Current setup at EuXFEL

- One primary node and one standby with asynchronous streaming replication.
- Postgresql with timescale extension
- Main reasons for choosing postgresql/timescale
    - open source
    - many data types
    - different purposes of the database (archive, event archive, metadata)
    - scalability
    - stablity
    - self-hosted system required
    - SQL
    - High ingestion performance (intensive tests)

Database design

- Data and metadata model is working so far, the database is operable and can store most DOOCS data types.
- Currently working un GUI tools and refinement of metadata

High Availability (HA)

- Replication as such is fairly easy to set up in postgresql.
    - Different models supported. We use asynchronous physical replication via slots.
- Additional HA Features are not parts of postgresql:
    - monitoring of database nodes
    - automatic failover from primary to standby in case of an error
    - automatic failback (failed master comes back up, needs to rewind and synchronize and added as new standby.
    - STONITH if failback is not working (disable previous primary)
    - IP switching
    - connection pooling
    - load balancing (read access can be done by replica)
- We tested different HA solution to manage failover/failback situations for postgresql.
    - pgpool (fairly complete but unreliable), pg_auto_failover (simple but not yet ready), repmgr (simple but limited)
    - repmgr + pgpool chosen for one replicated node
        - Possibly not the right solution for bigger clusters
    - Maybe switch to patroni or pgedge later on

Backup

- Made experiments with pg_backrest
- Rather slow for aprrox. 100TB of data
- Overall concept not yet clear
- Additional server or disk space needed
- pg_backrest may be the first solution to start with, possibly compare with barman

Scalability

- Timescale stopped multinote support

- Concept for horizontal scalability needed
- Concept for vertical scalability needed
- Not so relevant for EuXFEL, but certainly for PETRA IV.

ETL Tool

- How do DOOCS data get into the database?
- DOOCS RPC much too slow.
- Different ideas have been developed
- The new Asynchronous DOOCS / ZMQ seems promising
- ETL tool subscribes DOOCS channels and upload them chunk-wise
- Specifications of what to extract is currently being moved from JSON files to the database.
- Things to work on:
    - Filtering
    - Timestamps
    - Specification when channels are updated / initialized at start / updated at regular intervals.
    - Error handling (how to handle connection error, invalid data)
- Drawback:
    - Pub/Sub is CPU intensive
    - Special servers needed as data hubs