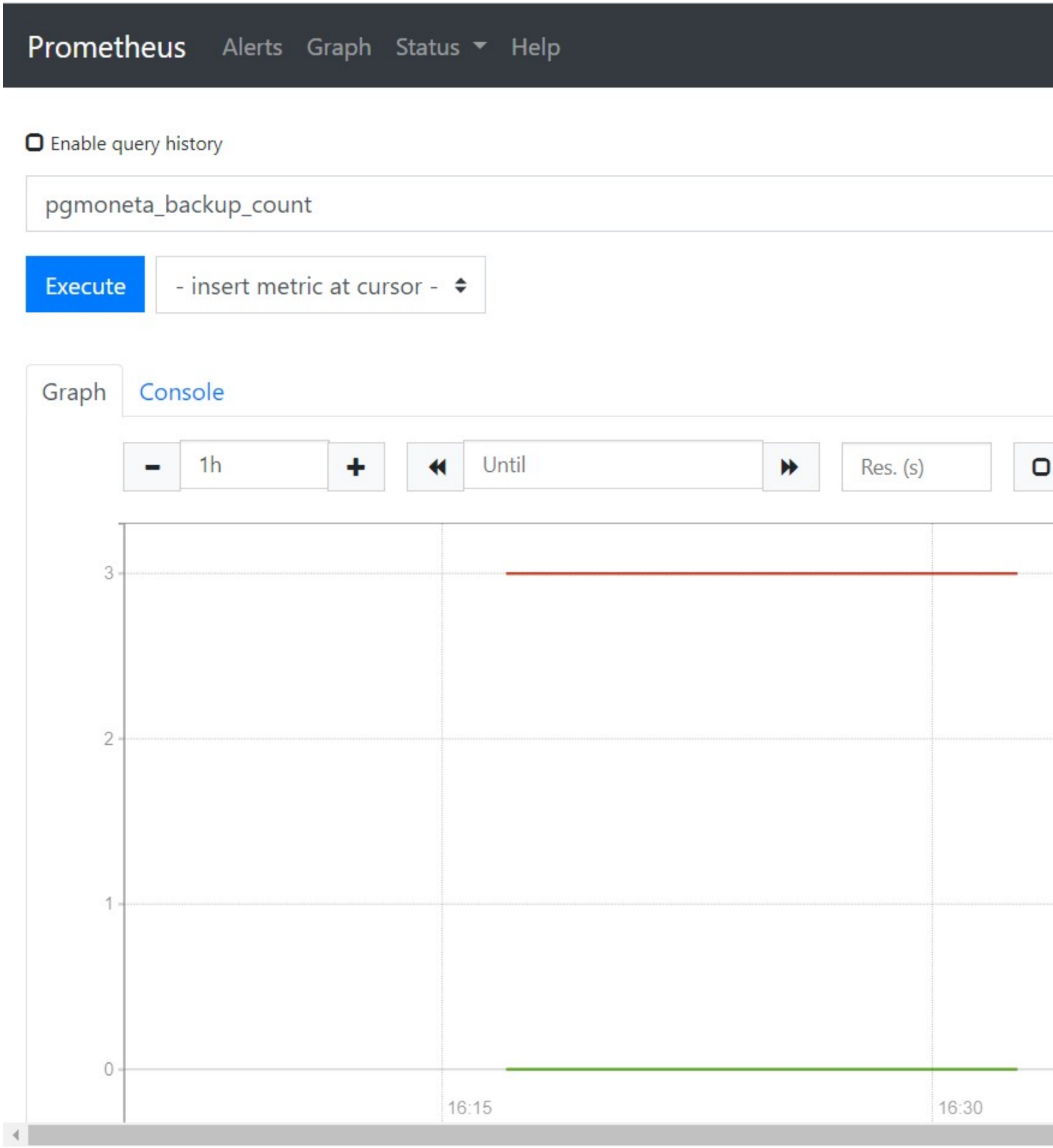
Enable the endpoint by adding

```
scrape_configs:
  - job_name: 'pgmoneta'
    metrics_path: '/metrics'
    static_configs:
      - targets: ['localhost:5001']
```

to the Grafana configuration.

Then the Prometheus service will query your **pgmoneta** metrics every 15 seconds and package them as time-series data. You can query your **pgmoneta** metrics and watch their changes as time passed in Prometheus web page (default port is `9090`).

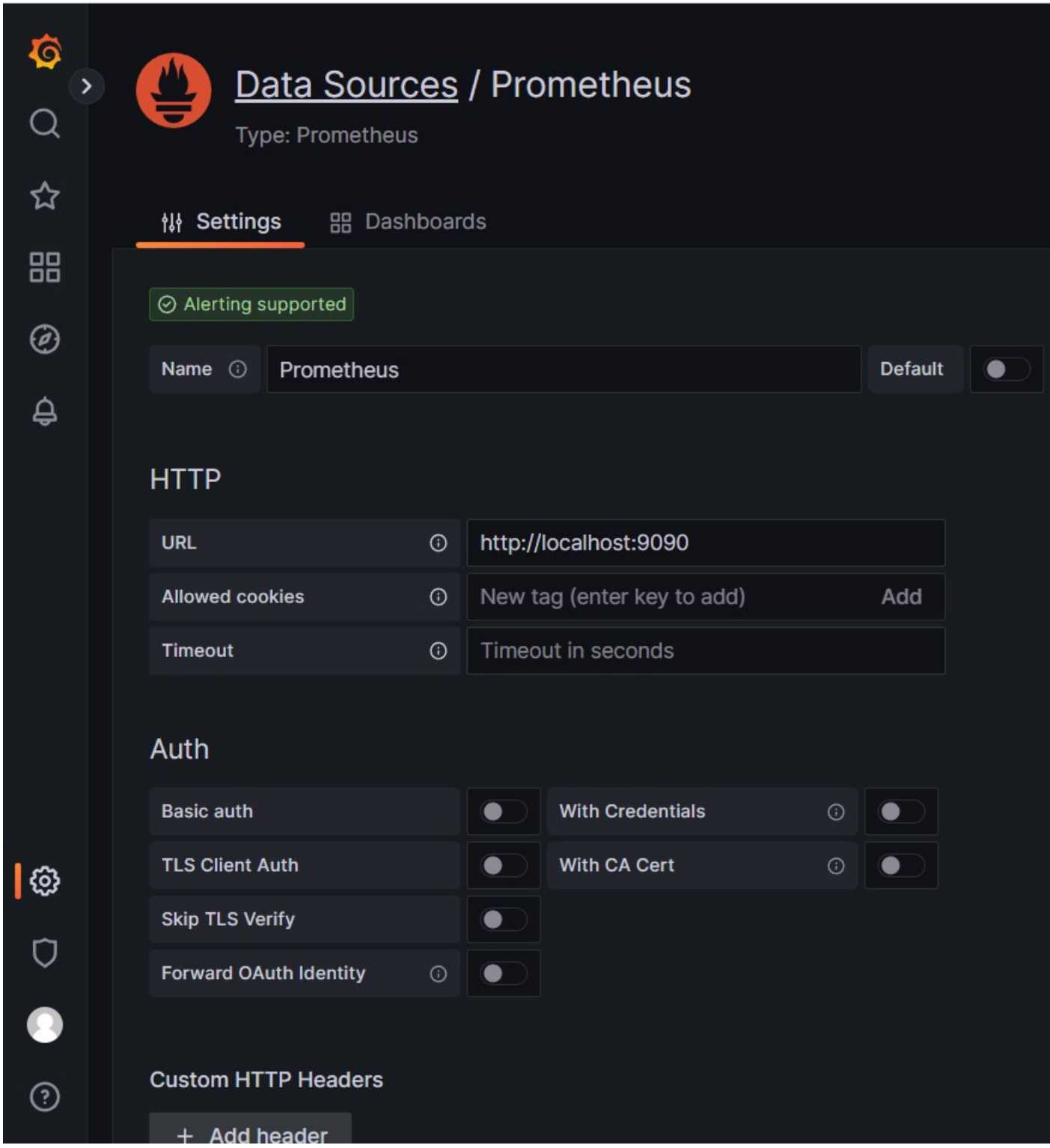

Prometheus
Alerts
Graph
Status
Help
Enable query history
pgmoneta_backup_count
Execute
- insert metric at cursor -
Graph
Console
Moment
Element
pgmoneta_backup_count{instance="localhost:5001",job="pgmoneta",name="primary"}
pgmoneta_backup_count{instance="localhost:5001",job="pgmoneta",name="secondary"}
Add Graph


Prometheus
Alerts
Graph
Status
Help
Enable query history
pgmoneta_backup_count
Execute
- insert metric at cursor -
Graph
Console
1h
Until
Res. (s)
3
2
1
0
16:15
16:30

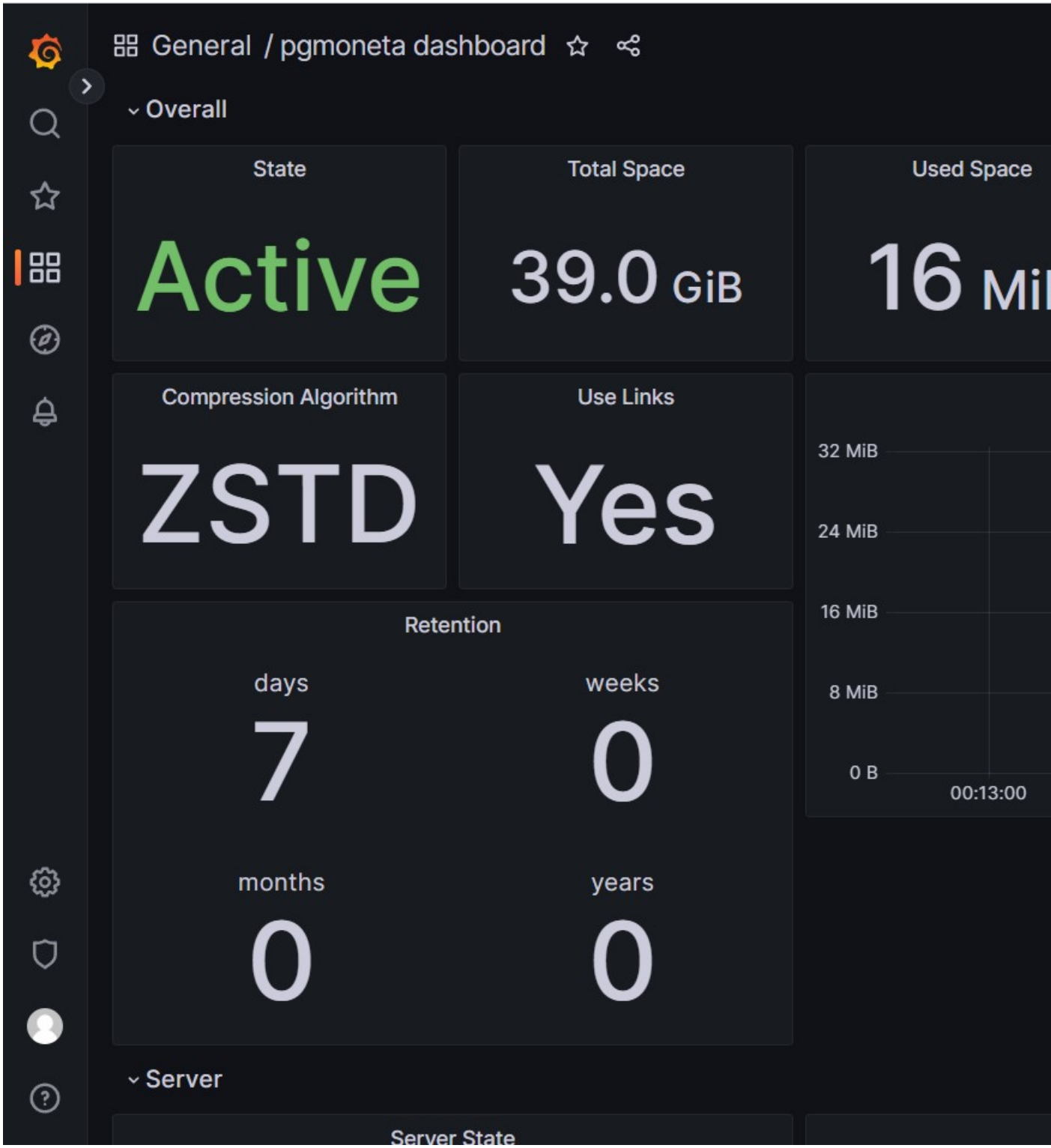
10.4.1 Import a Grafana dashboard

Although Prometheus provides capacity of querying and monitoring metrics, we can not customize graphs for each metric and provide a unified view.

As a result, we use Grafana to help us manage all graphs together. First of all, we should install Grafana in the computer you need to monitor **pgmoneta** metrics. You can browse Grafana web page with default port `3000`, default user `admin` and default password `admin`. Then you can create Prometheus data source of **pgmoneta**.



Finally you can create dashboard by importing `contrib/grafana/dashboard.json` and monitor metrics about **pgmoneta**.


General / pgmoneta dashboard
Overall
State
Active
Total Space
39.0 GiB
Used Space
16 Mi
Compression Algorithm
ZSTD
Use Links
Yes
32 MiB
24 MiB
16 MiB
8 MiB
0 B
00:13:00
Retention
days
7
weeks
0
months
0
years
0
Server
Server State

11 Adminstration access

You can access **pgmoneta** from a remote machine if you enable access.

11.1 Configuration

First, you need to enable remote access by adding

```
management = 5002
```

in `pgmoneta.conf` in the `[pgmoneta]` section.

11.2 Administrators

Next, you will need to add one or more administrators in `pgmoneta_admins.conf` through

```
pgmoneta-admin -f /etc/pgmoneta/pgmoneta_admins.conf user add
```

for example with a user name of `admin` and `secretpassword` as the password.

11.3 Restart pgmoneta

You have to restart **pgmoneta** to make the changes take effect.

11.4 Connect to pgmoneta

Then you will use the `pgmoneta-cli` tool to access **pgmoneta** with

```
pgmoneta-cli -h myhost -p 5002 -U admin status
```

to execute the `status` command after have entered the password.

11.5 Transport Level Security support

You can security the administration level interface by using Transport Level Security (TLS).

It is done by setting the following options,

```
[pgmoneta]
tls_cert_file=/path/to/server.crt
tls_key_file=/path/to/server.key
tls_ca_file=/path/to/root.crt

...
```

in `pgmoneta.conf`.

The client side setup must go into `~/.pgmoneta/` with the following files

```
~/.pgmoneta/pgmoneta.key
~/.pgmoneta/pgmoneta.crt
~/.pgmoneta/root.crt
```

They must have 0600 permission.

12 Shutdown

You can test the status of **pgmoneta** and shutdown either locally or from a remote machine.

12.1 ping

You can check if **pgmoneta** is running by

```
pgmoneta-cli ping
```

and check the output.

12.2 mode

pgmoneta detects when a server is down. You can bring a server online or offline using the mode command, like

```
pgmoneta-cli mode primary offline
```

or

```
pgmoneta-cli mode primary online
```

pgmoneta will keep basic services running for an offline server such that you can verify a backup or do a restore.

12.3 shutdown

You can shutdown **pgmoneta** by

```
pgmoneta-cli shutdown
```

and check the output.

13 Docker

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

13.1 Prerequisites

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

13.2 The image

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
```

Then, restart PostgreSQL for the changes to take effect:

```
sudo systemctl restart postgresql
```

Clone the Repository

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

Build the Docker Image

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

Using Docker

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

Using Podman

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

Rocky Linux 9-based image

Using Docker

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

Using Podman

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

Run pgmoneta as a Docker Container

Once the image is built, run the container using:

Using Docker

```
docker run -d --name pgmoneta --network host pgmoneta:latest
```

Using Podman

```
podman run -d --name pgmoneta --network host pgmoneta:latest
```

Verify the Container

Check if the container is running:

Using Docker

```
docker ps | grep pgmoneta
```

Using Podman

```
podman ps | grep pgmoneta
```

Check logs for any errors:

Using Docker

```
docker logs pgmoneta
```

Using Podman

```
podman logs pgmoneta
```

You can also inspect the exposed metrics at:

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

You can stop the container using

Using Docker

```
docker stop pgmoneta
```

Using Podman

```
podman stop pgmoneta
```

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

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

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

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

14 SSH

14.1 Prerequisites

First of all, you need to have a remote server where you can store your backups on.

Lets take an EC2 instance as an example, after launching an EC2 instance you need to add new user account with SSH access to the EC2 instance:

1. Connect to your Linux instance using SSH.
2. Use the adduser command to add a new user account to an EC2 instance (replace new_user with the new account name).

```
sudo adduser new_user --disabled-password
```

3. Change the security context to the new_user account so that folders and files you create have the correct permissions:

```
sudo su - new_user
```

4. Create a .ssh directory in the new_user home directory and use the chmod command to change the .ssh directory's permissions to 700:

```
mkdir .ssh && chmod 700 .ssh
```

5. Use the touch command to create the authorized_keys file in the .ssh directory and use the chmod command to change the .ssh/authorized_keys file permissions to 600:

```
touch .ssh/authorized_keys && chmod 600 .ssh/authorized_keys
```

6. Retrieve the public key for the key pair in your local computer:

```
cat ~/.ssh/id_rsa.pub
```

7. In the EC2 instance, run the cat command in append mode:

```
cat >> .ssh/authorized_keys
```

8. Paste the public key into the .ssh/authorized_keys file and then press Enter.
9. Press and hold Ctrl+d to exit cat and return to the command line session prompt.

To verify that the new user can use SSH to connect to the EC2 instance, run the following command from a command line prompt on your local computer:

```
ssh new_user@public_dns_name_of_EC2_instance
```

14.2 Modify the pgmoneta configuration

You need to create a directory on your remote server where backups can be stored in.

In addition, your local computer needs to have a storage space for 1 backup.

Change `pgmoneta.conf` to add

```
storage_engine = ssh
ssh_hostname =  your-public_dns_name_of_EC2_instance
ssh_username = new_user
ssh_base_dir = the-path-of-the-directory-where-backups-stored-in
```

under the `[pgmoneta]` section.

15 Azure

15.1 Prerequisites

First of all, you need to have an Azure account, an Azure storage account and a blob container.

A container organizes a set of blobs, similar to a directory in a file system. A storage account can include an unlimited number of containers, and a container can store an unlimited number of blobs.

To create an Azure storage account with the Azure portal:

1. Sign in to the Azure portal.
2. From the left portal menu, select Storage accounts to display a list of your storage accounts. If the portal menu isn't visible, click the menu button to toggle it on.
3. On the Storage accounts page, select Create.
4. On the Basics tab, provide a resource group name and storage account name. You can go for the default settings of the other fields.
5. Choose Next: Advanced.
6. On the Advanced tab, you can configure additional options and modify default settings for your new storage account. You can go for the default settings.
7. Choose Next: Networking.
8. On the Networking tab, you can go for the default settings.
9. Choose Next: Data protection.
10. On the Data protection tab, you can for the default settings.
11. Choose Next: Encryption.
12. On the Encryption tab, you can for the default settings.
13. Choose Next: Tags.
14. Choose Next: Review to see all of the choices you made up to this point. When you are ready to proceed, choose Create.

To create a blob container with the Azure portal:

1. In the navigation pane for the storage account, scroll to the `Data storage` section and select Containers.
2. Within the Containers pane, select the + `Container` button to open the New container pane.

3. Within the New Container pane, provide a Name for your new container.
4. Select Create to create the container.

To get the Azure storage account shared key which is required for pgmoneta configuration:

1. In the navigation pane for the storage account, scroll to the `Security + networking` section and select Access Keys.
2. Under key1, find the Key value. Select the Copy button to copy the account key.

You can use either of the two keys to access Azure Storage, but in general it's a good practice to use the first key, and reserve the use of the second key for when you are rotating keys.

15.2 Modify the pgmoneta configuration

You need to have a storage space for 1 backup on your local computer.

Change `pgmoneta.conf` to add

```
storage_engine = azure
azure_storage_account = the-storage-account-name
azure_container = the-container-name
azure_shared_key = the-storage-account-shared-key
azure_base_dir = directory-where-backups-will-be-stored-in
```

under the `[pgmoneta]` section.

16 S3

16.1 Prerequisites

First of all, you need to have an AWS account, an IAM user and S3 bucket.

To create an IAM user:

1. Sign in to the AWS Management Console and open the IAM console.
2. In the navigation pane, choose Users and then choose Add users.
3. Type the user name for the new user.
4. Select the type of access to be both programmatic access and access to the AWS Management Console.
5. Choose Next: Permissions.
6. On the Set permissions page, select attach existing policies directly, search for AmazonS3FullAccess and choose it, then choose Next: Review, then choose Add permissions.
7. Choose Next: Tags.
8. Choose Next: Review to see all of the choices you made up to this point. When you are ready to proceed, choose Create user.
9. To view the users' access keys (access key IDs and secret access keys), choose Show next to each password and access key that you want to see. To save the access keys, choose Download .csv and then save the file to a safe location.

You are now ready to create a S3 bucket, To create a S3 bucket:

1. Sign in to the AWS Management Console using your IAM user credentials and open the Amazon S3 console.
2. Choose Create bucket.
3. In Bucket name, enter a name for your bucket.
4. In Region, choose the AWS Region where you want the bucket to reside.
5. Keep the default values as it is and Choose Create bucket.

16.2 Modify the pgmoneta configuration

You need to have a storage space for 1 backup on your local computer.

Change `pgmoneta.conf` to add

```
storage_engine = s3
s3_aws_region = the-aws-region
s3_access_key_id = your-access-key-id-from-the-downloaded-file
s3_secret_access_key = your-secret-access-key-from-the-downloaded-file
s3_bucket = your-s3-bucket-name
s3_base_dir = directory-where-backups-will-be-stored-in
```

under the `[pgmoneta]` section.

17 Developer information

17.1 C programming

pgmoneta is developed using the C programming language so it is a good idea to have some knowledge about the language before you begin to make changes.

There are books like,

- C in a Nutshell
- 21st Century C

that can help you

17.1.1 Debugging

In order to debug problems in your code you can use gdb, or add extra logging using the `pgmoneta_log_XYZ()` API

17.2 Git guide

Here are some links that will help you

- How to Squash Commits in Git
- ProGit book

17.2.1 Basic steps

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>/pgmoneta.git
```

Add upstream

Do

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

```
./uncrustify.sh
```

to format the source code

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`.

17.3 Architecture

17.3.1 Overview

pgmoneta use a process model (`fork()`), where each process handles one Write-Ahead Log (WAL) receiver to PostgreSQL.

The main process is defined in main.c.

Backup is handled in backup.h (backup.c).

Restore is handled in restore.h (restore.c) with linking handled in link.h (link.c).

Archive is handled in achv.h (archive.c) backed by restore.

Write-Ahead Log is handled in wal.h (wal.c).

Backup information is handled in info.h (info.c).

Retention is handled in retention.h (retention.c).

Compression is handled in gzip_compression.h (gzip_compression.c), lz4_compression.h (lz4_compression.c), zstandard_compression.h (zstandard_compression.c), and bzip2_compression.h (bzip2_compression.c).

Encryption is handled in aes.h (aes.c).

17.3.2 Shared memory

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

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

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

17.3.3 Network and messages

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

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

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

17.3.4 Memory

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

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).

17.3.5 Management

pgmoneta 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 `pgmoneta-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
`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).

17.3.6 libev usage

libev is used to handle network interactions, which is “activated” upon an `EV_READ` event.

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.

17.3.7 Signals

The main process of **pgmoneta** supports the following signals `SIGTERM`, `SIGINT` and `SIGALRM` as a mechanism for shutting down. The `SIGABRT` is used to request a core dump (`abort()`).

The `SIGHUP` signal will trigger a reload of the configuration.

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

17.3.8 Reload

The `SIGHUP` signal will trigger a reload of the configuration.

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

- `hugepage`
- `libev`
- `log_path`
- `log_type`
- `unix_socket_dir`
- `pidfile`

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

The `SIGHUP` signal will trigger a full reload of the configuration. When `SIGHUP` is received, **pgmoneta** 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 specific services (metrics and management) using the current in-memory configuration. This signal is automatically triggered by `pgmoneta-cli conf set` when

applying runtime configuration changes that don't require a full restart. 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.

Services affected by SIGUSR1: - **Metrics service**: Restarted when `metrics` port is changed or enabled/disabled - **Management service**: Restarted when `management` port is changed or enabled/disabled

17.3.9 Prometheus

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

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.

17.3.10 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

Level	Description
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)`

17.3.11 Protocol

The protocol interactions can be debugged using Wireshark or pgprtdbg.

17.4 Encryption

17.4.1 Overview

AES Cipher block chaining (CBC) mode and AES Counter (CTR) mode are supported in **pgmoneta**. The default setup is no encryption.

CBC is the most commonly used and considered save mode. Its main drawbacks are that encryption is sequential (decryption can be parallelized).

Along with CBC, CTR mode is one of two block cipher modes recommended by Niels Ferguson and Bruce Schneier. Both encryption and decryption are parallelizable.

Longer the key length, safer the encryption. However, with 20% (192 bit) and 40% (256 bit) extra workload compare to 128 bit.

17.4.2 Encryption Configuration

`none`: No encryption (default value)

`aes` | `aes-256` | `aes-256-cbc`: AES CBC (Cipher Block Chaining) mode with 256 bit key length

`aes-192` | `aes-192-cbc`: AES CBC mode with 192 bit key length

`aes-128` | `aes-128-cbc`: AES CBC mode with 128 bit key length

`aes-256-ctr`: AES CTR (Counter) mode with 256 bit key length

`aes-192-ctr`: AES CTR mode with 192 bit key length

`aes-128-ctr`: AES CTR mode with 128 bit key length

17.4.3 Encryption / Decryption CLI Commands

decrypt

Decrypt the file in place, remove encrypted file after successful decryption.

Command

```
pgmoneta-cli decrypt <file>
```

encrypt

Encrypt the file in place, remove unencrypted file after successful encryption.

Command

```
pgmoneta-cli encrypt <file>
```

17.4.4 Benchmark

Check if your CPU have AES-NI

```
cat /proc/cpuinfo | grep aes
```

Query number of cores on your CPU

```
lscpu | grep '^CPU(s):'
```

By default openssl using AES-NI if the CPU have it.

```
openssl speed -elapsed -evp aes-128-cbc
```

Speed test with explicit disabled AES-NI feature

```
OPENSSL_ia32cap="~0x200000200000000" openssl speed -elapsed -evp aes-128-
   cbc
```

Test decrypt

```
openssl speed -elapsed -decrypt -evp aes-128-cbc
```

Speed test with 8 cores

```
openssl speed -multi 8 -elapsed -evp aes-128-cbc
```

```
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         39 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  12
  On-line CPU(s) list:   0-11
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
    CPU family:          6
    Model:               158
    Thread(s) per core:  2
    Core(s) per socket:  6
    Socket(s):           1
    Stepping:            10
    BogoMIPS:            5183.98
    Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
       pge mca cmov pat pse36 clflush mmx fxsr sse sse2 s
                         s ht syscall nx pdpe1gb rdtscp lm constant_tsc
                            rep_good nopl xtopology cpuid pni pclmulqdq
                            vmx ssse
                         3 fma cx16 pcid sse4_1 sse4_2 movbe popcnt aes
                            xsave avx f16c rdrand hypervisor lahf_lm abm 3
                            dnowpr
                         efetch invpcid_single pti ssbd ibrs ibpb stibp
                            tpr_shadow vnmi ept vpid ept_ad fsgsbase bmi1
                            avx2 s
                         mep bmi2 erms invpcid rdseed adx smap clflushopt
                            xsaveopt xsavec xgetbv1 xsaves flush_l1d
                            arch_capa
                         bilities
Virtualization features:
  Virtualization:        VT-x
  Hypervisor vendor:     Microsoft
  Virtualization type:   full
Caches (sum of all):
  L1d:                   192 KiB (6 instances)
  L1i:                   192 KiB (6 instances)
  L2:                    1.5 MiB (6 instances)
  L3:                    12 MiB (1 instance)
Vulnerabilities:
  Itlb multihit:         KVM: Mitigation: VMX disabled
  L1tf:                  Mitigation; PTE Inversion; VMX conditional cache
     flushes, SMT vulnerable
  Mds:                   Vulnerable: Clear CPU buffers attempted, no
     microcode; SMT Host state unknown
  Meltdown:              Mitigation; PTI
  Spec store bypass:     Mitigation; Speculative Store Bypass disabled via
      prctl and seccomp
```

```
  Spectre v1:             Mitigation; usercopy/swapgs barriers and __user
     pointer sanitization
  Spectre v2:             Mitigation; Full generic retpoline, IBPB
     conditional, IBRS_FW, STIBP conditional, RSB filling
  Srbds:                  Unknown: Dependent on hypervisor status
  Tsx async abort:        Not affected

openssl version: 3.0.5
built on: Tue Jul  5 00:00:00 2022 UTC
options: bn(64,64)
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -O2 -flto=auto -ffat-
    lto-objects -fexceptions -g -grecord-gcc-switches -pipe -Wall -Werror=
    format-security -Wp,-D_FORTIFY_SOURCE=2 -Wp,-D_GLIBCXX_ASSERTIONS -
    specs=/usr/lib/rpm/redhat/redhat-hardened-cc1 -fstack-protector-strong
    -specs=/usr/lib/rpm/redhat/redhat-annobin-cc1  -m64  -mtune=generic -
    fasynchronous-unwind-tables -fstack-clash-protection -fcf-protection -
    O2 -flto=auto -ffat-lto-objects -fexceptions -g -grecord-gcc-switches -
    pipe -Wall -Werror=format-security -Wp,-D_FORTIFY_SOURCE=2 -Wp,-
    D_GLIBCXX_ASSERTIONS -specs=/usr/lib/rpm/redhat/redhat-hardened-cc1 -
    fstack-protector-strong -specs=/usr/lib/rpm/redhat/redhat-annobin-cc1 -
    m64 -mtune=generic -fasynchronous-unwind-tables -fstack-clash-
    protection -fcf-protection -Wa,--noexecstack -Wa,--generate-missing-
    build-notes=yes -specs=/usr/lib/rpm/redhat/redhat-hardened-ld -specs=/
    usr/lib/rpm/redhat/redhat-annobin-cc1 -DOPENSSL_USE_NODELETE -DL_ENDIAN
     -DOPENSSL_PIC -DOPENSSL_BUILDING_OPENSSL -DZLIB -DNDEBUG -DPURIFY -
    DDEVRANDOM="\"/dev/urandom\"" -DSYSTEM_CIPHERS_FILE="/etc/crypto-
    policies/back-ends/openssl.config"
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192
    bytes  16384 bytes
AES-128-CBC *   357381.06k   414960.06k   416301.23k   416687.10k
    416175.45k   416268.29k
AES-128-CBC     902160.83k  1496344.68k  1514778.62k  1555236.52k
    1542537.22k  1569259.52k
AES-128-CBC d   909710.79k  2941259.46k  5167110.31k  5927086.76k
    6365967.70k  6349198.68k
AES-128-CBC 8  3912786.36k  8042348.31k  9870507.86k 10254096.38k
    10653332.82k 10310331.05k
AES-128-CBC 8d 4157037.26k 12337480.36k 26613686.27k 29902703.27k
    32306793.13k 31440366.25k

AES-128-CTR *   146971.83k   165696.94k   574871.64k   634507.61k
    676448.94k   668139.52k
AES-128-CTR     887783.06k  2255074.22k  4800168.19k  5930596.01k
    6431110.49k  6376062.98k
AES-128-CTR d   793432.63k  2181439.06k  4541298.09k  5743022.42k
    6480090.45k  6271221.76k
AES-128-CTR 8  3833975.47k 10832239.55k 23757293.40k 28413146.79k
    30514317.99k 30092356.27k
AES-128-CTR 8d 3456838.44k  9749773.91k 22107652.18k 27229352.28k
    30703026.18k 29387025.07k
```

```
AES-192-CBC      853380.50k  1238507.90k  1299788.12k  1257189.03k
    1272591.70k  1271840.77k
AES-192-CBC d    876094.29k  2843770.82k  4523019.52k  5177496.92k
    5442652.84k  5372559.36k
AES-192-CTR      869039.84k  2285946.18k  4229439.91k  5049118.04k
    5422994.77k  5309748.57k
AES-192-CTR d    789470.51k  2177050.05k  4194812.76k  4935891.63k
    5257865.90k  5323046.91k

AES-256-CBC      834298.24k  1100648.64k  1117826.90k  1104301.40k
    1130657.11k  1097285.63k
AES-256-CBC d    843079.68k  2714917.67k  4084088.23k  4510005.59k
    4557821.27k  4594783.57k
AES-256-CTR      811325.74k  2222582.89k  3749333.08k  4412143.27k
    4640549.55k  4554828.46k
AES-256-CTR d    730844.97k  2081179.20k  3673258.15k  4346793.64k
    4515722.58k  4594335.74k
```

*: AES-NI disabled; 8: 8 cores; d: decryption

17.5 RPM

pgmoneta can be built into a RPM for Fedora systems.

17.5.1 Requirements

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

17.5.2 Setup RPM development

```
rpmdev-setuptree
```

17.5.3 Create source package

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

17.5.4 Create RPM package

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

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

17.6 Building pgmoneta

17.6.1 Overview

pgmoneta 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]

17.6.2 Dependencies

Before building pgmoneta 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.

17.6.3 Debug Mode

When building in Debug mode, **pgmoneta** automatically enables various compiler flags to help with debugging, including AddressSanitizer (ASAN) and UndefinedBehaviorSanitizer (UBSAN) support when available.

The Debug mode can be enabled by adding -DCMAKE_BUILD_TYPE=Debug to your CMake command.

17.6.4 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+)

17.6.5 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

17.6.6 Building with Sanitizers

To build **pgmoneta** 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.

17.6.7 Running with Sanitizers

When running **pgmoneta** 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 ./pgmoneta
```

You can combine ASAN and UBSAN options:

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

17.6.8 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.

17.7 Test

Dependencies

To install all the required dependencies, simply run `<PATH_TO_PGMONETA>/pgmoneta/test/check.sh setup`. You need to install docker or podman separately.

Running Tests

To run the tests, simply run `<PATH_TO_PGMONETA>/pgmoneta/test/check.sh`. The script will build a PostgreSQL 17 image the first time you run it, and start a docker/podman container using the image (so make sure you at least have one of them installed and have the corresponding container engine started). The containerized postgres server will have a `repl` user with the replication attribute, a normal user `myuser` and a database `mydb`.

The script then starts pgmoneta and runs tests in your local environment. The tests are run locally so that you may leverage stdout to debug and the testing environment won't run into weird container environment issues, and so that we can reuse the installed dependencies and cmake cache to speed up development and debugging.

All the configuration, logs, coverage reports and data will be in `/tmp/pgmoneta-test/`, and a cleanup will run whether the script exits normally or not. pgmoneta will be force shutdown if it doesn't terminate normally. So don't worry about your local setup being tampered. The container will be stopped and removed when the script exits or is terminated.

To run one particular test case or suite (unfortunately check doesn't support running one single test at the moment), run `CK_RUN_CASE=<test_case_name> <PATH_TO_PGMONETA>/pgmoneta/test/check.sh` or `CK_RUN_SUITE=<test_case_name> <PATH_TO_PGMONETA>/pgmoneta/test/check.sh`. Alternatively, you can first export the environment variables and then run the script:

```
export CK_RUN_CASE=<test_case_name>
<PATH_TO_PGMONETA>/pgmoneta/test/check.sh
```

The environment variables will be automatically unset when the test is finished or aborted.

It is recommended that you **ALWAYS** run tests before raising PR.

Add Testcases

To add an additional testcase, go to testcases directory inside the `pgmoneta` project.

Create a `.c` file that contains the test suite and define the suite inside `/test/include/tssuite.sh`. Add the above created suite to the test runner in runner.c

Test Directory

After running the tests, you will find:

- **pgmoneta log:** `/tmp/pgmoneta-test/log/`
- **postgres log:** `/tmp/pgmoneta-test/pg_log/`, the log level is set to debug5 and has the application name (**pgmoneta**) shown in the log.
- **code coverage reports:** `/tmp/pgmoneta-test/coverage/`

If you need to create a directory runtime, create it under `/tmp/pgmoneta-test/base/`, which also contains `backup/`, `restore/`, `conf` and `workspace/`. Base directory will be cleaned up after tests are done. In `tscommon.h` you will find `TEST_BASE_DIR` and other global variables holding corresponding directories, fetched from environment variables.

Cleanup

<PATH_TO_PGMONETA>/pgmoneta/test/check.sh clean will remove the testing directory and the built image. If you are using docker, chances are it eats your disk space secretly, in that case consider cleaning up using docker system prune --volume. Use with caution though as it nukes all the docker volumes.

Port

By default, the pod exposes port 6432 for pgmoneta to connect to. This can be changed by export PGMONETA_TEST_PORT=<your-port> before running check.sh. Or you may also run PGMONETA_TEST_PORT=<your-port> ./check.sh.

Configuration

Name	Default	Value	Description
CK_RUN_CASE		test case name	Run one single test case
CK_RUN_SUITE		test suite name	Run one single test suite
PGMONETA_TEST_PORT	6432	port number	The port name pgmoneta use to connect to the db pod

17.7.1 Adding wal-related testcases

While moving towards the goal of building a complete test suite to test pgmoneta wal generation and replay mechanisms, we need to add some testcases that will generate wal files and then replay them. Currently we need to add testcases for the following wal record types:

Click to expand

- **XLOG**
  - XLOG_CHECKPOINT_SHUTDOWN
  - XLOG_CHECKPOINT_ONLINE
  - XLOG_NOOP
  - XLOG_NEXTOID
  - XLOG_SWITCH
  - XLOG_BACKUP_END
  - XLOG_PARAMETER_CHANGE
  - XLOG_RESTORE_POINT
  - XLOG_FPI
  - XLOG_FPI_FOR_HINT

  - XLOG_FPW_CHANGE
  - XLOG_END_OF_RECOVERY
  - XLOG_OVERWRITE_CONTRECORD

- **XACT**
  - XLOG_XACT_COMMIT
  - XLOG_XACT_ABORT
  - XLOG_XACT_PREPARE
  - XLOG_XACT_COMMIT_PREPARED
  - XLOG_XACT_ABORT_PREPARED
  - XLOG_XACT_ASSIGNMENT

- **SMGR**
  - XLOG_SMGR_CREATE
  - XLOG_SMGR_TRUNCATE

- **DBASE**
  - XLOG_DBASE_CREATE
  - XLOG_DBASE_DROP

- **TBLSPC**
  - XLOG_TBLSPC_CREATE
  - XLOG_TBLSPC_DROP

- **RELMAP**
  - XLOG_RELMAP_UPDATE

- **STANDBY**
  - XLOG_RUNNING_XACTS
  - XLOG_STANDBY_LOCK

- **HEAP2**
  - XLOG_HEAP2_FREEZE_PAGE
  - XLOG_HEAP2_VACUUM
  - XLOG_HEAP2_VISIBLE
  - XLOG_HEAP2_MULTI_INSERT
  - XLOG_HEAP2_PRUNE

- **HEAP**

  - XLOG_HEAP_INSERT
  - XLOG_HEAP_DELETE
  - XLOG_HEAP_UPDATE
  - XLOG_HEAP_INPLACE
  - XLOG_HEAP_LOCK
  - XLOG_HEAP_CONFIRM

- **BTREE**

  - XLOG_BTREE_INSERT_LEAF
  - XLOG_BTREE_INSERT_UPPER
  - XLOG_BTREE_INSERT_META
  - XLOG_BTREE_SPLIT_L
  - XLOG_BTREE_SPLIT_R
  - XLOG_BTREE_VACUUM
  - XLOG_BTREE_DELETE
  - XLOG_BTREE_UNLINK_PAGE
  - XLOG_BTREE_NEWROOT
  - XLOG_BTREE_REUSE_PAGE

- **HASH**

  - XLOG_HASH_INIT_META_PAGE
  - XLOG_HASH_INIT_BITMAP_PAGE
  - XLOG_HASH_INSERT
  - XLOG_HASH_ADD_OVFL_PAGE
  - XLOG_HASH_DELETE
  - XLOG_HASH_SPLIT_ALLOCATE_PAGE
  - XLOG_HASH_SPLIT_PAGE
  - XLOG_HASH_SPLIT_COMPLETE
  - XLOG_HASH_MOVE_PAGE_CONTENTS
  - XLOG_HASH_SQUEEZE_PAGE

- **GIN**

  - XLOG_GIN_CREATE_PTREE
  - XLOG_GIN_INSERT
  - XLOG_GIN_SPLIT
  - XLOG_GIN_VACUUM_PAGE
  - XLOG_GIN_DELETE_PAGE
  - XLOG_GIN_UPDATE_META_PAGE

  - XLOG_GIN_INSERT_LISTPAGE
  - XLOG_GIN_DELETE_LISTPAGE

- **GIST**

  - XLOG_GIST_PAGE_UPDATE
  - XLOG_GIST_PAGE_SPLIT
  - XLOG_GIST_DELETE

- **SEQ**

  - XLOG_SEQ_LOG

- **SPGIST**

  - XLOG_SPGIST_ADD_LEAF
  - XLOG_SPGIST_MOVE_LEAFS
  - XLOG_SPGIST_ADD_NODE
  - XLOG_SPGIST_SPLIT_TUPLE
  - XLOG_SPGIST_VACUUM_LEAF
  - XLOG_SPGIST_VACUUM_ROOT
  - XLOG_SPGIST_VACUUM_REDIRECT

- **BRIN**

  - XLOG_BRIN_CREATE_INDEX
  - XLOG_BRIN_UPDATE
  - XLOG_BRIN_SAMEPAGE_UPDATE
  - XLOG_BRIN_REVMAP_EXTEND
  - XLOG_BRIN_DESUMMARIZE

- **REPLORIGIN**

  - XLOG_REPLORIGIN_SET
  - XLOG_REPLORIGIN_DROP

- **LOGICALMSG**

  - XLOG_LOGICAL_MESSAGE

For every record type, we need to add a test case that will generate the wal record and then replay it. For all types, the reading and writing procedures will be the same, but the generation of the wal record will be different. To add testcases for a specific record type, you will need to follow the procedures mentioned in the previous section. To write the testcase itself, do the following: 1. Implement function `pgmoneta_test_generate_<type>_v<version>` in

test/libpgmonetatest/tswalutils/tswalutils_<version>.c (add the function prototype in test/include/tswalutils.h as well). This function is responsible for generating the wal record of the type you are adding that mimics a real PostgreSQL wal record. 2. Add this in the body of the testcase

```
START_TEST(test_check_point_shutdown_v17)
{
  test_walfile(pgmoneta_test_generate_check_point_shutdown_v17);
}
END_TEST
```

and replace pgmoneta_test_generate_check_point_shutdown_v17); with the function you implemented in step 1.

test_walfile is a function that will take care of the reading, writing and comparing of the wal file generated against the one read from the disk.

If the record type you are adding has differences between versions of PostgreSQL (13-17), you will need to implement a generate function per version (generate_rec_x -> generate_rec_x_v16, generate_rec_x_v17, etc.).

For the sake of simplicity, please create one test suite per postgres version where the implementation resides in test/libpgmonetatest/tswalutils/tswalutils_<version>.c and the testcases in test/testcases/test_wal_utils.c and add testcase per record type within this version. You can take a look at this testcase for reference.

17.8 Code Coverage

17.8.1 Automatic Code Coverage Detection

If both gcov and gcovr are installed on your system **and the compiler is set to GCC** (regardless of the build type), code coverage will be automatically enabled during the build process. The build scripts will detect these tools and set the appropriate flags. If either tool is missing, or if the compiler is not GCC, code coverage will be skipped and a message will indicate that coverage tools were not found or the compiler is not supported.

17.8.2 Generating Coverage Reports

After building the project with coverage enabled and running your testsuite, coverage reports will be generated automatically in the build/coverage directory if gcov and gcovr are available.

The following commands are used to generate the reports (executed automatically by the test scripts):

```
# Make sure the coverage directory exists
mkdir -p ./coverage

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

Important: These commands must be run from inside the `build` directory.

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

If you want to generate these reports manually after running your own tests, simply run the above commands from your `build` directory.

> **Note:** If the `coverage` directory does not exist, create it first using `mkdir -p ./coverage` before running the coverage commands.
>
> **Important:** `gcovr` only works with GCC builds. If you build the project with Clang, coverage reports will not be generated with `gcovr`.

17.8.3 Notes

- Make sure you have both `gcov` and `gcovr` installed before building the project to enable coverage.
- If coverage tools are not found, or the compiler is not GCC, coverage generation will be skipped automatically and a message will be shown.
- You can always re-run the coverage commands manually if needed.

17.9 WAL Reader

17.9.1 Overview

This document provides an overview of the `wal_reader` tool, with a focus on the `parse_wal_file` function, which serves as the main entry point for parsing Write-Ahead Log (WAL) files. Currently, the function only parses the given WAL file and prints the description of each record. In the future, it will be integrated with other parts of the code.

17.9.2 pgmoneta-walinfo

`pgmoneta-walinfo` is a command line utility designed to read and display information about PostgreSQL Write-Ahead Log (WAL) files. The tool provides output in either raw or JSON format, making it easy to analyze WAL files for debugging, auditing, or general information purposes.

In addition to standard WAL files, `pgmoneta-walinfo` also supports encrypted (**aes**) and compressed WAL files in the following formats: **zstd**, **gz**, **lz4**, and **bz2**.

Usage

```
pgmoneta-walinfo 0.19.0
  Command line utility to read and display Write-Ahead Log (WAL) files

Usage:
  pgmoneta-walinfo <file>

Options:
  -c,  --config      Set the path to the pgmoneta_walinfo.conf file
  -u,  --users       Set the path to the pgmoneta_users.conf file
  -RT, --tablespaces Filter on tablspaces
  -RD, --databases   Filter on databases
  -RT, --relations   Filter on relations
  -R,  --filter      Combination of -RT, -RD, -RR
  -o,  --output      Output file
  -F,  --format      Output format (raw, json)
  -L,  --logfile     Set the log file
  -q,  --quiet       No output only result
       --color       Use colors (on, off)
  -r,  --rmgr        Filter on a resource manager
  -s,  --start       Filter on a start LSN
  -e,  --end         Filter on an end LSN
  -x,  --xid         Filter on an XID
  -l,  --limit       Limit number of outputs
  -v,  --verbose     Output result
  -S,  --summary     Show a summary of WAL record counts grouped by
     resource manager
  -V,  --version     Display version information
  -m,  --mapping     Provide mappings file for OID translation
  -t,  --translate   Translate OIDs to object names in XLOG records
  -?,  --help        Display help
```

Raw Output Format

In `raw` format, the default, the output is structured as follows:

```
Resource Manager | Start LSN | End LSN | rec len | tot len | xid |
   description (data and backup)
```

- **Resource Manager**: The name of the resource manager handling the log record.

- **Start LSN**: The start Log Sequence Number (LSN).
- **End LSN**: The end Log Sequence Number (LSN).
- **rec len**: The length of the WAL record.
- **tot len**: The total length of the WAL record, including the header.
- **xid**: The transaction ID associated with the record.
- **description (data and backup)**: A detailed description of the operation, along with any related backup block information.

Each part of the output is color-coded:

- **Red**: Header information (resource manager, record length, transaction ID, etc.).
- **Green**: Description of the WAL record.
- **Blue**: Backup block references or additional data.

This format makes it easy to visually distinguish different parts of the WAL file for quick analysis.

Example

1. To view WAL file details in JSON format:

```
pgmoneta-walinfo -F json /path/to/walfile
```

2. To view WAL file details with OIDs in the records translated to object names:

Currently, `pgmoneta-walinfo` supports translating OIDs in two ways, 1. If the user provided `pgmoneta_user.conf` file, the tool will use it to get the needed credentials to connect to the database cluster and fetch the object names. directly from it.

```
pgmoneta-walinfo -c pgmoneta_walinfo.conf -t -u /path/to/pgmoneta_user.
   conf /path/to/walfile
```

2. If the user provided a mapping file that contains the OIDs and the corresponding object names, the tool will use it to translate the OIDs to the object names. This option helps if the user doesn't have the `pgmoneta_user.conf` file or doesn't want to use it.

```
pgmoneta-walinfo -c pgmoneta_walinfo.conf -t -m /path/to/mapping.json /
   path/to/walfile
```

User can get the needed info to create the file using these queries:

```
SELECT spcname, oid FROM pg_tablespace
SELECT datname, oid FROM pg_database
SELECT nspname || '.' || relname, c.oid FROM pg_class c JOIN pg_namespace
   n ON c.relnamespace = n.oid
```

In either ways, the user should use the `-t` flag to enable the translation. If user provided `pgmoneta_user.conf` file or the mapping file, the tool will do nothing if the `-t` flag is not provided.

User can create the `pgmoneta_user.conf` file by following the instructions in the `DEVELOPER.md`

After using this translation feature, the output will change XLOG records from something like this `Heap2 | 1/D8FFD1C0 | 1/D8FFEB50 | 59 | 59 | 958 | cutoff xid 0 flags 0x03 blkref #0: rel 1663/16399/16733 forknum 2 blk 0 blkref #1: rel 1663/16399/16733 forknum 0 blk 27597`

to this `Heap2 | 1/D8FFD1C0 | 1/D8FFEB50 | 59 | 59 | 958 | cutoff xid 0 flags 0x03 blkref #0: rel pg_default/mydb/test_tbl forknum 2 blk 0 blkref #1: rel pg_default/mydb/16733 forknum 0 blk 27597`

Example of `mappings.json` file:

```
{
    "tablespaces": [
        {"name1": "oid1"},
        {"name2": "oid2"}
    ],
    "databases": [
        {"name1": "oid1"},
        {"name2": "oid2"}
    ],
    "relations": [
        {"name1": "oid1"},
        {"name2": "oid2"}
    ]
}
```

which is basically three sections, each section contains array key value pairs. The key is the object name and the value is the oid.

Note 1: If both files (`pgmoneta_users.conf` & `mappings.json`) are provided, the tool will use the mapping file. Note 2: If there is an OID that wasn't in the server/mapping (whichever the user choose at that time), the oid will be written as it is.

e.g. `rel pg_default/mydb/16733` will be written as `rel pg_default/mydb/16733` if the OID 16733 wasn't in the server/mapping.

17.9.3 High-Level API Overview

The following section provides a high-level overview of how users can interact with the functions and structures defined in the `walfile.h` file. These APIs allow you to read, write, and manage Write-Ahead Log (WAL) files.

`struct walfile`

The `walfile` struct represents the core structure used for interacting with WAL files in PostgreSQL. A WAL file stores a log of changes to the database and is used for crash recovery, replication, and other purposes. Each WAL file consists of pages (each 8192 bytes by default), containing records that capture database changes.

Fields:

- **magic_number**: Identifies the PostgreSQL version that created the WAL file.
- **long_phd**: A pointer to the extended header (long header) found on the first page of the WAL file. This header contains additional metadata.
- **page_headers**: A deque of headers representing each page in the WAL file, excluding the first page.
- **records**: A deque of decoded WAL records. Each record represents a change made to the database and contains both metadata and the actual data to be applied during recovery or replication.

Function Overview

The `walfile.h` file provides three key functions for interacting with WAL files: `pgmoneta_read_walfile`, `pgmoneta_write_walfile`, and `pgmoneta_destroy_walfile`. These functions allow users to read from, write to, and destroy WAL file objects, respectively.

`pgmoneta_read_walfile`

```
int pgmoneta_read_walfile(int server, char* path, struct walfile** wf);
```

Description:

This function reads a WAL file from a specified path and populates a `walfile` structure with its contents, including the file's headers and records.

Parameters:

- **server**: The index of the Postgres server in Pgmoneta configuration.
- **path**: The file path to the WAL file that needs to be read.
- **wf**: A pointer to a pointer to a `walfile` structure that will be populated with the WAL file data.

Return:

- Returns `0` on success or `1` on failure.

Usage Example:

```
struct walfile* wf = NULL;
int result = pgmoneta_read_walfile(0, "/path/to/walfile", &wf);
if (result == 0) {
    // Successfully read WAL file
}
```

pgmoneta_write_walfile

```
int pgmoneta_write_walfile(struct walfile* wf, int server, char* path);
```

Description:

This function writes the contents of a `walfile` structure back to disk, saving it as a WAL file at the specified path.

Parameters:

- **wf**: The `walfile` structure containing the WAL data to be written.
- **server**: The index or ID of the server where the WAL file should be saved.
- **path**: The file path where the WAL file should be written.

Return:

- Returns `0` on success or `1` on failure.

Usage Example:

```
int result = pgmoneta_write_walfile(wf, 0, "/path/to/output_walfile");
if (result == 0)
{
    // Successfully wrote WAL file
}
```

pgmoneta_destroy_walfile

```
void pgmoneta_destroy_walfile(struct walfile* wf);
```

Description:

This function frees the memory allocated for a `walfile` structure, including its headers and records.

Parameters:

- **wf**: The `walfile` structure to be destroyed.

Usage Example:

```
struct walfile* wf = NULL;
int result = pgmoneta_read_walfile(0, "/path/to/walfile", &wf);
if (result == 0) {
    // Successfully read WAL file
}
pgmoneta_destroy_walfile(wf);
```

pgmoneta_describe_walfile

```
int pgmoneta_describe_walfile(char* path, enum value_type type, FILE*
    output, bool quiet, bool color,
                              struct deque* rms, uint64_t start_lsn,
                                 uint64_t end_lsn, struct deque* xids,
                              uint32_t limit, bool summary, char**
                                 included_objects);
```

Description:

This function reads a single WAL file at the specified `path`, filters its records based on provided parameters, and writes the formatted output to `output`.

Parameters:

- **path**: Path to the WAL file to be described
- **type**: Output format type (raw or JSON)
- **output**: File stream for output; if NULL, prints to stdout
- **quiet**: If true, suppresses detailed output
- **color**: If true, enables colored output for better readability
- **rms**: Deque of resource managers to filter on
- **start_lsn**: Starting LSN to filter records (0 for no filter)
- **end_lsn**: Ending LSN to filter records (0 for no filter)
- **xids**: Deque of transaction IDs to filter on
- **limit**: Maximum number of records to output (0 for no limit)
- **summary**: Show a summary of WAL record counts grouped by resource manager
- **included_objects**: Array of object names to filter on (NULL for all objects)

Return:

- Returns `0` on success or `1` on failure.

```
**‘pgmoneta_describe_walfiles_in_directory‘**

‘‘‘c
int pgmoneta_describe_walfiles_in_directory(char* dir_path, enum
    value_type type, FILE* output,
```

```
                                                    bool quiet, bool color, struct
                                                        deque* rms,
                                                    uint64_t start_lsn, uint64_t
                                                       end_lsn, struct deque* xids
                                                       ,
                                                    uint32_t limit, bool summary,
                                                       char** included_objects);
```

Description:

This function processes all WAL files in the directory specified by `dir_path`, applies the same filtering logic as `pgmoneta_describe_walfile`, and writes aggregated results to `output`.

Parameters:

- **dir_path**: Path to the directory containing WAL files
- **type**: Output format type (raw or JSON)
- **output**: File stream for output; if NULL, prints to stdout
- **quiet**: If true, suppresses detailed output
- **color**: If true, enables colored output for better readability
- **rms**: Deque of resource managers to filter on
- **start_lsn**: Starting LSN to filter records (0 for no filter)
- **end_lsn**: Ending LSN to filter records (0 for no filter)
- **xids**: Deque of transaction IDs to filter on
- **limit**: Maximum number of records to output (0 for no limit)
- **summary**: Show a summary of WAL record counts grouped by resource manager
- **included_objects**: Array of object names to filter on (NULL for all objects)

Return:

- Returns `0` on success or `1` on failure

17.9.4 Internal API Overview

struct partial_xlog_record

The partial_xlog_record struct represents an incomplete WAL XLOG record encountered during parsing. It is used to manage records that span multiple WAL files.

```
struct partial_xlog_record
{
   char* data_buffer;                  /**< Data portion of the record. */
   char* xlog_record;                  /**< Pointer to the xlog record. */
   uint32_t data_buffer_bytes_read;    /**< Length of the total data read
      in data_buffer. */
```

```
    uint32_t xlog_record_bytes_read;    /**< Length of the total data read
       in xlog_record buffer. */
};
```

Fields:

- **data_buffer**: Contains the data portion of the partially read WAL record.
- **xlog_record**: Points to the header structure containing metadata about the WAL record.
- **data_buffer_bytes_read**: Length of the total data read in data_buffer.
- **xlog_record_bytes_read**: Length of the total data read in xlog_record buffer.

`parse_wal_file`

This function is responsible for reading and parsing a PostgreSQL Write-Ahead Log (WAL) file.

Parameters

- **path**: The file path to the WAL file that needs to be parsed.
- **server_info**: A pointer to a `server` structure containing information about the server.

Description

The `parse_wal_file` function opens the WAL file specified by the `path` parameter in binary mode and reads the WAL records. It processes these records, handling various cases such as records that cross page boundaries, while ensuring correct memory management throughout the process.

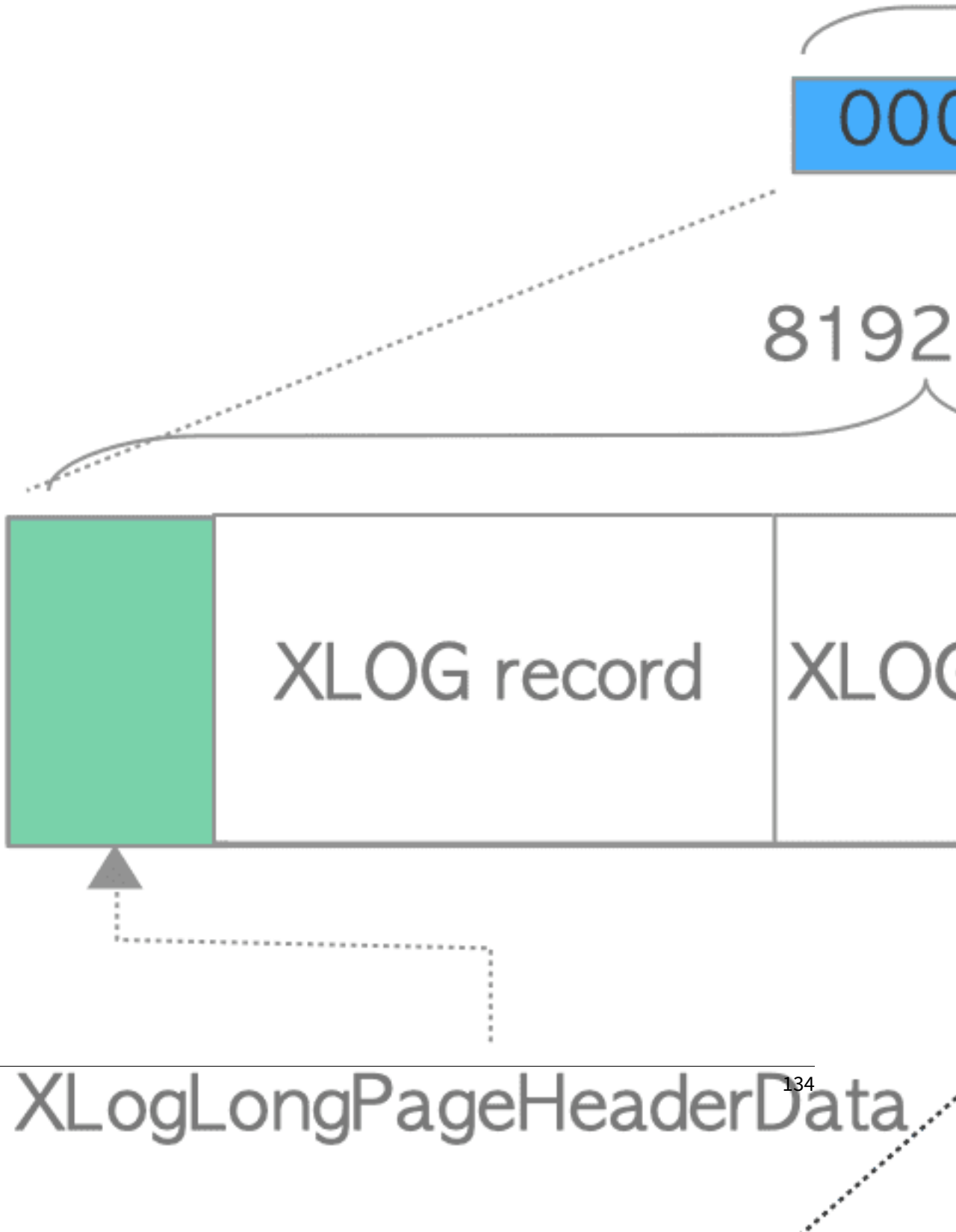
Usage Example

```
parse_wal_file("/path/to/wal/file", &my_server);
```

WAL File Structure

The image illustrates the structure of a WAL (Write-Ahead Logging) file in PostgreSQL, focusing on how XLOG records are organized within WAL segments.

Source: https://www.interdb.jp/pg/pgsql09/03.html


000
8192
XLOG record
XLOG
XLogLongPageHeaderData

A WAL segment, by default, is a 16 MB file, divided into pages of 8192 bytes (8 KB) each. The first page contains a header defined by the XLogLongPageHeaderData structure, while all subsequent pages have headers described by the XLogPageHeaderData structure. XLOG records are written sequentially in each page, starting at the beginning and moving downward.

The figure highlights how the WAL ensures data consistency by sequentially writing XLOG records in pages, structured within larger 16 MB WAL segments.

17.9.5 Resource Managers

In the context of the WAL reader, resource managers (rm) are responsible for handling different types of records found within a WAL file. Each record in the WAL file is associated with a specific resource manager, which determines how that record is processed.

Resource Manager Definitions

Each resource manager is defined in the `rm_[name].h` header file and implemented in the corresponding `rm_[name].c` source file.

In the `rmgr.h` header file, the resource managers are declared as an enum, with each resource manager having a unique identifier.

Resource Manager Functions

Each resource manager implements the `rm_desc` function, which provides a description of the record type associated with that resource manager. In the future, they will be extended to implement the `rm_redo` function to apply the changes to another server.

Supporting Various WAL Structures in PostgreSQL Versions 13 to 17

The WAL structure has evolved across PostgreSQL versions 13 to 17, requiring different handling for each version. To accommodate these differences, we have implemented a wrapper-based approach, such as the factory pattern, to handle varying WAL structures.

Below are the commit hashes for the officially supported magic values in each PostgreSQL version:

1. PostgreSQL 13 - 0xD106
2. PostgreSQL 14 - 0xD10D
3. PostgreSQL 15 - 0xD110
4. PostgreSQL 16 - 0xD113
5. PostgreSQL 17 - 0xD116
6. PostgreSQL 18 - 0xD118

`xl_end_of_recovery` is an example of how we handle different versions of structures with a wrapper struct and a factory pattern.

```
struct xl_end_of_recovery_v16 {
    timestamp_tz end_time;
    timeline_id this_timeline_id;
    timeline_id prev_timeline_id;
};

struct xl_end_of_recovery_v17 {
    timestamp_tz end_time;
    timeline_id this_timeline_id;
    timeline_id prev_timeline_id;
    int wal_level;
};

struct xl_end_of_recovery {
    int pg_version;
    union {
        struct xl_end_of_recovery_v16 v16;
        struct xl_end_of_recovery_v17 v17;
    } data;
    void (*parse)(struct xl_end_of_recovery* wrapper, void* rec);
    char* (*format)(struct xl_end_of_recovery* wrapper, char* buf);
};

xl_end_of_recovery* create_xl_end_of_recovery(int pg_version) {
    xl_end_of_recovery* wrapper = malloc(sizeof(xl_end_of_recovery));
    wrapper->pg_version = pg_version;

    if (pg_version >= 17) {
        wrapper->parse = parse_v17;
        wrapper->format = format_v17;
    } else {
        wrapper->parse = parse_v16;
        wrapper->format = format_v16;
    }

    return wrapper;
}

void parse_v16(xl_end_of_recovery* wrapper, void* rec) {
    memcpy(&wrapper->data.v16, rec, sizeof(struct xl_end_of_recovery_v16))
        ;
}

void parse_v17(xl_end_of_recovery* wrapper, void* rec) {
    memcpy(&wrapper->data.v17, rec, sizeof(struct xl_end_of_recovery_v17))
        ;
}

char* format_v16(xl_end_of_recovery* wrapper, char* buf) {
    struct xl_end_of_recovery_v16* xlrec = &wrapper->data.v16;
```

```
    return pgmoneta_format_and_append(buf, "tli %u; prev tli %u; time %s",
                                      xlrec->this_timeline_id, xlrec->
                                         prev_timeline_id,
                                      pgmoneta_wal_timestamptz_to_str(
                                         xlrec->end_time));
}

char* format_v17(xl_end_of_recovery* wrapper, char* buf) {
    struct xl_end_of_recovery_v17* xlrec = &wrapper->data.v17;
    return pgmoneta_format_and_append(buf, "tli %u; prev tli %u; time %s;
       wal_level %d",
                                      xlrec->this_timeline_id, xlrec->
                                         prev_timeline_id,
                                      pgmoneta_wal_timestamptz_to_str(
                                         xlrec->end_time),
                                      xlrec->wal_level);
}
```

17.9.6 WAL Change List

This section lists the changes in the WAL format between different versions of PostgreSQL.

xl_clog_truncate

17

```
struct xl_clog_truncate
{
   int64 pageno;              /**< The page number of the clog to truncate
       */
   transaction_id oldestXact;  /**< The oldest transaction ID to retain */
   oid oldestXactDb;        /**< The database ID of the oldest transaction
       */
};
```

16

```
struct xl_clog_truncate
{
   int64 pageno;              /**< The page number of the clog to truncate
       */
   transaction_id oldestXact;  /**< The oldest transaction ID to retain */
   oid oldestXactDb;        /**< The database ID of the oldest transaction
       */
};
```

xl_commit_ts_truncate

17:

```
typedef struct xl_commit_ts_truncate
{
    int64       pageno;
    TransactionId oldestXid;
} xl_commit_ts_truncate;
```

16:

```
typedef struct xl_commit_ts_truncate
{
    int         pageno;
    TransactionId oldestXid;
} xl_commit_ts_truncate;
```

xl_heap_prune

17:

```
typedef struct xl_heap_prune
{
    uint8       reason;
    uint8       flags;

    /*
     * If XLHP_HAS_CONFLICT_HORIZON is set, the conflict horizon XID
        follows,
     * unaligned
     */
} xl_heap_prune;
#define SizeOfHeapPrune (offsetof(xl_heap_prune, flags) + sizeof(uint8))
```

16:

```
typedef struct xl_heap_prune
{
    TransactionId snapshotConflictHorizon;
    uint16      nredirected;
    uint16      ndead;
    bool        isCatalogRel;   /* to handle recovery conflict during
       logical
                                 * decoding on standby */
    /* OFFSET NUMBERS are in the block reference 0 */
} xl_heap_prune;
#define SizeOfHeapPrune (offsetof(xl_heap_prune, isCatalogRel) + sizeof(
   bool))
```

xlhp_freeze_plan

Removed `xl_heap_freeze_page`

17:

```
typedef struct xlhp_freeze_plan
{
    TransactionId xmax;
    uint16      t_infomask2;
    uint16      t_infomask;
    uint8       frzflags;

    /* Length of individual page offset numbers array for this plan */
    uint16      ntuples;
} xlhp_freeze_plan;
```

spgxlogState

(Doesn't need to be changed)

17:

```
typedef struct spgxlogState
{
    TransactionId redirectXid;
    bool        isBuild;
} spgxlogState;
```

16:

```
typedef struct spgxlogState
{
    TransactionId myXid;
    bool        isBuild;
} spgxlogState;
```

xl_end_of_recovery

```
typedef struct xl_end_of_recovery
{
    TimestampTz end_time;
    TimeLineID  ThisTimeLineID; /* new TLI */
    TimeLineID  PrevTimeLineID; /* previous TLI we forked off from */
    int         wal_level;
} xl_end_of_recovery;
```

16:

```
typedef struct xl_end_of_recovery
{
    TimestampTz end_time;
    TimeLineID  ThisTimeLineID; /* new TLI */
    TimeLineID  PrevTimeLineID; /* previous TLI we forked off from */
} xl_end_of_recovery;
```

16 → 15

gingxlogSplit

16: same for `gin_xlog_update_meta`

```
typedef struct ginxlogSplit
{
    RelFileLocator locator;
    BlockNumber rrlink;         /* right link, or root's blocknumber if
       root
                                 * split */
    BlockNumber leftChildBlkno; /* valid on a non-leaf split */
    BlockNumber rightChildBlkno;
    uint16      flags;          /* see below */
} ginxlogSplit;
```

15:

```
typedef struct ginxlogSplit
{
    RelFileNode node;
    BlockNumber rrlink;         /* right link, or root's blocknumber if
       root
                                 * split */
    BlockNumber leftChildBlkno; /* valid on a non-leaf split */
    BlockNumber rightChildBlkno;
    uint16      flags;          /* see below */
} ginxlogSplit;
```

gistxlogDelete

16:

```
typedef struct gistxlogDelete
{
    TransactionId snapshotConflictHorizon;
    uint16      ntodelete;      /* number of deleted offsets */
    bool        isCatalogRel;   /* to handle recovery conflict during
       logical
                                 * decoding on standby */

    /* TODELETE OFFSET NUMBERS */
    OffsetNumber offsets[FLEXIBLE_ARRAY_MEMBER];
} gistxlogDelete;
#define SizeOfGistxlogDelete    offsetof(gistxlogDelete, offsets)
```

15:

```
typedef struct gistxlogDelete
{
    TransactionId latestRemovedXid;
    uint16      ntodelete;      /* number of deleted offsets */

    /*
     * In payload of blk 0 : todelete OffsetNumbers
     */
} gistxlogDelete;
#define SizeOfGistxlogDelete    (offsetof(gistxlogDelete, ntodelete) +
   sizeof(uint16))
```

gistxlogPageReuse

16:

```
typedef struct gistxlogPageReuse
{
    RelFileLocator locator;
    BlockNumber block;
    FullTransactionId snapshotConflictHorizon;
    bool        isCatalogRel;   /* to handle recovery conflict during
       logical
                                 * decoding on standby */
} gistxlogPageReuse;
#define SizeOfGistxlogPageReuse (offsetof(gistxlogPageReuse, isCatalogRel)
    + sizeof(bool))
```

15:

```
typedef struct gistxlogPageReuse
{
    RelFileNode node;
    BlockNumber block;
    FullTransactionId latestRemovedFullXid;
} gistxlogPageReuse;

#define SizeOfGistxlogPageReuse (offsetof(gistxlogPageReuse,
   latestRemovedFullXid) + sizeof(FullTransactionId))
```

xl_hash_vacuum_one_page

16:

```
typedef struct xl_hash_vacuum_one_page
{
    TransactionId snapshotConflictHorizon;
    uint16      ntuples;
    bool        isCatalogRel;   /* to handle recovery conflict during
       logical
                                 * decoding on standby */
```

```
    /* TARGET OFFSET NUMBERS */
    OffsetNumber offsets[FLEXIBLE_ARRAY_MEMBER];
} xl_hash_vacuum_one_page;
#define SizeOfHashVacuumOnePage offsetof(xl_hash_vacuum_one_page, offsets)
```

15:

```
typedef struct xl_hash_vacuum_one_page
{
    TransactionId latestRemovedXid;
    int         ntuples;

    /* TARGET OFFSET NUMBERS FOLLOW AT THE END */
} xl_hash_vacuum_one_page;
#define SizeOfHashVacuumOnePage \
    (offsetof(xl_hash_vacuum_one_page, ntuples) + sizeof(int))
```

xl_heap_prune

16:

```
typedef struct xl_heap_prune
{
    TransactionId snapshotConflictHorizon;
    uint16      nredirected;
    uint16      ndead;
    bool        isCatalogRel;   /* to handle recovery conflict during
       logical
                                 * decoding on standby */
    /* OFFSET NUMBERS are in the block reference 0 */
} xl_heap_prune;
#define SizeOfHeapPrune (offsetof(xl_heap_prune, isCatalogRel) + sizeof(
   bool))
```

15:

```
typedef struct xl_heap_prune
{
    TransactionId latestRemovedXid;
    uint16      nredirected;
    uint16      ndead;
    /* OFFSET NUMBERS are in the block reference 0 */
} xl_heap_prune;
#define SizeOfHeapPrune (offsetof(xl_heap_prune, ndead) + sizeof(uint16))
```

xl_heap_freeze_plan

16:

```
typedef struct xl_heap_freeze_plan
```

```
{
    TransactionId xmax;
    uint16      t_infomask2;
    uint16      t_infomask;
    uint8       frzflags;

    /* Length of individual page offset numbers array for this plan */
    uint16      ntuples;
} xl_heap_freeze_plan;
```

15:

```
typedef struct xl_heap_freeze_tuple
{
    TransactionId xmax;
    OffsetNumber offset;
    uint16      t_infomask2;
    uint16      t_infomask;
    uint8       frzflags;
} xl_heap_freeze_tuple;
```

xl_heap_freeze_page

16:

```
typedef struct xl_heap_freeze_page
{
    TransactionId snapshotConflictHorizon;
    uint16      nplans;
    bool        isCatalogRel;   /* to handle recovery conflict during
       logical
                                 * decoding on standby */

    /*
     * In payload of blk 0 : FREEZE PLANS and OFFSET NUMBER ARRAY
     */
} xl_heap_freeze_page;
```

15:

```
typedef struct xl_heap_freeze_page
{
    TransactionId cutoff_xid;
    uint16      ntuples;
} xl_heap_freeze_page;
```

xl_btree_reuse_page

16:

```
typedef struct xl_btree_reuse_page
```

```
{
    RelFileLocator locator;
    BlockNumber block;
    FullTransactionId snapshotConflictHorizon;
    bool        isCatalogRel;   /* to handle recovery conflict during
       logical
                                 * decoding on standby */
} xl_btree_reuse_page;
```

15:

```
typedef struct xl_btree_reuse_page
{
    RelFileNode node;
    BlockNumber block;
    FullTransactionId latestRemovedFullXid;
} xl_btree_reuse_page;
```

xl_btree_delete

16:

```
typedef struct xl_btree_delete
{
    TransactionId snapshotConflictHorizon;
    uint16      ndeleted;
    uint16      nupdated;
    bool        isCatalogRel;   /* to handle recovery conflict during
       logical
                                 * decoding on standby */

    /*----
     * In payload of blk 0 :
     * - DELETED TARGET OFFSET NUMBERS
     * - UPDATED TARGET OFFSET NUMBERS
     * - UPDATED TUPLES METADATA (xl_btree_update) ARRAY
     *----
     */
} xl_btree_delete;
```

15:

```
typedef struct xl_btree_delete
{
    TransactionId latestRemovedXid;
    uint16      ndeleted;
    uint16      nupdated;

    /* DELETED TARGET OFFSET NUMBERS FOLLOW */
    /* UPDATED TARGET OFFSET NUMBERS FOLLOW */
    /* UPDATED TUPLES METADATA (xl_btree_update) ARRAY FOLLOWS */
```

```
} xl_btree_delete;
```

spgxlogVacuumRedirect

16:

```
typedef struct spgxlogVacuumRedirect
{
    uint16      nToPlaceholder; /* number of redirects to make
       placeholders */
    OffsetNumber firstPlaceholder;  /* first placeholder tuple to remove
       */
    TransactionId snapshotConflictHorizon;  /* newest XID of removed
       redirects */
    bool        isCatalogRel;   /* to handle recovery conflict during
       logical
                                 * decoding on standby */

    /* offsets of redirect tuples to make placeholders follow */
    OffsetNumber offsets[FLEXIBLE_ARRAY_MEMBER];
} spgxlogVacuumRedirect;
```

15:

```
typedef struct spgxlogVacuumRedirect
{
    uint16      nToPlaceholder; /* number of redirects to make
       placeholders */
    OffsetNumber firstPlaceholder;  /* first placeholder tuple to remove
       */
    TransactionId newestRedirectXid;    /* newest XID of removed redirects
        */

    /* offsets of redirect tuples to make placeholders follow */
    OffsetNumber offsets[FLEXIBLE_ARRAY_MEMBER];
} spgxlogVacuumRedirect;
```

---

15 → 14

xl_xact_prepare

15:

```
ctypedef struct xl_xact_prepare
{
    uint32      magic;          /* format identifier */
    uint32      total_len;      /* actual file length */
    TransactionId xid;          /* original transaction XID */
```

```
    Oid         database;       /* OID of database it was in */
    TimestampTz prepared_at;    /* time of preparation */
    Oid         owner;          /* user running the transaction */
    int32       nsubxacts;      /* number of following subxact XIDs */
    int32       ncommitrels;    /* number of delete-on-commit rels */
    int32       nabortrels;     /* number of delete-on-abort rels */
    int32       ncommitstats;   /* number of stats to drop on commit */
    int32       nabortstats;    /* number of stats to drop on abort */
    int32       ninvalmsgs;     /* number of cache invalidation messages
       */
    bool        initfileinval;  /* does relcache init file need
       invalidation? */
    uint16      gidlen;         /* length of the GID - GID follows the
       header */
    XLogRecPtr  origin_lsn;     /* lsn of this record at origin node */
    TimestampTz origin_timestamp;   /* time of prepare at origin node */
} xl_xact_prepare;
```

14:

```
typedef struct xl_xact_prepare
{
    uint32      magic;          /* format identifier */
    uint32      total_len;      /* actual file length */
    TransactionId xid;          /* original transaction XID */
    Oid         database;       /* OID of database it was in */
    TimestampTz prepared_at;    /* time of preparation */
    Oid         owner;          /* user running the transaction */
    int32       nsubxacts;      /* number of following subxact XIDs */
    int32       ncommitrels;    /* number of delete-on-commit rels */
    int32       nabortrels;     /* number of delete-on-abort rels */
    int32       ninvalmsgs;     /* number of cache invalidation messages
       */
    bool        initfileinval;  /* does relcache init file need
       invalidation? */
    uint16      gidlen;         /* length of the GID - GID follows the
       header */
    XLogRecPtr  origin_lsn;     /* lsn of this record at origin node */
    TimestampTz origin_timestamp;   /* time of prepare at origin node */
} xl_xact_prepare;
```

xl_xact_parsed_commit

15:

```
typedef struct xl_xact_parsed_commit
{
    TimestampTz xact_time;
    uint32      xinfo;

    Oid         dbId;           /* MyDatabaseId */
```

```
    Oid         tsId;           /* MyDatabaseTableSpace */

    int         nsubxacts;
    TransactionId *subxacts;

    int         nrels;
    RelFileNode *xnodes;

    int         nstats;
    xl_xact_stats_item *stats;

    int         nmsgs;
    SharedInvalidationMessage *msgs;

    TransactionId twophase_xid; /* only for 2PC */
    char        twophase_gid[GIDSIZE];  /* only for 2PC */
    int         nabortrels;     /* only for 2PC */
    RelFileNode *abortnodes;    /* only for 2PC */
    int         nabortstats;        /* only for 2PC */
    xl_xact_stats_item *abortstats; /* only for 2PC */

    XLogRecPtr  origin_lsn;
    TimestampTz origin_timestamp;
} xl_xact_parsed_commit;
```

14:

```
typedef struct xl_xact_parsed_commit
{
    TimestampTz xact_time;
    uint32      xinfo;

    Oid         dbId;           /* MyDatabaseId */
    Oid         tsId;           /* MyDatabaseTableSpace */

    int         nsubxacts;
    TransactionId *subxacts;

    int         nrels;
    RelFileNode *xnodes;

    int         nmsgs;
    SharedInvalidationMessage *msgs;

    TransactionId twophase_xid; /* only for 2PC */
    char        twophase_gid[GIDSIZE];  /* only for 2PC */
    int         nabortrels;     /* only for 2PC */
    RelFileNode *abortnodes;    /* only for 2PC */

    XLogRecPtr  origin_lsn;
    TimestampTz origin_timestamp;
```

```
} xl_xact_parsed_commit;
```

xl_xact_parsed_abort

15:

```
typedef struct xl_xact_parsed_abort
{
    TimestampTz xact_time;
    uint32      xinfo;

    Oid         dbId;           /* MyDatabaseId */
    Oid         tsId;           /* MyDatabaseTableSpace */

    int         nsubxacts;
    TransactionId *subxacts;

    int         nrels;
    RelFileNode *xnodes;

    int         nstats;
    xl_xact_stats_item *stats;

    TransactionId twophase_xid; /* only for 2PC */
    char        twophase_gid[GIDSIZE];  /* only for 2PC */

    XLogRecPtr  origin_lsn;
    TimestampTz origin_timestamp;
} xl_xact_parsed_abort;
```

14:

```
typedef struct xl_xact_parsed_abort
{
    TimestampTz xact_time;
    uint32      xinfo;

    Oid         dbId;           /* MyDatabaseId */
    Oid         tsId;           /* MyDatabaseTableSpace */

    int         nsubxacts;
    TransactionId *subxacts;

    int         nrels;
    RelFileNode *xnodes;

    TransactionId twophase_xid; /* only for 2PC */
    char        twophase_gid[GIDSIZE];  /* only for 2PC */

    XLogRecPtr  origin_lsn;
    TimestampTz origin_timestamp;
```

```
} xl_xact_parsed_abort;
```

xlogrecord.h flags

15:

```
#define BKPIMAGE_APPLY          0x02    /* page image should be restored
                                         * during replay */
/* compression methods supported */
#define BKPIMAGE_COMPRESS_PGLZ  0x04
#define BKPIMAGE_COMPRESS_LZ4   0x08
#define BKPIMAGE_COMPRESS_ZSTD  0x10

#define BKPIMAGE_COMPRESSED(info) \
    ((info & (BKPIMAGE_COMPRESS_PGLZ | BKPIMAGE_COMPRESS_LZ4 | \
              BKPIMAGE_COMPRESS_ZSTD)) != 0)
```

14:

```
#define BKPIMAGE_IS_COMPRESSED      0x02    /* page image is compressed */
#define BKPIMAGE_APPLY      0x04    /* page image should be restored
   during
                                     * replay */
```

---

14 → 13

xl_heap_prune

14:

```
typedef struct xl_heap_prune
{
    TransactionId latestRemovedXid;
    uint16      nredirected;
    uint16      ndead;
    /* OFFSET NUMBERS are in the block reference 0 */
} xl_heap_prune;
```

13:

```
typedef struct xl_heap_clean
{
    TransactionId latestRemovedXid;
    uint16      nredirected;
    uint16      ndead;
    /* OFFSET NUMBERS are in the block reference 0 */
} xl_heap_clean;
```

xl_heap_vacuum

14:

```
typedef struct xl_heap_vacuum
{
    uint16      nunused;
    /* OFFSET NUMBERS are in the block reference 0 */
} xl_heap_vacuum;
```

13:

```
typedef struct xl_heap_cleanup_info
{
    RelFileNode node;
    TransactionId latestRemovedXid;
} xl_heap_cleanup_info;
```

xl_btree_metadata

14:

```
typedef struct xl_btree_metadata
{
    uint32      version;
    BlockNumber root;
    uint32      level;
    BlockNumber fastroot;
    uint32      fastlevel;
    uint32      last_cleanup_num_delpages;
    bool        allequalimage;
} xl_btree_metadata;
```

13:

```
typedef struct xl_btree_metadata
{
    uint32      version;
    BlockNumber root;
    uint32      level;
    BlockNumber fastroot;
    uint32      fastlevel;
    TransactionId oldest_btpo_xact;
    float8      last_cleanup_num_heap_tuples;
    bool        allequalimage;
} xl_btree_metadata;
```

xl_btree_reuse_page

14:

```
typedef struct xl_btree_reuse_page
{
    RelFileNode node;
    BlockNumber block;
    FullTransactionId latestRemovedFullXid;
} xl_btree_reuse_page;
```

13:

```
typedef struct xl_btree_reuse_page
{
    RelFileNode node;
    BlockNumber block;
    TransactionId latestRemovedXid;
} xl_btree_reuse_page;
```

xl_btree_delete

14:

```
typedef struct xl_btree_delete
{
    TransactionId latestRemovedXid;
    uint16      ndeleted;
    uint16      nupdated;

    /* DELETED TARGET OFFSET NUMBERS FOLLOW */
    /* UPDATED TARGET OFFSET NUMBERS FOLLOW */
    /* UPDATED TUPLES METADATA (xl_btree_update) ARRAY FOLLOWS */
} xl_btree_delete;
```

13:

```
typedef struct xl_btree_delete
{
    TransactionId latestRemovedXid;
    uint32      ndeleted;

    /* DELETED TARGET OFFSET NUMBERS FOLLOW */
} xl_btree_delete;
```

xl_btree_unlink_page

14:

```
typedef struct xl_btree_unlink_page
{
    BlockNumber leftsib;        /* target block's left sibling, if any */
    BlockNumber rightsib;       /* target block's right sibling */
    uint32      level;          /* target block's level */
```

```
    FullTransactionId safexid;  /* target block's BTPageSetDeleted() XID
       */

    /*
     * Information needed to recreate a half-dead leaf page with correct
     * topparent link.  The fields are only used when deletion operation's
     * target page is an internal page.  REDO routine creates half-dead
        page
     * from scratch to keep things simple (this is the same convenient
     * approach used for the target page itself).
     */
    BlockNumber leafleftsib;
    BlockNumber leafrightsib;
    BlockNumber leaftopparent;  /* next child down in the subtree */

    /* xl_btree_metadata FOLLOWS IF XLOG_BTREE_UNLINK_PAGE_META */
} xl_btree_unlink_page;
```

13:

```
typedef struct xl_btree_unlink_page
{
    BlockNumber leftsib;        /* target block's left sibling, if any */
    BlockNumber rightsib;       /* target block's right sibling */

    /*
     * Information needed to recreate the leaf page, when target is an
     * internal page.
     */
    BlockNumber leafleftsib;
    BlockNumber leafrightsib;
    BlockNumber topparent;      /* next child down in the branch */

    TransactionId btpo_xact;    /* value of btpo.xact for use in recovery
       */
    /* xl_btree_metadata FOLLOWS IF XLOG_BTREE_UNLINK_PAGE_META */
} xl_btree_unlink_page;
```

17.9.7 Additional Information

For more details on the internal workings and additional helper functions used in `parse_wal_file`, refer to the source code in `wal_reader.c`.

17.10 Core APIs

pgmoneta 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.

17.10.1 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`	
`int32_t`	`ValueInt32`	

type	type enum	free behavior (no-op if left blank)
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	pgmoneta_json_destroy(), this will recursively destroy internal data
struct json*	ValueJSONRef	
struct deque*	ValueDeque	pgmoneta_deque_destroy(), this will recursively destroy internal data
struct deque*	ValueDequeRef	
struct art *	ValueART	pgmoneta_art_destroy(), this will recursively destroy internal data
struct art *	ValueARTRef	

type	type enum	free behavior (no-op if left blank)
`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

pgmoneta_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 pgmoneta_value_from_float/pgmoneta_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.

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

pgmoneta_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);
```

pgmoneta_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.

pgmoneta_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.

pgmoneta_value_type

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

pgmoneta_value_destroy

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

pgmoneta_value_to_float/pgmoneta_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.

pgmoneta_value_from_float/pgmoneta_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.

```
pgmoneta_value_create(ValueFloat, pgmoneta_value_from_float(float_val), &
   val);
```

pgmoneta_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.

17.10.2 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

pgmoneta_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.

pgmoneta_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 (`pgmoneta_value_from_float` / `pgmoneta_value_from_double`). The function acquires write lock if thread safe is enabled.

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

pgmoneta_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)
{
   pgmoneta_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};
...

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

pgmoneta_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).

```
pgmoneta_deque_add(deque, "Hello", (uintptr_t)"world", ValueString);
char* tag = NULL;
char* value = (char*)pgmoneta_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
pgmoneta_deque_add(deque, "Hello", (uintptr_t)"world", ValueString);
char* value = (char*)pgmoneta_deque_poll(deque, NULL);

printf("%s!\n", value) // "world!"

// remember to free it!
free(value);
```

pgmoneta_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).

```
pgmoneta_deque_add(deque, "Hello", (uintptr_t)"world", ValueString);
char* tag = NULL;
```

```
char* value = (char*)pgmoneta_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
pgmoneta_deque_add(deque, "Hello", (uintptr_t)"world", ValueString);
char* value = (char*)pgmoneta_deque_poll_last(deque, NULL);

printf("%s!\n", value) // "world!"

// remember to free it!
free(value);
```

pgmoneta_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).

pgmoneta_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).

pgmoneta_deque_iterator_create

Create a deque iterator, note that iterator is **NOT** thread safe

pgmoneta_deque_iterator_destroy

Destroy a deque iterator

pgmoneta_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.

pgmoneta_deque_iterator_has_next

Check if iterator has next value without advancing it.

pgmoneta_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 `pgmoneta_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
pgmoneta_deque_iterator_create(deque, &iter);
while (pgmoneta_deque_iterator_next(iter)) {
    if (iter->tag == NULL) {
        pgmoneta_deque_iterator_remove(iter);
    }
    else {
        printf("%s: %s\n", iter->tag, (char*)pgmoneta_value_data(iter->
           value));
    }
}
pgmoneta_deque_iterator_destroy(iter);
```

pgmoneta_deque_size

Get the current deque size, the function acquires the read lock

pgmoneta_deque_empty

Check if the deque is empty

pgmoneta_deque_to_string

Convert the deque to string of the specified format.

pgmoneta_deque_list

Log the deque content in logs. This only works in TRACE log level.

pgmoneta_deque_sort

Merge sort the deque. The time complexity is O(log(n)).

pgmoneta_deque_get

Get the data with a specific tag from the deque.

The time complexity for getting a node is O(n).

pgmoneta_deque_exists

Check if a tag exists in deque.

pgmoneta_deque_remove

Remove all the nodes in the deque that have the given tag.

pgmoneta_deque_clear

Remove all the nodes in the deque.

pgmoneta_deque_set_thread_safe

Set the deque to be thread safe.

17.10.3 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

pgmoneta_art_create

Create an adaptive radix tree

pgmoneta_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.

pgmoneta_art_insert_with_config

Insert a key value pair with a customized configuration. The idea and usage is identical to `pgmoneta_deque_add_with_config`.

pgmoneta_art_contains_key

Check if a key exists in ART.

pgmoneta_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 `pgmoneta_art_contains_key`.

pgmoneta_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.

pgmoneta_art_delete

Delete a key from ART. Note that the function returns success(i.e. 0) even if the key does not exist.

pgmoneta_art_clear

Removes all the key value pairs in the ART tree.

pgmoneta_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.

pgmoneta_art_destroy

Destroy an ART.

pgmoneta_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.

pgmoneta_art_iterator_destroy

Destroy an ART iterator. This will recursively destroy all of its key value entries.

pgmoneta_art_iterator_remove

Remove the key value pair the iterator points to. Note that currently the function just invokes `pgmoneta_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.

pgmoneta_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.

pgmoneta_art_iterator_has_next

Check if the iterator has next value without advancing it.

```
pgmoneta_art_iterator_create(t, &iter);
while (pgmoneta_art_iterator_next(iter)) {
    printf("%s: %s\n", iter->key, (char*)pgmoneta_value_data(iter->value))
        ;
}
pgmoneta_art_iterator_destroy(iter);
```

17.10.4 JSON

JSON is essentially built upon deque and ART. Find its definition and implementation in json.h and json.c.

Note that this document will not cover the iterative parsing APIs, since those are still experimental.

APIs

pgmoneta_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.

pgmoneta_json_destroy

Destroy a JSON object

pgmoneta_json_put

Put a key value pair into the json object. This function invokes `pgmoneta_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 `pgmoneta_json_append` from then on. Note that unlike ART, JSON only takes certain types of value. See JSON introduction for details.

pgmoneta_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 `pgmoneta_json_put` from then on.

pgmoneta_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.

pgmoneta_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.

pgmoneta_json_get

Get and unwrap the value data from a JSON item. If the JSON object is an array, the function returns 0.

pgmoneta_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.

pgmoneta_json_contains_key

Check if the JSON item contains a specific key. It always returns false if the object is an array.

pgmoneta_json_array_length

Get the length of a JSON array

pgmoneta_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.

pgmoneta_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.

pgmoneta_json_iterator_has_next

Check if the object has the next entry or key value pair.

pgmoneta_json_iterator_destroy

Destroy the JSON iterator.

pgmoneta_json_parse_string

Parse a JSON string into a JSON object.

pgmoneta_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.

pgmoneta_json_to_string

Convert the JSON object to string.

pgmoneta_json_print

A convenient wrapper to quickly print out the JSON object.

pgmoneta_json_read_file

Read the JSON file and parse it into the JSON object.

pgmoneta_json_write_file

Convert the JSON to string and write it to a JSON file.

18 Troubleshooting

18.1 Could not get version for server

If you get this `FATAL` during startup check your PostgreSQL logins

```
psql postgres
```

and

```
psql -U repl postgres
```

And, check the PostgreSQL logs for any error.

Setting `log_level` to `DEBUG5` in `pgmoneta.conf` could provide more information about the error.

19 Acknowledgement

19.1 Authors

pgmoneta was created by the following authors:

```
Jesper Pedersen <jesperpedersen.db@gmail.com>
David Fetter <david@fetter.org>
Will Leinweber <will@bitfission.com>
Luca Ferrari <fluca1978@gmail.com>
Nikita Bugrovsky <nbugrovs@redhat.com>
Mariam Fahmy <mariamfahmy66@gmail.com>
Jichen Xu <kyokitisin@gmail.com>
Saurav Pal <resyfer.dev@gmail.com>
Bokket <bokkett@gmail.com>
Haoran Zhang <andrewzhr9911@gmail.com>
Hazem Alrawi <hazemalrawi7@gmail.com>
Shahryar Soltanpour <shahryar.soltanpour@gmail.com>
Shikhar Soni <shikharish05@gmail.com>
Nguyen Cong Nhat Le <lenguyencongnhat2001@gmail.com>
Chao Gu <chadraven369@gmail.com>
Luchen Zhao <lucian.zlc@gmail.com>
Joan Jeremiah J <joanjeremiah04@gmail.com>
Iury Santos <iuryroberto@gmail.com>
Palak Chaturvedi <palakchaturvedi2843@gmail.com>
Jakub Jirutka <jakub@jirutka.cz>
Mario Rodas
Annupamaa <annu242005@gmail.com>
Ashutosh Sharma <ash2003sharma@gmail.com>
Mohab Yaser <mohabyaserofficial2003@gmail.com>
Georg Pfuetzenreuter <mail@georg-pfuetzenreuter.net>
Ahmed Ashour <a8087027@gmail.com>
Sangkeun J.C. Kim <jchrys@me.com>
Tejas Tyagi <tejastyagi.tt@gmail.com>
Aryan Arora <aryanarora.w1@gmail.com>
Arshdeep Singh <balarsh535@gmail.com>
Din Xin Chen <s990346@gmail.com>
Mingzhuo Yin <yinmingzhuo@gmail.com>
Vanes Angelo <k124k3n@gmail.com>
Bassam Adnan <mailbassam@gmail.com>
```

19.2 Committers

```
Jesper Pedersen <jesperpedersen.db@gmail.com>
Haoran Zhang <andrewzhr9911@gmail.com>
```

19.3 Contributing

Contributions to **pgmoneta** 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 X as well.

20 License

```
Copyright (C) 2025 The pgmoneta 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.
```