Engineer

Backup

a copy of data against loss, and the making of it

What it is

A backup is what the data gets restored from when the original is deleted, corrupted or lost along with the machine that held it, so it has to live somewhere those events do not reach. The NIST glossary defines it as a copy of files and programs whose purpose is to make recovery easier. The noun, plural backups, names the copies, and the verb, written as two words, names the act: a nightly job backs up the database and leaves a backup behind.

Neighbouring things get mistaken for it. A replica receives every change within seconds, which protects against a dead disk and copies a mistaken `DELETE` just as faithfully. A snapshot freezes a volume at one moment, and becomes a backup only once it is stored where the original's failures cannot follow. A backup is defined by history and separation: it returns the data as it stood at an earlier point, from a place that survived whatever took the original.

Databases make this concrete. The PostgreSQL manual describes three fundamentally different approaches. An SQL dump, made with `pg_dump`, writes out the statements that rebuild the database, as a consistent snapshot of the moment the dump began. A file system level backup copies the data directory itself. Continuous archiving combines one such base backup with the write-ahead log (WAL) the server produces afterwards, and replaying that log can restore the database to any moment since the base backup. That ability is called point-in-time recovery.

Use the concept in a decision

Put "What is a backup?" to work in a real decision. We can help relate the concept to the system you are building or buying. Backups that are tested rather than assumed, and a written plan for the morning a server does not come back on.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Automated 100 critical data backups using Barman — 100 critical backups automated across the databases

Automated 100 critical data backups using Barman Get in touch →

Why it matters

Two numbers turn a general wish to be safe into a design. NIST defines the recovery point objective, RPO, as the point in time to which data must be recovered after an outage: in practice, how much recent work the business can afford to lose. The recovery time objective, RTO, is how long a system can stay in recovery before the outage harms what the organization is there to do.

The two pull on different parts of the design. RPO is set by how often a copy is taken: a nightly dump can lose a day of writes, while WAL archiving shrinks the loss to the last few moments of log still on their way to the archive. RTO is set by how long a restore takes. The PostgreSQL manual notes that the server replays every WAL segment since the last base backup, which can take a while when that backup is old, so how often to take base backups is an RTO decision as much as a storage one.

Both numbers are claims about a restore, which is why a backup is judged by restoring it. A job that exits with status zero shows that bytes were written somewhere. Whether they rebuild a working system inside the RTO is answered by restoring on a schedule, somewhere other than production, and comparing what comes back.

Separation is the other half. A copy in the same account, under the same credentials, shares the failures it is meant to survive: a closed account, a stolen key, a script run against the wrong target. The useful question for any copy is which single event would take it out with the original.

What it looks like in practice

At the smallest scale, the company's backup and restore-test implementation was designed to extend recovery beyond a provider snapshot. An Ansible role installs restic scripts and systemd units for a database dump where configured, encrypted deduplicated snapshots, retention pruning and integrity checks. A separate monthly workflow restores a known file to a scratch location and compares its bytes. Backup and restore have separate success signals, and the timers require a configured repository. The passphrase comes from a protected

file rather than a systemd-parsed environment value. Those are implemented controls and activation prerequisites: the implementation record dated 2026-10-27 says the timers were disabled without a repository and records no completed live restore. It supports the automation work, not a claim that recovery has succeeded or that the host can already be lost safely.

At a larger scale, a hundred critical data backups were automated with Barman and Google Cloud for a product that lives on its GIS data. Barman handled PostgreSQL, Google Cloud held the offsite copies, and Bash and Python orchestrated and validated the runs. Integrity checks confirmed that each backup was good rather than merely present.

On a third product the work was to define the recoverable position instead of assuming it. Backup retention lives in the Bicep template beside the database, so point-in-time restore is a property of the template rather than a setting someone once clicked in a portal, and the environment around it is code too. The audit wrote the gaps down: geo-redundancy is switched off, and the RTO is proposed rather than measured, with the drill that would measure it described and waiting to run.

Where it goes wrong

Backup problems tend to stay hidden until a restore is needed, so each is worth looking for on a quiet day.

A file copy of a running database restores as a broken cluster. Copying PostgreSQL's data directory with `tar` while the server runs collects files from different moments, and the manual states that the server must be shut down to get a usable backup this way. A filesystem snapshot taken at one atomic instant does work if it includes the WAL, which the server replays on start as it would after a crash; with data and WAL on different filesystems, a simultaneous snapshot may be impossible. `pg_dump`, or a base backup for continuous archiving, gives a consistent copy while the server keeps running.

A gap in the WAL archive ends point-in-time recovery at the gap. Recovery needs an unbroken sequence of archived WAL files reaching back to the start of the base backup. When the archive command keeps failing, PostgreSQL keeps retrying and WAL piles up in `pg_wal/`, so the first symptom is often a filling disk. The manual's advice is to set up and test archiving before the first base backup, and to monitor it for delays.

The job stops, and the silence looks like success. A timer job that breaks writes its error to a log nobody reads, and each quiet day passes for a good one. Monitoring has to treat silence as the alarm: a check-in sent only after a successful run, to a service that pages when it is late, catches a job that has stopped as well as one that fails loudly.

The key goes missing, or arrives changed. Encryption makes access depend on the keys. restic supports multiple passwords for one repository; its warning about irrecoverable data loss applies when no usable repository key remains. The key has to live somewhere that survives the machine. In the restic setup above, the change could have come from the supervisor itself: it interprets escape sequences in environment values, so a passphrase with a backslash, handed over that way, would have reached restic as different bytes from the ones the repository was created with, and without a word of warning. Reading it from a file closes that route.

Retention removes the copy that was needed. Corruption is often noticed weeks after it happened, and a policy that keeps seven daily copies has by then replaced every clean copy with a damaged one. A tiered policy keeps a long tail at low cost, and restic's `--dry-run` shows what a policy would remove before anything goes:

```
restic forget --keep-daily 7 --keep-weekly 5 --keep-monthly 12 \
  --prune --dry-run
```

`forget` removes snapshots and `prune` removes the data only they referenced, so after a prune the old copies are gone for good.

The integrity check passes and the restore still breaks. By default, `restic check` verifies the repository's structure without reading every pack file, since that means downloading all of them; `--read-data` reads everything, and `--read-data-subset=1/5` reads one fifth, so five runs from `1/5` to `5/5` cover the lot. A full read proves only that the stored bytes are intact. Whether they rebuild a working database, with the right roles and settings, is something a restore shows.

Further reading

- NIST glossary: backup — the definition used here and the publications behind it.
- NIST SP 800-34 Rev. 1 — the contingency planning guide that defines RPO and RTO.
- PostgreSQL: Backup and Restore — the three approaches and what each one costs.
- PostgreSQL: Continuous Archiving and Point-in-Time Recovery — base backups, WAL archiving and restoring to a chosen moment.
- restic: Preparing a new repository — passwords, password files and the warning about losing them.
- restic: Removing backup snapshots — retention policies, `forget`, `prune` and `--dry-run`.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is a backup?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/backup/