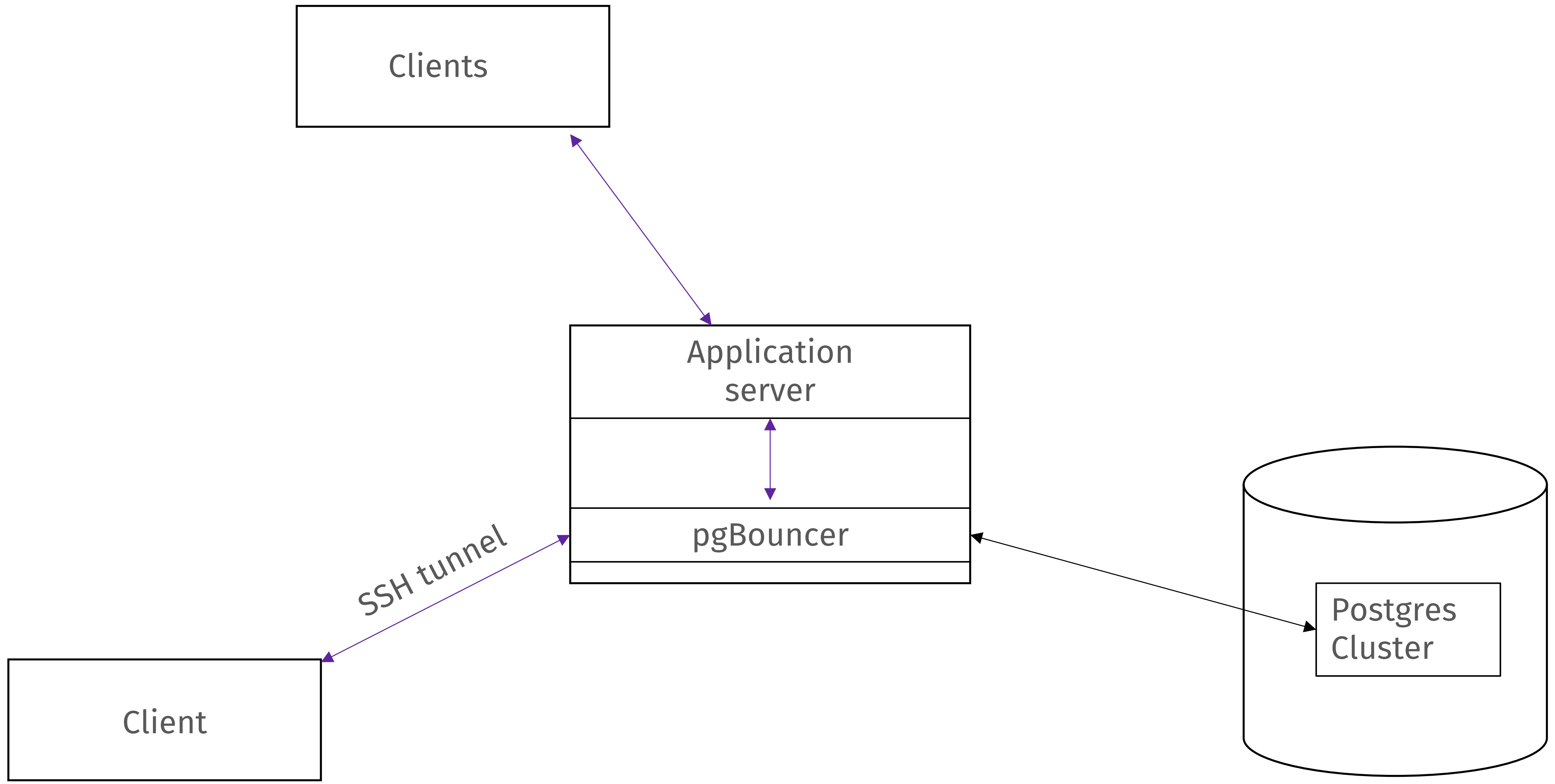
Today. Tomorrow. Together

AGENDA

Physical setup
The connection service file
The password file
Remote Access to PostgreSQL
Using pg_hba.conf for Remote Access management
Don't use superuser access
Don't use PUBLIC in databases
Default privileges
Row-level security (RLS)
Auditing
Regular Patching
Alter system V17
Maintain V17

PHYSICAL SETUP



THE PASSWORD FILE

The file .pgpass in a user's home directory can contain passwords to be used if the connection requires a password (and no password has been specified otherwise).

This file should contain lines of the following format:

```
hostname:port:database:username:password
hostname:port:*:username:password
*:*:*:username:password
```

chmod 0600 ~/.pgpass

THE CONNECTION SERVICE FILE

The connection service file ~/.pg_service.conf. is an “INI file” format file and an example file is provided in the PostgreSQL installation at share/pg_service.conf.sample.

```
# comment
[p16]
host=127.0.0.1
port=5416
user=pgr

[p17]
host=127.0.0.1
port=5417
user=pgr
```

```
psql service=p16
```

REMOTE ACCESS TO POSTGRESQL

To restrict access to unix socket, modify the postgresql.conf file and set listen_addresses to `''` then only unix socket access is possible.

To restrict access to localhost only, modify the postgresql.conf file and set listen_addresses to localhost.

If only few users need access, then SSH tunneling could be used.

If localhost not possible then set listen_addresses to an ip address and not to **'*'**

The port that Postgres listens on "should" be set to something different than the default **5432**

USING pg_hba.conf FOR REMOTE ACCESS MANAGEMENT

The pg_hba.conf is evaluated from the top to the bottom, line by line and when the first match is found the rest of the file is ignored.

So, the general format of pg_hba.conf is like this:

```
# TYPE DATABASE USER ADDRESS METHOD

TYPE = local, host, hostssl, hostgssenc, ...

DATABASE = database name, all, @file-with-databases, ...

USER = username, all, @file-with-users, ...

ADDRESS = IP 4/6 med subnet

METHOD = trust, reject, scram-sha-256, md5, password, …
```

USING 'pg_hba.conf' FOR REMOTE ACCESS MANGESMANT

```
# Allow all user to connect to any database via Unix domain socket without password
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
  local   all           all                                     peer

# Allow any user from any host with IP address 192.168.93.x to connect to database
# "dev" with scram-sha-256 password.
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
  host  dev             all             192.168.93.0/24         scram-sha-256

# Reject any user from any host with IP address 192.168.94.x to connect
# to database "postgres
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
  host  postgres        all             192.168.94.0/24         reject

# Reject any user from any IP address to connect to database
# TYPE  DATABASE        USER            ADDRESS                 METHOD
  host  all             all             0.0.0.0/0               reject
```

DEFAULT PASSWORD

password_encryption = scram-sha-256 # (scram-sha-256 , md5)

The default is scram-sha-256 from Postgres version 15.

Note that older clients might lack support for the SCRAM authentication mechanism, and hence not work with passwords encrypted with SCRAM-SHA-256. See Section 21.5 for more details.

Add in to postgresql.conf

DON'T USE SUPERUSER ACCESS

It is always good if the number of people who have administrative access to a security critical system is as limited as possible. How far you want to go here depends on your security needs:

- ✓ Use personalized superuser accounts for the administrators, so that you can quickly revoke access to an administrator when needed.
- ✓ Use superusers only where really required. For example, you don’t need a superuser for replication connections, or to take a backup.
- ✓ Restrict access with superuser accounts in pg_hba.conf. Ideally, only local connections are allowed. If you don’t want that, restrict access to the personal system of the administrator.

PASSWORD'S ?

Use password expiration :

```
CREATE ROLE "peter.gram@itm8.dk"
WITH LOGIN PASSWORD '*' VALID UNTIL '2024-12-31';
```

Postgres will not warn or inform the user that the password is expiring but a nice DBA will send a mail to the users that time to password change is up ☺

DON'T USE PUBLIC BEFORE VERSION 15

In PostgreSQL 14 and in prior versions, **by default anybody can create a table**. The new table will simply end up in the PUBLIC schema.

Recommendations concerning schema PUBLIC (PRIOR to PG v15):

1. Remove PUBLIC from your database permissions
2. Remove PUBLIC from your public-schema permissions

DON'T USE PUBLIC

DON'T USE PUBLIC AFTER VERSION 15

In PostgreSQL 15, a fundamental change took place which is relevant to every user who happens to work with permissions: The default permissions of the public schema have been modified. This is relevant because it might hurt you during application deployment. You need to be aware of how it may affect you.

In version 15+ , **only the database owner can create objects in the public schema, or a user granted all schema PUBLIC**.

DON'T USE PUBLIC

itm8®

DEFAULT PRIVILEGES

Default privileges GRANT's or REVOKE's rights to objects on create. The drawback is the the default privilege must be created before the object and it is not applied to exiting objects.

```
ALTER DEFAULT PRIVILEGES IN SCHEMA my_schema GRANT INSERT ON TABLES
TO my_dev_role;
```

ROW-LEVEL SECURITY (RLS)

The essence of the row-level security method is setting privileges to table rows. It relates to both the existing rows and the new rows that result from the INSERT, UPDATE, or DELETE commands.

```
ALTER TABLE my_table ENABLE ROW LEVEL SECURITY;

CREATE POLICY select_all_rows
ON my_table FOR SELECT
USING (true);

CREATE POLICY update_own_rows
ON my_table FOR UPDATE
USING (current_user = owner)
WITH CHECK (current_user = owner);
```

P.S: To create or update this RLS policy, you have to be the table owner.

AUDITING

The standard PostgreSQL does not provide many options for auditing.

We can log this type of data in the logfile :

```
# Log successful and unsuccessful connection attempts.
log_connections = on

# Log terminated sessions.
log_disconnections = on

# Log all executed SQL statements (none (off), ddl, mod, and all).
log_statement = all
```

Note : There is a Postgres extension pgAudit (https://github.com/pgaudit)

REGULAR PATCHING

Minor releases never change the internal storage format and are always compatible with earlier and later minor releases of the same major version number. For example, version 15.6 is compatible with version 15.0 and/or version 15.1.

To update between compatible versions, you simply replace the executables while the server is down and restart the server.

The data directory remains unchanged — **minor upgrades are that simple and fast !**

ALTER SYSTEM FROM V17

The **allow_alter_system** (boolean) parameter controls if you can use the ALTER SYSTEM command

Note that this setting must not be regarded as a security feature. It only disables the ALTER SYSTEM command. It does not prevent a superuser from changing the configuration using other SQL commands. A superuser has many ways of executing shell commands at the operating system level, and can therefore modify postgresql.auto.conf regardless of the value of this setting.

This parameter only controls the use of ALTER SYSTEM. The settings stored in postgresql.auto.conf take effect even if allow_alter_system is set to off.

MAINTAIN FROM V17

The permission can be granted on a per-table basis using the **maintain** privilege and on a per-role basis via the **pg_maintain** predefined role.

Permitted operations are :

VACUUM, ANALYZE, REINDEX, REFRESH MATERIALIZED VIEW, CLUSTER, and LOCK TABLE.

```
GRANT MAINTAIN ON <table> TO peter;
```

TRANSACTION_TIMEOUT V17

The transaction_timeout parameter terminate any session that spans longer than the specified amount of time in a transaction.

The limit applies both to explicit transactions (started with BEGIN) and to an implicitly started transaction corresponding to a single statement. If this value is specified without units, it is taken as milliseconds. A value of zero (the default) disables the timeout.

If transaction_timeout is shorter or equal
to idle_in_transaction_session_timeout or statement_timeout then the longer timeout is ignored.

Setting transaction_timeout in postgresql.conf is not recommended because it would affect all sessions !

Let’s build today’s
and tomorrow’s IT.
Together?

Peter.Gram@itm8.dk

Tlf. +45 5374 7107

itm8