BUILD WITH CONFIDENCE

A Deep Dive into Database Security with PostgreSQL

Roberto Mello

crunchy data

About me

- Working with databases and PostgreSQL since 1999
- BS/MS in Computer Science from Utah State
- Principal Solutions Architect with Crunchy Data
- Outside of work: more tech stuff (raspberry pis, home automation, home servers, etc)
- Outside of tech: snowboarding, Austrian economics, scouting, travelling

Why Postgres

Established, Reliable and Secure

Feature Rich

Extensibility

No Central Owner

Hiring (Open Source)

The Technical Details

- Datatypes
- Transactional DDL
- Foreign Data Wrappers
- Concurrent Index Creation
- Conditional indexes
- JSON
- Common Table Expressions
- Fast column addition
- Listen/Notify
- Upsert
- Partitioning
- Per transaction sync replication
- Window function
- JSON/JSONB
- Continued innovation

Crunchy Data

- Leading Team in Postgres (core contributors)
- Certified Open Source Postgres Distribution
- Leader in Postgres Technology for Kubernetes
- Crunchy Bridge: Fully Managed Multi-Cloud Postgres
- Enterprise Focus and Go to Market Approach

crunchy data

Enterprise Ready Software Built on Open Source

Open Source Software

- Postgres
- Backups
- Disaster Recovery
- High availability
- Connection scaling
- Monitoring

Crunchy Data Subscription Value Add

- Packaging
- Integration
- Documentation
- Automation
- Certification
- 24x7 Commercial Support

Crunchy Data Products

Crunchy Postgres

Crunchy Postgres is Production Ready Postgres

Backups

Data is one of your most valuable assets, Crunchy Postgres ensures your data is safe and backed up so you can sleep easy.

High availability

With high availability you can trust your database is online.

Monitoring

Get the insights you need to know what is happening with your database with monitoring included.

Disaster recovery

When the worst happens, we have you covered. Never worry about data corruption or loss with point-in-time recovery.

Connection scaling

With built in connection scaling you can easily scale to tens of thousands of connections for your database.

No lock-in

With Crunchy Data and open-source Postgres you are not locked-in to proprietary technology.

crunchy data

Postgres Anywhere

BARE METAL, VMS, CLOUD

Crunchy Postgres

Crunchy Certified PostgreSQL is production ready Postgres.

INCLUDES:

- Backups
- Disaster recovery
- High availability
- Monitoring
- Automation
- Self managed

KUBERNETES

Crunchy Postgres for Kubernetes

Cloud Native Postgres on Kubernetes powered by Crunchy Postgres Operator.

INCLUDES:

- Simple provisioning
- Backups and DR included
- High availability
- Seamless upgrades
- Scale from 1 to thousands of databases
- Self managed

FULLY MANAGED CLOUD

Crunchy Bridge

The fully managed Postgres option on your choice of Cloud provider.

INCLUDES:

- AWS, Azure or GCP
- Continuous protection
- Backups
- Point in Time Recovery
- Full superuser access
- The developer experience you want

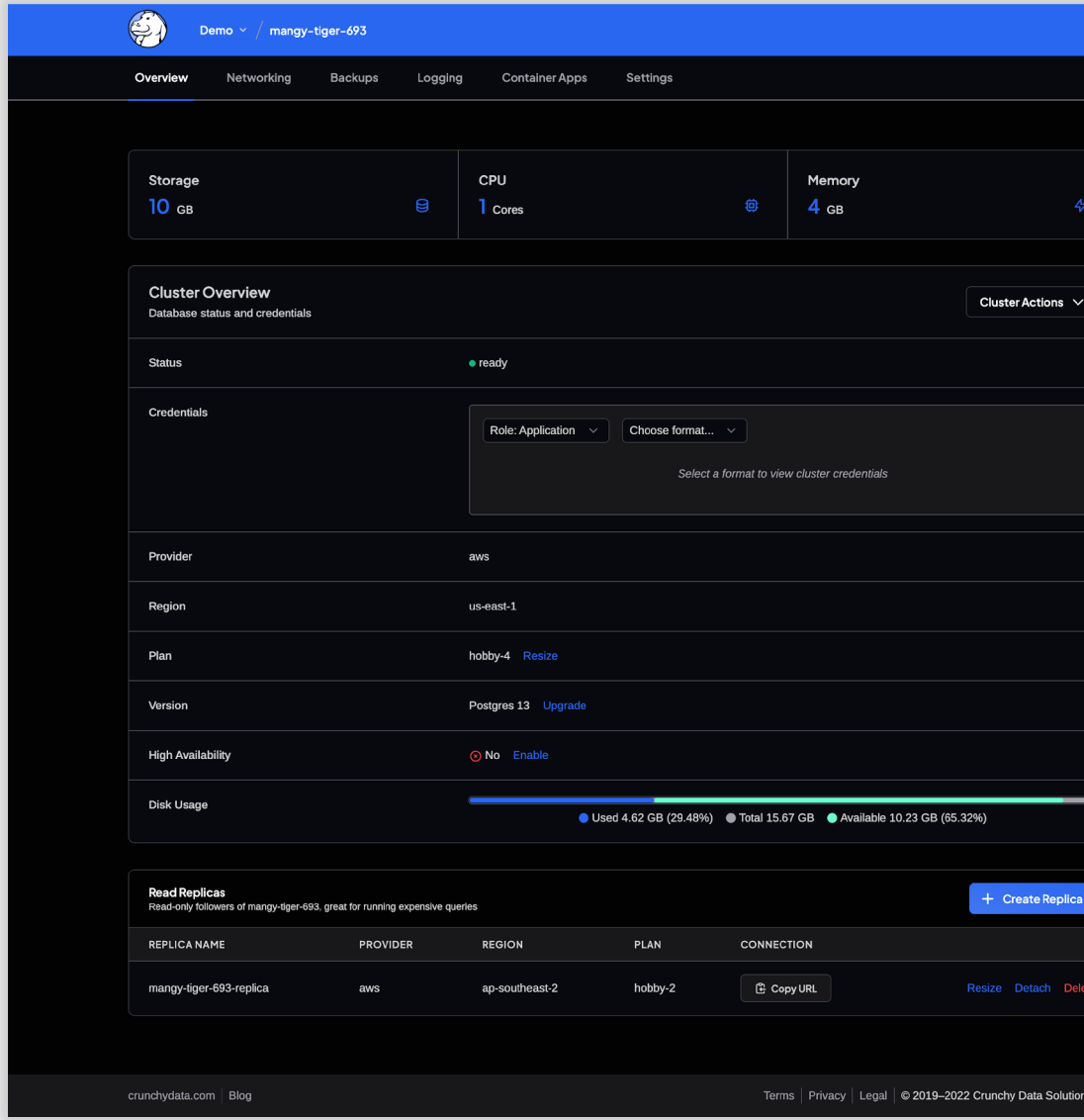
crunchy data

Crunchy Bridge

Cloud Postgres built for a superior developer experience

On Crunchy Bridge, Developer Experience (DX) means a toolset that gets you into the zone. Our platform has the tools necessary for developers to execute the optimum decision for performance and scale today while looking into the future.

“The Crunchy team is awesome to work with, very responsive, very smart, and are clearly committed to getting the customer issues solved. We are working with Crunchy for the people and the technology.” Seth Pollack, RivalIQ Founder



crunchy data

What is a Database?

- An advanced SQL frontend to the file system
- A data access platform
- Daemon listening on (multiple) port(s)
- Files underneath
  - Local or remote storage?
- Transaction logs
- Backups

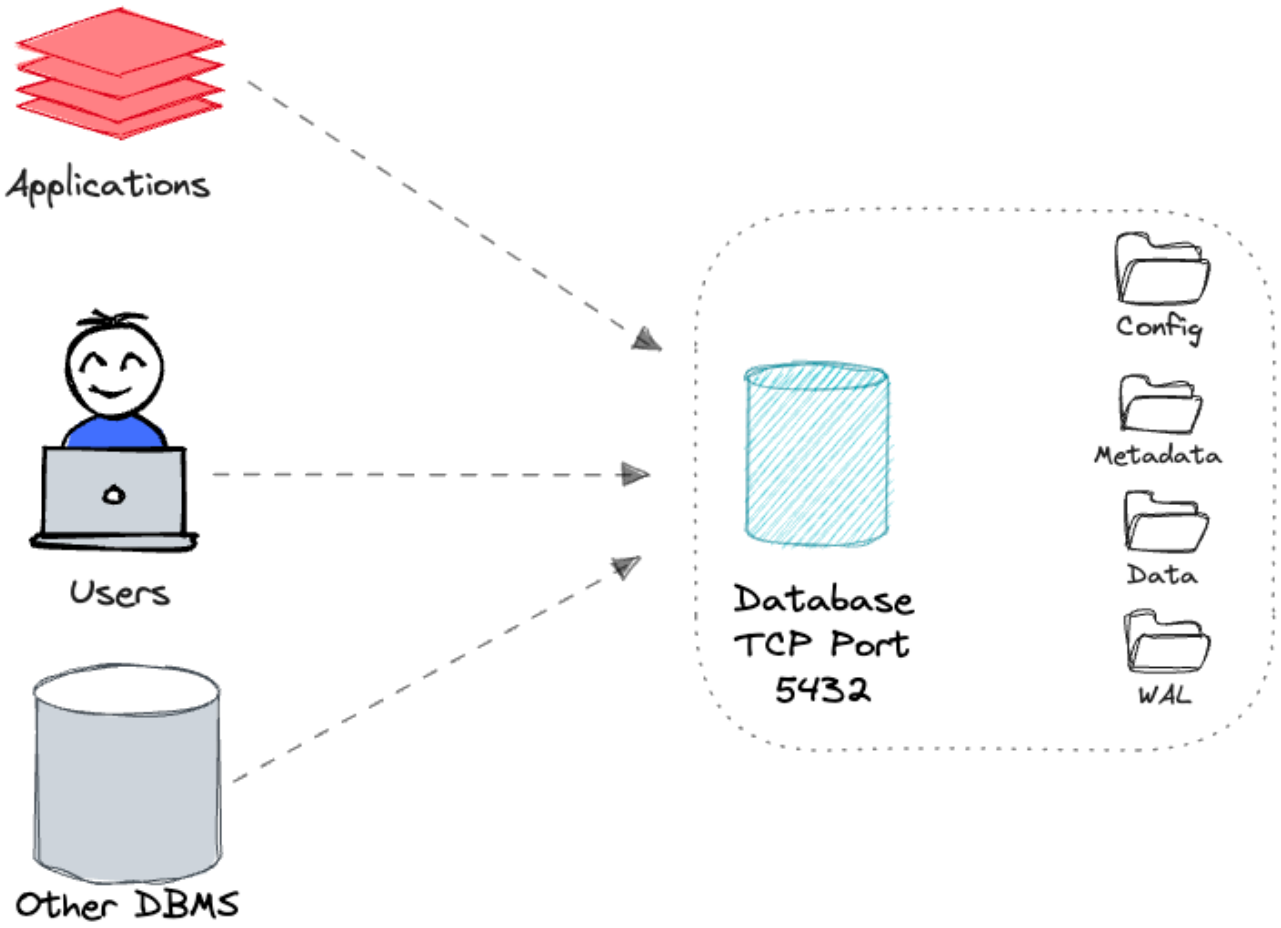
What is a Database?

```
# ss -tlnup | grep 5432
tcp   LISTEN 0      144          127.0.0.1:5432       0.0.0.0:*    users:(("postgres",pid=2260689,fd=5))

$ ls -la /var/lib/postgresql/16/main

total 88
drwx------ 19 postgres postgres 4096 Oct 11 18:32 .
drwxr-xr-x  3 postgres postgres 4096 Sep 18 11:52 ..
drwx------  5 postgres postgres 4096 Sep 18 11:52 base
drwx------  2 postgres postgres 4096 Oct 11 18:01 global
drwx------  2 postgres postgres 4096 Sep 18 11:52 pg_commit_ts
drwx------  2 postgres postgres 4096 Sep 18 11:52 pg_dynshmem
drwx------  4 postgres postgres 4096 Oct 11 18:32 pg_logical
drwx------  4 postgres postgres 4096 Sep 18 11:52 pg_multixact
drwx------  2 postgres postgres 4096 Sep 18 11:52 pg_notify
drwx------  2 postgres postgres 4096 Sep 18 11:52 pg_replslot
drwx------  2 postgres postgres 4096 Sep 18 11:52 pg_serial
drwx------  2 postgres postgres 4096 Sep 18 11:52 pg_snapshots
drwx------  2 postgres postgres 4096 Oct 11 18:32 pg_stat
drwx------  2 postgres postgres 4096 Sep 18 11:52 pg_stat_tmp
drwx------  2 postgres postgres 4096 Sep 18 11:52 pg_subtrans
drwx------  2 postgres postgres 4096 Sep 18 11:52 pg_tblspc
drwx------  2 postgres postgres 4096 Sep 18 11:52 pg_twophase
-rw-------  1 postgres postgres    3 Sep 18 11:52 PG_VERSION
drwx------  3 postgres postgres 4096 Sep 18 11:52 pg_wal
drwx------  2 postgres postgres 4096 Sep 18 11:52 pg_xact
-rw-------  1 postgres postgres   88 Sep 18 11:52 postgresql.auto.conf
-rw-------  1 postgres postgres  130 Oct 11 18:01 postmaster.opts
```

Database Security

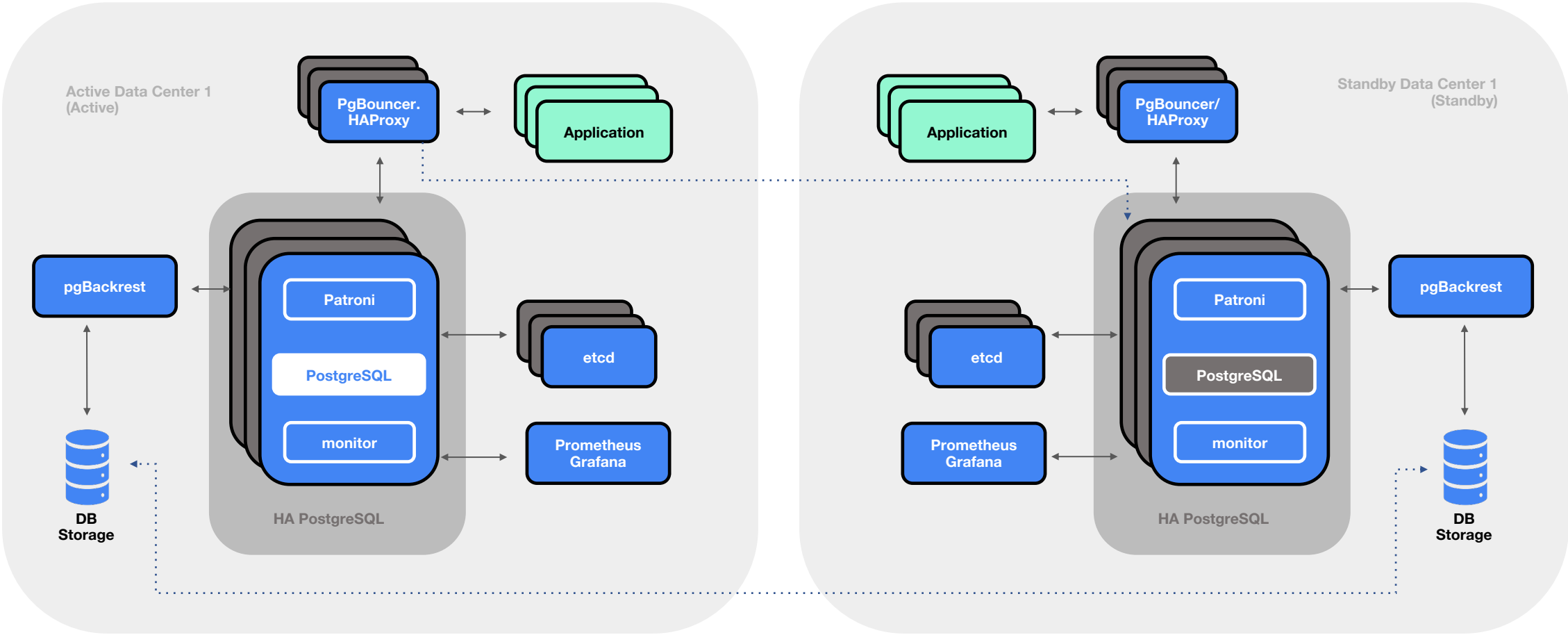
- Who can access the database?
- Who can authenticate?
- What data can they access?
- What data can they change?
- Who did what when? (auditing)

Attack Vectors – simple view

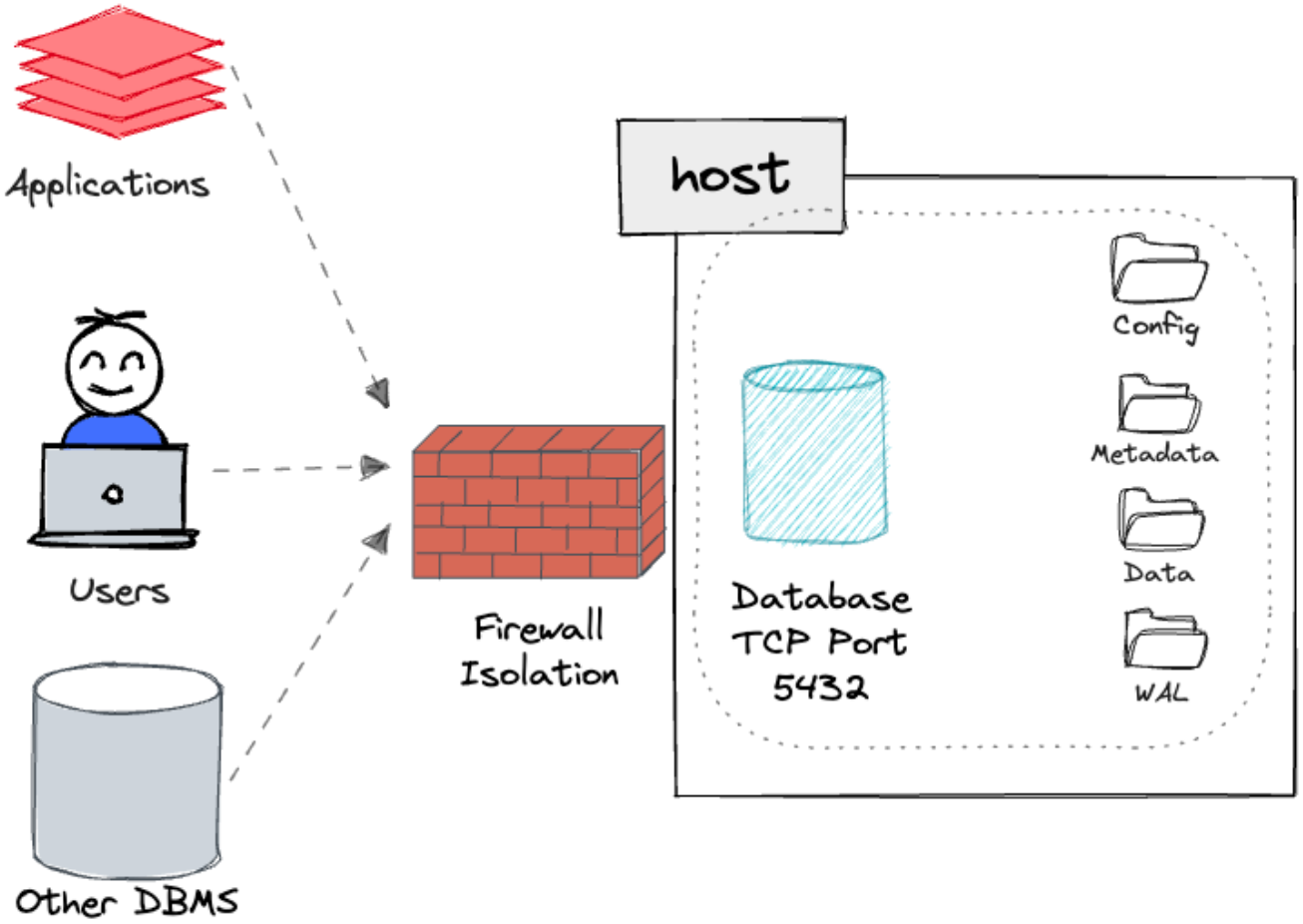


- Who can access the database?
- Who can authenticate?
- What data can they access?
- What data can they change?
- Who did what when? (auditing)
- How can we recover?

Attack Vectors – Multi Data Center with HA/DR



External Factors



- Pre-authentication
- Physical security
- Network security
- Sideway attacks
  - Backups
  - Applications
- Virtualization attacks

Who can Authenticate to the DB?

- Principle of Least Privilege
  - IP ranges
  - Users/Roles
- Users and Roles in PostgreSQL
  - A user is a role with login privilege
  - Users and Groups are the same
- Use different users
  - Very low-level user with minimal privileges
  - Strong user to do more, like create/drop tables
  - Superuser should be avoided (just like root)
- Always use SSL

Who can Authenticate to the DB?

- **Host-based authentication (pg_hba.conf)**

- **Which users can connect…**
- **…to which databases**
- **…connecting from which IP ranges (or locally)**
- **…with (non)encrypted connections**
- **…using which authentication methods**

Who can Authenticate to the DB?

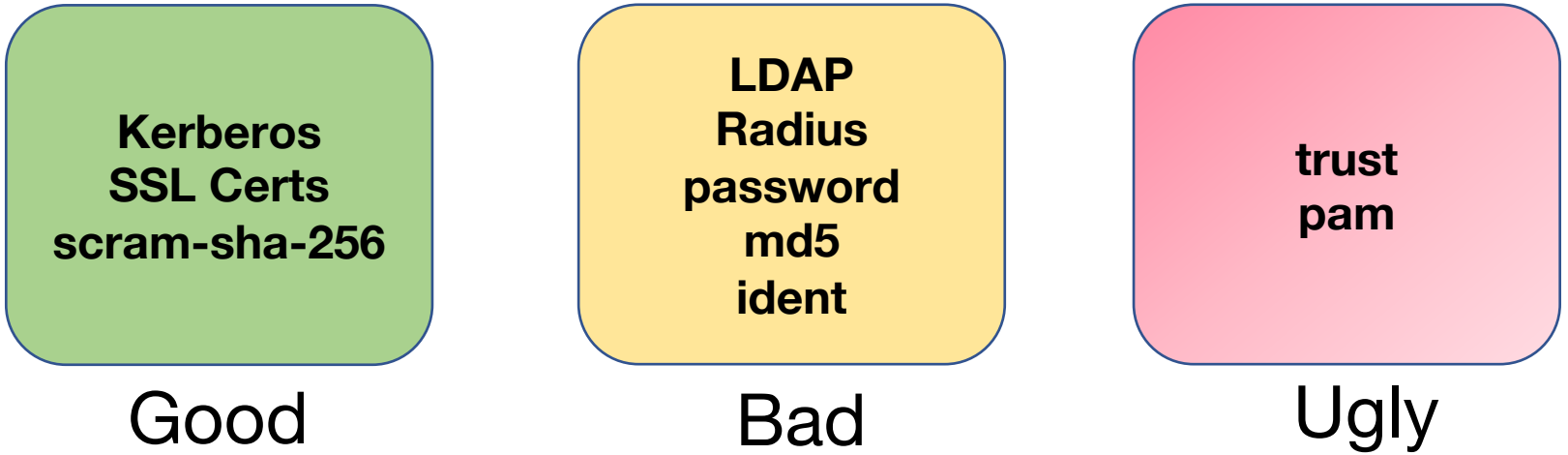
- pg_hba.conf
- Reloaded via SIGHUP or “SELECT pg_reload_conf()”

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
# Database administrative login by Unix domain socket
local   all             postgres                                peer
# "local" is for Unix domain socket connections only
local   all             all                                     peer

# IPv4 / IPV6 local connections
host    app1            user_foo        127.0.0.1/32            scram-sha-256
host    app2            +group_bar      ::1/128                 scram-sha-256

# SSL connections with regex matching databases and users (PG 16)
hostssl "/^db\d{1,4}$" "/^user\d{1,4}$" 10.10.2.0/24            scram-sha-256
```

Who can Authenticate to the DB?

- **Many companies deploy some sort of directory**
  - **Active Directory**
  - **FreeIPA**

- **Common mistake is to use LDAP authentication instead of Kerberos**



Who can Authenticate to the DB?

- **scram-sha-256 is a *big* improvement over md5**
  - **does not reveal the user's cleartext password to the server**
  - **is designed to prevent replay attacks**

- **SCRAM's Channel Binding is a feature that allows for mutual authentication (client/server) to prevent MITM attacks**

- **For extra security, require SCRAM channel binding**

- `channel_binding=require` **in the connection string**
- `PGCHANNELBINDING=require` **environment variable**

https://www.citusdata.com/blog/2020/07/28/securely-authenticate-with-scram-in-postgres-13/

What can they see or do?

```
CREATE ROLE name [ [ WITH ] option [ ... ] ]

where option can be:
 SUPERUSER | NOSUPERUSER                  # Avoid superusers
 | CREATEDB | NOCREATEDB                  # Can it create databases?
 | CREATEROLE | NOCREATEROLE              # Can it create other roles?
 | INHERIT | NOINHERIT                    # PG 16: GRANT WITH INHERIT true
 | LOGIN | NOLOGIN                        # Can it login? (user)
 | REPLICATION | NOREPLICATION            # Role will have replication privs
 | BYPASSRLS | NOBYPASSRLS                # Bypass or not Row Level Security
 | CONNECTION LIMIT connlimit              # Default is no limit (-1)
 | [ ENCRYPTED ] PASSWORD 'password' | PASSWORD NULL
 | VALID UNTIL 'timestamp'                 # Cut off access after this timestamp
 | IN ROLE role_name [, ...]               # Added as member of existing roles
 | IN GROUP role_name [, ...]              # Same as "IN ROLE"
 | ROLE role_name [, ...]                  # Add existing roles into this new one
 | ADMIN role_name [, ...]                 # Add existing roles into this, being able
                                           # add new roles into it (WITH ADMIN)
```

What can they see or do?

Several built-in roles

```
\du pg*
                    List of roles
          Role name             |  Attributes
--------------------------------+--------------
 pg_checkpoint                  | Cannot login
 pg_create_subscription         | Cannot login
 pg_database_owner              | Cannot login
 pg_execute_server_program      | Cannot login
 pg_monitor                     | Cannot login
 pg_read_all_data               | Cannot login
 pg_read_all_settings           | Cannot login
 pg_read_all_stats              | Cannot login
 pg_read_server_files           | Cannot login
 pg_signal_backend              | Cannot login
 pg_stat_scan_tables            | Cannot login
 pg_use_reserved_connections    | Cannot login
 pg_write_all_data              | Cannot login
 pg_write_server_files          | Cannot login
```

What can they see or do?

- Create ROLEs with appropriate permissions to database objects
- Databases, schemas, tables, views, functions, sequences
- `GRANT` those roles to specific users
- `GRANTs` can be column based
- Remember Principle of Least Privilege

What can they see or do?

```
GRANT SELECT ON TABLE table_name TO ro_role;
GRANT INSERT, UPDATE, DELETE ON TABLE table_name TO rw_role;
GRANT EXECUTE ON FUNCTION function_name(argument_types) TO role;
GRANT SELECT ON ALL TABLES IN SCHEMA schema_name TO role;
GRANT USAGE ON SCHEMA schema_name TO role;
GRANT CONNECT ON DATABASE database_name TO role;
GRANT TEMP ON DATABASE database_name TO role;
GRANT USAGE, SELECT ON SEQUENCE sequence_name TO role;
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA schema_name TO role;
GRANT role TO another_role;
```

GRANT can be column based

```
GRANT SELECT (column_name), UPDATE (column_name) ON TABLE
table_name TO role;
```

Security Definer function/stored proc

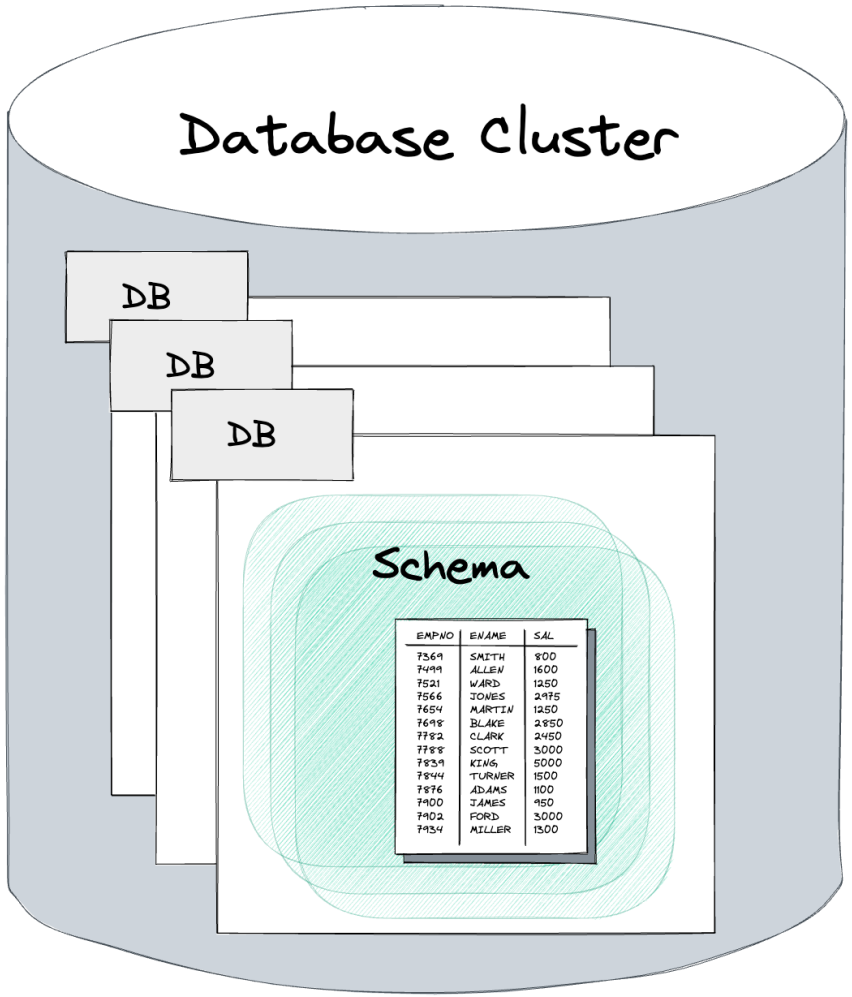
- Resist the temptation to make a role a superuser
- If you need a complicated set of permissions, create a stored procedure with SECURITY DEFINER privilege
- Be careful to avoid allowing a security hole (see docs)

Security Definer function/stored proc

```
-- As user with elevated privileges

CREATE OR REPLACE FUNCTION insert_sensitive_data(data text) RETURNS void AS
$$
BEGIN
    -- Check if the current user has the required privileges
    IF current_user = 'trusted_user' THEN
        INSERT INTO sensitive_data (data_value) VALUES (data);
    ELSE
        RAISE EXCEPTION 'Permission denied.';
    END IF;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

-- As regular user: Grant execute privilege to a trusted user

GRANT EXECUTE ON FUNCTION insert_sensitive_data(text) TO trusted_user;
```

What can they see or do?

Understand the chain of objects and their permissions



Cluster

↓

Database

↓

Schema

↓

Relation/Table

Row Level Security

- In Postgres since 2014
  - MySQL doesn’t have it. Views only
- Fine-grained Access Control
- Restrict, on a per-user basis
  - Which rows are visible to normal queries (per user)
  - What can be inserted, updated, deleted
- Policies control visibility
- Used alongside GRANT/REVOKE

Row Level Security

```
SET ROLE postgres;

CREATE POLICY security_clearance ON hero
  USING (pg_has_role(current_user, clearance, 'MEMBER'));

ALTER TABLE hero ENABLE ROW LEVEL SECURITY;-- Everyone but superusers+owner
ALTER TABLE hero FORCE ROW LEVEL SECURITY; -- Everyone but superusers
```

```
                    Table "public.hero"
         Column | Type | Collation | Nullable | Default
----------------+------+-----------+----------+---------
 name           | text |           |          |
 secret_identity | text |           |          |
 powers         | text |           |          |
 weaknesses     | text |           |          |
 clearance      | text |           |          |
Policies (forced row security enabled):
   POLICY "security_clearance"
     USING (pg_has_role(CURRENT_USER,
            (clearance)::name, 'MEMBER'::text))
```

Mandatory Access Control

- sepgsql
  - SELinux bindings
- RBAC type enforcement covers most DB objects
- Can be combined with custom SELinux policy for powerful control

What about data on disk?

- Transparent Data Encryption is still a WIP in Postgres, available in closed products
- Pgcrypto – extension to encrypt data in the database
  - PGP, OpenSSL; hashing and encryption
  - Not very practical
  - Do not save the key next to the data itself!
- PostgreSQL Anonymizer
  - Extension to mask or anonymize PII in the database
- Outside of PostgreSQL
  - Operating System
  - Storage device / cloud storage

What about the superuser?

- Bypass all DAC (Discretionary Access Control)
- Bypass all Row Level Security
- Can load any library
- Can run `COPY...PROGRAM` (execute arbritrary shell cmds)
- Can run `ALTER SYSTEM` (change conf settings with SQL)
- Create/execute any function
- Others

set_user extension

- `GRANT EXECUTE` on `set_user()` and/or `set_user_u()` to otherwise unprivileged users
- Can switch the effective user when needed to perform specific actions
- Optional enhanced logging ensures an audit trail
- Once one or more unprivileged users able to run `set_user_u()`, ALTER superuser to NOLOGIN
- Multiplex unprivileged users, e.g. with connection pools

Who did what, and when?

- Database logs
  - Ship them somewhere else (or outsource)
  - Have a sensible logging policy (don't log connections)
- PostgreSQL parameters
  - log_connections
  - log_statement (ddl, mod, all) – beware in production
- Log Rotation/truncation/removal
- PGAudit extension
  - Detailed session and/or object logging
  - Who did exactly what, and when
  - Produces audit logs required for compliance

What Else Can Go Wrong?

- Keep server updated to the latest minor release
  - Only bug fixes. Don’t hesitate to upgrade.
- Extensions have their own upgrade/patching cycle
- Ecosystem software (pgbackrest, patroni, pgbouncer)
  - needs to match versions of Postgres they work with
  - have their own release cycle
- Do not delete WAL files

Don’t Forget Backups

- Database dumps are NOT backups
- Storage snapshots are NOT (reliable) backups
- Everybody needs Point In Time Recovery (PITR) – why?
- DB “Backups” are comprised of separate operations
  - Restore
  - Recovery – replaying of transaction logs up to a consistent state
- If you’re not exercising *documented* restores, you can’t assume your backups are valid

Don’t Forget Backups

- Use specialized software like pgBackRest
  - Backup encryption
  - multi-repository, delta restores, file bundling, block incremental

- Last words heard in the operations room:

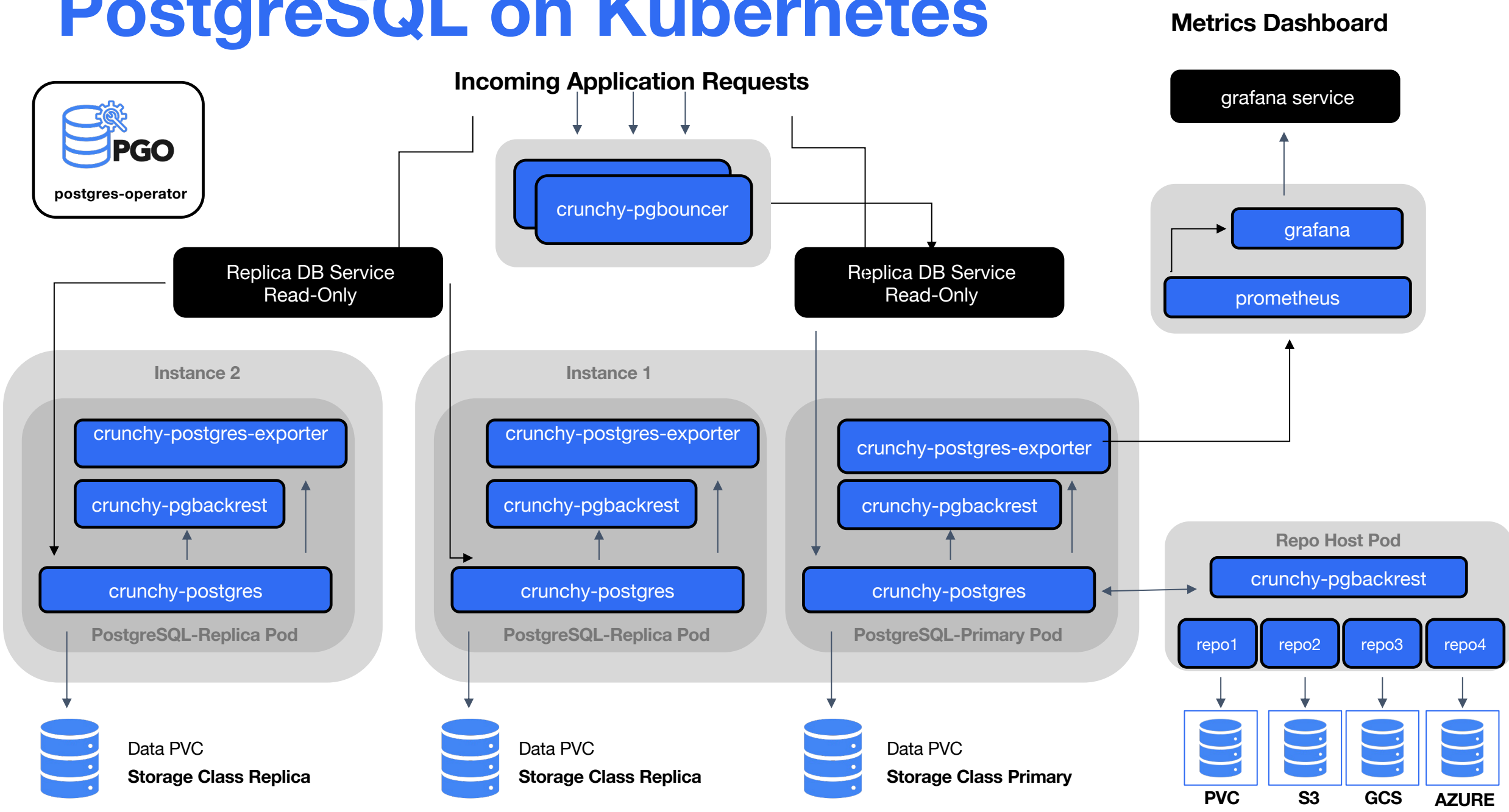
  “But I thought *you*
  did the backups!”

Should I run PostgreSQL in a container?

You already are running PostgreSQL in a container

```
# ps -fae | grep postgres
postgres  248985       1   0 Feb22 ?        00:00:02 /usr/pgsql-13/bin/postgres -D
/app/pgdata/hippo.13
...

# ll /proc/248985/ns
total 0
lrwxrwxrwx 1 postgres postgres 0 Feb 24 12:59 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 postgres postgres 0 Feb 24 12:59 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 postgres postgres 0 Feb 24 12:59 mnt -> 'mnt:[4026531840]'
lrwxrwxrwx 1 postgres postgres 0 Feb 24 12:59 net -> 'net:[4026531992]'
lrwxrwxrwx 1 postgres postgres 0 Feb 24 12:59 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 postgres postgres 0 Feb 24 13:11 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 postgres postgres 0 Feb 24 13:11 time -> 'time:[4026531834]'
lrwxrwxrwx 1 postgres postgres 0 Feb 24 13:11 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 postgres postgres 0 Feb 24 12:59 user -> 'user:[4026531837]'
lrwxrwxrwx 1 postgres postgres 0 Feb 24 12:59 uts -> 'uts:[4026531838]'
```

PostgreSQL on Kubernetes



It is a container, what could go wrong?

Managing compute resources

Overprovisioning
Control Plane
No Monitoring / Capacity Planning
Proper Storage

Container resource limits

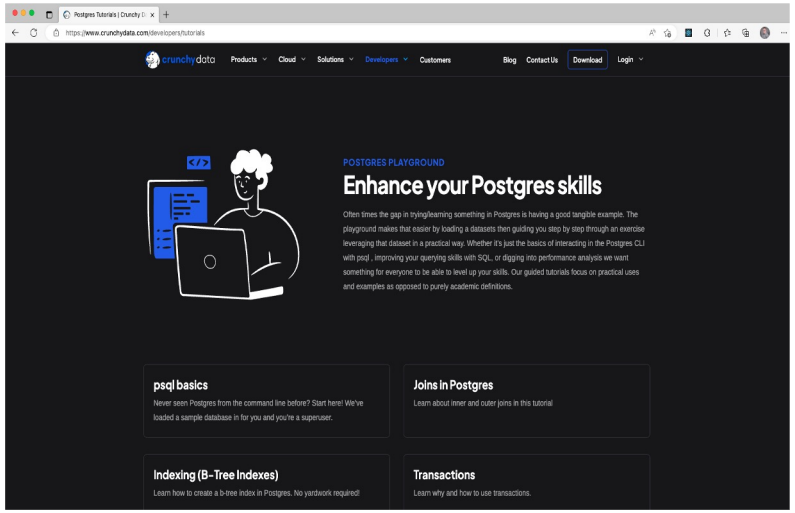
Minimal CPU*:
<number of active concurrent sessions> / 2.5

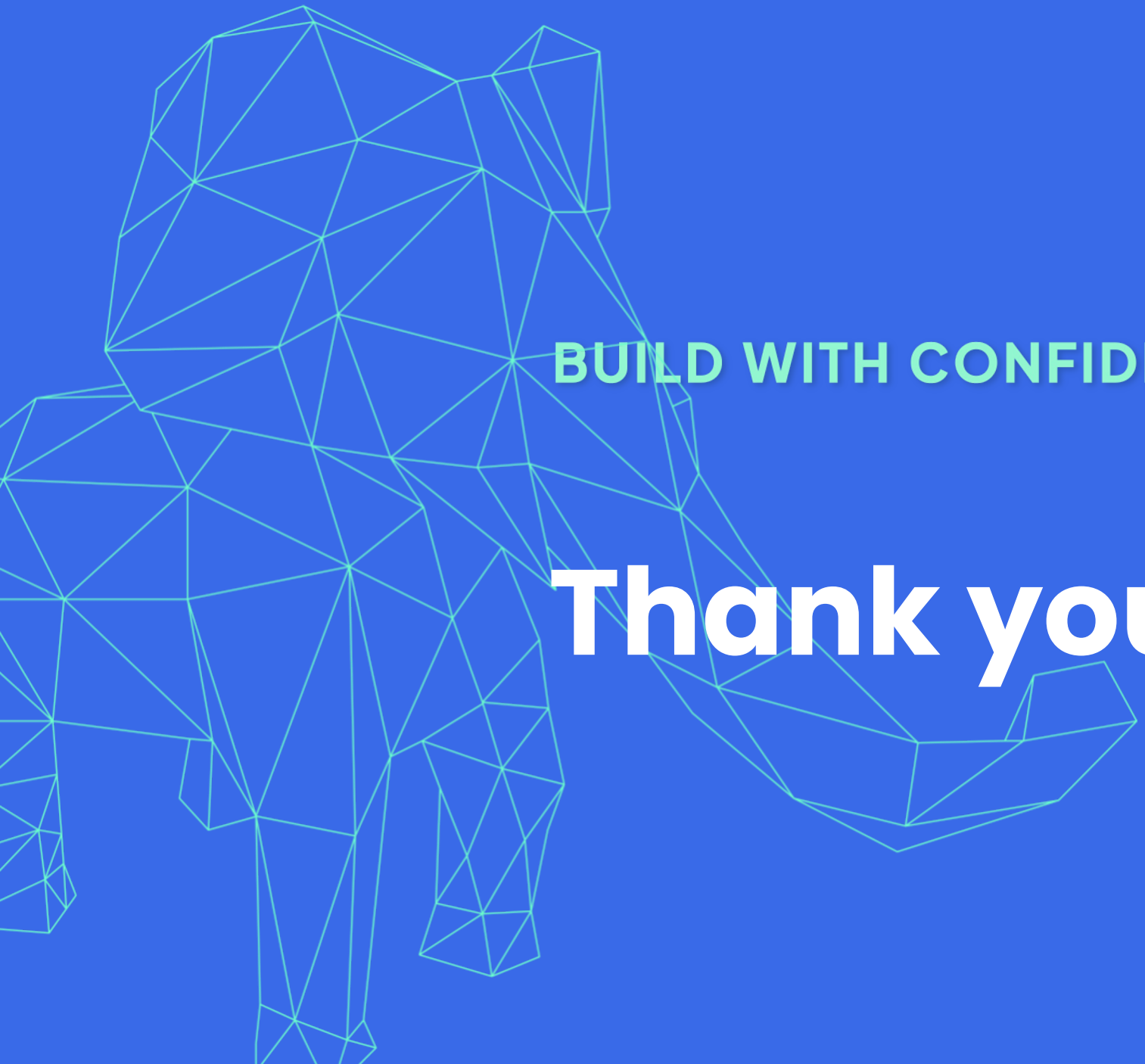
Minimal Memory*:
(work_mem * <avg active concurrent sessions> * 2) +
(maintenance_work_mem * autovacuum_max_workers)
+ shared_buffers + (max_connections * 20MB) < 70%

Immature Kubernetes Process/Procedures

Acknowledgements and Resources

- **Greg Sabino Mullane**
- **Joe Conway**
- **Crunchy Data**
- **All of the amazing PostgreSQL community**
- **Run and Learn PostgreSQL in your browser** https://www.crunchydata.com/developers/tutorials
- **Learn PostgreSQL** https://www.crunchydata.com/developers/
- PGPedia: https://pgpedia.info/





BUILD WITH CONFIDENCE

Thank you

crunchy data