METRO|NOM

Use Docker, Go, Rest and some more to build a PostgreSQL multi cloud self service

PGCONF US, NEW YORK

MARCH 2020

About me

Heiko Onnebrink - DBA – developer - open-source contributor

300 relatives in China + 3 kids in Germany

working 25 year as Oracle employee, trainer, consultant & freelancer

I decided 3 years ago to stop doing anything with Oracle products

Currently principal domain architect of core infrastructure @ METRO|NOM

heiko.onnebrink@metronom.com

@HeikoOnnebrink

THE BIGGEST SOFTWARE COMPANY YOU NEVER HEARD ABOUT

METRONOM – SETTING THE PACE IN FOOD AND TECHNOLOGY

METRO | NOM

- 150.000 Employees
- €37 Billion
- 25 Countries

- 2000 Employees
- IT-Services
- IT-Solutions

Relational Database @ METRO|NOM

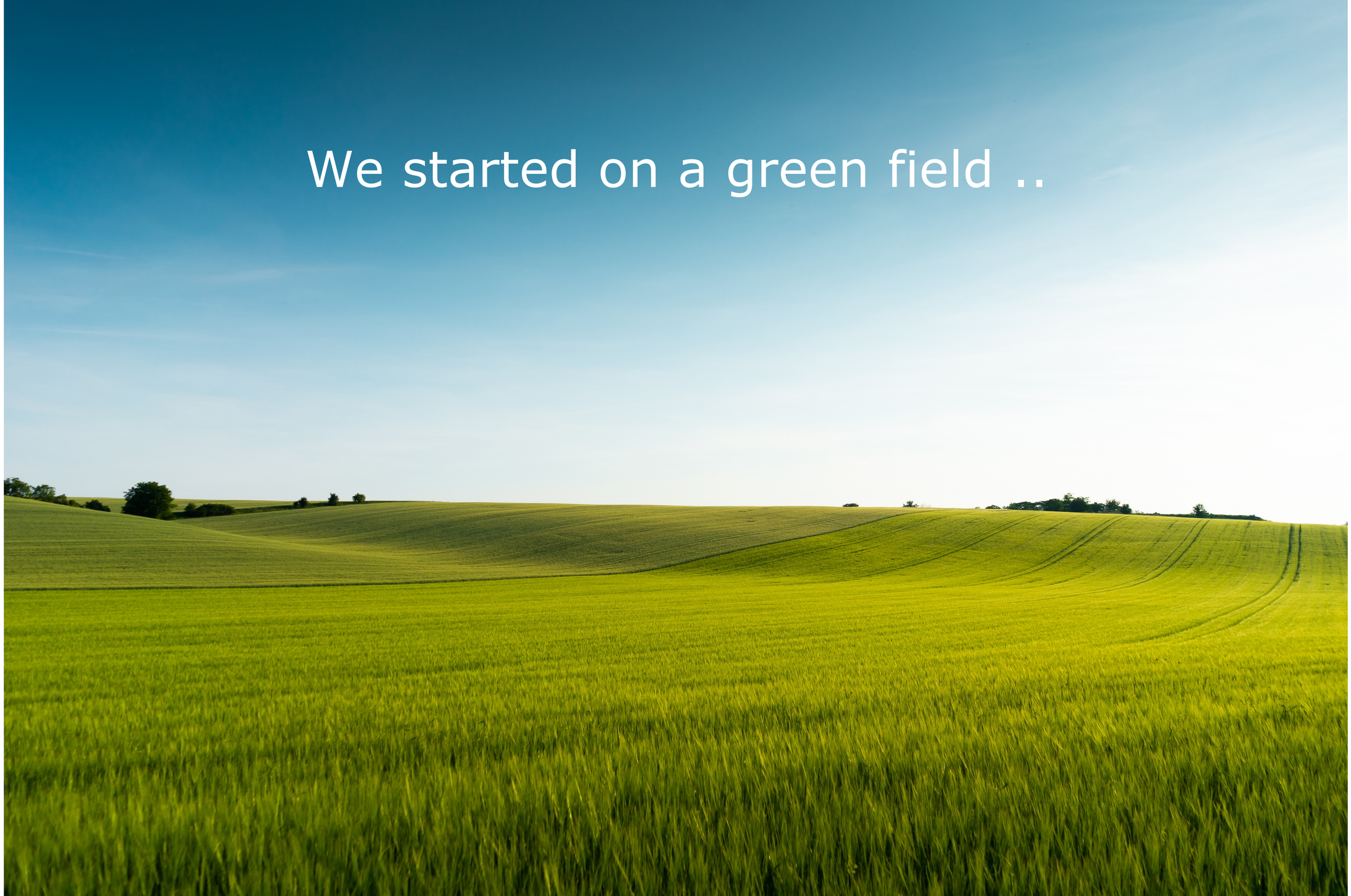
Facts

- IT grown over decades
- 3000+ Oracle databases
- ticket driven organisation
- slow delivery time
- low level of automatisation

Challenges

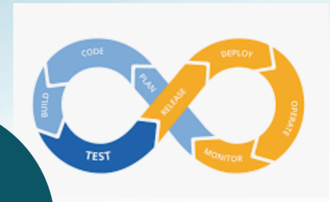
- Digital Readiness projects
- alternative RDBMS
- support DevOps / DBaaS
- full automation
- self service
- run across any cloud, on prem and public

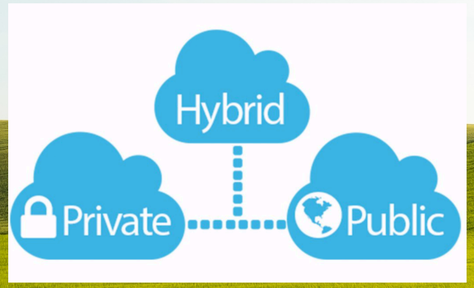

TIME FOR
SOMETHING
NEW!

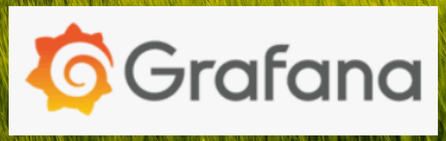

We started on a green field ..


.. and looked around what‘s out there

.. and looked around what's out there



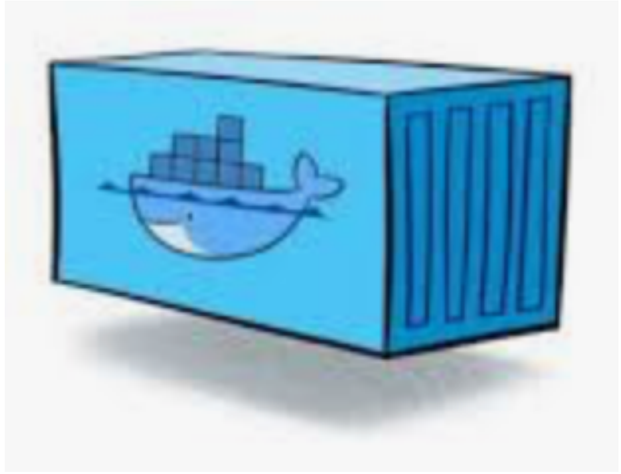




CHAPTER # 1

Containers

We started and build some containers ..



The PostgreSQL container

Based on Ubuntu Image

PostgreSQL database with GIS

WAL-G for wal shipping

FDW for PostgreSQL and Oracle

Oracle instant client

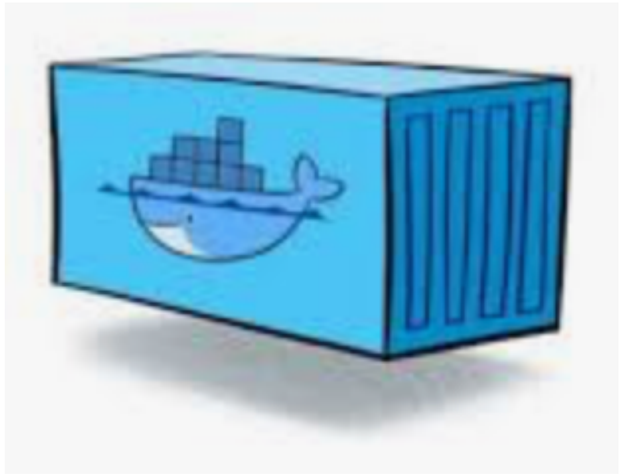
pg_activity

pg_audit

Logical Decoding plugins (Json/Protobuf) for Kafka Integration

Timescaledb & Prometheus Extension

... and an entrypoint / shell script which contains the logic



The PostgreSQL container (continued)

Features:

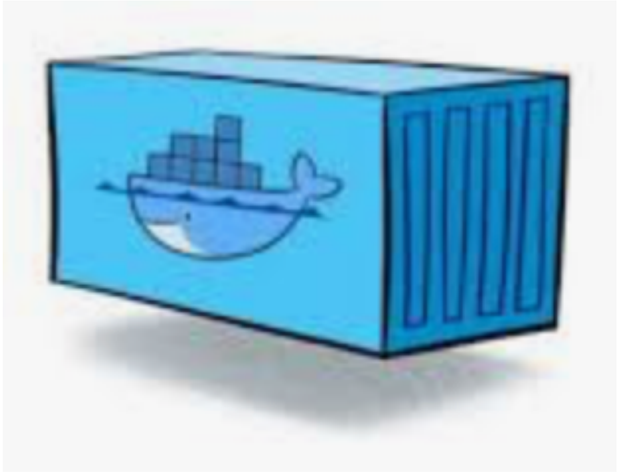
create a new database

- from scratch
- as a slave of some existing db
- as a clone of some existing db

run a major release upgrade of a database

run a full / PIT /restore-point based recovery

WAL-G



The Backup Container

Created from the PostgreSQL container, so it already contains WAL-G binaries.

.. just contains a different entrypoint / shell script with some logic

Features:

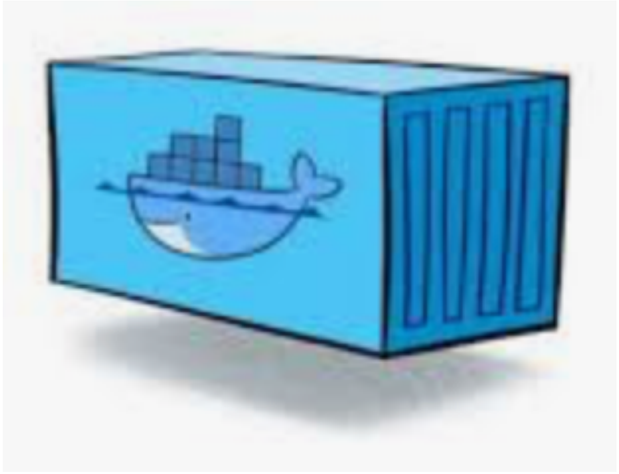
write backups straight to object storage (S3, SWIFT, GCS)

allow full or incremental backups

added functionality to allow self-contained backups for database not running wal shipping (by temporarily activating wal shipping during backup execution)

also used to do backup cleanup on object storage

The Metric Container



There was already something .. so lets use it ..

Features:

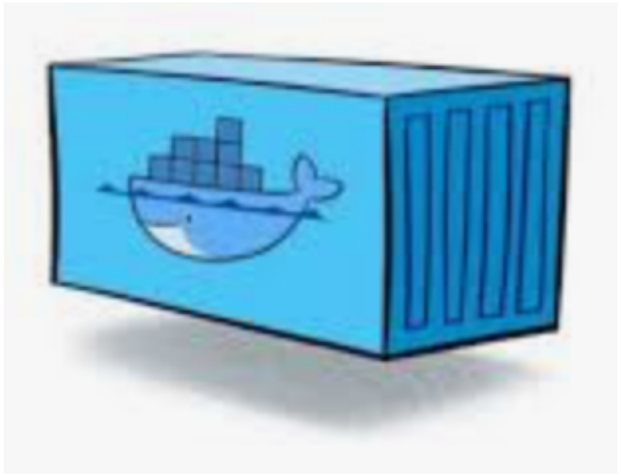
out of the box metric sources for most relevant OS level metrics

out of the box PostgreSQL metrics

easy extensible to add any custome metrics

easy to containerize

open architecture.. can be used also for any other metric source



The backend and helper container

used wherever possible ready build containers from docker hub

Features:

Influxdb : time series database to store db and host metrics

Grafana : web based dashboard to visualize metrics

pgAdmin : web based admin tool

pgBadger: transforms db logs into HTTP web content

goStatic : minimalistic HTTP server used to publish pgBadger reports

we were able now to ..

spin up some db like this:

docker run registry.metronom.com/rdb-dev/postgresql:11

we were able now to ..

create db with usr/pwd, mount volume inside and ship wal to S3 storage :

```
docker run --rm –d --name postgresql –d
-v /dockerdata/postgresql:/postgresqldata
-e username=heiko –e password=supersecret –db=appdb1
-e DBUUID=5b54d046-c8f3-447c-8e56-247b7493b4df
-e BACKUP_TYPE=S3
-e BACKUP_URI=//backup.postgresqlbackup.dus30hcp01.asf.madm.net
-e BACKUP_CREDENTIALS=“someid:somesecret"
-e PG_WRITE_ARCHIVE=true
registry.metronom.com/rdb-dev/postgresql:11
```

and we were able to do all the other day2 operations :

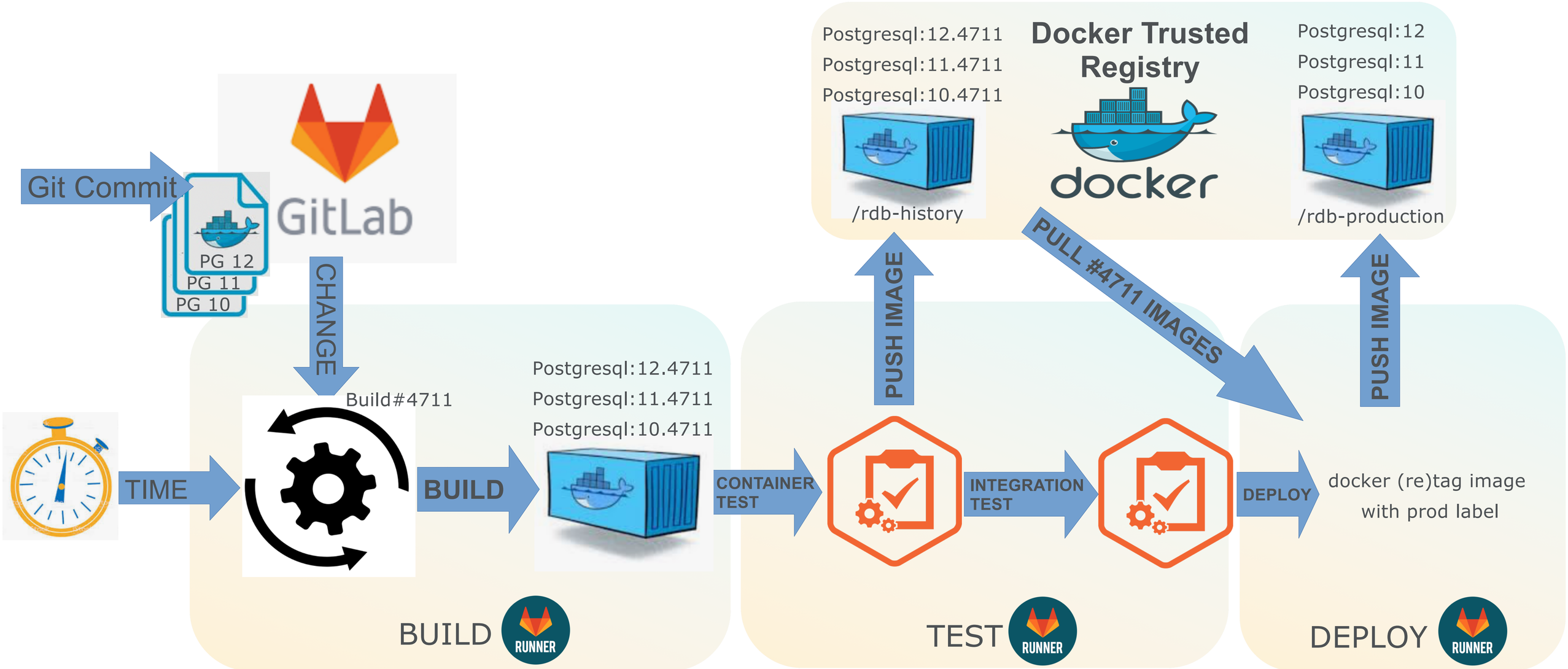
- take a backup from a running db to some object store
- run a major PostgreSQL release upgrade
- restore/recover a db with all recovery options
- collect metrics and ship them to some metric repo
- publish metrics using Grafana container
- offer pgAdmin web
- publish pgBadger reports to web using HTTP container on DB node

But who will continously build and test all these containers ..

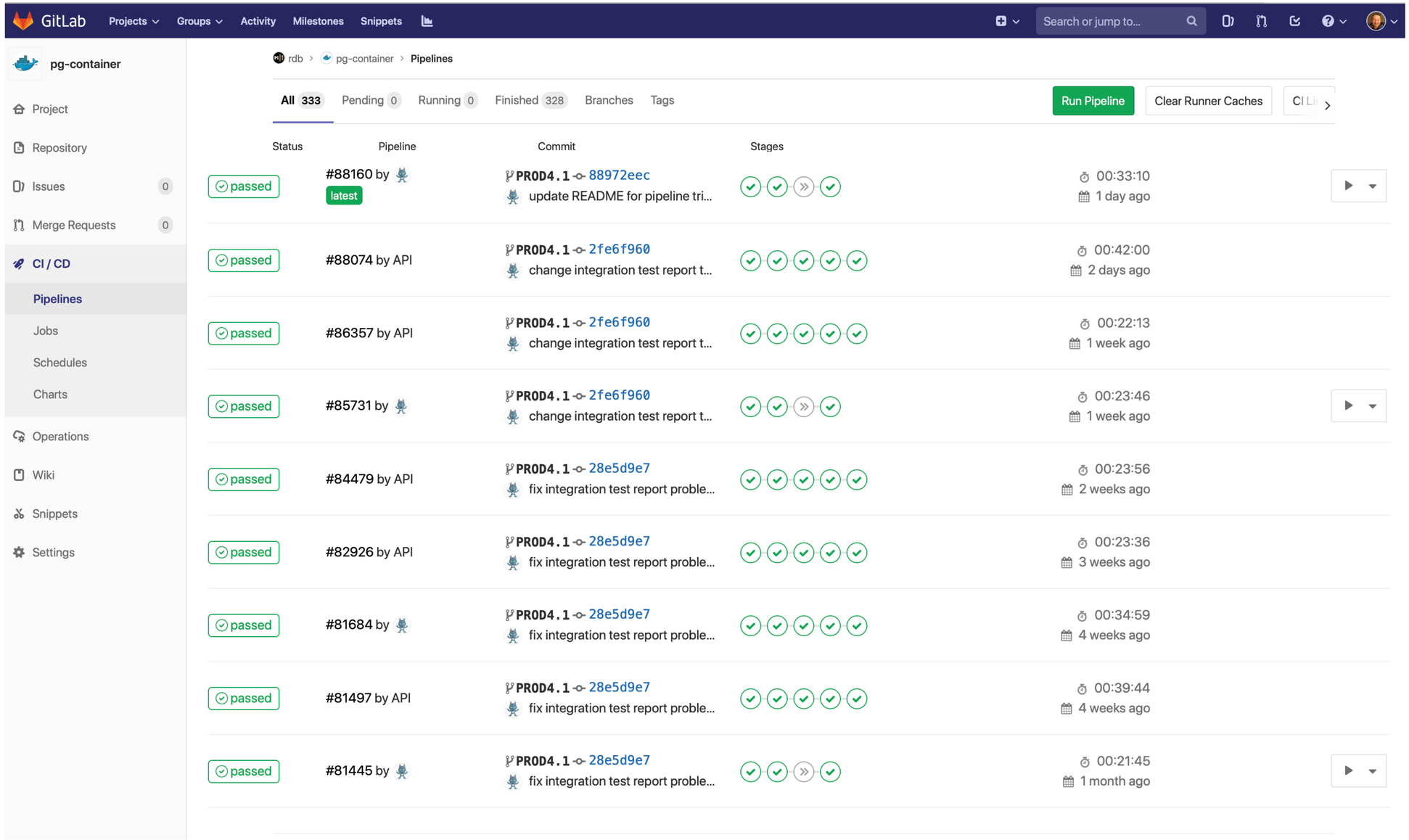
CHAPTER # 2

The container build pipeline

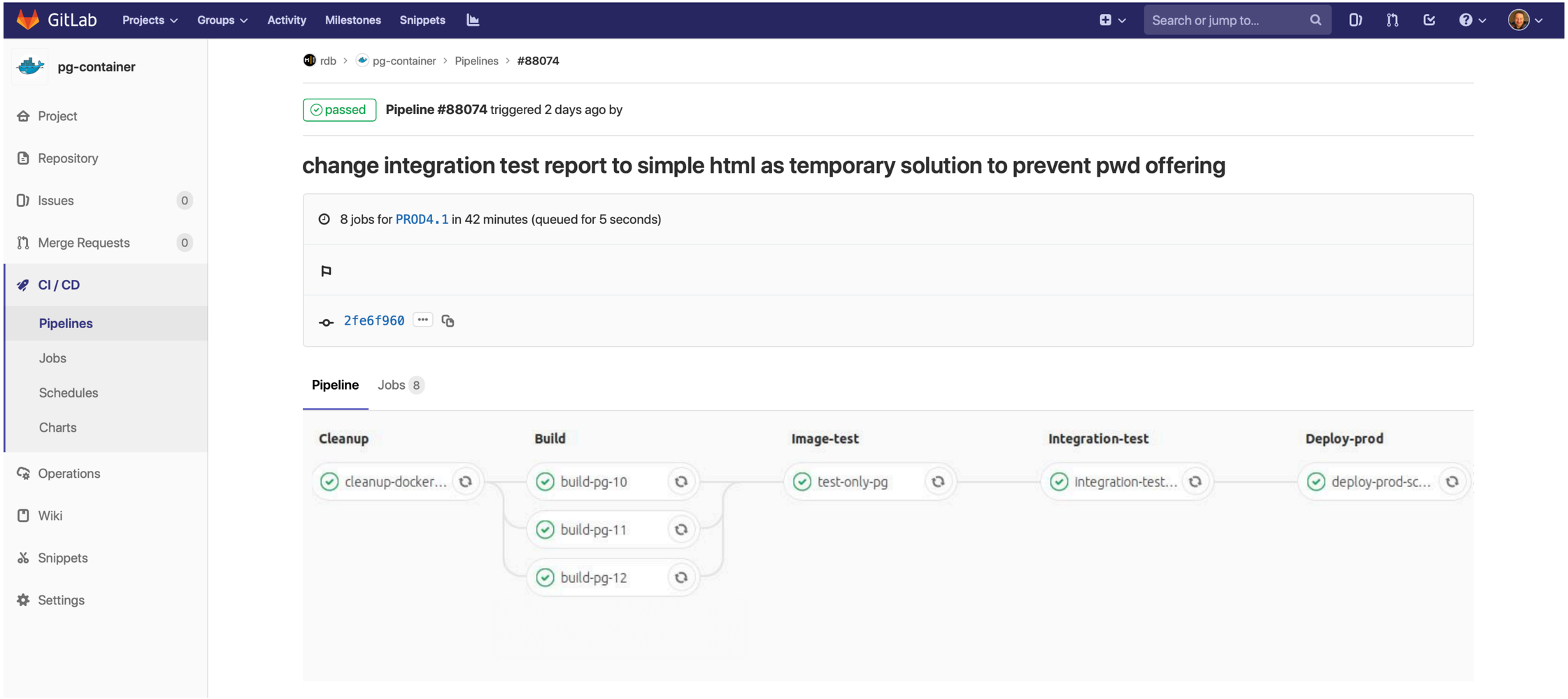
We created next some container build pipeline ..



Container build pipeline (continued)



Container build pipeline (continued)

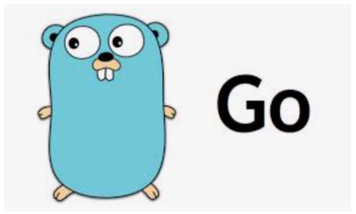


How do we make now a Self Service out of it

CHAPTER # 3

The API first approach !

We build some REST API ..



RESTful

follows micro service principals

written in Go

consumes other REST APIs of cloud and storage providers

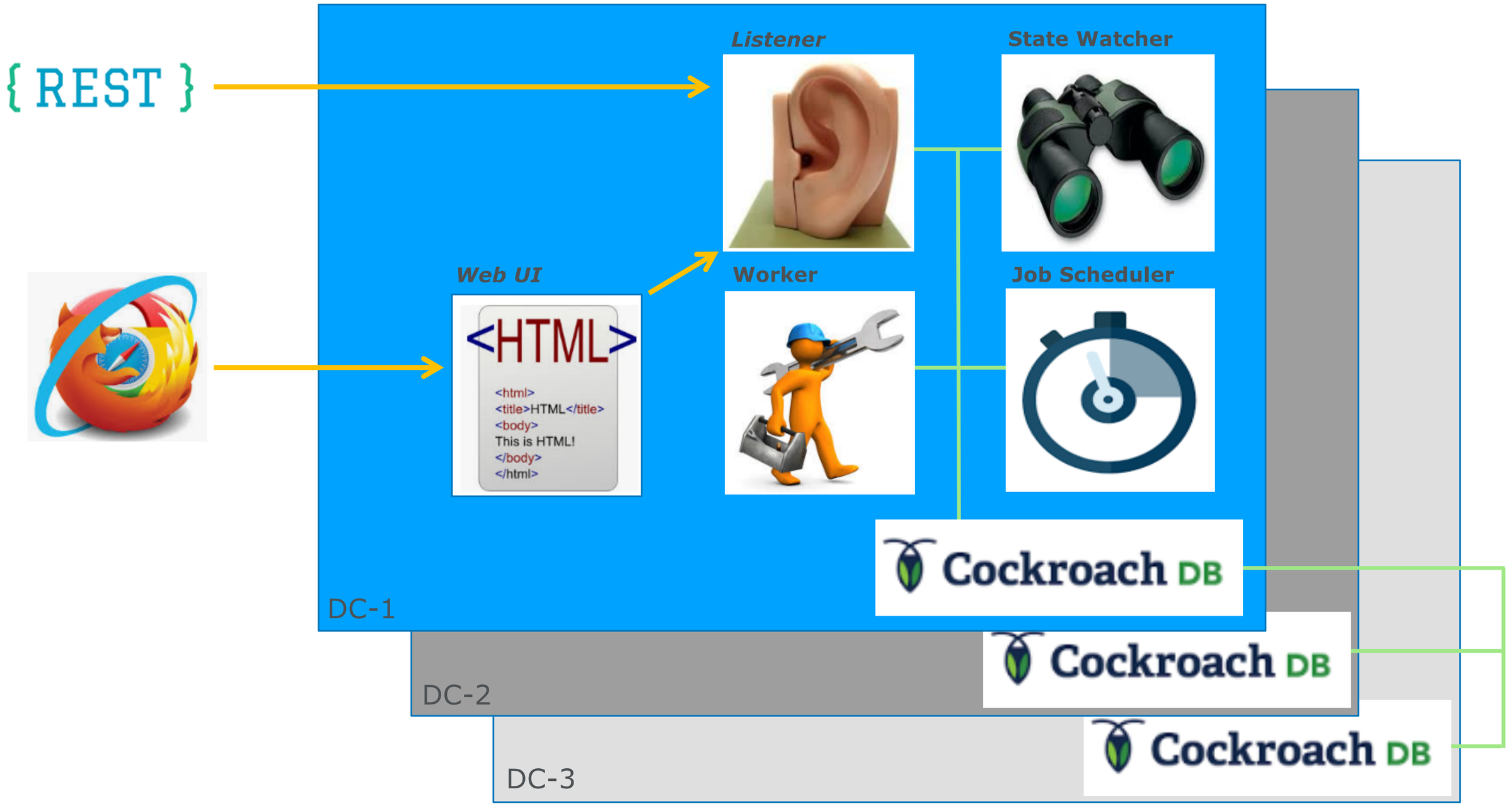
ultra high availability as it is runs across several DCs

backed by Cockroach DB (solving for us all the distributed problems)

secured by OAuth 2.0

terminated by a distributed load balancer

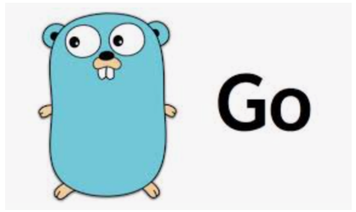
PostgreSQL DBaaS Restful - API



CHAPTER # 4

The self service

In the end we build the Self Service portal ..



written in Go (using Go HTML templates, jQuery, Twitter bootstrap)

designed with the target to have a great user experience

wizard for db creation incl. jobs and reports

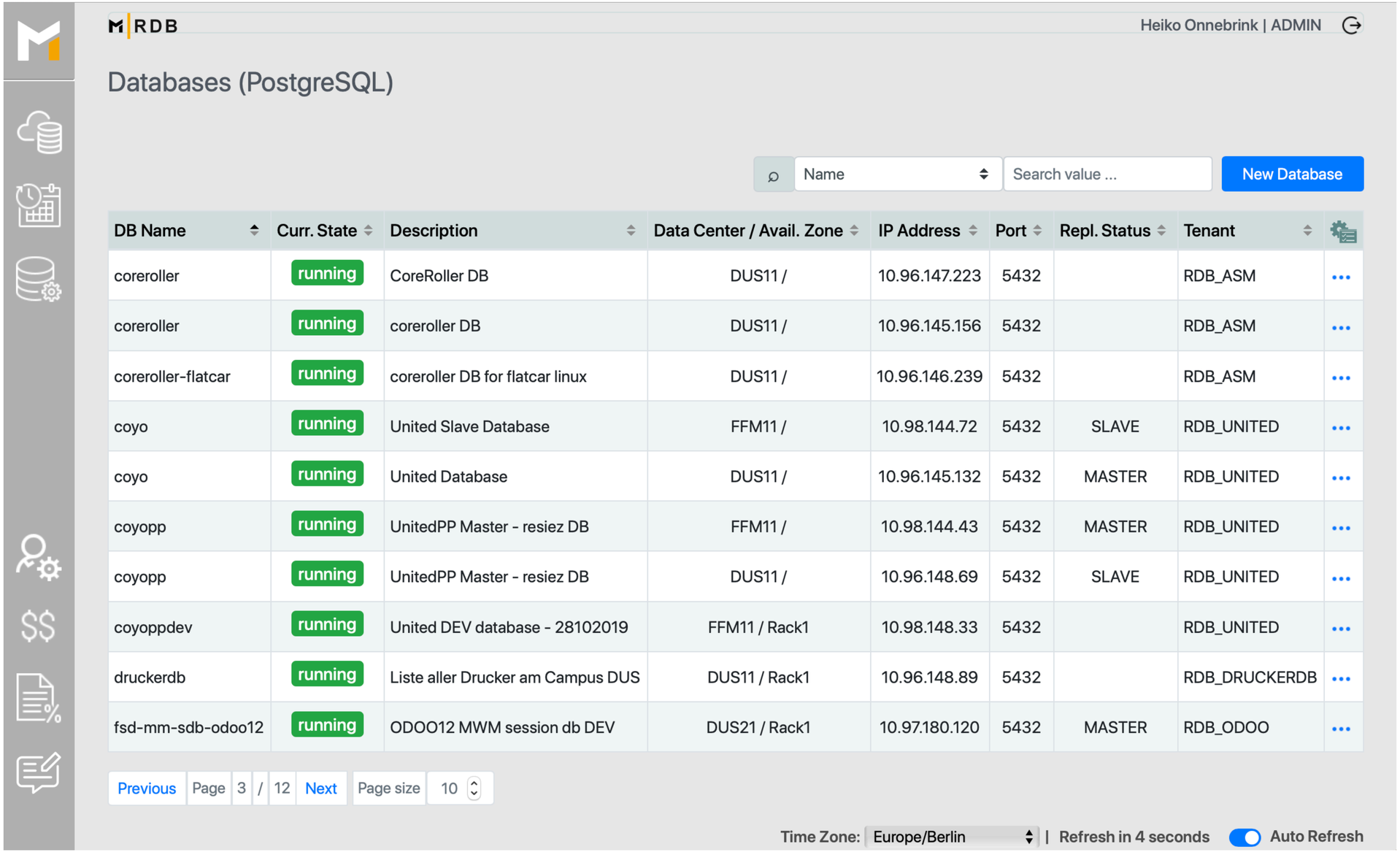
allow self-service config/backup/restore/recovery

expose dashboards and pgBadger reports to DevOps teams

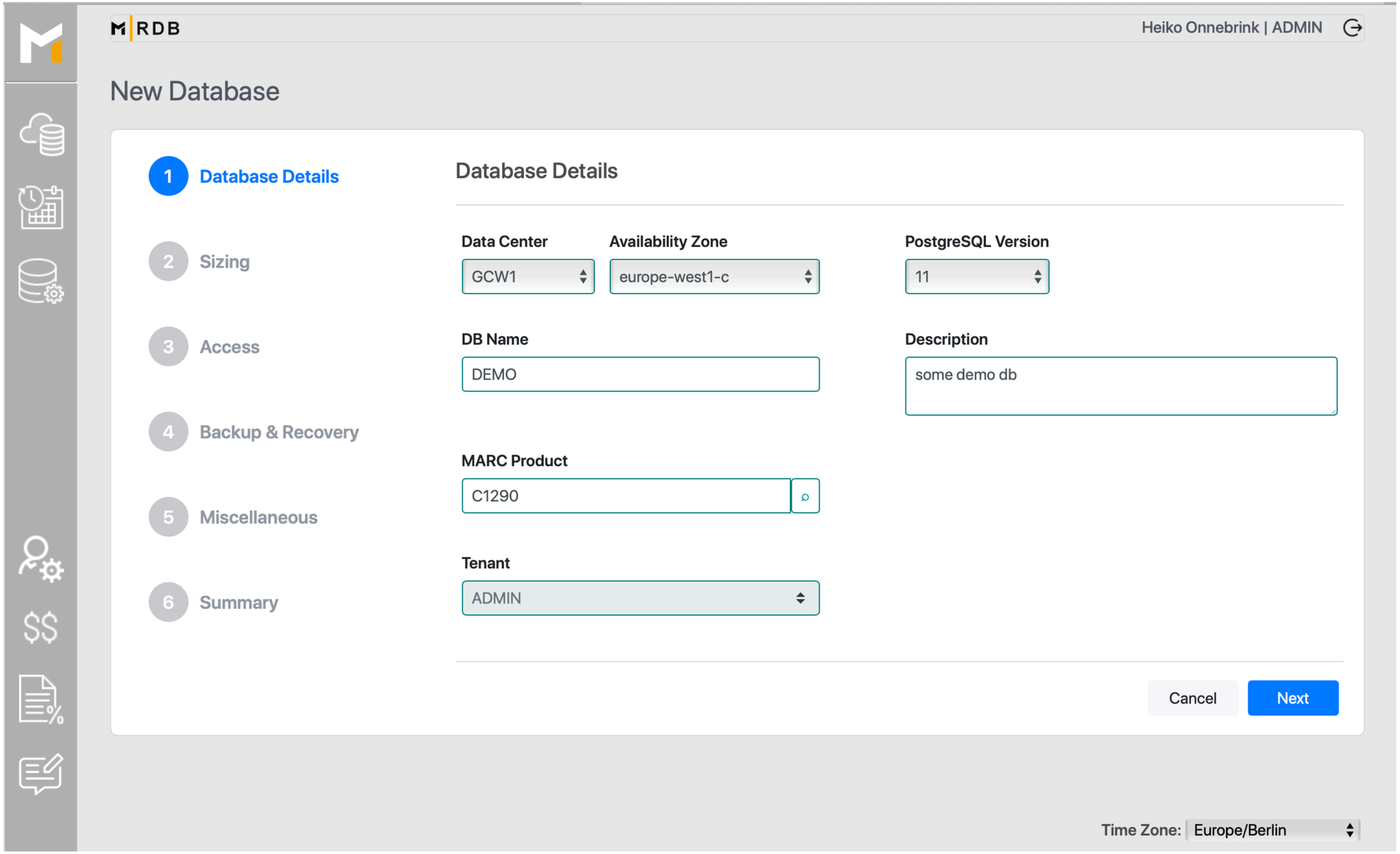
pay per use

integrate customer feedback so that we better learn how to improve our service

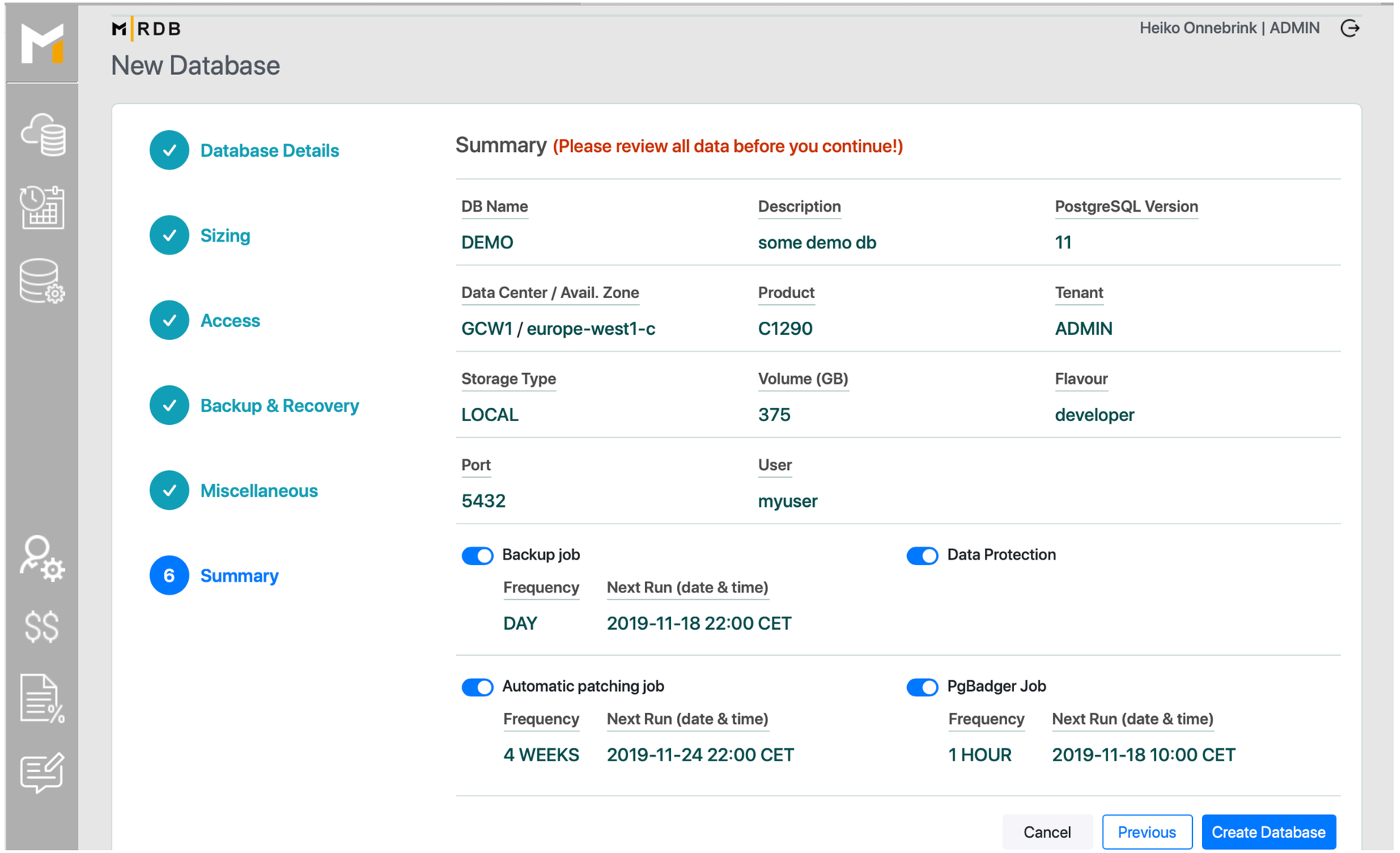
Intuitive to use..



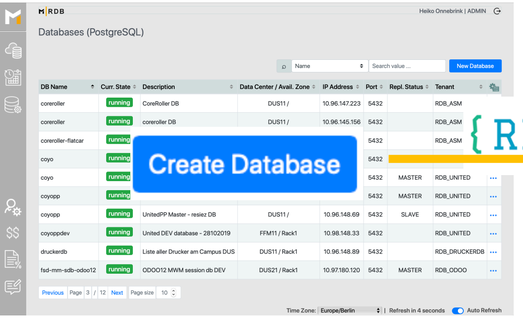
wizard driven DB (and job) creation..



summary screen ..



What happens in the background when a DB gets created

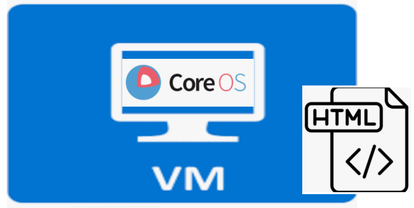


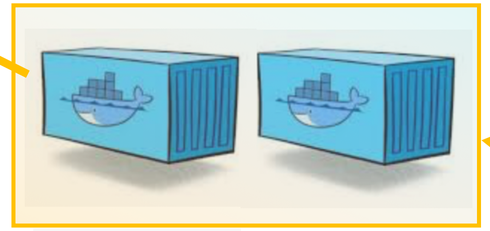
{ REST }

{ REST }
Ignition Config

build machine

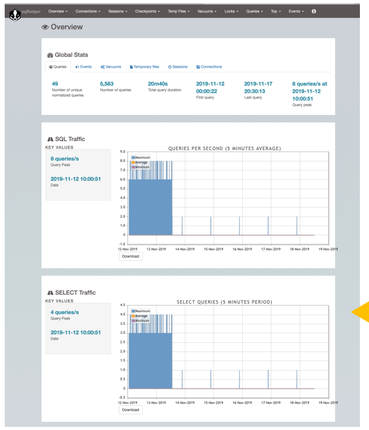
{ REST }
request container start

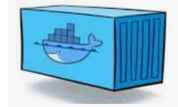




Grafana influxdb

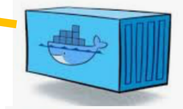
send metrics



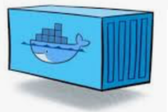


PostgreSQL

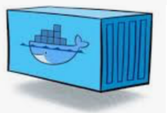
ship WAL



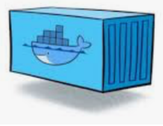
Metric Collector



pgBadger

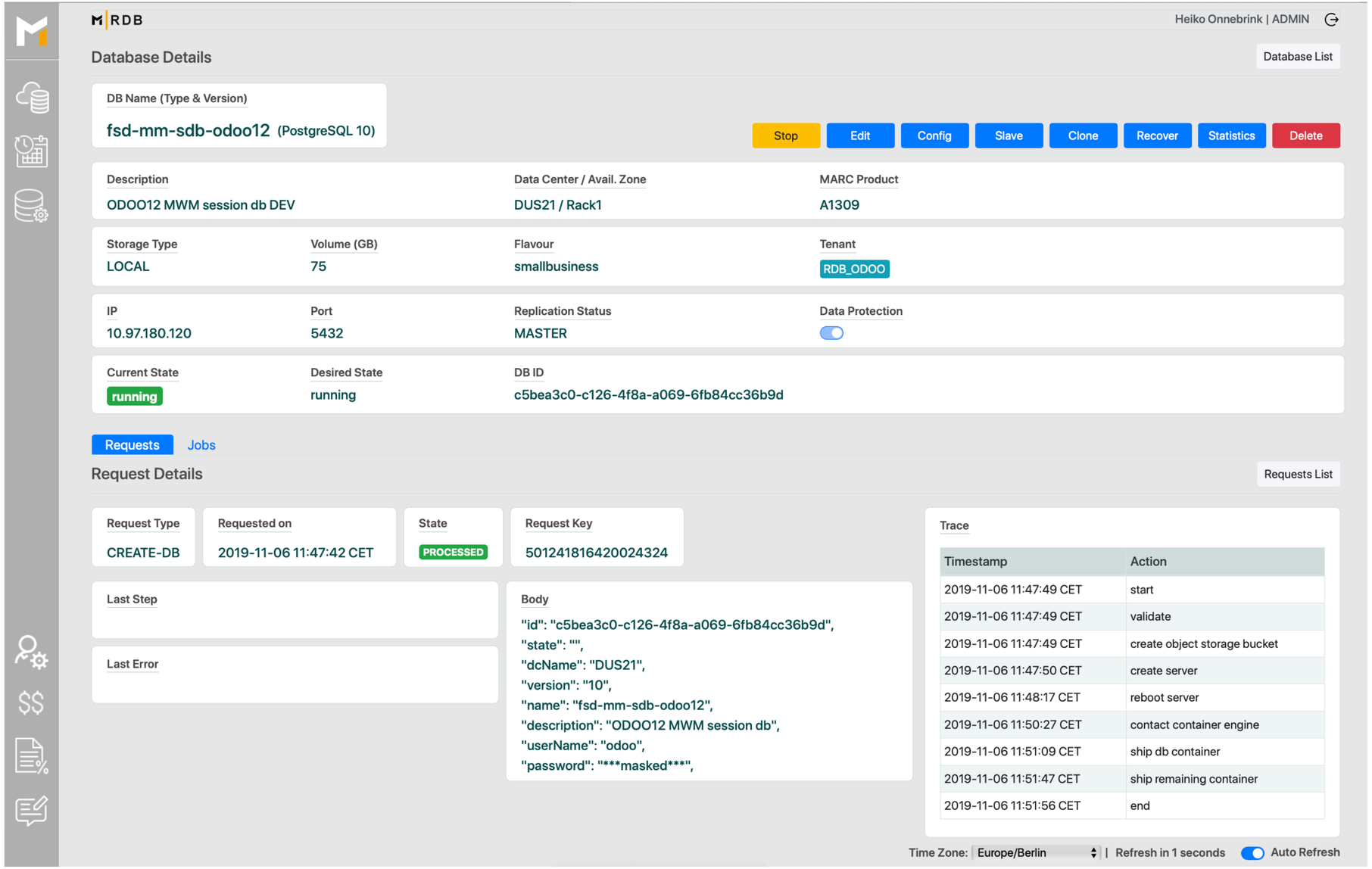


HTTP server

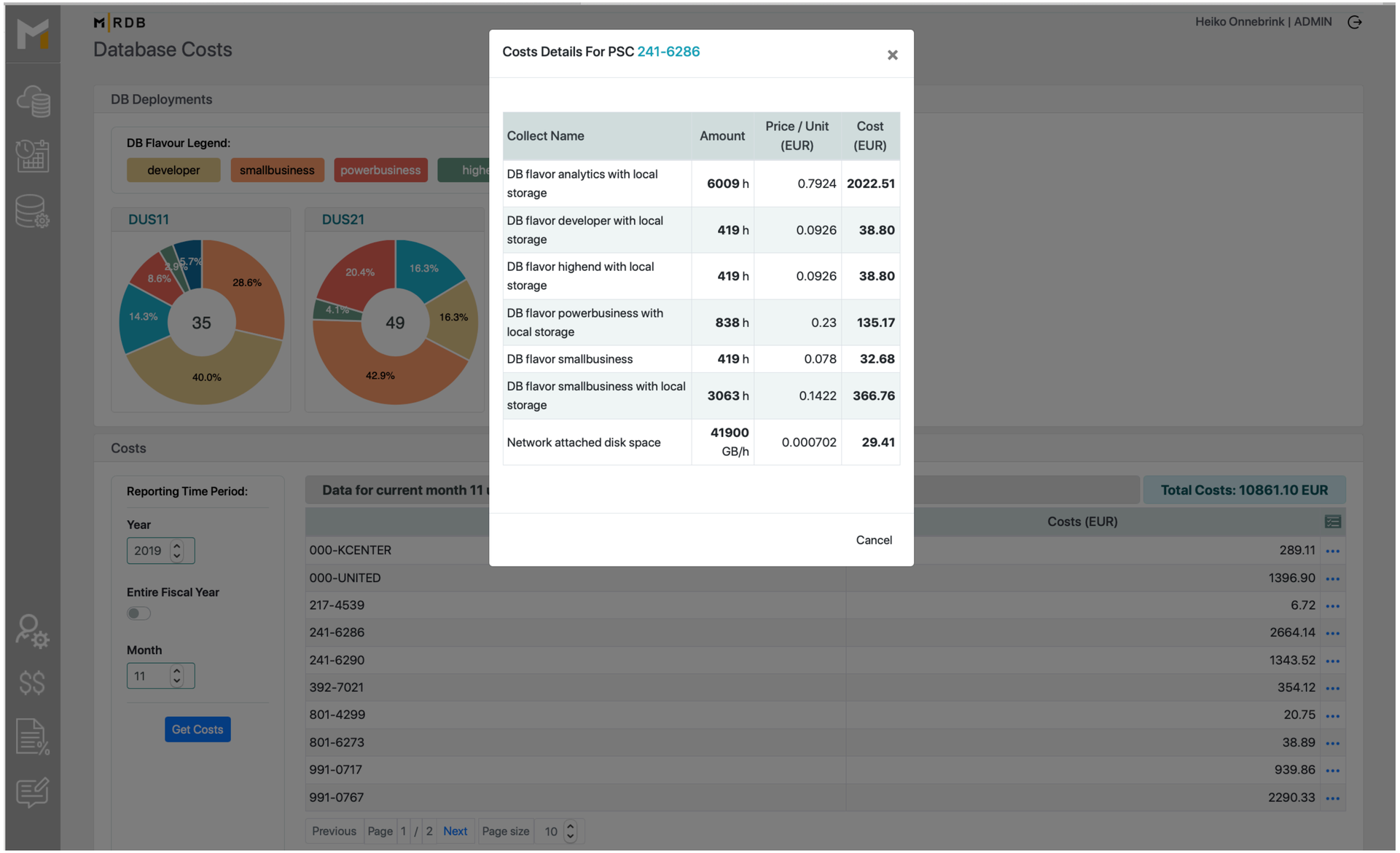


WAL-G

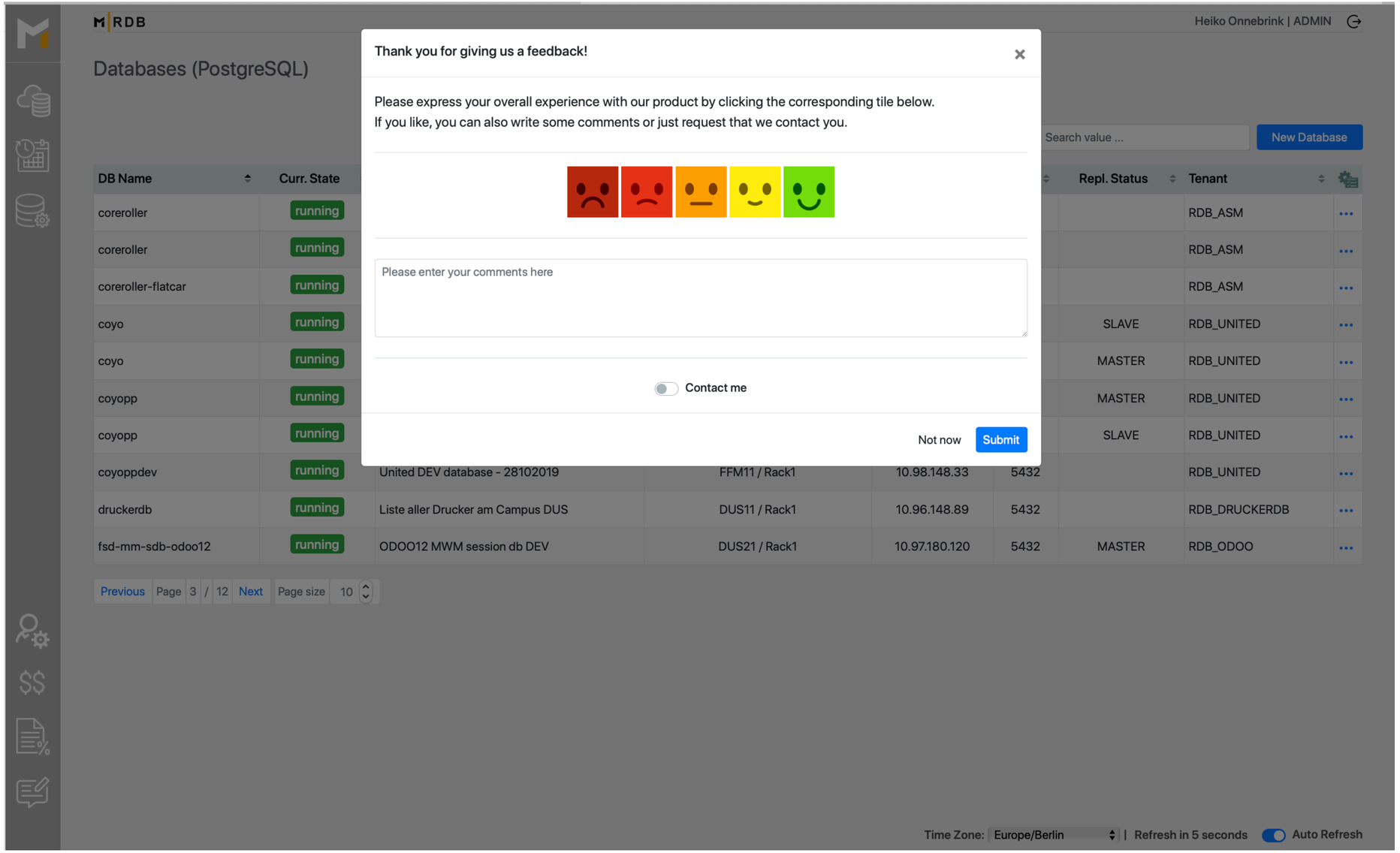
DB details with request and job views ..



cost details per tenant ..



and we value customer feedback..



LAST CHAPTER

lessons learned

- automize everything you do more than once
- use automated testing from the very beginning
- know your customer, build what he needs and make him productive
- start with a minimal viable product (MVP)
- grow from here feature by feature
- it takes some time to get there.. but its worth the effort

once you got there you will gain time, quality and happy customer..

.. and have much more time to go to conferences and give a talk ☺

Thank you for your attention!

the PostgreSQL team @

METRO | NOM