TUTORIAL

How to Configure PostgreSQL Streaming Replication Securely

Learn how to configure PostgreSQL streaming replication securely with TLS, least-privilege roles, replication slots, validation checks, and production-ready safeguards.

Databases - July 05, 2026

What you will build

PostgreSQL streaming replication gives you a physical standby that continuously replays WAL from a primary server. The operational problem is simple: if you add replication without securing the transport, credentials, and server roles, you expand the blast radius of a compromise and can expose sensitive data in transit or at rest.

In this tutorial, you will build a secure primary-to-standby replication setup with TLS, a dedicated replication user, least-privilege access controls, and validation checks that prove the standby is actually receiving and replaying WAL. By the end, you should be able to decide whether streaming replication fits your recovery and security requirements, implement it safely, and verify the configuration before production use.

Prerequisites and stop-here checks

Before you begin, confirm the environment supports physical replication and that you are allowed to modify both servers. Streaming replication requires compatible PostgreSQL major versions and access to the primary?s data directory or a fresh base backup path on the standby. If your version, package layout, or service manager differs, verify the exact file locations and restart commands before making changes.

Stop here if any of the following are true:

- You do not have administrative access on both nodes.
- The primary is already under heavy write load and you cannot take a base backup window.
- You have not confirmed the allowed network path between standby and primary.
- You cannot enforce TLS for the replication connection.
- You plan to reuse a general-purpose database account for replication.

A secure configuration depends on planning the trust boundary first. If you already have a pattern for role-based access control in your database estate, use the same discipline here; the replication account should do only replication work and nothing else, similar to the access-control approach described in How to Secure NoSQL Databases with Role-Based Access Control.

Decide the security model before you touch configuration

Goal

Define how the standby authenticates, how the network is restricted, and what evidence you will use to confirm the setup is safe.

Action

Choose these controls up front:

- Dedicated replication role with only the replication attribute or equivalent minimal privilege.
- TLS for client connections from standby to primary.
- Network path restriction so only the standby can reach the replication port on the primary.

- Replication slot if you need WAL retention guarantees and can operationally manage slot hygiene.
- Base backup source and retention policy so the standby can be rebuilt without guesswork.

Expected output

You should have a concrete design for:

- which host connects to which host,
- which account is used,
- which certificate or trust model is required,
- and whether you will use a physical replication slot.

Validation

Document the expected connection string, the standby host IP, the primary host IP, and the exact port. If you cannot state all four, the design is not ready.

Common failure

Teams often start with pg_hba.conf and primary_conninfo before agreeing on trust boundaries. That usually leads to broad allow rules, reused credentials, or later outages when TLS or slot settings are tightened without a rollback plan.

Prepare the primary securely

Goal

Create a replication identity and allow only the standby to authenticate for replication traffic.

Action

Create a dedicated replication user on the primary. Use a strong secret managed through your normal secret distribution process, not a shared shell history or ad hoc file.

Restrict replication access in pg_hba.conf to the standby host or subnet as narrowly as possible. Prefer a host-specific rule over a broad CIDR range.

Then make sure the primary is configured to support replication traffic and sufficient WAL retention. Exact parameter names and defaults vary by version, so verify them for your release before changing them. Common settings to review include wal_level, max_wal_senders, and max_replication_slots if you plan to use a slot.

Expected output

The primary accepts replication connections only from the standby address and only for the dedicated replication role.

Validation

Use the primary to confirm the rule order and test with a connection attempt from any non-standby host. The connection should fail. Also confirm the role does not have unnecessary privileges such as schema ownership, object creation, or login access to normal application databases.

Common failure

A permissive pg_hba.conf entry placed above a narrow one can silently defeat your access control. Another common mistake is leaving the replication role with broader database access than needed.

Enforce transport security with TLS

Goal

Prevent credential exposure and reduce the risk of interception between standby and primary.

Action

Configure PostgreSQL to accept TLS connections and ensure the standby uses TLS when connecting. Depending on your certificate model, this may involve server certificates on the primary and client trust material on the standby.

On the standby, set the connection string so it requires or verifies TLS rather than falling back to plain TCP. A typical pattern is to use sslmode=verify-full when you have proper hostname validation and a trusted certificate chain.

If you use mutual TLS, verify the required client certificate files and file permissions on the standby. Keep private keys readable only by the PostgreSQL service account.

Expected output

Replication traffic is encrypted in transit and the standby fails closed if the TLS trust chain is invalid.

Validation

Confirm the replication session is using SSL by checking the server-side connection view and the standby logs. Also verify that a connection attempt with an invalid certificate, wrong hostname, or missing trust chain fails as expected.

Common failure

A frequent weak point is setting up certificates but leaving sslmode too permissive. If the client still accepts non-validated TLS, you have encryption without strong server identity verification.

Take a base backup for the standby

Goal

Create a consistent starting point for the standby before replication begins.

Action

Stop the standby PostgreSQL service if needed, empty or replace the data directory according to your operational procedure, and take a base backup from the primary. Use the built-in physical backup tool available in your version.

A typical pattern is:

The -R option writes the replication connection settings for the standby in supported versions. Check your version?s behavior because file layout and generated settings differ across releases.

Expected output

The standby contains a consistent copy of the primary data directory and has the replication connection information needed to start recovery.

Validation

Verify the backup completed without errors and that the standby data directory contains the expected structure. If pg_basebackup fails, do not start the standby with a partial directory.

Common failure

The most common mistake is starting the standby before the base backup is complete or after copying files manually. That can produce subtle corruption or recovery failures later.

Configure the standby for secure replay

Goal

Start the standby in recovery mode so it connects securely, receives WAL, and replays changes.

Action

Make sure the standby has the correct connection settings, TLS trust material, and any recovery signal or configuration files required by your version. On modern PostgreSQL releases, standby behavior is typically driven by a recovery signal file and replication settings generated by the backup step or configured explicitly.

If you need to pin the primary endpoint, define the host name carefully and ensure certificate names match. If you rely on a VIP or load balancer, confirm that the presented certificate still matches the hostname used in primary_conninfo.

For operational clarity, treat this setup like any production security control: the secure defaults and access boundaries should be explicit, not implied. If you maintain a checklist for encryption and access controls elsewhere, a resource such as NoSQL Database Security Checklist for Access Control and Encryption can be a useful model for the level of evidence you should require before go-live.

Expected output

The standby starts in recovery mode, connects only to the approved primary, and begins receiving WAL through the secure channel.

Validation

Check the standby logs for connection success and replay progress. Confirm that the standby reports it is recovering rather than serving writes. If the standby comes up as a writable primary, stop and investigate immediately.

Common failure

A misnamed host, certificate mismatch, or wrong file permissions on the key material will usually prevent the standby from connecting. A more dangerous failure is a seemingly healthy standby that is not actually receiving WAL because the connection succeeded earlier but later stopped.

Use replication slots carefully

Goal

Prevent WAL from being removed before the standby receives it, without creating an unbounded retention problem.

Action

If your design requires guaranteed WAL retention for one or more standbys, create a physical replication slot on the primary and bind the standby to it. This can help during network outages, but it also means the primary will retain WAL until the slot catches up.

Example conceptually:

Then configure the standby to use that slot name.

Expected output

The primary retains necessary WAL until the standby has consumed it.

Validation

Confirm the slot is active when the standby is connected and that retained WAL usage stays within expected bounds. Monitor disk growth on the primary during standby outages.

Common failure

Slots are a common source of disk exhaustion. If the standby is broken, paused, or forgotten, WAL can accumulate until the primary fills its storage. Use a slot only if you have alerting and a repair process.

Validate that replication is actually secure and healthy

Goal

Prove the standby is both connected and protected by the controls you intended.

Action

Check the following on the primary and standby:

- the standby is connected with SSL/TLS,
- the replication role is the only account used,
- the connection source matches the approved IP,
- WAL replay is progressing,
- the standby lag is within the operational threshold you set.

Useful server-side checks often include views that expose replication state, sender activity, and recovery status. The exact fields vary slightly by version, so verify the view names and columns in your release.

Expected output

You can confirm the standby is in recovery, receiving WAL from the expected primary, and using the secure transport and approved identity.

Validation

A minimal validation sequence is:

- Confirm the standby reports recovery mode.
- Confirm the primary shows an active replication connection.
- Confirm the connection uses SSL.
- Confirm replay lag is acceptable.
- Confirm unauthorized hosts cannot connect.

Common failure

It is easy to verify only one side of replication and miss a partial failure. For example, the primary may show an active sender while the standby has stopped replaying due to disk pressure or a certificate problem.

Operational follow-up after go-live

Goal

Keep the setup secure and recoverable after initial deployment.

Action

Add the following to your operational runbook:

- certificate expiration checks and rotation timing,
- replication role password rotation if you use password auth,
- alerting for replication lag, sender disconnects, and slot buildup,
- periodic restore or failover rehearsal,
- review of pg_hba.conf changes and firewall rules after every network update.

Also confirm who owns the standby rebuild procedure. Secure replication is not finished when the standby starts; it is finished when you can rebuild, validate, and monitor it without relying on tribal knowledge.

Expected output

You have a documented process for keeping the replication path secure and for recovering it when a certificate, slot, network rule, or node fails.

Validation

Run a controlled test where you stop the standby, verify that alerts fire, and then restore it from a new base backup or your approved recovery method. This proves the security controls do not block operational recovery.

Common failure

Many teams secure the initial setup but never test rebuild and monitoring. That leaves them with a fragile system that looks healthy until the first real incident.

Finished state

A secure PostgreSQL streaming replication setup should have a dedicated replication role, narrow host-based access rules, TLS-encrypted transport, a consistent base backup, and a standby that can prove it is in recovery and replaying WAL from the expected primary. If you can verify those conditions consistently, you have moved from an ad hoc replication link to a production-ready control plane for read availability and disaster recovery.