Design and Operation of a PostgreSQL High Availability Architecture

Issue/Rev. No.: 1.0
Date: 28/1/2026
Author: Massimo Costantini
Affiliation: INAF - Istituto Nazionale di Astrofisica
Osservatorio Astronomico di Trieste
Gruppo IA2
Approved by: Cristina Knapic

Abstract

A legacy set of PostgreSQL databases at INAF was previously distributed across heterogeneous machines running obsolete software, with backup as the only protection mechanism and no form of High Availability. This Technical Report describes the design and deployment of a consolidated PostgreSQL infrastructure based on physical replication and a homogeneous, up-to-date platform.

The proposed architecture is built on Rocky Linux 9.3 and PostgreSQL 16.1 and relies on a production master, multiple replicas, and a DNS-based indirection layer. All databases are replicated at cluster level, enabling consistent failover through native PostgreSQL mechanisms. The document describes the resulting system, its operational model and the procedures that ensure service continuity in a scientific production environment.

Contents

1 Introduction

Several production services at INAF rely on a heterogeneous set of *PostgreSQL* [1] databases that have been developed and maintained over many years. Prior to the work described in this report, this ecosystem consisted of a dozen independent databases deployed across multiple machines, running on different obsolete operating systems and *PostgreSQL* versions.

In this legacy configuration, the only form of protection against failures was periodic backup. No replication, failover mechanism or redundancy was in place. As a consequence, a hardware or software failure on any of the hosting machines implied service downtime, manual recovery procedures, and potentially long restoration times. This situation represented a significant operational risk for services that are increasingly critical for both internal workflows and external users.

The primary objective of this project was to consolidate and modernize this fragmented environment by:

- Upgrading the underlying operating systems and *PostgreSQL* instances to a current, supported platform (Rocky Linux 9.3 [2] and PostgreSQL 16.1)
- Migrating all databases into a unified and controlled infrastructure
- Introducing a *High Availability (HA)*[1] architecture based on physical replication
- Providing a clear and reproducible operational model for failover and disaster recovery

Rather than adopting a complex orchestration framework, the design favors a transparent and robust solution based on native *PostgreSQL* mechanisms and standard system tools. The resulting architecture is centered on a production *master* node, one or more *slave* nodes in physical replication, and a *DNS-based*[2] indirection layer that allows services to be redirected in case of failure.

This report describes the design, deployment and operation of the resulting *High Availability* architecture. It focuses on the physical replication model, the role of *DNS* in service continuity, the monitoring and backup strategy and the operational procedures required to promote a replica to production in case of failure. The document aims to serve both as technical documentation of the implemented system and as a reference model for similar deployments in scientific infrastructures.

---

[1] *High Availability (HA)* refers to the design of systems that ensure continuous service operation with minimal downtime, even in the presence of hardware or software failures, through redundancy, replication and controlled failover mechanisms.

[2] The *Domain Name System (DNS)* is a distributed naming service that translates human-readable hostnames into network addresses, enabling clients to locate services independently of their physical network location.

2 System Overview

The new infrastructure is based on a small set of homogeneous nodes running Rocky Linux 9.3 and PostgreSQL 16.1. Two physical servers form the core of the system, while a third virtual machine provides an additional replication target.

The architecture is composed of:

- One **master** node, hosting the production *PostgreSQL instance*
- One **slave** node on a physical server, configured as a physical replica in read-only mode
- One **slave** node on a virtual server, also configured as a physical replica
- A *DNS* alias (*CNAME*[3]) that identifies the production database endpoint independently of the underlying host

All client applications connect to the database through the *DNS* alias rather than addressing a specific machine. This indirection layer decouples services from the physical location of the *master* node and enables transparent redirection in case of failure.

The use of physical replication ensures that the entire *PostgreSQL cluster* is replicated, including all databases, roles and global objects. The replicas are therefore exact binary copies of the *master* at the storage level and can be promoted without logical reconfiguration.

The system provides two main operational advantages:

- If the *master* node becomes unavailable, one of the replicas can be promoted to production with a controlled and deterministic procedure
- Replica nodes can be used for read-only workloads, such as reporting or heavy analytical queries, without impacting production performance

[3] A *CNAME (Canonical Name)* record is a *DNS* entry that maps an alias hostname to another canonical hostname, allowing a service to be redirected transparently without changing client configuration.

3 Architecture Design

The architecture is deliberately simple and relies on native *PostgreSQL* features. Physical streaming replication is used to continuously ship *WAL*[4] records from the *master* to each replica.

The *master* node is the only instance that accepts write operations. Replica nodes run in hot-standby mode and continuously apply *WAL* segments, remaining in near-real-time synchronization with the *master*.

A *DNS CNAME* record represents the logical identity of the production database service. At any time, the alias points to the current *master* node. Client applications are unaware of the underlying topology and always connect to the same hostname.

Initially, the system consisted of:

- One physical *master* node
- One physical replica node

In a later phase, a second replica was added on a virtual machine, extending the topology to three nodes. This additional replica provides increased resilience and allows maintenance or testing activities without impacting the physical pair.

This design avoids external clustering frameworks and favors explicit operational control. Failover is not automatic: it is performed manually by an administrator following a documented procedure. This choice reflects the need for predictability and traceability in a scientific production environment.

[4] *Write-Ahead Logging (WAL)* is a *PostgreSQL* mechanism that records all changes to the database in a sequential log before they are applied, ensuring data durability and enabling crash recovery and physical replication.

4 Databases Scope

The *PostgreSQL cluster* hosts a set of fifteen databases, including system templates and application-specific instances:

- aao
- datab
- gms
- hosted
- lbt
- mwa
- svas
- tng
- vialactea
- vialacteadevel
- vospace
- postgres (system)
- template0 (template)
- template1 (template)
- contrib_regression

These databases support a variety of services and user communities, each with different access patterns and operational requirements.

Because physical replication operates at the cluster level, all databases are replicated as a single coherent unit. No per-database configuration is required: every object, table, role and configuration change performed on the *master* is automatically propagated to all replicas.

This approach guarantees strong consistency across the entire environment and eliminates the risk of partial or divergent states between databases.

5 Replication Monitoring

The correctness and timeliness of replication are primarily verified through *PostgreSQL* log files, located on each node under:

/mnt/db-storage/pgsql/16/data/log

On the *master*, logs record *WAL* generation and replication slots activity. On replica nodes, logs provide information about *WAL* reception, replay progress and any interruption in streaming.

Operational monitoring consists of:

- Verifying that replicas continuously receive and apply *WAL* segments
- Detecting delays or disconnections in streaming replication
- Identifying I/O or storage issues that may prevent replicas from keeping up with the *master*

In practice, log inspection allows administrators to determine whether the system is operating normally or whether a replica is lagging or disconnected. This is particularly important before performing a failover, as the selected replica must be sufficiently up-to-date to guarantee data integrity.

6 Backup Strategy

In addition to replication, a full backup system is implemented on the production node. Backups are stored under:

/mnt/db-storage/pgsql/16/backups

Backup operations are scheduled via crontab on the *master* node. These jobs include:

- Periodic *PostgreSQL* backups
- Additional scheduled tasks related to other services

Backups serve a different purpose from replication. While replication protects against node failure and provides continuity of service, backups protect against logical errors, accidental data loss or corruption propagated through replication.

It is therefore essential that backup jobs run only on the active production node. Running them simultaneously on multiple nodes would be inefficient and could produce inconsistent results.

For this reason, the activation of the crontab is tightly coupled with the role of the node: only the current *master* must execute scheduled jobs.

7 Failover Procedure

In case the production *master* becomes unavailable, one of the replicas must be promoted to take over the production role. This operation is manual and consists of four coordinated steps:

1. *DNS* update
   Modify the *DNS* configuration so that the *CNAME* alias points to the selected replica instead of the failed *master*

2. Disable scheduled jobs on the failed node
   On the failed (if accessible) or demoted machine, comment out the crontab entries to prevent backups and other periodic jobs from running

3. Enable scheduled jobs on the new *master*
   On the node that will become the new production *master*, uncomment the crontab so that backups and service-related tasks are activated

4. Promote the replica
   On the selected node, as the postgres user, execute:

   ```
   pg_ctl promote
   ```

This command terminates recovery mode and converts the replica into a writable *master*. After promotion, the node assumes the full production role. Client applications, redirected through *DNS*, transparently reconnect to the new *master* without configuration changes.

This procedure provides a deterministic and auditable failover path. Although not automatic, it ensures that each transition is explicitly controlled, minimizing the risk of *split-brain*[5] conditions or unintended role changes.

[5] *Split-brain* is a failure condition in distributed systems in which two or more nodes simultaneously believe they are the active *master*, leading to divergent states and potential data corruption due to concurrent, uncoordinated writes.

8 Operational Considerations and Limitations

The proposed architecture intentionally avoids fully automated failover mechanisms. While tools such as *Patroni*[6], *Pacemaker*[7] or external orchestration frameworks can provide automatic role switching, they also introduce additional layers of complexity, dependencies and opaque behavior.

In the context of a scientific infrastructure, where traceability and predictability are essential, the chosen design favors explicit human control over autonomous decision-making. Each role transition is performed deliberately by an administrator, following a documented procedure. This approach ensures that:

- The state of replication is verified before promotion
- *DNS* changes are synchronized with database role changes
- Scheduled tasks are activated only on the correct node
- The risk of *split-brain* scenarios is minimized

This design choice implies a number of operational constraints:

- Failover is not instantaneous and requires human intervention
- Availability during nights or holidays depends on the presence of on-call personnel
- The procedure must be well documented and periodically rehearsed

However, these limitations are balanced by a high degree of transparency. The system state is always observable through standard *PostgreSQL* tools and log files. There are no hidden state machines or background agents that may trigger unexpected transitions.

The addition of a second replica on a virtual server further improves resilience. It enables maintenance on physical hosts, provides an additional recovery path, and allows testing of promotion procedures without impacting production. The architecture can be extended with additional replicas without structural changes.

Future enhancements may include:

- Automated health checks and alerting on replication status
- Periodic verification of replica lag

---

[6] *Patroni* is an open-source cluster management framework for *PostgreSQL* that automates *High Availability* by coordinating leader election, failover and configuration using a distributed consensus backend.

[7] *Pacemaker* is a *High Availability* cluster resource manager that monitors services and system components and automatically orchestrates failover between nodes in case of failures.

- Semi-automated failover scripts that assist administrators while preserving manual control

These improvements can be layered on top of the existing design without altering its core principles.

9 Conclusion and Future Work

This project transformed a fragmented and legacy *PostgreSQL* environment into a unified, modern and resilient infrastructure. A set of heterogeneous and obsolete database instances, previously protected only by backups, has been consolidated into a coherent *High Availability* architecture based on physical replication.

The resulting system provides:

- A homogeneous and up-to-date platform (Rocky Linux 9.3, PostgreSQL 16.1)
- Full-cluster physical replication across multiple nodes
- A *DNS-based* indirection layer that decouples services from physical hosts
- A clear and deterministic failover procedure
- Integrated backup and operational workflows

The architecture achieves a balance between robustness and simplicity. By relying on native *PostgreSQL* mechanisms and standard system tools, it avoids unnecessary complexity while delivering effective protection against node failures. The explicit operational model ensures that every role transition is controlled, observable and auditable.

Future work will focus on incremental automation and monitoring. Possible developments include:

- Integration with monitoring systems to provide real-time visibility on replication health
- Alerting mechanisms for lag or disconnection of replicas
- Assisted failover tools that guide administrators through the promotion procedure
- Periodic disaster recovery drills to validate operational readiness

The experience gained through this deployment demonstrates that a carefully designed physical replication architecture can provide strong availability guarantees for scientific services, while remaining transparent, maintainable and fully under operational control.

10 References

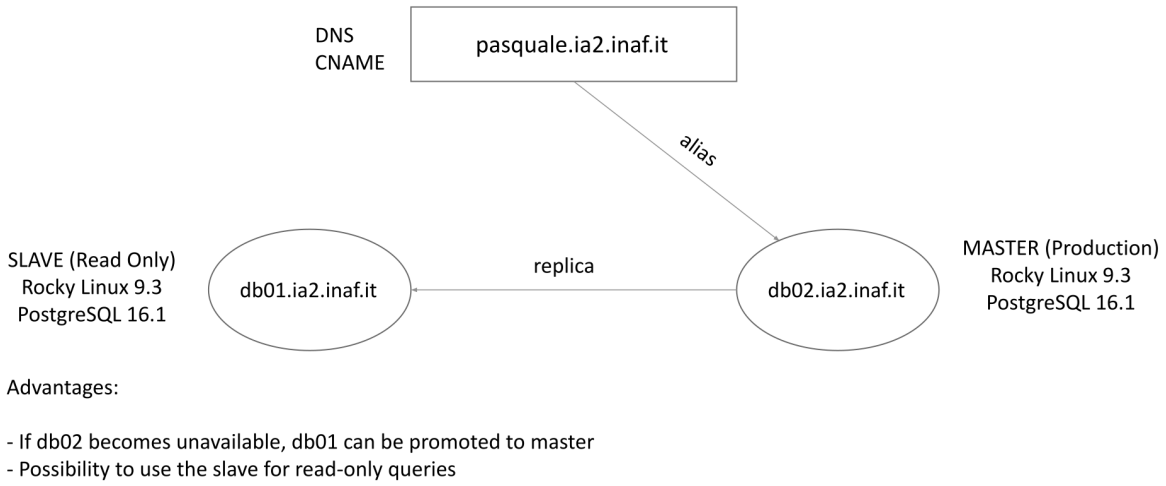
[1] *PostgreSQL* is a powerful, open source object-relational database system:
https://www.postgresql.org

[2] *Rocky Linux* is an open-source enterprise operating system designed to be 100% bug-for-bug compatible with Red Hat Enterprise Linux:
https://rockylinux.org

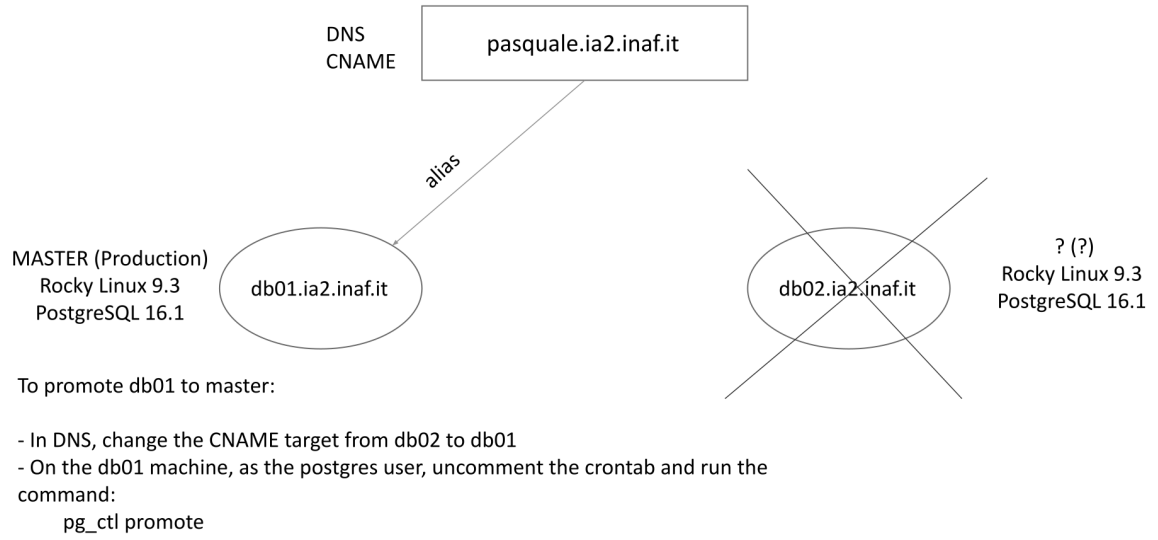
11 Appendices (Schema Overview)

The following schemas summarize the logical and physical layout of the *High Availability* architecture described in this report. They provide a compact visual representation of the system in its different stages of evolution, highlighting the role of the DNS alias, the master–slave relationship and the extension of the topology with an additional replica.

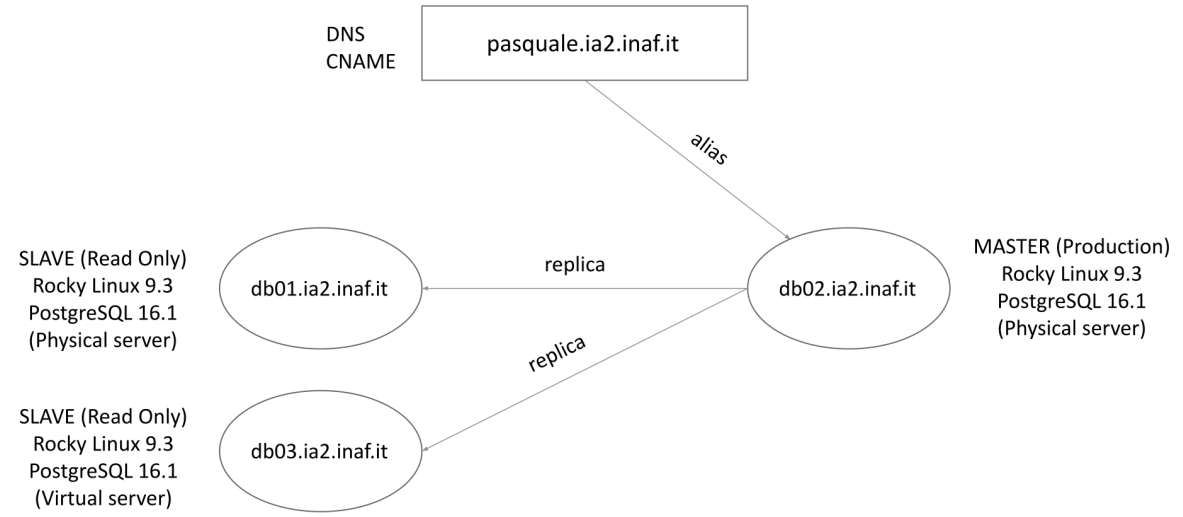
These diagrams are intended to complement the textual description by illustrating how client access is decoupled from the physical nodes and how replication enables controlled failover and service continuity:



Schema 1: This diagram illustrates the basic *High Availability* setup, composed of a production *master* and a single physical replica. Client applications connect through a *DNS CNAME* alias pointing to the *master* node. Physical replication continuously mirrors the entire *PostgreSQL cluster* to the slave, which remains in read-only mode and can be promoted in case of failure.



Schema 2: This schema depicts the transition from *slave* to *master*. The *DNS* alias is reassigned from the failed production node to the replica and the replica is promoted using the native *PostgreSQL* pg_ctl promote command. This sequence enables service continuity while preserving explicit administrative control over the role change.



Schema 3: The final diagram shows the complete deployment, including a second replica hosted on a virtual server. The *master* node streams *WAL* data to both replicas. This extended topology increases resilience, allows maintenance without service interruption and provides an additional recovery path while preserving the same operational model.