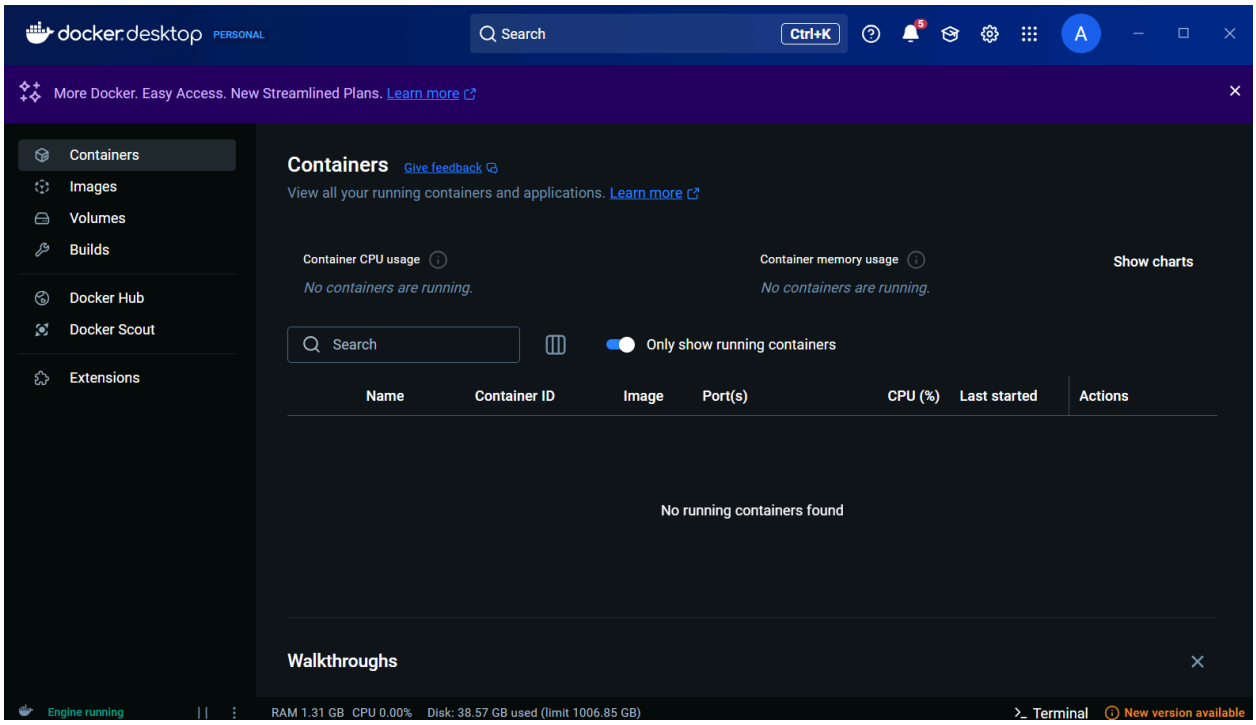
ENVIRONMENTAL SETUP AND LEARNING PROCESS

DOCKER AND ENVIRONMENT SETUP

To establish a robust development environment with all dependencies and ensure portability, Docker is used. The goal was to create an environment where any change in the codebase reflects in the project without manual intervention, as this is a full-fledged product rather than a single-file program. Docker Compose plays a crucial role in achieving this by managing multiple containers efficiently.

Docker Desktop is installed from the website. Then, the project folders are cloned from the GitHub repository, and Docker Compose is used to set up the environment, ensuring that pgwatch, Grafana, and the PostgreSQL database are running for monitoring.



git clone https://github.com/cybertec-postgresql/pgwatch.git && cd pgwatch
docker compose -f ./docker/docker-compose.yml up --detach

To gain a deeper understanding of the PostgreSQL server, the pgAdmin container is also included:

docker compose -f ./docker/docker-compose.yml up --detach pgadmin

The different components of the project run on various ports:

PgAdmin: port 80

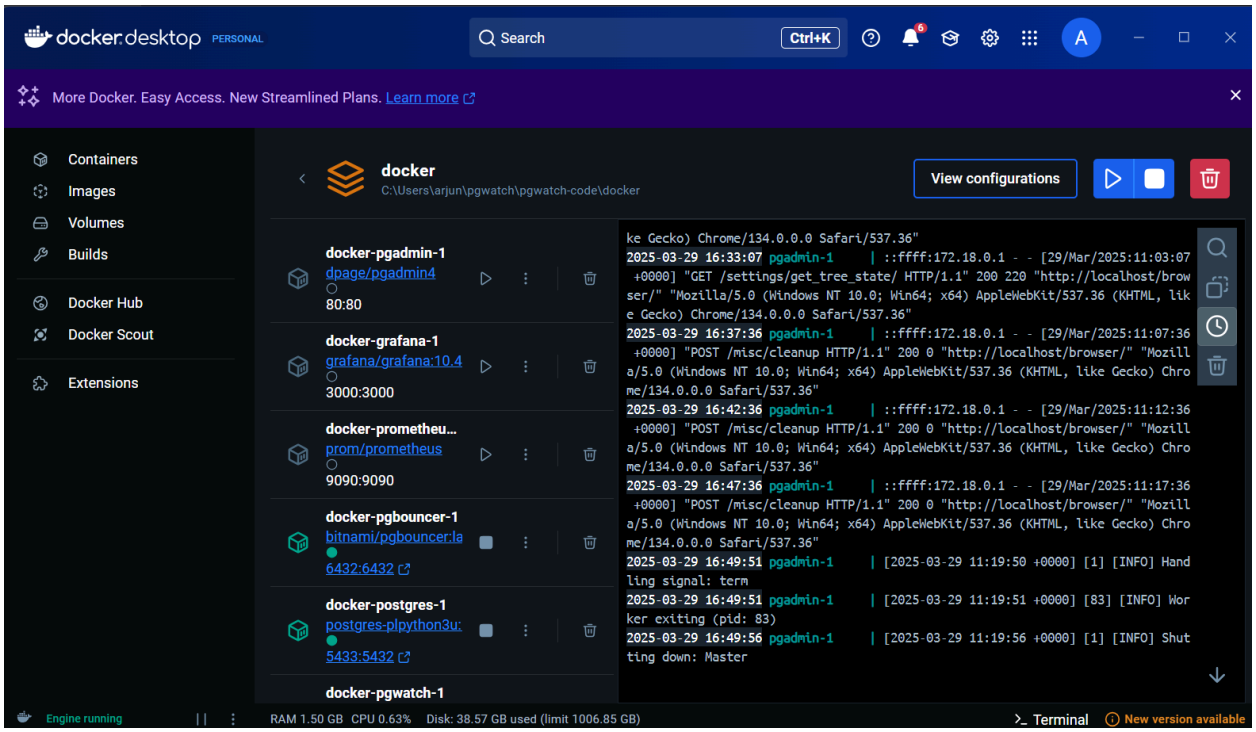
Pgwatch: port 8080

Grafana: port 3000

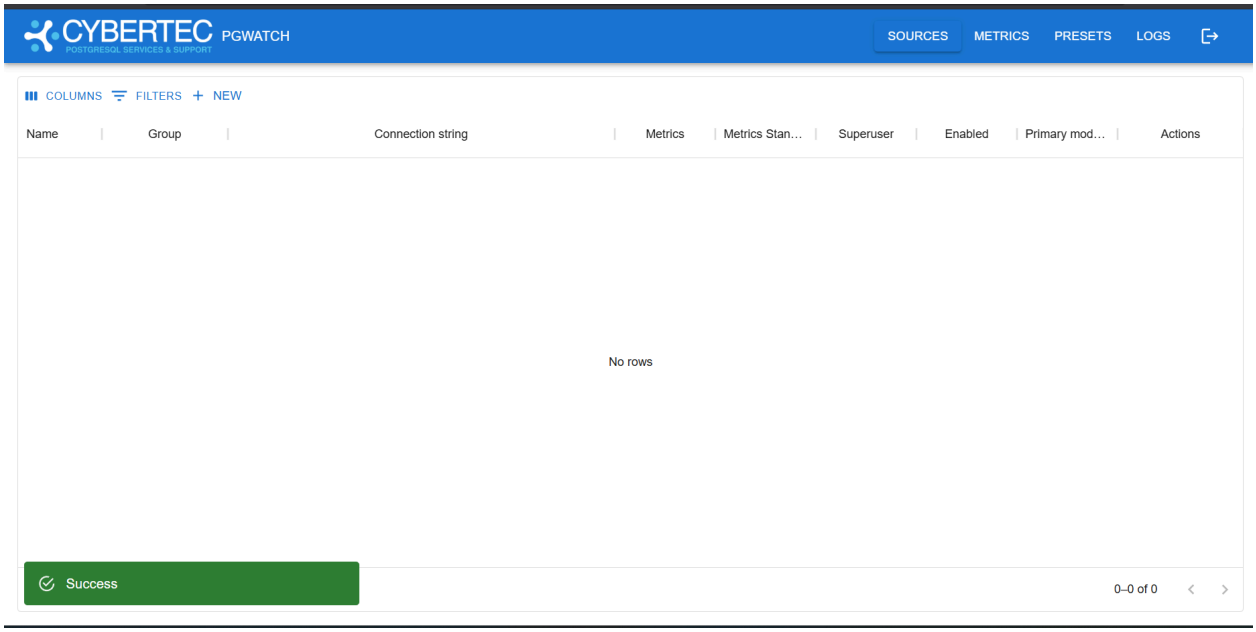
Prometheus: port 9090

PostgreSQL: port 5433 (originally on 5432, but changed locally to 5433 to accommodate pgwatch)

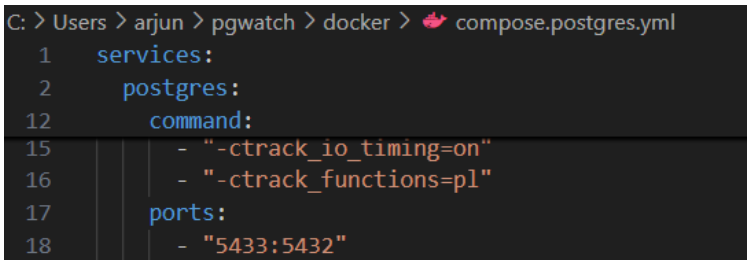
Using:
docker compose -f ./docker/docker-compose.yml up --detach



we effectively set up a development environment where changes in the codebase are reflected in the running application without the need for manual reconfiguration, streamlining the development process.

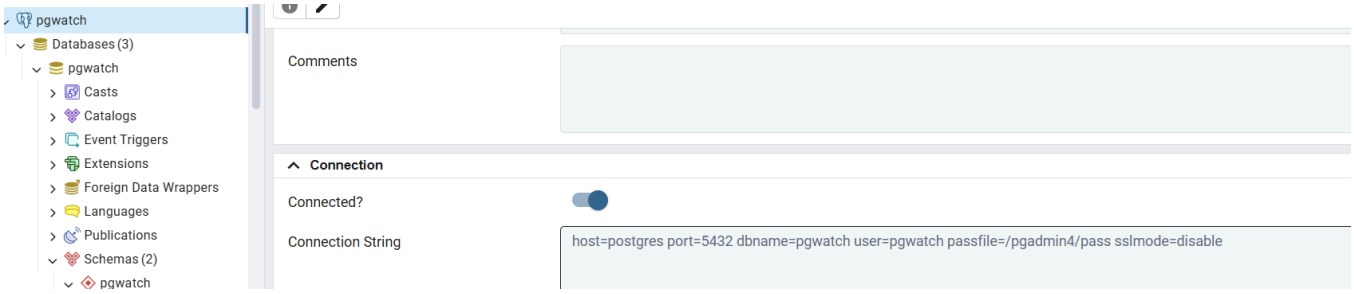


The pgwatch container is changed to run 5433 on the local ports by modifying the compose.postgres.yml file in the docker folder.

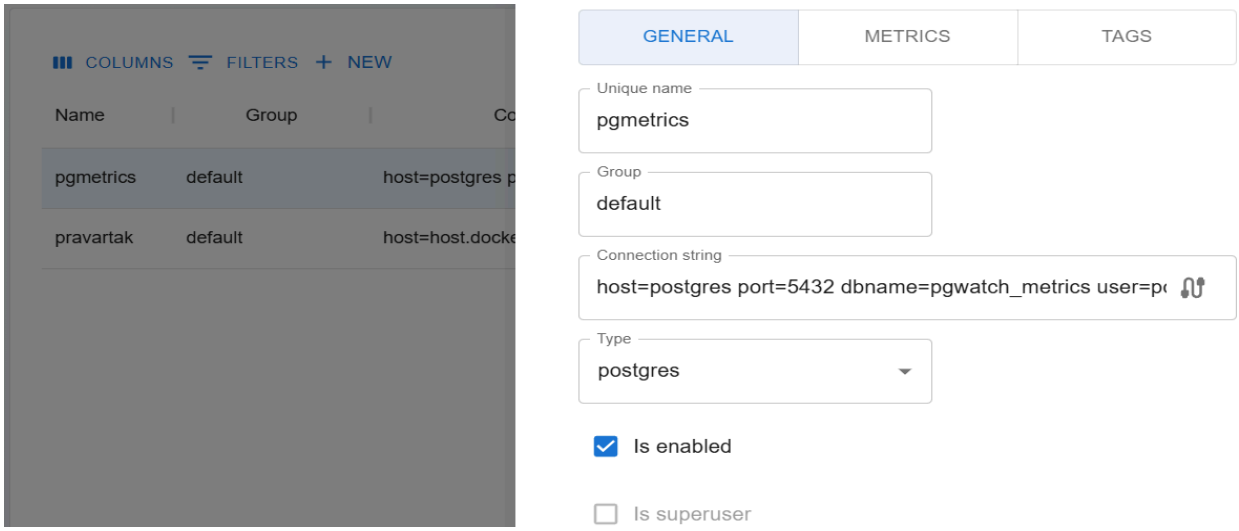


CONNECTING DATABASES TO BE MONITORED

Two databases where connected with pgwatch to be monitored. We can connect them using a connection string, which was obtained from the properties of the databases in pgadmin.



This connection string is given in pgwatch(ui) to connect the database to be monitored.This is done for a database in the postgres container we installed.



To connect a database from the postgres locally installed,which is another port, through research the below given modification to hostname was used.

As a base requirement the following instructions are run on the database for connecting and fetching metrics.

CREATE ROLE pgwatch WITH LOGIN PASSWORD 'secret';
-- For critical databases it might make sense to ensure that the user account
-- used for monitoring can only open a limited number of connections
-- (there are according checks in code, but multiple instances might be launched)
ALTER ROLE pgwatch CONNECTION LIMIT 5;
GRANT pg_monitor TO pgwatch;
GRANT CONNECT ON DATABASE mydb TO pgwatch;
GRANT EXECUTE ON FUNCTION pg_stat_file(text) to pgwatch; -- for wal_size metric
GRANT EXECUTE ON FUNCTION pg_stat_file(text, boolean) TO pgwatch;

For more detailed metrics, like “per query” performance aggregates, the following should be done:

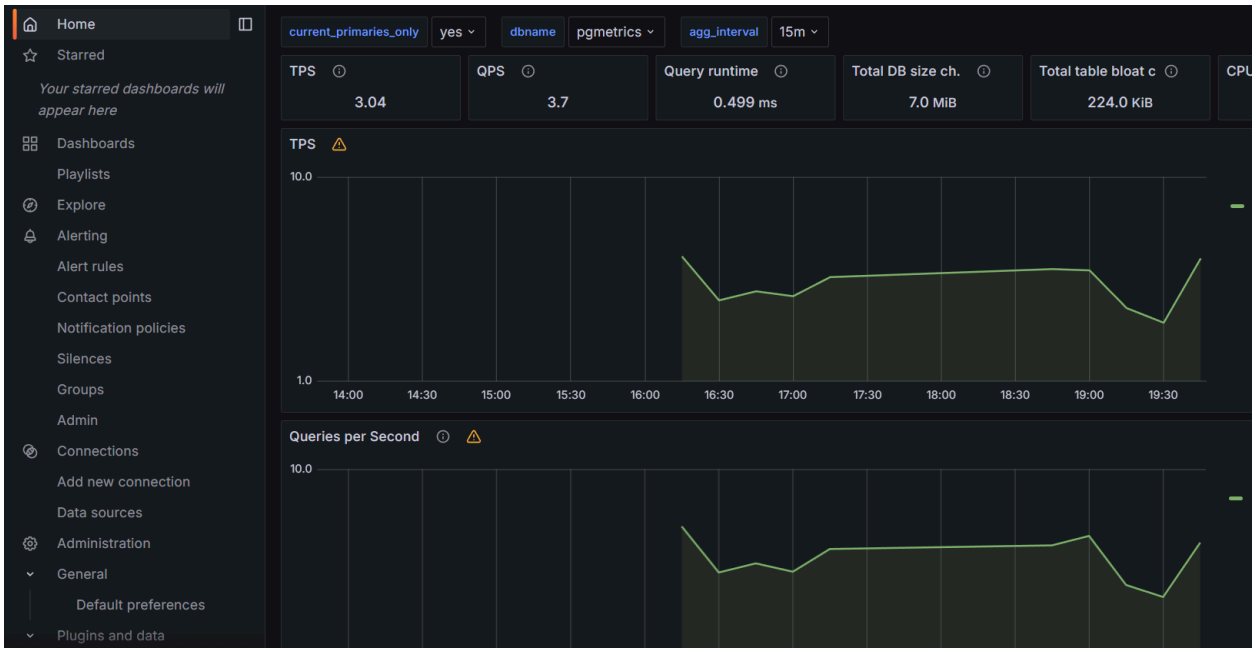
Add pg_stat_statements to your server config (postgresql.conf) and restart the server.

shared_preload_libraries = 'pg_stat_statements'
track_io_timing = on

After restarting activate the extension in the monitored DB. Assumes Postgres superuser.

psql -c "CREATE EXTENSION IF NOT EXISTS pg_stat_statements"

After connecting them in this way, they where monitored and was seen as from grafana ui.



EXPLORING THE DATABASES OF THE PROJECT

The databases where data is stored is explored. It contains a database pgwatch_metrics where the metrics are stored as well as pgwatch where the description for the metrics are stored.

```
                                                List of databases
     Name      |  Owner   | Encoding | Locale Provider |  Collate   |   Ctype    | Locale | ICU Rules |   Access privileges
---------------+----------+----------+-----------------+------------+------------+--------+-----------+-----------------------
 pgwatch       | pgwatch  | UTF8     | libc            | en_US.utf8 | en_US.utf8 |        |           |
 pgwatch_metrics | pgwatch  | UTF8     | libc            | en_US.utf8 | en_US.utf8 |        |           |
 postgres      | postgres | UTF8     | libc            | en_US.utf8 | en_US.utf8 |        |           |
 template0     | postgres | UTF8     | libc            | en_US.utf8 | en_US.utf8 |        |           | =c/postgres          +
               |          |          |                 |            |            |        |           | postgres=CTc/postgres
 template1     | postgres | UTF8     | libc            | en_US.utf8 | en_US.utf8 |        |           | =c/postgres          +
               |          |          |                 |            |            |        |           | postgres=CTc/postgres
(5 rows)
```

Here we can see how changing the database(pgwatch) can be seen in the ui of pgwatch.

```
pgwatch=> update metric set description='updated' where name='archiver';
UPDATE 1
```

Name	Description	InitSQL	Node status	Gauges	Instance level?	Storage name
archiver	updated			is_failing_int,seconds_sin…	✓	
backends				*		
backup_age_pgb…		CREATE EXTENSION IF …			✓	
backup_age_walg		CREATE EXTENSION IF …			✓	
bgwriter			primary		✓	

The database pgwatch_metrics contain the the tables where the recorded metrics are stored, like db_stats,db_size,stat_statements_calls etc where different live metrics that are recorded are stored.

```
You are now connected to database "pgwatch_metrics" as user "pgwatch".
pgwatch_metrics=> \dt
                    List of relations
 Schema |              Name               |       Type        |  Owner
--------+---------------------------------+-------------------+---------
 public | archiver                        | partitioned table | pgwatch
 public | backends                        | partitioned table | pgwatch
 public | bgwriter                        | partitioned table | pgwatch
 public | change_events                   | partitioned table | pgwatch
 public | checkpointer                    | partitioned table | pgwatch
 public | configuration_changes           | partitioned table | pgwatch
 public | configured_dbs                  | partitioned table | pgwatch
 public | cpu_load                        | partitioned table | pgwatch
 public | db_size                         | partitioned table | pgwatch
 public | db_stats                        | partitioned table | pgwatch
 public | index_changes                   | partitioned table | pgwatch
```

Each of these contain four columns, time,dbname,data and tagdata. Thus we can see that it is a time series database that stores the recorded data from each table at intervals of time.
Below given are some examples of recorded data
db_stats:

	time timestamp with time zone	dbname text	data jsonb
1	2025-03-21 12:45:04.328285+00	pgmetrics	{"sys_id": "7483764217247694875", "blks_hit": 30290, "sessions": 27, "blks_rea
2	2025-03-21 12:46:09.807231+00	pgmetrics	{"sys_id": "7483764217247694875", "blks_hit": 38692, "sessions": 28, "blks_rea
3	2025-03-21 12:47:15.363337+00	pgmetrics	{"sys_id": "7483764217247694875", "blks_hit": 42009, "sessions": 31, "blks_rea
4	2025-03-21 12:48:20.853984+00	pgmetrics	{"sys_id": "7483764217247694875", "blks_hit": 44488, "sessions": 34, "blks_rea
5	2025-03-21 12:49:26.327884+00	pgmetrics	{"sys_id": "7483764217247694875", "blks_hit": 47013, "sessions": 37, "blks_rea
6	2025-03-21 12:50:31.717514+00	pgmetrics	{"sys_id": "7483764217247694875", "blks_hit": 51033, "sessions": 39, "blks_rea

stat_statements_calls:

	time timestamp with time zone	dbname text	data jsonb
1	2025-03-28 07:28:54.073532+00	pgmetrics	{"calls": 1643, "sys_id": "7483764217247694875", "total_time": 969.332, "real_d
2	2025-03-28 07:29:59.87421+00	pgmetrics	{"calls": 2133, "sys_id": "7483764217247694875", "total_time": 1243.451, "real_
3	2025-03-28 07:31:05.846601+00	pgmetrics	{"calls": 2235, "sys_id": "7483764217247694875", "total_time": 1253.156, "real_
4	2025-03-28 07:32:11.697991+00	pgmetrics	{"calls": 2352, "sys_id": "7483764217247694875", "total_time": 1297.566, "real_
5	2025-03-28 07:33:17.628176+00	pgmetrics	{"calls": 2503, "sys_id": "7483764217247694875", "total_time": 1342.008, "real_
6	2025-03-28 07:34:23.634523+00	pgmetrics	{"calls": 2631, "sys_id": "7483764217247694875", "total_time": 1366.202, "real_

From exploring, it seems as if grafana shows the metrics like tps, qps etc by querying this databases with the appropriate functions to get the values for them.

BASIC UNDERSTANDING OF SOURCE CODE

Frontend (Web UI)

- Framework: React.js (v18.2.0)
- Language: TypeScript
- UI Components: Material-UI (MUI) v5

Backend

Language: Go (Golang)
Architecture: Microservices-based with several components:

- webserver: Main API server
- metrics: Metrics collection and processing
- sources: Data sources
- sinks: Data output handlers
- reaper: Cleanup/maintenance tasks
- db: Database interactions