Setting up a PGPool II Cluster

Oliver Freyermuth[1]
Michael Hübner[1]
Peter Wienemann[1]

[1]Physikalisches Institut, Universität Bonn

HEPiX 2021

UNIVERSITÄT BONN

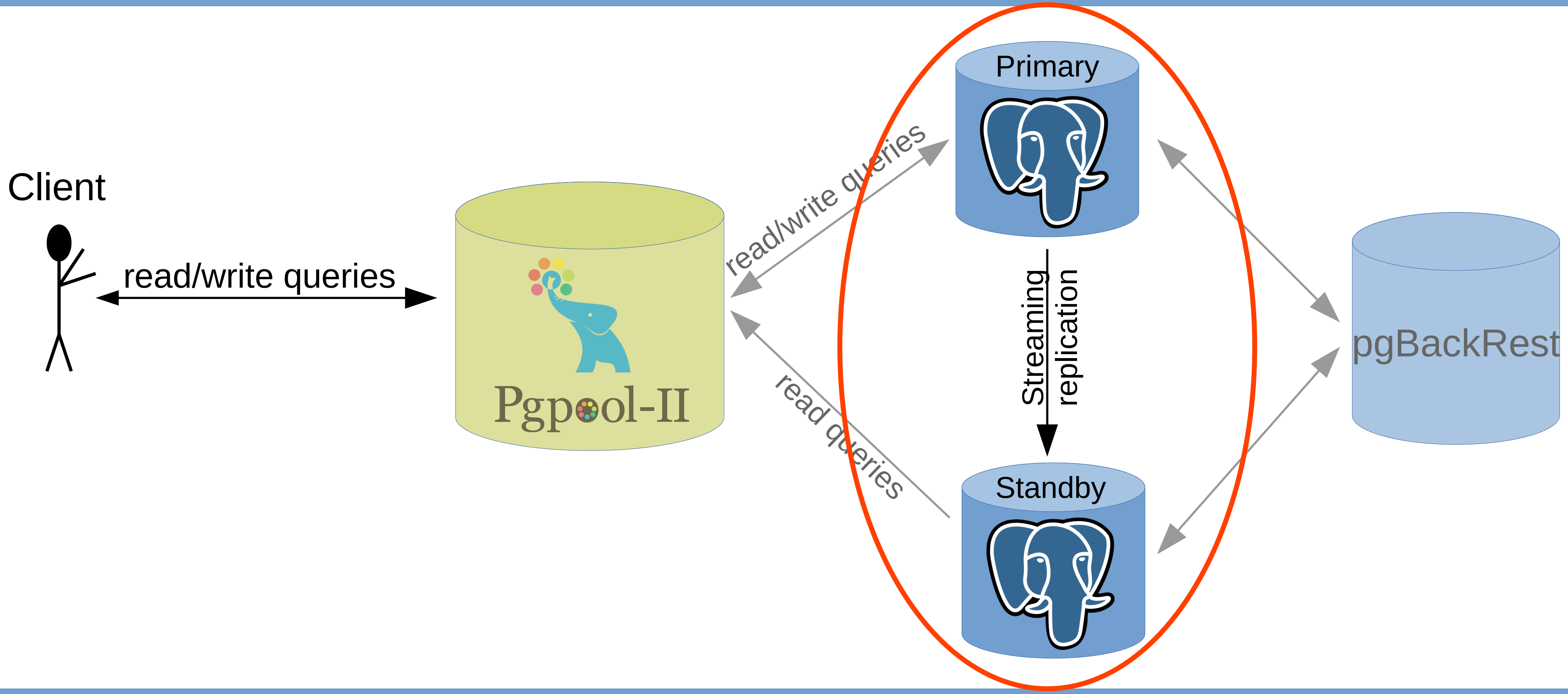
Our Use-Case

- Monitoring of institute machines (office, server room, printers, environment,...) using Zabbix
  - PostgreSQL database
  - Data kept for several years
    - Different granularity for different time scales (more finely for recent data, only trends for older data)

Requirements of the Setup

- Continuous monitoring
  - Highly available such that database is always reachable
  - Redundancy in case of backend failure
  - Minimal downtimes (e.g. during planned maintenances, failure of systems,...)
- Easily recoverable
  - Backups

The Setup - Backends



PostgreSQL Backends

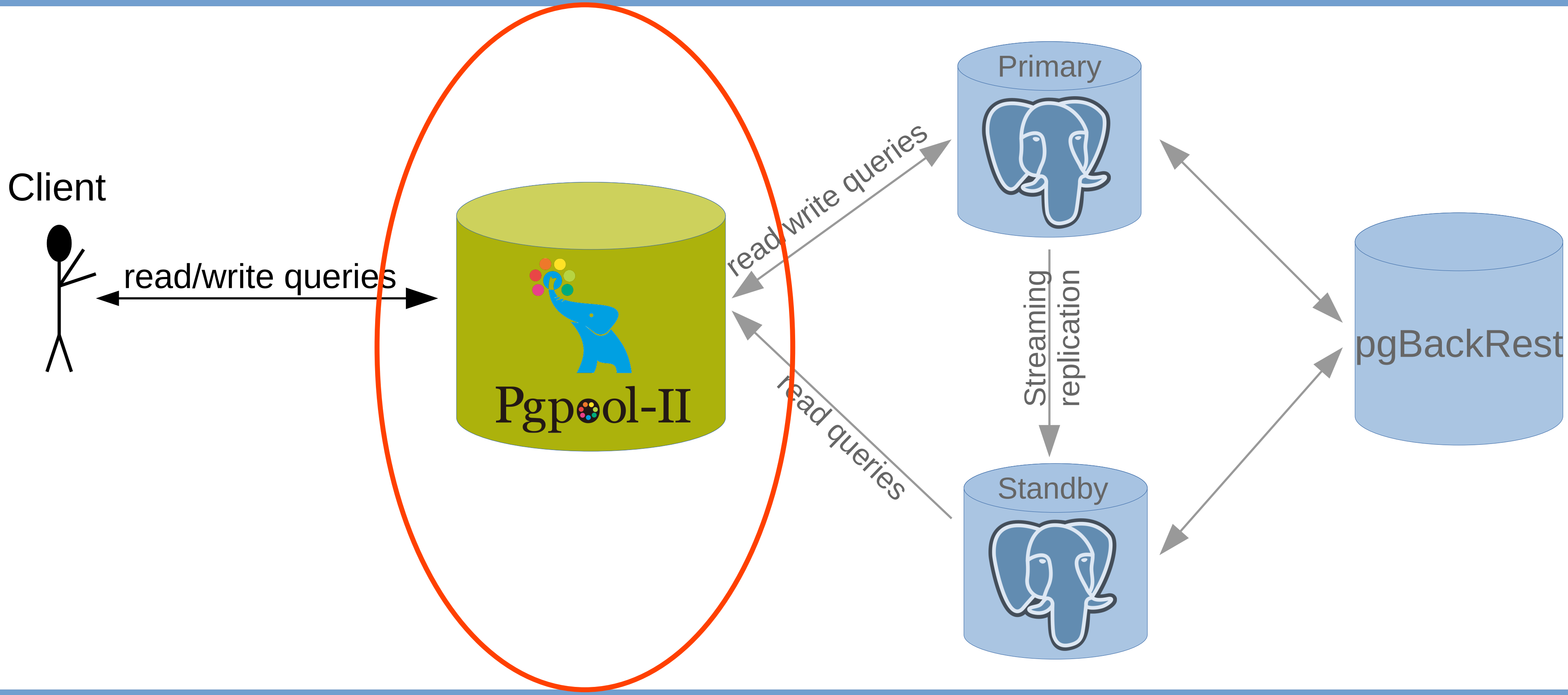
- Open Source database solution (BSD license)
  - Object-relational database management system
  - Long history of active development (30+ years)
  - Extensions
  - ACID (atomicity, consistency, isolation, durability) compliant
- Most important features for our (server) setup
  - Streaming replication: continuous WAL archiving
  - Authentification methods (pg_hba.conf)
    - SSL encrypted connection using password authentication (MD5 hashes) between backends and for communication with PGPool II-Frontend

PostgreSQL Backends

UNIVERSITÄT BONN

- Puppet module: https://github.com/puppetlabs/puppetlabs-postgresql
  - Automatic installation
  - Handling of configs, firewall, etc.
  - Explicitly no automatic start of the daemon!
- Does not provide
  - Handling of failover
  - Load balancing

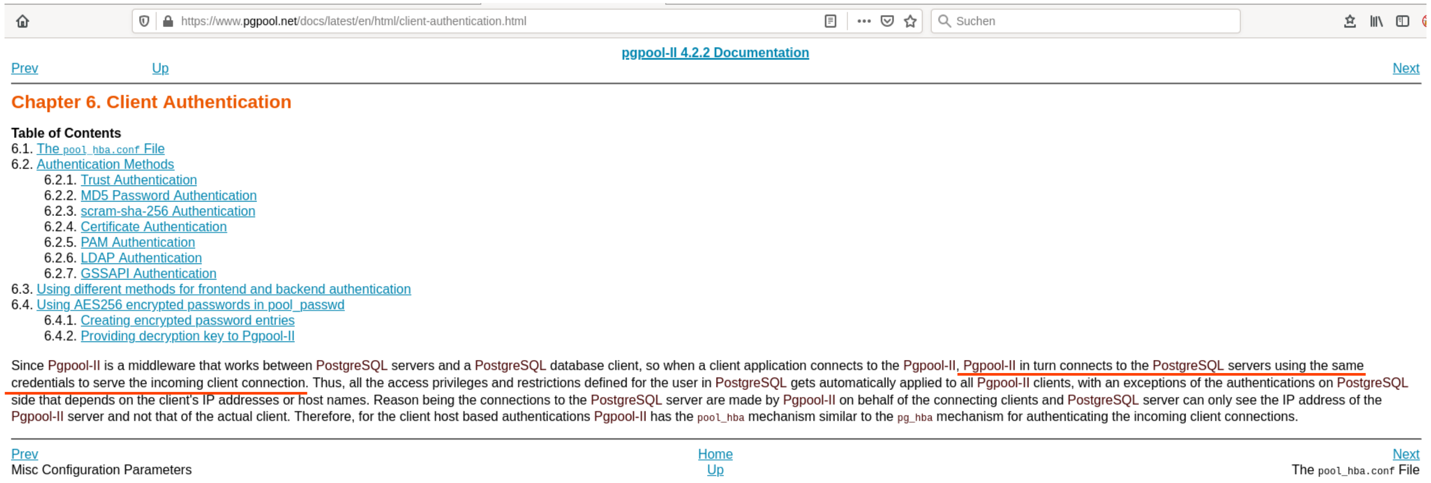
The Setup - Frontend



PGPool II Frontend

- Open Source middleware between clients and PostgreSQL servers (BSD license)
- What does it provide for our setup?
  - Authentification (pgpool.hba): Client ↔ PGpool II ↔ PostgreSQL backends
    - Password authentication + SSL encrypted connection
    - Same credentials are passed on to backends
  - Watchdog: steady health check of system
  - Automatic failover routines
    - Automatic triggers
    - Executes scripts one has to provide
  - Load balancing (only for read queries)
- Puppet: https://github.com/mwhahaha/puppet-pgpool
  - Somewhat actively maintained
  - Our PR was merged pretty fast (https://github.com/mwhahaha/augeas-pgpool/pull/2)
    - PGPool sometimes changes naming schemes and allowed values, so it does need active maintenance

PGPool II Frontend - Authentication



https://www.pgpool.net/docs/latest/en/html/client-authentication.html

pgpool-II 4.2.2 Documentation

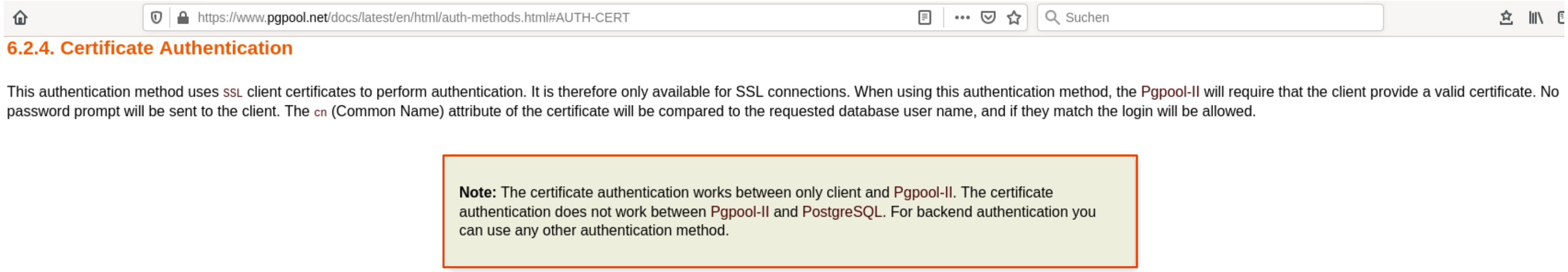
Prev Up Next

Chapter 6. Client Authentication

Table of Contents

Since Pgpool-II is a middleware that works between PostgreSQL servers and a PostgreSQL database client, so when a client application connects to the Pgpool-II, Pgpool-II in turn connects to the PostgreSQL servers using the same credentials to serve the incoming client connection. Thus, all the access privileges and restrictions defined for the user in PostgreSQL gets automatically applied to all Pgpool-II clients, with an exceptions of the authentications on PostgreSQL side that depends on the client's IP addresses or host names. Reason being the connections to the PostgreSQL server are made by Pgpool-II on behalf of the connecting clients and PostgreSQL server can only see the IP address of the Pgpool-II server and not that of the actual client. Therefore, for the client host based authentications Pgpool-II has the pool_hba mechanism similar to the pg_hba mechanism for authenticating the incoming client connections.

Prev | Home | Next
Misc Configuration Parameters | Up | The pool_hba.conf File



https://www.pgpool.net/docs/latest/en/html/auth-methods.html#AUTH-CERT

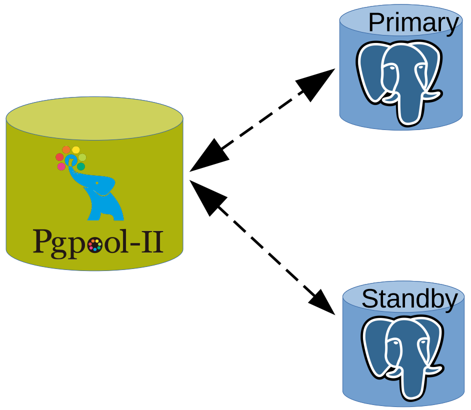
6.2.4. Certificate Authentication

This authentication method uses SSL client certificates to perform authentication. It is therefore only available for SSL connections. When using this authentication method, the Pgpool-II will require that the client provide a valid certificate. No password prompt will be sent to the client. The cn (Common Name) attribute of the certificate will be compared to the requested database user name, and if they match the login will be allowed.

Note: The certificate authentication works between only client and Pgpool-II. The certificate authentication does not work between Pgpool-II and PostgreSQL. For backend authentication you can use any other authentication method.

Health Checking

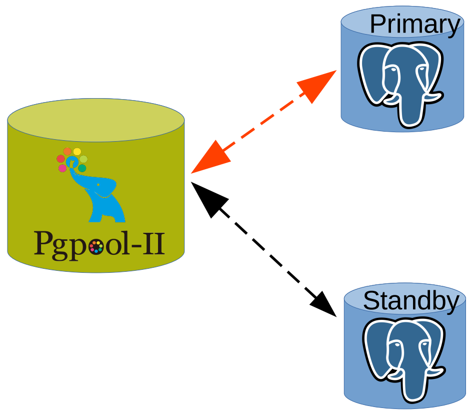
- Dedicated health check user
  - Steered by PGPool frontend
  - Connects to PostgreSQL backends (template table in postgres DB)
- Set up on PostgreSQL backends
  - Create health check user with corresponding role
  - Allow connection in pg_hba.conf
- Set up on PGPool frontend
  - Configure health check user and password
  - Important parameters
    - health_check_period: interval between checks in seconds
    - health_check_timeout: timeout in seconds
    - health_check_max_retries: maximum number of retries before failover
    - health_check_retry_delay: interval between failed checks in seconds



Health Checking

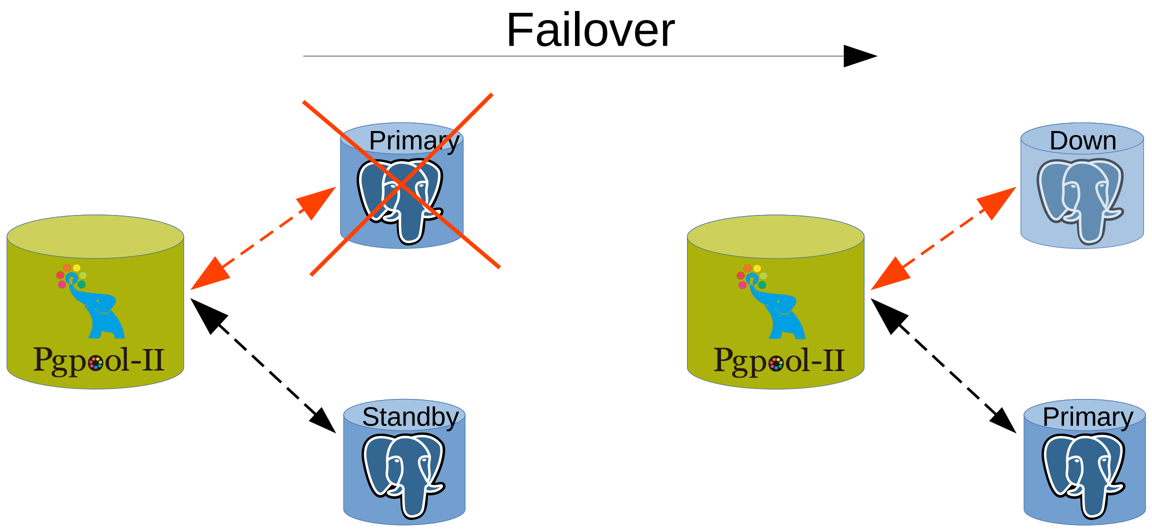
UNIVERSITÄT BONN

- Important to choose parameters carefully
- Observations during commissioning
  - Sometimes PGPool marks backends as ‚down‘ even if they are healthy
  - Explanation: DNS lookup took too long → with given health check parameters, health check failed → node unhealthy (marked as down)
- Solution
  - Local DNS caching
  - Increase grace period before health check triggers (8*(20+5) = 200 s)
    - health_check_period = 5
    - health_check_timeout = 20
    - health_check_max_retries = 8
    - health_check_retry_delay = 5



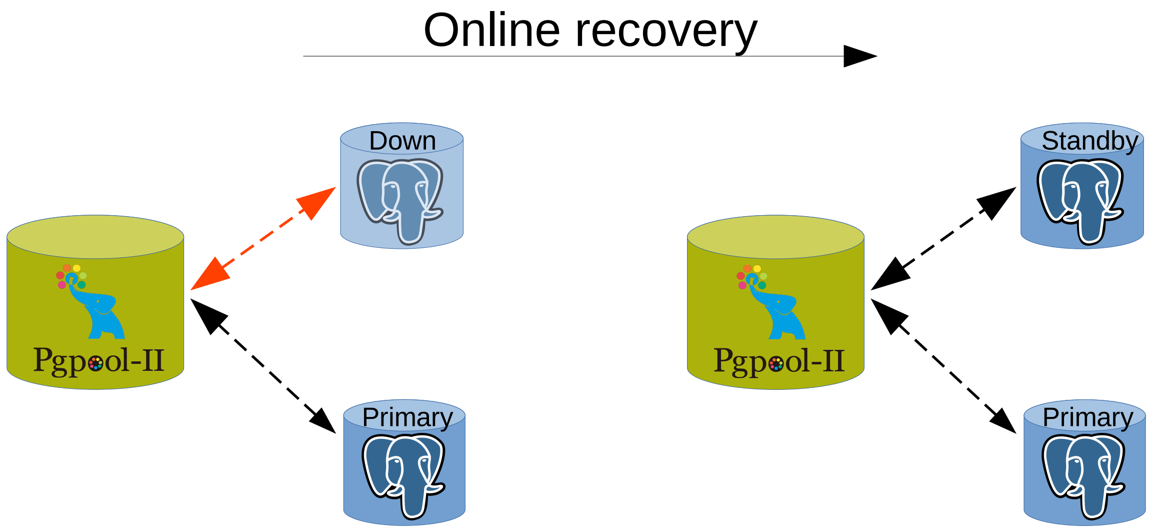
PGPool II Frontend - Scripts

- Failover (on Frontend)
  - Connect to new master
  - Promote PostgreSQL instance to primary
  - Create replication slot
- Online recovery (on Frontend)
  - Connect to node that needs recovery
  - Stop PostgreSQL instance
  - Recover from primary using pg_rewind
  - Create recovery.conf
    - PgBackRest information
    - Set node to Standby
  - Start PostgreSQL instance

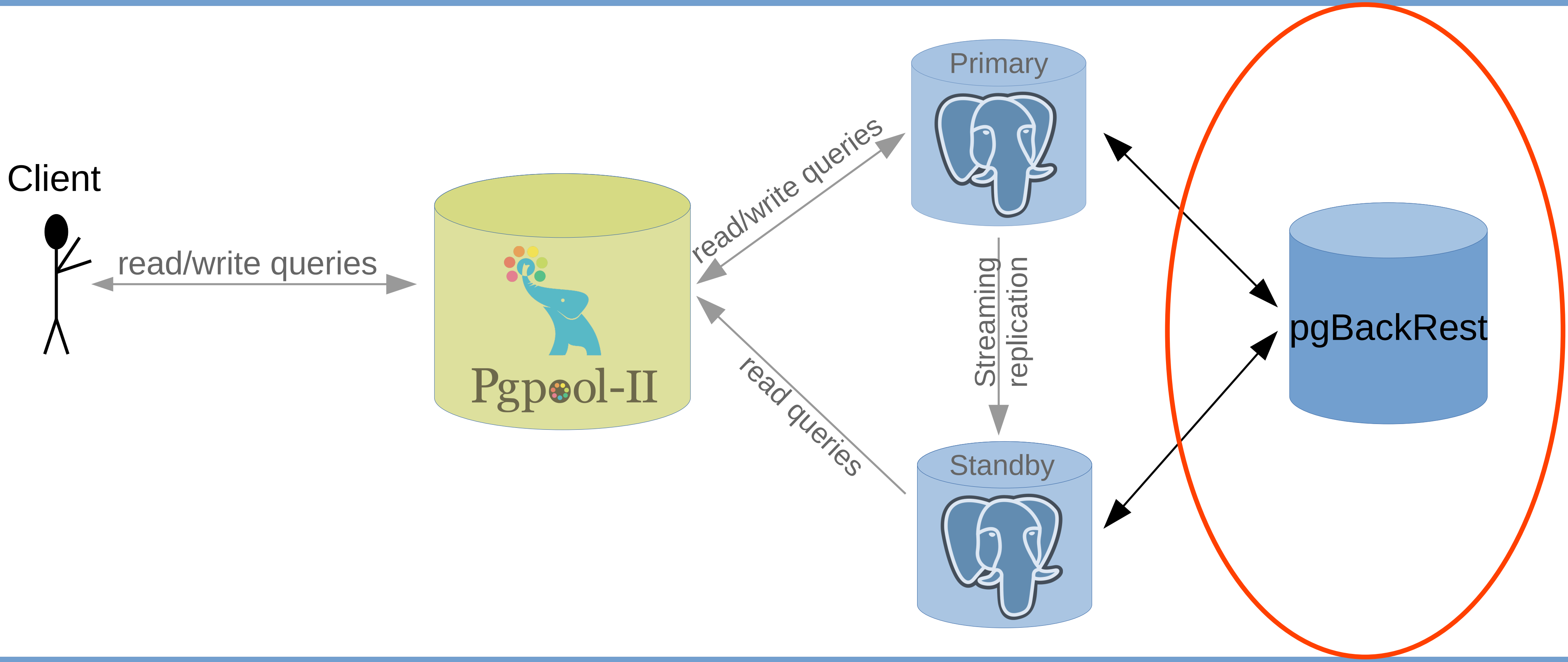


PGPool II Frontend - Scripts

- Failover (on Frontend)
  - Connect to new master
  - Promote PostgreSQL instance to primary
  - Create replication slot
- Online recovery (on Frontend)
  - Connect to node that needs recovery
  - Stop PostgreSQL instance
  - Recover from primary using pg_rewind
  - Create recovery.conf
    - PgBackRest information
    - Set node to Standby
  - Start PostgreSQL instance



The Setup - Backup

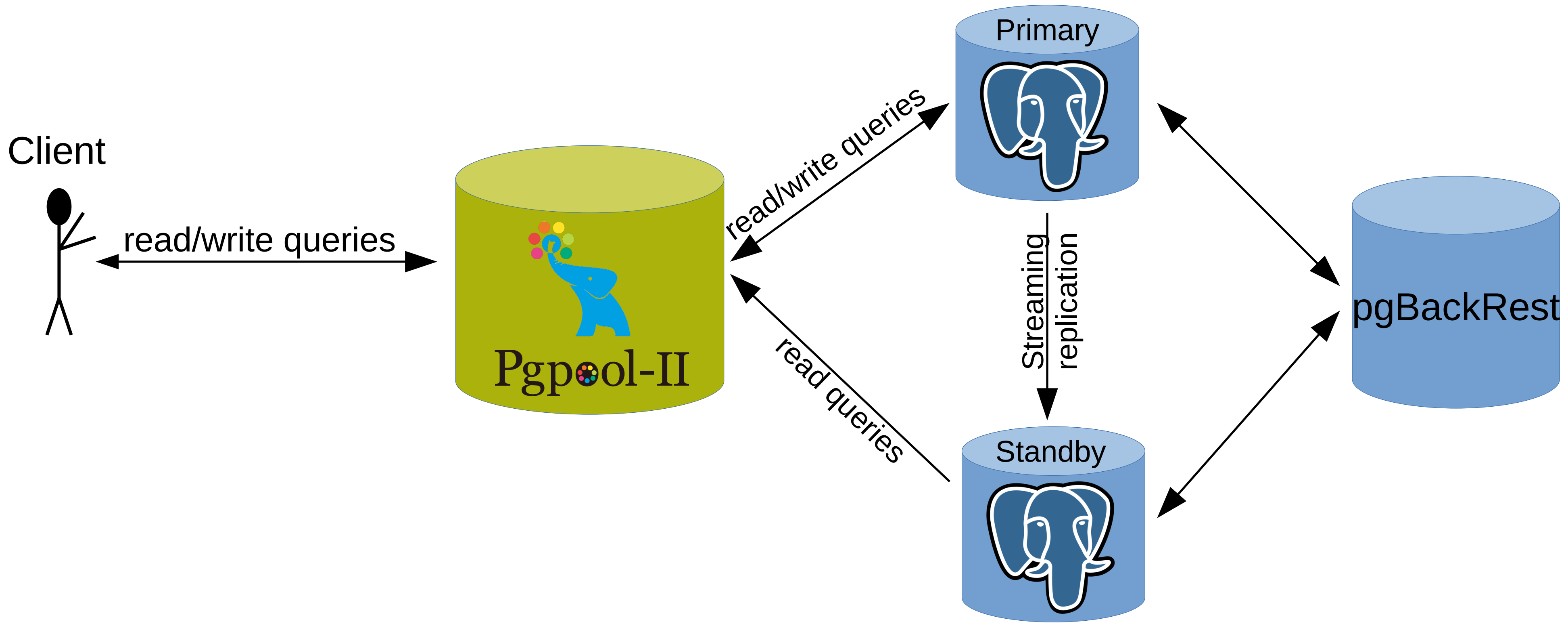


pgBackRest

- Open source backup solution for PostgreSQL (MIT license)
- What does it provide?
  - Full, differential, incremental backups
  - Retention options
  - Compressed and encrypted
  - S3 backend backup
- Setup
  - Backup run per cronjob on frontend
    - 1x full backup per week
    - 6x incremental backups per week
  - Retention
    - 12 full backups (3 months)
    - 32 incremental backups (includes full backups in count → 4 weeks)

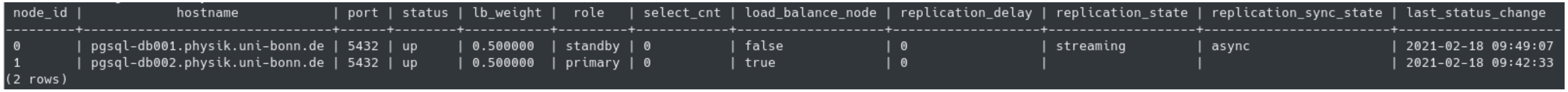
```
if (PGPool is not running):
    exit
change to user „postgres“
if (all backends up):
    if (cannot ping standby):
        send alert mail
        exit
    ssh to standby
    run backup
    if (error during backup):
        send alert mail
        exit
else:
    log error
    exit
```

The Full Setup



General Functionality Tests

- Cluster status



```
 node_id |             hostname              | port | status | lb_weight |  role   | select_cnt | load_balance_node | replication_delay | replication_state | replication_sync_state | last_status_change
---------+-----------------------------------+------+--------+-----------+---------+------------+-------------------+-------------------+-------------------+------------------------+---------------------
 0       | pgsql-db001.physik.uni-bonn.de | 5432 | up     | 0.500000  | standby | 0          | false             | 0                 | streaming         | async                  | 2021-02-18 09:49:07
 1       | pgsql-db002.physik.uni-bonn.de | 5432 | up     | 0.500000  | primary | 0          | true              | 0                 |                   |                        | 2021-02-18 09:42:33
(2 rows)
```

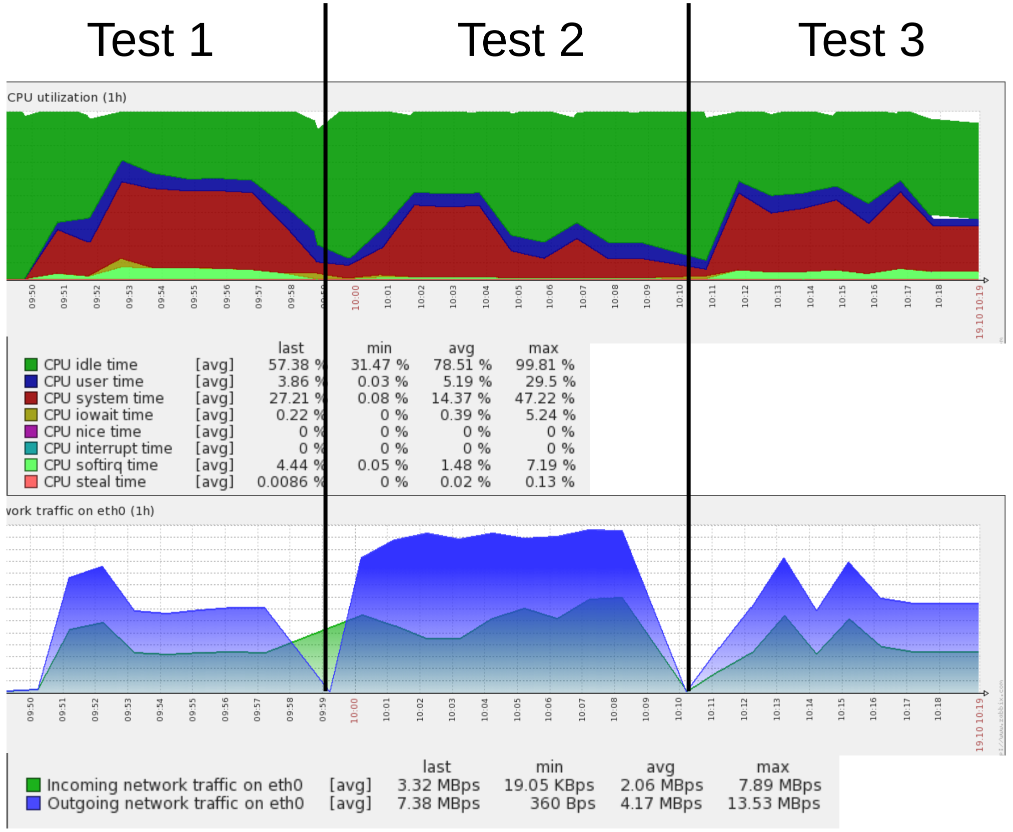
- Failover
- Currently testing upgrades
- Pitfalls encountered when
  - Attaching a node (never really need to do this by hand)
  - Promote a node to primary (never really need to do this by hand)

Resource Usage

- Idea: test performance of setup when importing Zabbix database
- Setup
  - Frontend installed on VM
  - Backends installed on baremetal machines
- Scenarios
  - Test 1: Frontend on AMD based hypervisor node in different building
  - Test 2: Frontend on local Intel based hypervisor node
  - Test 3: Frontend on local AMD based hypervisor node
- Questions to answer
  - Does latency matter (on our scale)?
  - What are limiting factors?

Resource Usage

- Scenarios
  - Test 1: Frontend on AMD based hypervisor node in different building
  - Test 2: Frontend on local Intel based hypervisor node
  - Test 3: Frontend on local AMD based hypervisor node
- What we can learn
  - Latency does not impact performance at our scale
  - Single core perfomance matters
    - Older AMD hypervisor **do not** offer hardware support for SSL algorithms → saturation on frontend
    - Newer Intel hypervisor **does** offer hardware support → saturation on backends



		last	min	avg	max
🟩 CPU idle time	[avg]	57.38 %	31.47 %	78.51 %	99.81 %
🟦 CPU user time	[avg]	3.86 %	0.03 %	5.19 %	29.5 %
🟥 CPU system time	[avg]	27.21 %	0.08 %	14.37 %	47.22 %
🟨 CPU iowait time	[avg]	0.22 %	0 %	0.39 %	5.24 %
🟪 CPU nice time	[avg]	0 %	0 %	0 %	0 %
🟦 CPU interrupt time	[avg]	0 %	0 %	0 %	0 %
🟩 CPU softirq time	[avg]	4.44 %	0.05 %	1.48 %	7.19 %
🟥 CPU steal time	[avg]	0.0086 %	0 %	0.02 %	0.13 %

		last	min	avg	max
🟩 Incoming network traffic on eth0	[avg]	3.32 MBps	19.05 KBps	2.06 MBps	7.89 MBps
🟦 Outgoing network traffic on eth0	[avg]	7.38 MBps	360 Bps	4.17 MBps	13.53 MBps

Disaster Drill

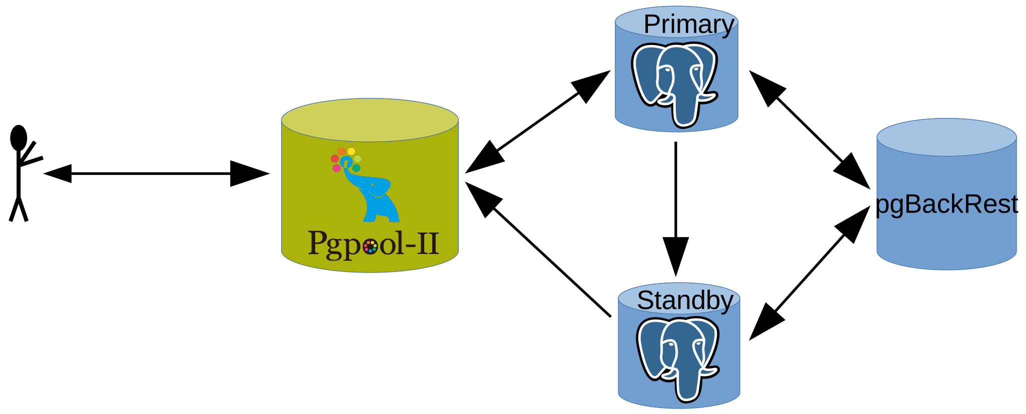
- Scenario: complete loss of both backends
- Strategy
  - Stop PGPool on frontend (if not broken as well)
  - Reinstall backend nodes using puppet
    - PostgreSQL instances will not be running
  - Restore database on one node using pgBackRest
    - Run puppet once to create directory structure first
  - Start frontend
    - Restored backend node will become primary
  - Recover second backend node using PGPool
    - If difficulties arise → manual recovery with pg_basebackup

Outlook: TimescaleDB

- Extension for PostgreSQL (TSL license)
  - Can use all community features for free unless hosted as a Database-as-a-Service
- Provides very effective compression for time-series databases („94% - 97% compression rates“)
  - Zabbix database is a perfect candidate
  - Monitoring is effectively time-series with one number per point in time
- Compression achieved via delta-delta encoding
  - Delta encoding: only store change w.r.t. previous data point
  - Delta-delta encoding: apply delta encoding on delta encoded data

Summary & Outlook

- Summary
  - Introduced PGPool cluster setup to be used for Zabbix monitoring
  - Highlighted some of the most important features and pitfalls
- Outlook
  - Test updates/upgrades of setup, e.g. PostgreSQL version upgrade 11 → 12
  - Activate TimescaleDB for Zabbix database



UNIVERSITÄT BONN

Backup