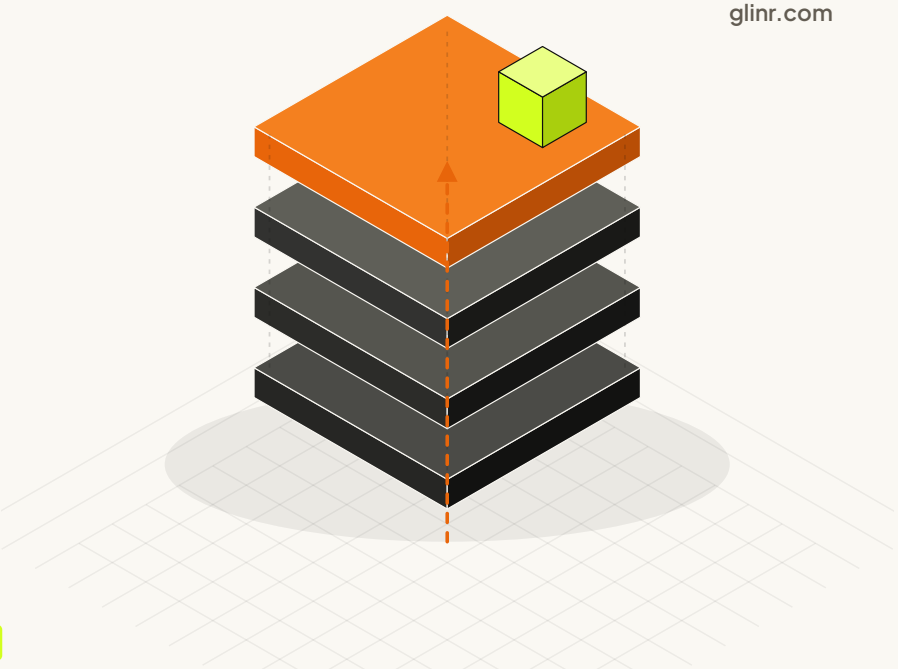
GLINR STUDIOS

glinr.com



WHITEPAPER · GLINR-WP-2026-101

Restore Is the Product

Backups you can prove: measuring RPO and RTO, mapping threats to surviving copies, and testing restores before you need them

glinr.com

github.com/glincker

GLINR Studios
November 2026 · Version 0.1.0

A GLINR Studios publication
glinr.com

Restore Is the Product

Backups you can prove: measuring RPO and RTO, mapping threats to surviving copies, and testing restores before you need them

ABSTRACT

A backup job reports that it ran. It does not report that your data can come back. In tests on one laptop, four backup jobs that saved nothing useful all exited 0, flipped bytes in a restic repository passed the default check, and a one-database restore that took 0.21 s still needed four rounds of fixes before the application could log in. This paper treats a backup as a system you measure, starting from the restore: four gates a copy must pass, public incidents mapped to the gate each one failed, the 3-2-1-1-0 rule recast as a table of failure domains, and a method to declare and then measure RPO and RTO on a clean host. It closes with a runbook and an evidence checklist. The scripts and raw results are published.

KEYWORDS

backups · restore testing · RPO · RTO · failure domains · immutable backups · ransomware · disaster recovery · restore drills

Author	GLINR Studios
Version	0.1.0
Date	November 2026
Report ID	GLINR-WP-2026-101
License	CC BY 4.0
Source	github.com/glincker/wp-2026-101-backup-restore-testing
Web	glinr.com/research/2026-101-backups-you-can-restore
Review	Not externally reviewed
Verified	November 2026
Contact	support@glinr.com

AT A GLANCE

The problem
A backup job reports that it ran, not that your data can come back. Most teams find the gap during a restore, when it costs the most.

What this adds
A method, not a product: four gates a copy must pass, public incidents mapped to the gate each failed, and a way to declare then measure RPO and RTO on a clean host. Scripts and raw results are published.

Who it is for
People who run their own stack: a few servers, a database, object storage and files, and no full-time backup owner.

How to use it
Copy the tables into a review, declare your RPO and RTO, and run the Prove it check ending each chapter.

KEY FINDINGS

01 **Exit 0 is intent, not evidence.**
Four jobs that saved nothing useful all exited 0. One dumped the wrong database: 1,030 bytes against 285,501, restored to zero tables.[M] Ch. 1, 7

02 **Damage can be invisible.**
After 16 flipped bytes in a restic repository, a new backup and the default check both exited 0. Only the full data check failed.[M] Ch. 1, 7

03 **A fast copy is not a fast recovery.**
A one-database restore took 0.21 s, yet the application could not log in until four rounds of fixes, because roles and settings live outside the dump.[M] Ch. 5

04 **Timings are a range, not a number.**
Ten times the data took 3.3 to 6.9 times as long to restore, and the same suite varied by 1.0 to 9.6 times between two runs.[M] Ch. 4

05 **Immutable is only as strong as its account.**
S3 Object Lock in compliance mode can be removed early only by deleting the AWS account, so a locked copy in production's own account is exposed to whoever controls it.[S] Ch. 3, 6

06 **Nobody owned the test.**
GitLab's 2017 backups used pg_dump 9.2 against PostgreSQL 9.6, and its postmortem says nobody was responsible for testing the procedure.[S] Ch. 1, 9

CITE AS

GLINR Studios. *Restore Is the Product*. November 2026. GLINR-WP-2026-101, version 0.1.0. glinr.com/research/2026-101-backups-you-can-restore

DISCLOSURE

GLINR Studios builds Levelrail, a self-hosted deployment platform. No product is recommended in this paper. Tools appear only as test subjects, with versions.
Funding: No external funding. This is the studio's own open-source work, paid for by the studio.

CLAIM LABELS

Claims carry a small tag: [M] measured by us, [S] sourced from a primary source, [D] derived with the arithmetic shown, [O] opinion. No unverified claim is printed.

RIGHTS

Copyright © GLINR Studios. Licensed CC BY 4.0: share and adapt with attribution. Sourced claims cite the references.

Contents

Preface 3

CHAPTERS

1 **A backup job is not a backup** 4
Four gates a copy has to pass, read through GitLab's 2017 postmortem.

2 **What actually destroys data** 6
Which failures take out which copies, across five public incidents.

3 **3-2-1-1-0 as a failure-domain table** 8
Why three copies can still be one failure, and an audit table for yours.

4 **RPO and RTO as numbers you measure** 10
Declare an RPO and an RTO, restore onto a clean host, and compare.

5 **What gets backed up** 12
The layers of a service, and what a one-database dump leaves behind.

6 **Credential compromise and immutable copies** 14
A copy that stolen credentials cannot delete, and what it still misses.

7 **Verification, silent failure and restore drills** 16
What each check can and cannot see, measured on damaged repositories.

8 **Restore at scale and under pressure** 19
Why handing back one customer outlasts copying their bytes.

9 **Runbook, ownership and the evidence checklist** 21
Who owns the restore, and the least evidence they should hold.

APPENDICES

A **Restore-drill harness specification** 23
The drill as a specification: inputs, steps and pass criteria.

B **Restore-drill record** 24
A one-page record to fill in after every drill.

C **Failure-domain audit** 25
Audit your copies against the failure domains.

D **Threat by copy matrix** 26
Which threat each copy survives, as a matrix.

E **Runbook skeleton** 27
The page the owner opens when the incident is declared.

F **Evidence checklist** 28
Ten rows an owner should be able to show today.

BACK MATTER

How to verify this paper 29

Glossary 30

References 31

About this paper and license 32

READING PATHS

Fifteen minutes
Key findings, chapter 1, then Appendix F.

You run databases
Chapters 4, 5 and 7.

You worry about attackers
Chapters 3 and 6.

You own the process
Chapters 8 and 9, then Appendices B and E.

Preface

We aimed a `pg_dump` at the wrong database, on purpose. It exited 0 and wrote 1,030 bytes, where the database we meant to dump gives 285,501. Nothing complained [1][M]. A backup job tells you that it ran. It does not tell you that you can get your data back, and those are different facts.

So this paper starts at the restore. It treats a backup as a system you measure: you declare how much data you can lose and how long you can be down, then you prove both by restoring onto a clean host with a stopwatch running. The incidents we use are public, and each one shows one way a copy lets you down. Where two sources disagree we print both and do not reconcile them. The measurements are ours, from one laptop, and they are labeled as indicative.

We wrote it for people who run their own stacks: a few servers, a database, some object storage and files, and nobody whose full-time job is backups. If you have a platform team, you will recognize a lot of it. The tables are built to be lifted into a review.

> HOW TO READ THIS
> Chapter 1 gives the model: four gates that a copy has to pass. After that, jump to the chapter that matches the worry in front of you. Each one opens with a question and a line saying which gates it bears on, and ends with a "Prove it" check you can run this week. The appendices are printable pages, from a drill record to an evidence checklist. The small tags after claims say how each is known: M measured by us, S sourced, D derived, O opinion.

Disclosure. GLINR Studios builds Levelrail, a self-hosted deployment platform. No product is recommended in this paper. Tools appear only as test subjects, with versions. Beyond this note, the studio's own work is not mentioned.

GLINR Studios

CHAPTER 1

A backup job is not a backup §

The question: what does "trustworthy" mean for a backup, and why do copies that exist still fail on the day you need them?

GitLab's postmortem of its database outage of 31 January 2017 says that "the backup procedure was using `pg_dump` 9.2, while our database is running PostgreSQL 9.6". It says Azure disk snapshots "were not enabled for any of the database servers" and that GitLab.com "was not using WAL archiving". In the end, "GitLab.com had to be restored using a copy of the staging database" [2][s].

None of that is exotic. Each item is a tool version, a disabled setting or a missing archive, the kind of detail an exit code never shows[o]. This chapter measures the distance between a backup job that ran and a backup you can restore.

1.1 Four gates, not one §

A backup is a promise about a future restore. We split that promise into four gates. A copy is trustworthy when it has passed all four on evidence, and until then it is an assumption. The gates are our own framing, not a standard. They are separate questions, and a green job answers only the first, and weakly.

Gate	Passes when	Proof that counts	Proof that does not
1 Exists	A new, non-empty copy sits somewhere you can name	Size, object count and date compared with the previous run	The job's exit code
2 Readable	A tool can open the copy and confirm it is intact	A check that reads the stored data	A check that skips the stored data (restic's default `check`, in our test, chapter 7[M])
3 Restorable	It restores onto a clean host and the application works	A restore followed by a probe that logs in, reads and writes as the application does	The restore command finished
4 Fast enough	The restore meets the recovery time objective (RTO) and recovery point objective (RPO) you accepted, defined in chapter 4	A timed restore at today's data size, with the median and the worst of at least three runs	A timing from a smaller dataset or a quiet day

The Google SRE book draws the line at gate 3: "The most important difference between backups and archives is that backups can be loaded back into an application, while archives cannot" [3][s]. It also warns that "replication and redundancy are not recoverability" [3][s], so a replica is not a fifth gate that lets you skip the others. Its "one lesson" from the chapter is: "you only know that you can recover your recent state if you actually do so" [3][s].

A green job proves, at most, that a file was written.

Gates 1 and 2 can be checked cheaply and often. Gates 3 and 4 can only be proven by a restore someone ran. A passing restore does not prove gate 2: in our restic test a restore passed while `check --read-data` failed (chapter 7). That is why every gate in the figure below is drawn with a dashed outline.

1.2 GitLab 2017, gate by gate §

We asked which gate each line of the postmortem belongs to. The placement is our reading, not GitLab's. Where the postmortem is silent, the figure shows a test of ours instead, marked as such. Gate 4 is also where the recovery point and recovery time objectives live, which chapter 4 defines and measures.

GATE 1: THE COPIES AND THE ALERT PATH

The dump job used a `pg_dump` older than the server it dumped. Azure disk snapshots were not enabled for the database servers, and there was no WAL archiving. Recovery came from a copy of the staging database. On the alerting, the postmortem says: "DMARC was not enabled for the cronjob emails, resulting in them being rejected by the receiver" [2][s]. An alert that is rejected on arrival cannot warn anyone, so we count the alert path as part of gate 1[o].

GATE 2: NOTHING TO PLACE, SO WE TESTED IT

We cannot place any line of the postmortem at gate 2, so here is our own case. In a restic 0.19.1 repository holding a 50 MB dataset (28.5 MB on disk) we flipped 16 bytes in the middle of a data pack. A new backup exited 0 and the default `restic check` exited 0 and reported no errors. Only `restic check --read-data` exited 1 [1][M]. One case is enough to show a check can pass without reading the data it vouches for. Chapter 7 has the full table, including a case where a restore passed while the deep check failed.

GATE 3: AN UNTESTED PROCEDURE

Asked why the backup procedure was not tested on a regular basis, the postmortem answers that there was no ownership, so nobody was responsible for testing it [2][s]. Chapter 9 quotes the sentence. To see what an untested restore looks like when the copy is fine, we restored a `pg_dump -Fc` of one database onto a fresh PostgreSQL 14.18 cluster that lacked the application roles. The restore took 0.21 s and ignored 12 errors, each naming a role that does not exist. The data arrived. The roles, grants, ownership and settings did not, and the application could not log in until four rounds of fixes were done [1][M]. Chapter 5 gives the account.

GATE 4: HOURS, AND TWO ANSWERS FOR THE LOSS

The postmortem says GitLab.com "was down for about 18 hours" and that "restoring GitLab.com took over 18 hours" [2][s]. On the data lost it gives two windows. The introduction says modifications between 17:20 and 00:00 UTC on 31 January were lost. The impact section says data "created between January 31st 17:20 UTC and 23:30 UTC has been lost" [2][s]. That is 6 h 40 min against 6 h 10 min, our arithmetic[D]. GitLab's best estimate of the damage is "roughly 5,000 projects, 5,000 comments

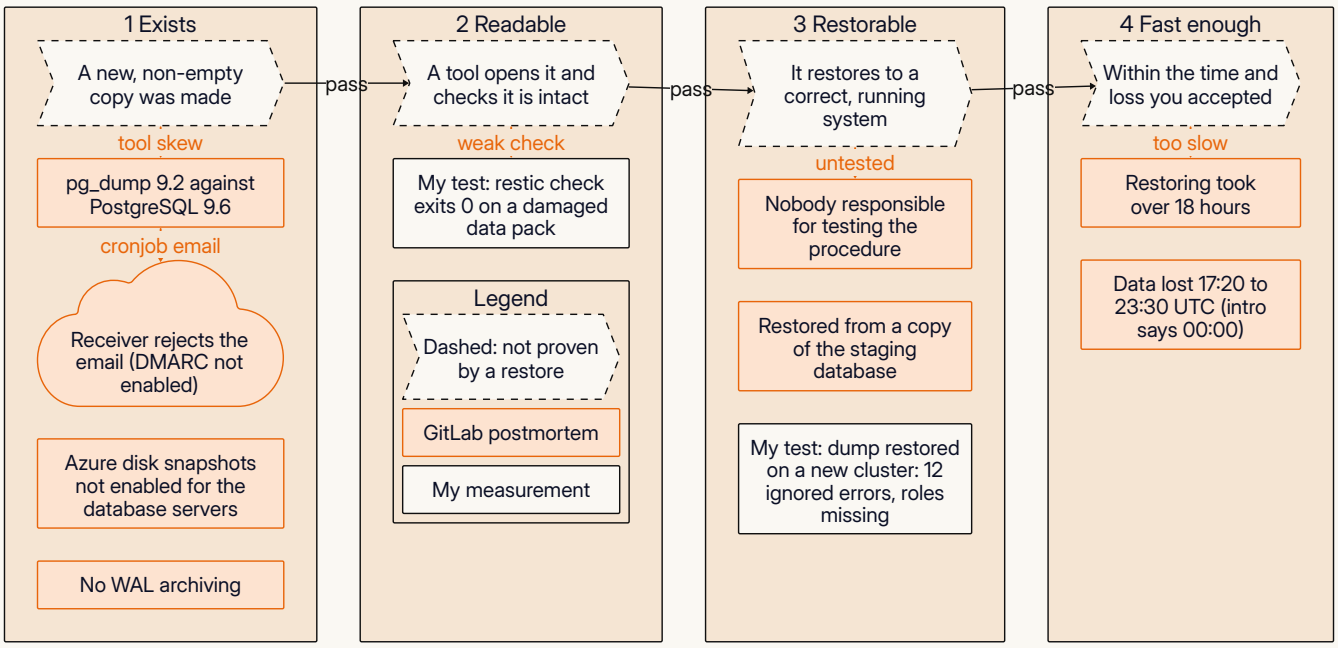


Figure 1.1. The GitLab 2017 postmortem placed on the four gates, with our own measurements where the postmortem has nothing to say. Each gate is dashed because, until a restore proves it, it is an assumption.

and 700 new user accounts", an estimate by the company itself [2][S].

Our own timings are laptop seconds, not hours, and the useful part is their shape: ten times the data did not take ten times as long, and the same suite run twice differed from itself by up to 9.6 times on single steps [1][M]. Chapter 4 gives the numbers and the limits. Record a range with a date.

1.3 What a green job tells you §

The GitLab case is about copies that were missing or unusable when needed. We also wanted jobs that succeed and leave nothing restorable, so we built four, each exiting 0 on the same laptop.

Job	Exit	What it left behind
`pg_dump -Fc` pointed at the wrong database	0	1,030 bytes instead of 285,501 (the intended database was a separate small pgbench database, see chapter 5). Restored into a new database, it gave 0 tables in the public schema
`pg_dump` piped into `gzip` while `pg_dump` fails	0	A valid 20-byte gzip file holding 0 bytes, which `gzip -t` accepts. With `set -o pipefail` the pipeline exits 1
restic, borg and tar with zstd backing up an empty mount point	0	A snapshot or archive with 0 files, and a 348-byte tar.zst file
`restic forget --keep-last 2 --prune` after a third, empty backup	0	The last good snapshot deleted, a repository of 9,237 bytes, and `restic restore latest` returning 0 of 1,030 files

The mount dataset has 1,030 files, one more than the 1,029 of the damage tests in chapter 7, because the mount copy carries a canary file. The first three jobs were each caught by a check of a few lines: a size floor and a look at the archive's table of contents, `set -o pipefail`, or a canary file and a file count compared with the previous run [1][M]. We did not test a check for the fourth. These are simple examples and not a monitoring design. What matters is that all four would show green on a dashboard that only reads exit codes.

A gate 1 check can be this small. The sketch below combines two of the checks that caught our cases, `pipefail` and a size floor. We ran them as separate steps and not as this script, and `stat -f %z` is the macOS spelling (GNU `stat` uses `-c %s`). With `pipefail` the pipeline fails when `pg_dump` fails, `set -e` stops the script there, and the floor comes from the environment so each system sets its own.

```
set -eo pipefail
pg_dump -Fp "$DB" | gzip > "$OUT"
[ "$(stat -f %z "$OUT")" -ge "$MIN_DUMP_BYTES" ]
```

A job that exits 0 has told you the process finished. Everything else in this chapter is evidence you still have to go and get.

> **WHAT WOULD CHANGE OUR MIND**
> If a check that never reads the stored data found damage as reliably as one that does, gates 1 and 2 could merge. Our restic test says it does not, for that one tool and that kind of damage, so repeats on other tools and damage patterns could move this.

> **WHEN DID YOU LAST RESTORE?**
> Write down the date of your last full restore and how many minutes it took. If you cannot, write "unknown". Then write one line per gate naming the artifact that proves it: a restore log, not a job log, with the date, the command, the exit code, the minutes and who ran it.

CHAPTER 2

What actually destroys data §

The question: which failures destroy data, and which kind of copy is still standing after each one?

In the model of chapter 1: threats to gates 1, 2 and 3.

Chapter 1 asked whether a copy can be trusted. This chapter asks what, specifically, it has to survive.

UniSuper had duplication in two geographies. Its joint statement with Google Cloud says it had that as a protection against outages and loss, and that the misconfiguration caused deletion across both of these geographies [4][s]. Two geographies protect against a fire or a regional power cut. They did not protect against a misconfiguration that deleted the subscription holding both[o].

A copy is a bet against a specific threat, and a bet against one threat says nothing about the others. So stop counting copies. List the threats and ask each kind of copy whether it would still be there[o].

2.1 Three ways a copy fails you §

Reach. The threat can get to the copy because the copy shares something with the thing that was destroyed: a site, an account, a set of credentials, a provider. Chapter 3 calls each of these a failure domain and turns them into a per-copy audit. The sections on facility loss, your own script and the control panel are about reach.

Carry. The copy holds the damage unchanged. The SRE book makes the point about replication in one line: "replication and redundancy are not recoverability" [3][s]. A backup taken after the damage carries it as well, and the section on damage below points to a measurement[M].

Absence. The copy was never made, or is empty, and nothing said so. The last section is about absence.

2.2 One provider deletion, read through the three failures §

Google's post says one input parameter was left blank when an internal tool was used to provision the customer's Private Cloud, so the system assigned a then unknown default fixed one year term, after which the Private Cloud was deleted [5][s]. The two accounts describe the scope in different terms. UniSuper's statement speaks of two geographies. Google's post speaks of one customer in one cloud region and one of that customer's multiple Private Clouds, across two zones.

On the copies that survived, the accounts agree in part. UniSuper's statement says it had backups in place with an additional service provider. Google's post says backups stored in Google Cloud Storage in the same region were not impacted and, along with third party backup software, aided the rapid restoration [4], [5][s]. They differ on which backups were used, so we make no claim about which copy did the work. Google also says the customer's and its own teams worked 24x7 over several days to recover the Private Cloud, restore network and security configurations and applications, and recover data [5][s]. Survival and speed are different properties. Chapter 4 measures the second one.

Our reading is that the replicated pair died together because it shared a failure domain with the thing destroyed, and that anything outside that domain had a chance[o]. Nobody drew the domain for this threat in advance. The matrix exists to draw it.

2.3 The matrix §

The table asks eight threats of five kinds of copy. Read it by rows, not columns. A row with a "No" in every cell you actually have is a loss you have accepted, whether or not anyone decided to.

Delete columns D and E, which a small stack may well lack, and two rows have no survivor: stolen credentials and ransomware[D]. Both lose every copy that production's own credentials can reach. Ransomware is the credentials row with a clock on it. Encrypting instead of deleting means every copy written after the encryption starts holds ciphertext, so only versions older than the attacker's arrival help[o].

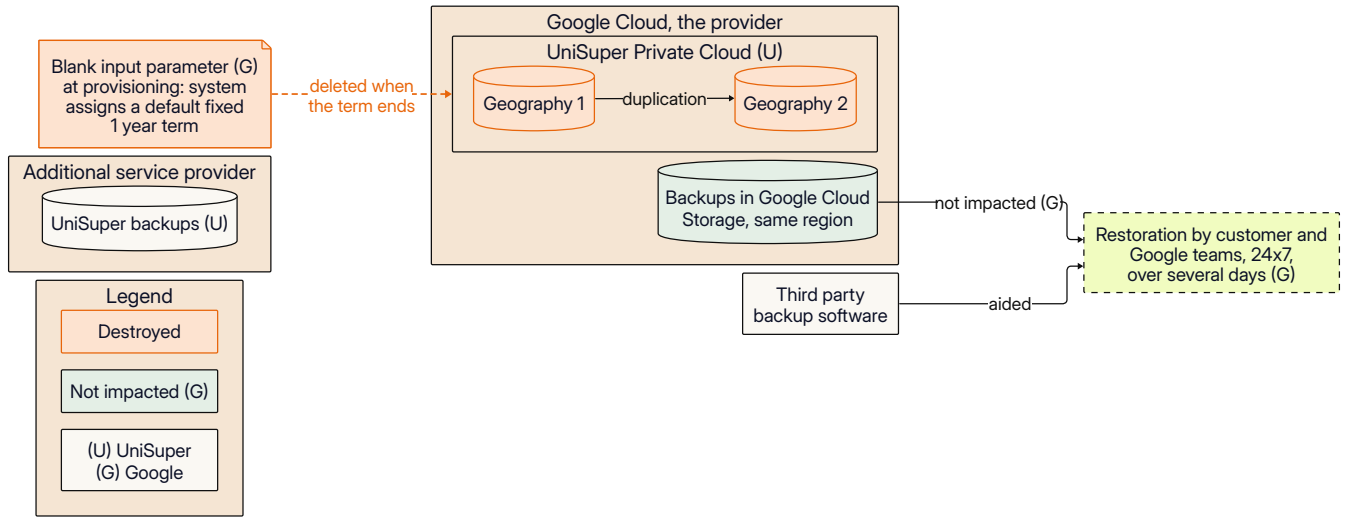


Figure 2.1. UniSuper and Google Cloud, May 2024, as the two first-party accounts describe it. The duplicated pair sat inside the scope of the deletion. Google says backups in Cloud Storage in the same region were not impacted, and UniSuper says it had backups with an additional provider. The two accounts differ on which backups were used in the restoration, so the figure draws no arrow from the additional provider and does not say where the third party software ran.

Threat	A. Live replica	B. Snapshot, same account and site	C. Backup, same provider and logins	D. Backup, other provider and logins	E. Write-once or offline copy
Disk or host failure	**Yes**	**If** stored apart from the disk	**Yes**	**Yes**	**Yes**
Fire or loss of the site	**If** at another site	**No**	**If** at another site	**Yes**	**If** kept off site
Mistaken delete or script	**No** replicates it	**If** older and out of the script's reach	**If** out of the script's reach	**Yes**	**Yes**
Bad data written (bug, corruption)	**No** replicates it	**If** a version predates it	**If** a version predates it	**If** a version predates it	**If** a version predates it
Provider deletes or closes the account	**No**	**No**	**If** outside the deleted scope	**Yes**	**If** outside the provider
Stolen credentials	**No**	**No**	**No**	**If** production's keys cannot delete	**If** the lock cannot be shortened
Ransomware	**No**	**No**	**No**	**If** a version predates the attack	**If** a version predates the attack
Backup job fails or writes nothing	**Yes** not made by a job	**If** a check fails loudly	**If** a check fails loudly	**If** a check fails loudly	**If** a check fails loudly

Table 2.1. Which kind of copy is still standing after each threat. Yes: outside the threat's reach. If: only when the note holds. No: it dies with the original. The cells are our reasoning, not measurements, and each is a claim to prove with a restore drill. Retention is assumed to reach back before the damage began.

To read a row, take "Provider deletes or closes the account" with UniSuper in hand. The live replica and the snapshot in the same account die with the account: two No cells. UniSuper's statement says the deletion crossed both of its geographies, so duplicating its data across two geographies gave no protection against this threat [4][s]. A backup with the same provider survives only if it sits outside the deleted scope, an If, and Google says backups stored in Google Cloud Storage in the same region were not impacted [5][s]. A backup with another provider is a Yes, and UniSuper says it had backups in place with an additional service provider [4][s]. That is the row working: kinds of copy that look different, and a threat that treated some of them as one.

FACILITY LOSS

Disk failure is the easy row, and replication is a fine answer to it[o]. The facility row is where the "backup" label misleads. In the Bati Courtage decision against OVH-cloud, reported by Blocks and Files (23 March 2023), the contract described the backup as physically isolated from the infrastructure of the VPS, yet the backup was in the same building as the VPS, which sat in SBG2 [6][s]. Chapter 3 works the case through.

YOUR OWN SCRIPT

Atlassian's update says a script was executed with the wrong execution mode and the wrong list of IDs. The team supplied the IDs of entire cloud sites instead of the IDs of the app being deactivated, and the script combined a recoverable mark-for-deletion option with a permanent-delete option [7][s]. A live replica is built to follow a change like that one[o]. Atlassian says it maintains immutable backups, and its post-incident review says no customer lost more than five minutes of data, yet its update says it takes between 4 and 5 elapsed days to hand a site back to a customer [7], [8][s]. Chapter 8 tells that incident in full. A backup that survives a mistaken script can still mean a 13 day incident[p].

SOMEONE ELSE IN THE CONTROL PANEL

Code Spaces shows how much one control panel can reach. The Register reports that an attacker gained access to the company's Amazon EC2 control panel, and the company said most of its data, backups, machine configurations and offsite backups were either partially or completely deleted [9][s]. Chapter 6 gives the report and what it does not say.

2.4 Damage that gets copied §

The bad-data row asks about time rather than place. Every copy made after the damage carries the damage[o]. How often does stored data get damaged? Bairavasundaram and co-authors examined about 1.53 million disk drives over 41 months and reported more than 400,000 instances of checksum mismatches, with nearline disks showing about an order of magnitude more than enterprise-class drives [10][s]. That is a 2008 study (FAST '08), not a rate for your disks, and real damage takes other forms than our synthetic flips.

Whether your tools notice damage is a measurement, and chapter 7 reports one. We flipped bytes in stored restic and borg repositories and in one page of a live PostgreSQL table. The default `restic check` and a new restic backup passed over the damage, and in one case so did a restore of a snapshot, while `restic check --read-data` caught it. borg's default check noticed. PostgreSQL without data checksums noticed nothing at all on the backup path [1][m].

2.5 Copies that were never made §

The last row is the one where every job-made copy depends on a single thing: a check that fails loudly[o]. GitLab's 2017 postmortem lists a dump job built on the wrong tool version, an alert path whose emails were rejected, and no snapshots or WAL archiving [2][s].

A reader story in The Register's On Call column (7 February 2025) describes a log that reads "Backup completed successfully" and a tape that turned out blank when someone tried the restore [11]. It is an anecdote.

We built the same failure on purpose: four jobs that exited 0 and left nothing restorable, including a retention run that deleted the last good snapshot (the table in chapter 1) [1][m]. Three of the four were caught by a few lines of checking, which chapter 7 builds properly.

> **WHICH ROWS HAVE NO SURVIVOR?**
> Fill the table for your own stack, using the copies you actually have. Mark each cell Yes, If or No, and write the condition next to every If. Count the rows where every copy dies. Then pick the bad-data row, damage one file in a scratch copy of a backup, and see which of your commands notices.

CHAPTER 3

3-2-1-1-0 as a failure-domain table §

The question: what counts as a separate copy once you account for shared accounts, regions, credentials and software versions?

In the model of chapter 1: gates 1 and 3, counted by failure domain.

Bati Courtage had a backup, and it sat in the same building as the VPS. A counter that sees two objects says two. One that asks what each object depends on says one.

Bati Courtage's contract described its backup option as physically isolated from the infrastructure holding its VPS. Blocks and Files reports that the backup was in the same building as the VPS, which sat in SBG2, and that Bati Courtage was awarded EUR 101,102 in compensatory damages (decision of 3 February) [6][s]. For Bluepad, production was in SBG1 and the backup in SBG2, and the award was EUR 153,837 (ordered 16 March) [6][s]. The two awards sum to EUR 254,939 (our arithmetic, the article gives no total)[p]. As of the article's date, 23 March 2023, OVH had appealed the Bati Courtage decision and reportedly intended to appeal the Bluepad decision. Later status was not checked.

3.1 Count domains, not copies §

A failure domain is a set of things that fail together for one cause. The model is one rule: **a copy is separate only if it shares no failure domain with the original.** It uses four domains. The list and the incident mapping are ours, built from the incidents below, and the list is not exhaustive[o]. Counting copies has one extra rule for backup tools: snapshots in one repository share its packs, so they count as one copy[p]. Chapter 7 shows a new snapshot deduplicating against a damaged blob without reading it.

Domain	Fails together when	Seen in
Account	one owner, subscription or billing identity is deleted, suspended or closed	UniSuper
Region or facility	one building, site or region is lost	OVHcloud
Credentials	one identity or key can delete both the original and the copy	Code Spaces
Software version	one tool at one version wrote the copy and must read it back	GitLab

The audit is mechanical. Write production's value for each domain, then each copy's. A cell is **shared** when it equals production's. The count of shared cells says through how many of the four domains one event could take both. It is a screen, not a risk model, and says nothing about probability[o]. The table drops the medium. In the example stack below, disk snapshots and object storage sit behind the same account and the same key, so the account and credential columns already count what a second medium was meant to buy.

On a plain count the stack looks generous: production plus four copies, two kinds of storage, two copies outside Region 1. On the domain count it has one copy. The last two columns make it worse: nothing immutable, and one copy ever restored.

3.2 The old rule and its vendor extension §

Veeam states 3-2-1 as three copies of data on two media types, with at least one copy off-site [12][s]. It then writes: "At Veeam, we extended the rule to 3-2-1-1-0. It adds one immutable copy, and results in zero recovery errors." [12][s] So the term is vendor material. We keep it as a checklist and redefine it in failure-domain terms.

Bluepad shows why every cell needs evidence. Production was in SBG1 and the backup in SBG2, and two labels do not tell you whether those places fail together.

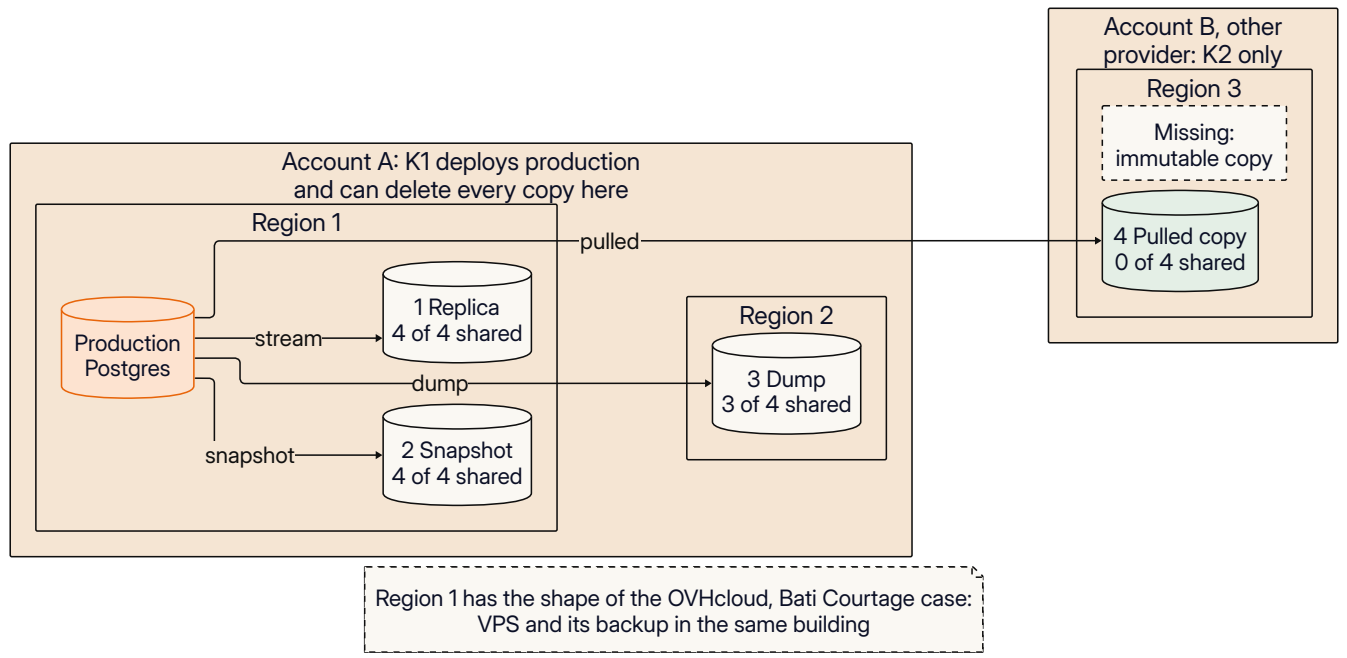


Figure 3.1. One hypothetical stack. Four copies of one Postgres look like plenty. Only the pulled copy in Account B shares none of the four domains with production. "n of 4 shared" counts the cells that equal production's.

Copy	Account	Region	Credential	Version domain	Shared	Immutable	Last restore drill
Production Postgres	A	R1	K1	engine v16	n/a	no	n/a
1 Streaming replica	A	R1	K1	engine v16	4 of 4	no	never
2 Nightly volume snapshot	A	R1	K1	engine v16	4 of 4	no	never
3 Nightly dump in a bucket	A	R2	K1	dump tool v16	3 of 4	no	never
4 Pulled copy, other provider	B	R3	K2	backup tool v2	0 of 4	no	passed

Figure 3.2. Audit of the same stack. Circled cells equal production's. Values are illustrative, not measured. The last two columns are the "1" and the "0" of 3-2-1-1-0: no copy is immutable, and only copy 4 has ever been restored.

The provider's own documentation does. Writing that answer down is the audit.

3.3 Account: the quiet shared cell §

UniSuper's joint statement with Google Cloud (8 May 2024) says UniSuper had duplication in two geographies as protection against outages and loss, and that an inadvertent misconfiguration during provisioning resulted in the deletion of its Private Cloud subscription and caused deletion across both geographies [4]ˢ. Google's own post (24 May 2024) describes the scope differently, as chapter 2 shows [5]ˢ.

Our reading, not either source's, is that if both geographies were deleted by the deletion of one subscription, they count as one copy as far as the account domain goesᵒ. UniSuper's statement also says it had backups with an additional service provider, and Google's post says backups in Google Cloud Storage in the same region were not affected. The two sources differ on which backups did the work, so we draw no conclusion about that.

3.4 Credentials: what "immutable" promises §

Code Spaces is the closest case we have to the credential domain: one control panel reached production data, snapshots and what the company called offsite backups. Chapter 6 gives the report and what it does not say [9]ˢ.

An immutable copy is a claim about a credential domain: immutable against which credential, held by whom. Chapter 6 reads the documentation for each mechanism. One result matters for this table. S3 Object Lock in compliance mode can be removed early only by deleting the associated AWS account, so a locked copy in production's own account is exposed to whoever controls that account [13]ˢ.

3.5 Software version: it fails on write and on read §

The GitLab postmortem says the backup procedure was using `pg_dump` 9.2 while the database ran PostgreSQL 9.6 [2]ˢ. The PostgreSQL 18 documentation (read 2026-11-11) says `pg_dump` cannot dump from a server with a newer major version than its own and refuses to try rather than risk an invalid dump [14]ˢ. We read this domain as failing in two waysᵒ. First, a copy written by the same version as production, such as a replica or a physical snapshot, depends on that one version. Second, a copy written by a different version depends on a tool pairing: the tool that wrote it, the server it read and the tool that must read it back all need to agree. The GitLab case is a mismatch on the writing side. Nothing in our sources or measurements covers the reading side: our restore tests used the same PostgreSQL 14.18 build on both hosts. The audit table shows only the first way. Record the restore host's version next to each copy's, so a mismatch is visible before a restore.

3.6 The zero §

"Zero recovery errors" has to be falsifiable. An exit code of 0 is not that, and a restore that finishes can still leave the application unable to log in (chapters 1 and 5).

3.7 Using the audit §

Each row of the generic matrix in chapter 2 asks which copy survives a given threat. The audit asks the same question from the other side: for each copy, which domain does a threat act through? Facility loss acts through the region, provider-side deletion through the account, a stolen or mistaken key through the credential, a bad tool release through the version. A copy dies for a threat when it shares that threat's domain with productionᵒ. Every cell needs evidence you can show someone.

Domain	Evidence worth writing down
Account	The account or subscription identifier from the provider console, and who can close or delete it.
Region or facility	The provider's own documentation of what the location name means: one building, one site or one region. A label alone is not evidence.
Credentials	Every identity with delete rights on the copy. If any of them also deploys production, the cell is shared.
Software version	The version output of the writing tool, and of the tool on the host where you will restore.

The audit has blind spots. It weights the four domains equally, and they are not equal. It does not see dependencies outside its columns, such as where the key that decrypts a copy is kept, and it cannot tell a complete copy from a partial one. Add a column when a copy needs one, and let a drill, not the table, have the last wordᵒ.

> HOW MANY COPIES ARE REALLY SEPARATE?
> Pick your three best copies. For each, write the account, region, credential and tool version, and circle every cell equal to production's. A copy with all four circled can be taken by any event that acts through one of these four domains. Write down how many copies have none circled. That is your copy count.

CHAPTER 4

RPO and RTO as numbers you measure §

The question: how do you derive an RPO and an RTO from what you can afford to lose, and then measure the values you actually get?

In the model of chapter 1: gate 4.

So far the question was whether a copy exists and is separate. Now the question is how fresh it is and how long it takes to come back.

Atlassian reported two figures for one incident. Its post-incident review of 29 April says "no customer lost more than five minutes of data" [8][s]. Its update of 13 April says "it takes between 4 and 5 elapsed days to hand a site back to a customer" [7][s]. The first is a recovery point and the second a recovery time, and neither predicts the other. Chapter 8 tells the incident in full.

A recovery point and a recovery time are different quantities on different clocks. A team that has measured the first has not measured the second.

4.1 Two numbers, two clocks §

NIST SP 800-34 Rev. 1 gives both definitions [15][s]. The recovery point objective (RPO) is the point in time, before a disruption or outage, to which mission and business process data can be recovered, given the most recent backup copy. The recovery time objective (RTO) is the maximum time a system resource can remain unavailable before there is an unacceptable impact[s]. NIST also defines a maximum tolerable downtime (MTD), and says that unlike RTO, RPO is not considered part of it[s]. So one number is a moment in the data's history and the other is a stretch of downtime. We express the RPO as that moment's age at the time of failure, so the two can sit side by side in minutes[o]. Chapter 1 used both terms for gate 4. This is where they get measured.

The SRE book chapter on data integrity uses neither acronym. It asks "How much recent data can you afford to lose during a recovery effort?" and how quickly you need to recover [3][s]. We use the NIST terms.

A declared RPO and RTO are hypotheses. A restore drill is the measurement.

The two numbers can sit far apart. A nightly dump of a small database has an RPO near a day (derived below) and, on our laptop, a machine-side restore of seconds[M]. Atlassian's two figures are the reverse: at most five minutes of loss for any customer, and four to five elapsed days to hand a site back [7], [8][s]. Figure 4.1 shows where each number comes from and how each one is timed.

4.2 Deriving the declared numbers §

Start from loss, not from tooling. NIST ties the RPO to how much data loss the business process can tolerate [15][s]. Sort your data by what losing it costs: rebuildable from another source, re-enterable by hand, gone for good. The strictest class you cannot rebuild sets the RPO, written as a duration of writes, for example "no more than 15 minutes of orders"[o]. One RPO for a whole stack hides that logs and orders have different answers.

The achieved RPO is the age of the newest restorable point, not the backup interval. A point is restorable only once it has finished, uploaded and, if you check, verified[o]. Take a dump that starts at 02:00 and finishes uploading at 02:40. Until 02:40 the next day the newest restorable point is the one from 02:00 the day before, so just before 02:40 it is 24 hours 40 minutes old[p]. The worst-case RPO is the interval plus the delay before a point becomes restorable. These times are an example, not a measurement.

A log-based path changes the arithmetic. The PostgreSQL 18 documentation (accessed 2026-11-11) says continuous archiving can restore the database to its state at any time since the base backup, and that this needs a continuous sequence of archived WAL files extending back at least to the start of that backup [16][s]. It follows that recovery cannot pass a gap in the sequence, so the RPO of this path is the age of the newest archived file in the unbroken run[p].

For the RTO, start from the longest outage the owner will accept. NIST counts RTO as part of MTD [15][s], so we work backwards: the declared RTO is what remains of the tolerable downtime after you subtract the parts you cannot shorten[o]. The clock starts at the failure, not at the restore command[o]. Detection, decision and access

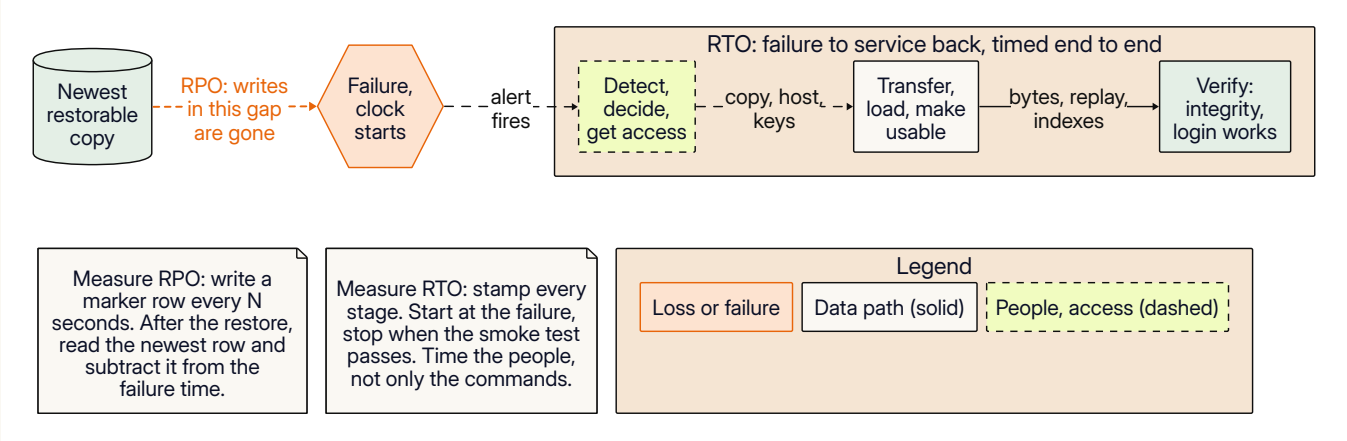


Figure 4.1. Two clocks on one incident. The RPO is the age of the newest restorable copy at the moment of failure. The RTO runs from the failure to a working service, and its first stage is people, not bytes.

are on it. Seven stages sit on the RTO clock. Four of them are on it whether or not anyone times them: detection (failure until a person knows, through the alert path and who is paged), decision (which copy, which host, which runbook), access (keys, credentials and a clean host to restore onto) and transfer (size divided by sustained throughput). Our drills timed none of those four, and transfer was not a factor because everything ran on a local SSD with no network. They timed the last three, shown below (the later of two runs).

Stage	What is on the clock	What our drills timed[M]
Load	Copy or replay into the engine	500 MB of files: 1.4 to 3.0 s across three tools. 322.7 MB database: 5.87 s logical, 4.31 s copy for a physical backup taken idle
Make usable	WAL replay, checkpoint, indexes, planner statistics, roles, grants	27% of a logical restore was not rows. 69% of copy plus start was not the copy for a backup taken under load. A login needed four fix rounds
Verify	Integrity check, then an application login	Login probe 0.11 s once roles existed. Diagnosis time not timed

Krebs on Security reports experts saying that the biggest reason victims or their insurers still pay, even with reliable backups, is that nobody at the victim organization tested in advance how long restoring would take [17][S]. It also reports Coveware's chief executive saying that most companies that pay either do not have properly configured backups or have not tested whether they can recover, and, on restoring remote backups over a slow link, that "someone pulls out a calculator and realizes it's going to take 69 years" [17][S]. That is expert opinion and one illustrative anecdote in a news article, not data. The arithmetic itself is short: transfer time is size divided by sustained throughput. Take 2 TB (decimal), which is 16 trillion bits, over a link that sustains 100 million bits per second. That is 160,000 seconds, a little over 44 hours[D]. The inputs are an example, and our drills never crossed a network.

4.3 What our drills measured §

METHOD AND LIMITS

Every figure tagged M comes from these runs. They ran on one Apple M4 Max laptop (36 GiB RAM, internal SSD) with restic 0.19.1, borg 1.4.5 and PostgreSQL 14.18 [1]. The data was synthetic, the page cache warm, nothing crossed a network, and every timing is the median of three repetitions. The whole suite ran twice, five minutes apart, and the figures here are from the later run unless stated. Timings differed between the two runs by 1.0 to 9.6 times, while all 69 exit codes in the corruption, fresh-host and empty-backup tests were identical, so read timings as orders of magnitude and exit codes as results. The laptop was on battery with a nearly full disk and in ordinary use, which affects timings. The time a person spends reading each error is in none of these runs, and we expect it to dominate, and the harness has no pager, so alert time is not measured either. The experiments were not pre-registered, and the scripts and raw output are published.

A restore at one size does not predict another. We restored a 50 MB file set and a 500 MB file set (54.7 and 526.6 MB exact). About ten times the data took 3.3 to 6.9 times as long: restic 0.92 s to 3.03 s, borg 0.41 s to 2.84 s, tar with zstd 0.39 s to 1.44 s[M]. In the earlier run the same ratios were 1.5, 3.1 and 2.1, and an idle physical copy went from 0.45 s to 4.31 s. A drill therefore gives you a range. Record the median and the worst of several runs, and declare against the worst[O].

Loading rows is most of a logical restore, but not all of it. A logical PostgreSQL restore of a 322.7 MB database (2,000,000 rows) took 5.87 s with `pg_restore` (range 2.45 to 13.77) in one set of runs. A separate set timed it by section: loading rows 4.44 s, building primary keys, indexes and constraints 1.44 s, `ANALYZE` 0.16 s (which `pg_restore` does not do for you) and the pre-data section 0.03 s, 6.07 s in total. Everything except loading rows is 1.63 s, about 27% of that total[M]. The dump compressed 57 to 1, so decompression time is understated for real data.

A physical restore of the same cluster (`pg_basebackup`, copy into place, start, first query) took 4.96 s for a backup taken while idle: 4.31 s copying and 0.65 s starting. A backup taken while writes were running needed 235 MB of WAL replayed and an end-of-recovery checkpoint before the first query. Startup took 3.69 s at the median and 41 s in the worst repetition, and its advantage over the logical path fell within the noise[M].

The result that matters most for the RTO took the least machine time. A restore of a one-database `pg_dump -Fc` onto a fresh cluster took 0.21 s, and the application still could not log in until four rounds of fixes were done. The machine time for those fixes was 0.43 s, and the person doing them is the cost that does not show here. Chapter 5 has the account[M].

4.4 Recording the result §

A declared number is a sentence. A measured number is a row in a drill report. For the RPO, put a marker row in the data: a scheduled job inserts a timestamp every N seconds, and after the drill restore you subtract the newest surviving row from the moment of failure.

```
insert into rpo_marker (ts) values (now());
select :'t_fail'::timestamptz - max(ts) as
measured_rpo from rpo_marker;
```

The resolution is N seconds and it covers only the database it lives in. Files and object stores need their own marker, such as a small file written on the same schedule[O]. For the RTO, stamp the failure instant, each stage, and the moment a scripted smoke test passes. A harness can time the machine stages (chapter 7 builds one). The human stages need a game day with a stopwatch and an operator who did not write the runbook[O].

Report three things per drill: the median, the worst of at least three runs, and the date. A line looks like this: declared RTO 60 min, measured median 95 min, worst 140 min (an example, not a result). When measured exceeds declared, the declared number was wrong. You can fix the system or change the declared number, and both are legitimate. Leaving the old sentence on the wiki is not.

HOW LONG DID THAT RESTORE TAKE?

Time one restore this week and write the minutes next to your declared RTO. If you have no declared RTO, write "none declared" next to the measured number.

CHAPTER 5

What gets backed up §

The question: what does a one-database dump leave behind, and which layers of a service did we not test at all?

In the model of chapter 1: what a restore at gate 3 has to cover.

Chapter 4 timed the machine. This chapter is about the thing the machine did not do: bring back everything the application needed.

The job was one line: `pg_dump -Fc appdb`. It produced a 1.4 MB archive. We restored the file onto a fresh PostgreSQL 14.18 cluster and `pg_restore` printed 12 errors, every one of them a role that did not exist. The rows were all there. The application could not log in. Getting a working login took four rounds of fix and retry, and when it worked the database was still owned by the wrong role, had the wrong collation and was missing two cluster settings [1][M].

```
$ createdb appdb && pg_restore -d appdb appdb.dump
pg_restore: while PROCESSING TOC:
pg_restore: from TOC entry 7; 2615 16508 SCHEMA app
app_owner
pg_restore: error: could not execute query: ERROR:
role "app_owner" does not exist
Command was: ALTER SCHEMA app OWNER TO app_owner;
[...]
pg_restore: warning: errors ignored on restore: 12
```

The job did what we told it to. The gap was between what we named and what the application needed.

5.1 A stack of five layers §

A service is a stack of layers: configuration and secrets, an object store, a database, volumes of files, and the host underneath. A backup of one layer says nothing about the others. What counts as consistent also changes from layer to layer. We use the word to mean that the restored copy is usable without a person repairing it first, and that is our own vocabulary, not a standard's[O].

1. Configuration and secrets: complete and current. Everything needed to recreate the service from text, including the values nobody wrote down.
2. Object store: every object version the database points to still exists and is the version the database expects.
3. Database: the data files agree with the log that goes with them, or the export was read from one snapshot.
4. Volumes and file trees: files that belong together were read together, and the tree you read was the tree you meant.
5. Host: the operating system and packages the rest depends on can be installed again at versions the rest accepts.

For each layer, write down how it is captured and what that method cannot capture. The rest of this chapter does that for the layers we could measure, and the closing table numbers its rows to match this list.

5.2 Databases: two captures, two meanings §

A logical export with `pg_dump` makes consistent exports even while the database is in use and does not block other users. The same page says that except in simple cases it is generally not the right choice for regular production backups [14][S]. A physical base backup plus archived write-ahead log (WAL) supports point-in-time recovery: you can restore the database to its state at any time since the base backup was taken. Recovery needs an unbroken sequence of archived WAL files reaching back at least to the start of the backup, and the base backup need not be perfectly consistent, because log replay corrects it. `pg_dump` output cannot be used as part of that

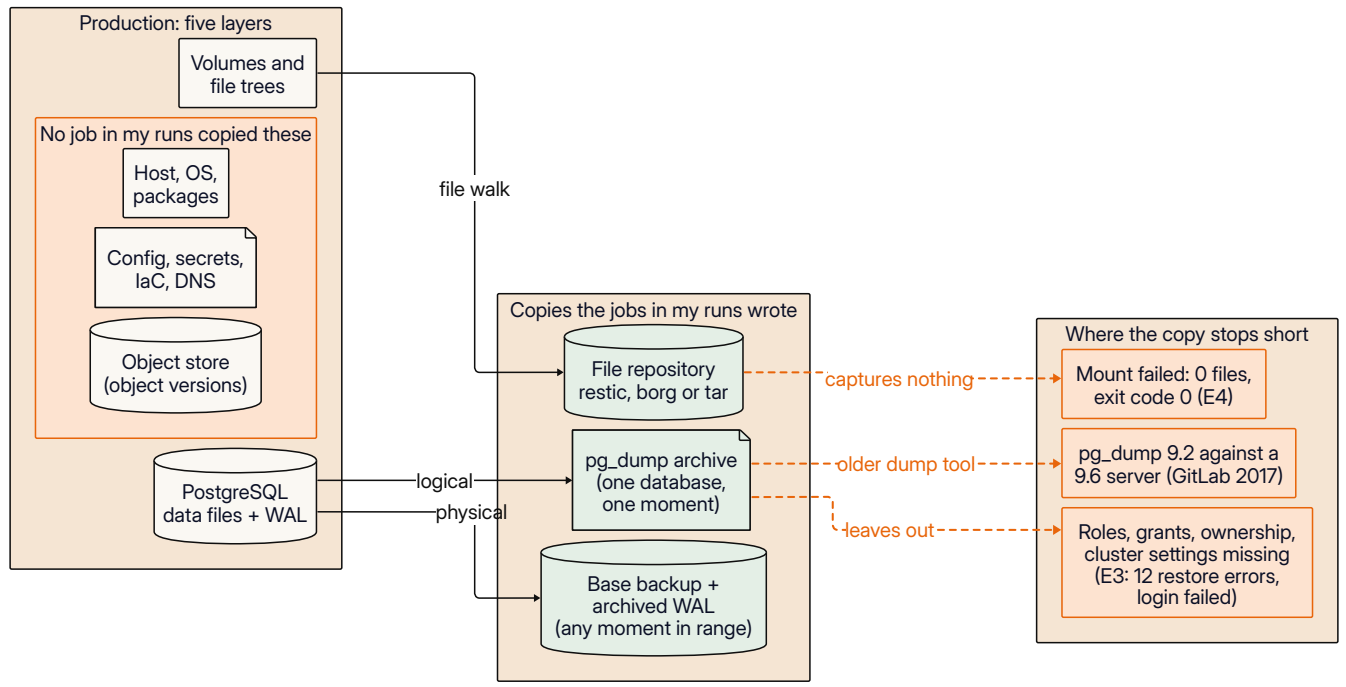


Figure 5.1. Five layers, the copies the jobs in our runs wrote from them, and where each copy stops short. Orange dashed paths are failures we measured, plus one published incident. Three layers had no copy in any of our jobs.

scheme [16][s]. The documentation adds that the archive command must return a zero exit status if and only if it succeeds, and that you should set up and test archiving before taking the first base backup [16][s]. These are the PostgreSQL 18 documents, read on 2026-11-11. Our own runs used 14.18.

Tools wrap the physical path. pgBackRest 2.60.0 offers full, differential and incremental backups, and its user guide says backing up a running cluster requires WAL archiving to be enabled [18][s].

So the choice is also a choice of recovery point. A dump gives you the moment it ran. Base backup plus WAL gives you any moment inside the archived range[o]. GitLab's 2017 postmortem describes a setup without a physical path, with no WAL archiving and no Azure disk snapshots for the database servers [2][s]. Version skew between the dump tool and the server is a failure domain of its own, and chapter 3 covers it.

The two paths also spend a restore differently. A logical restore rebuilds keys and indexes and leaves planner statistics to a separate `ANALYZE`, because `pg_restore` does not restore them. A physical backup taken under write load replays WAL before the first query, 235 MB of it in our run [1][M]. Chapter 4 has the timings.

5.3 What a one-database dump leaves behind §

The source cluster in our fresh-host test had six roles, a schema owned by a non-superuser, grants and default privileges, a database with a non-default collation and its own time zone setting, two extensions, a per-role setting, and two cluster settings made with `ALTER SYSTEM`. The backup job ran `pg_dump -Fc appdb` and nothing else. We restored on a fresh cluster of the same PostgreSQL build, two ways, compared below. A third path, `pg_restore -C` without the globals, recreated the database with its collation but hit the same three missing roles (13 errors) [1][M]. The wrong-database test of chapter 1 used a separate small pgbench database, also named `appdb` on its own scratch cluster, whose intended dump was 285,501 bytes. The six-role database here dumps to 1.4 MB.

What the application needs	`createdb`, then `pg_restore`	Globals first, then `pg_restore -C`
Roles	Missing, 12 errors ignored on restore	Present
Ownership and grants	Everything owned by the restoring superuser, grants gone	Present
Default privileges and role setting	Missing	Present
Database collation	Default one, `C.UTF-8` instead of `en_US.UTF-8`	Present
`ALTER SYSTEM` settings	Missing	Missing
First login probe	Failed, four fix rounds	Passed first time

The missing file was 1,482 bytes. `pg_dumpall --globals-only` produced it in 50 lines, and the job never ran it. The two cluster settings were still missing after the globals-first path, so even the better path left something behind[M].

Machine time was 0.21 s for the first restore plus 0.43 s of scripted fixes, against 0.23 s for the globals-first path. That is only the machine's share: every fix round in real life includes a person reading the error, finding the cause and writing the fix, and that time is not in these numbers (chapter 4). Scope: both hosts ran the same PostgreSQL build, so missing extension packages were not tested; we wrote the fix loop ourselves, so a person would take different steps; and we tested no other database engine.

5.4 Files, volumes and object stores §

File tools copied bytes exactly. In our runs restic 0.19.1, borg 1.4.5 and `tar` piped into `zstd` restored a 500 MB mixed dataset byte for byte, three of three runs per tool, checked with `diff -rq` [1][M]. That checks the copy. It says nothing about the selection.

So we simulated a failed mount: each tool was pointed at a mount point that was an empty directory. All three exited 0 and saved nothing (chapter 1 has the table). Then came retention. The restic repository held one good snapshot and one empty one, and a third backup ran against the same empty mount. `restic forget --keep-last 2 --prune` exited 0, kept the two empty snapshots and pruned the good one: the repository fell from 28.5 MB to 9,237 bytes, and `restic restore latest` returned 0 of 1,030 files [1][M]. A canary file inside the mount and a file count compared with the previous run caught the empty mount in our scripts.

An object store has a different unit, the object version. S3 Object Lock, for example, works only in buckets that have versioning enabled and protects only the specific object version it is applied to [13][s]. We ran no object store backend and did not test snapshots of a running database, so this paper makes no claim about either. Still, a row can point at an object the restore did not bring back, and an object can exist that no row points to. Decide which layer is captured first and which is restored first, and write it down[o].

5.5 Configuration, secrets, IaC and DNS §

Configuration, secrets, infrastructure as code (IaC) and DNS are layers we did not measure at all. They are the ones we expect teams to forget[o]. Our fresh-host run shows a small version of it: the configuration that made the database usable lived in places the dump did not reach. UniSuper and Google Cloud attribute the 2024 deletion of UniSuper's Private Cloud subscription to an inadvertent misconfiguration during provisioning [4][s]. Google's own post says the recovery included restoring network and security configurations and applications as well as data [5][s], so recovery there covered more than data.

One secret needs its own line. If the backup repository is encrypted, as restic's default and our borg `repokey` runs were, the repository key is an input to every restore. Whether it survives the loss of the host it protects is something to write down[o].

For each layer, the runbook gets a record: how it is captured and how often, what that method cannot capture, and what has to exist before it can be restored. Our fresh-host run supplies one hard ordering: roles come before the data that refers to them[M]. Two of the five layers had a job and a check in our runs: the database (layer 3) and files and volumes (layer 4). Configuration and secrets, the object store and the host had none, so this paper makes no claim about them. Configuration and secrets are the ones we expect teams to forget[o]. The record for all three is yours to fill.

> WHAT WAS NOT IN THE BACKUP?
> Restore your configuration and secrets to a clean host without touching production. Write down everything you needed that was not in the backup. That list is your forgotten layer.

CHAPTER 6

Credential compromise and immutable copies §

The question: how do you make a copy that an attacker holding your credentials cannot delete, and what does that copy still not protect against?

In the model of chapter 1: gate 1 under attack.

Chapter 5 listed what a restore has to cover. Chapter 6 asks who can destroy it, starting with whoever holds your keys.

Start with Code Spaces, whose damage list is short. The Register reported on 18 June 2014 that an attacker had gained access to the company's EC2 control panel, and that the company said he had removed EBS snapshots, S3 buckets, all AMIs, some EBS instances and several machine instances. In the company's words, most of its data, backups, machine configurations and offsite backups were "either partially or completely deleted", and it could not operate beyond that point [9]ˢ.

The report does not say how access was gained or how accounts were laid out, so we use it only for what the company's statement supports: one control panel reached production data, snapshots and, by the company's account, the backups.

6.1 One copy with two names §

A backup protects you against an attacker only if the attacker's credentials cannot delete it. That is a property of identities, not of disks or data centers. Make two lists: every identity that can write to production, and every identity that can delete or overwrite your newest backup. If one name is on both, you have one copy stored in two places, and a single stolen key removes bothᵒ.

Ransomware shows the same thing from the other side. Anything production can write to, it can overwrite. A backup target mounted or credentialed from the production host can be encrypted or deleted by the same attackerᵒ. Replication copies the damage as it copies the dataᵒ.

Three controls help, and each moves the delete permission somewhere production cannot reach. One is a credential that can add but not remove. One is a store that refuses removal for a set time. One is a direction of connection in which production holds no key to the backup at all. The figure below puts them in one design. It is an argument, not a reconstruction of Code Spaces, whose account layout the report does not give. "Zone A" in it is the production credential domain and "zone B" is the backup credential domain.

6.2 Control 1: add without remove §

The first control is a credential that can create new backups but cannot remove or change old ones. The restic rest-server README (accessed 2026-11-11) describes an append-only mode that allows creation of new backups but prevents deletion and modification of existing ones, and suggests it for systems that have a potential of being hacked [19]ˢ.

Borg's append-only mode is weaker than it sounds. Per the Borg 1.4.5 documentation (accessed 2026-11-11), append-only means Borg will never overwrite or delete committed data, but that restricts only the low-level repository structure. So `borg delete` and `borg prune` are still allowed, though they do not free disk space. A transaction log lets you roll back to an earlier state, provided `borg compact` has not been run since then, and the documentation warns that a later write that is not append-only permanently removes objects previously marked deleted [20]ˢ. By that warning, the rollback window closes once `borg compact` runs or a later write is not append-only, whoever runs it.

An append-only flag the client can switch off protects nothing, so the server or the store must enforce it, outside anything the production host holdsᵒ. And someone still has to run retention. We read no documented retention procedure for the restic setup and tested none for

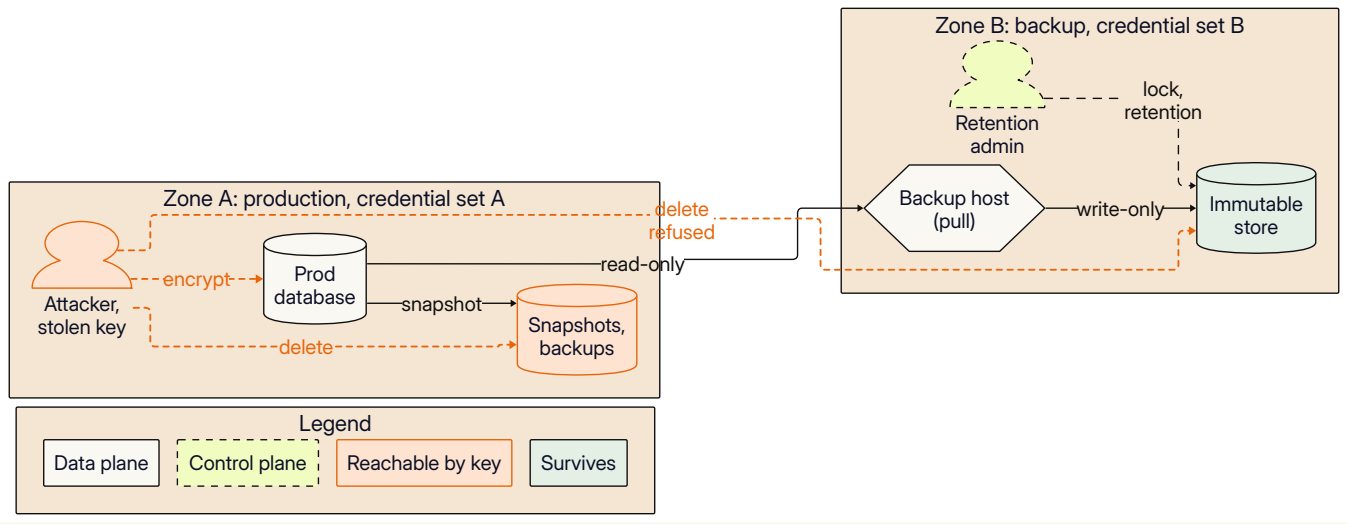


Figure 6.1. The stolen key reaches everything that zone A's identities can delete, snapshots and same-zone backups included. Zone B shares no credential with it, so the delete is refused. The Register reports snapshots, S3 buckets and AMIs removed and the company saying its backups and offsite backups were partly or wholly deleted, but not how its accounts were laid out, so zone B is a design, not a finding.

Mechanism	Refuses	Still gets through
restic rest-server, append-only	Deleting or modifying existing backups	The README states no caveats; retention from a separate admin path was not checked
borg 1.4.5, append-only	Overwriting or deleting committed data	`borg delete` and `borg prune` still run and free no space; rollback only before `borg compact`
S3 Object Lock, governance	Deletion by principals without special permission	A principal with `s3:BypassGovernanceRetention` and the bypass header, which the S3 console sends by default
S3 Object Lock, compliance	Overwrite or delete by any user, root included; shortening retention	Deleting the AWS account, the only early path AWS documents
B2 Object Lock, compliance	Removing the lock, by any user	Clients with the right app key capabilities can extend retention dates
B2 Object Lock, governance	Not stated in the pages we read	Clients with the appropriate app key capabilities can modify or override the lock

Table 6.1. What each mechanism refuses and what still passes, as documented, accessed 2026–11–11 [13], [19], [20], [21][s]. Not tested by us.

either tool, so make it your first experiment on a scratch repository.

6.3 Control 2: a store that refuses §

The second control moves the refusal into the storage service. Amazon S3 Object Lock (AWS S3 User Guide, accessed 2026-11-11) uses a write-once-read-many (WORM) model, works only in buckets that have S3 Versioning enabled, and protects only the specific object version it is applied to [13][s]. The page says a simple delete with no version ID returns 200 OK and adds a delete marker on top, while a permanent delete of a locked version, with its version ID, returns 403.

Against a stolen credential, the lock is worth whatever its mode allows.

Governance mode is a guard rail, not a wall: a stolen credential that carries the bypass permission and sends the bypass header removes the lock. Compliance mode closes that door, with one exception that shapes the design. AWS documents that the only way to delete a compliance-locked object before its retention date is to delete the associated AWS account [13][s]. So an immutable copy in production's own account is exposed to account closure by whoever controls that account. Put the locked copy in a different account, with credentials production never sees[o]. The page describes no account-closure safeguards, so we say nothing about how hard closure is.

A lock also expires. After the retain-until date passes, the version can be overwritten or deleted [13][s]. A lock shorter than your time to notice an intrusion protects nothing, so retention has to outlast detection[o]. Ransomware adds encryption to deletion: a backup written after the encryption holds ciphertext, so what helps is a version older than the attacker's arrival, kept for longer than it takes you to notice[o].

> IMMUTABLE IS NOT THE SAME AS GOOD
> A lock keeps whatever you wrote. In one scripted run with restic 0.19.1 and borg 1.4.5, a backup of a mount point left as an empty directory, as after a failed mount, exited 0 for both tools and saved a snapshot (restic) or archive (borg) with 0 files, where the mounted run had saved 1,030 files[M]. An object lock would have preserved that empty snapshot for its whole retention period[o]. Data that ransomware encrypted before the next scheduled run gets the same treatment: the pipeline stores the ciphertext without complaint[o]. Keep several restore points, for longer than detection takes.

6.4 Control 3: pull §

The third control reverses who connects to whom. In a push design, the production host holds a credential for the backup store, so a compromised host holds it too. In a pull design, a backup host in the other zone reaches into production with a read-only credential, and production holds nothing that points at the backup[o]. The backup host now holds a read key to production and becomes a target itself, so give it the narrowest read permission you can and no route in from the internet.

6.5 The retention identity §

The one identity that legitimately deletes backups is the retention job. It is a deleter with a key that can remove every backup, and an attacker needs only that key. We hit the plain version of this with no attacker involved: after a third, empty backup, `restic forget --keep-last 2 --prune` exited 0, left two empty snapshots and a repository of 9,237 bytes, and `restic restore latest` returned 0 files (chapter 1) [1][M]. That was one machine, one tool version and one scripted scenario, so it shows a mechanism and not a frequency. A retention job that cannot tell a good backup from an empty one will delete the last good copy on schedule, and a stolen retention credential does the same thing on purpose[o]. Give retention its own identity, in zone B, and keep that identity off every host that deploys or runs production.

6.6 Who can delete the newest backup? §

List every credential that can delete or overwrite your newest backup, then mark which of them can also write or deploy production. The example below is invented, a single-server stack, not data from an incident.

Credential	Writes prod	Deletes backup	Verdict
Deploy token	yes	no	Fine, no backup key
Prod host role	yes	yes	Finding: two names
Backup job (push)	no	add only	Fine, store enforced
Retention job	no	yes	Finding, unless zone B only
Account owner	yes	yes	Finding: move the copy

A lock keeps a copy, and it says nothing about how long you will wait to get it back. Chapters 4 and 8 cover that wait.

> WHO CAN DELETE THE NEWEST BACKUP?
> Make the two lists for your own stack. Any row with two yes answers means one name is on both lists. Fix the identity before you buy more storage: a third copy that shares the delete permission with the first two adds capacity and no protection.

CHAPTER 7

Verification, silent failure and restore drills §

The question: how do you know the bits you restore are the bits you wrote, and how do you make that check run unattended and speak up when it fails?

In the model of chapter 1: gates 2 and 3 as drill levels.

A lock keeps whatever you wrote, including an empty snapshot. Chapter 1 said a green job proves, at most, that a file was written. This chapter covers the checks that would have gone red and the ones that stayed green when they should not have. The wrong-database dump is the clean example: `pg_dump -Fc postgres` exited 0 and wrote 1,030 bytes where the database we meant to dump produces 285,501, and restoring it also exited 0 and left zero tables in the public schema [1][M]. Each tool answered whether the process finished, and the question was whether the data survived. Every M figure in this chapter comes from the runs described in chapter 4.

7.1 Exit 0 describes the process, not the data §

A success code means a program finished without hitting an error it knew how to detect. Chapter 1 lists four jobs that exit 0 and leave nothing restorable. For each we wrote the cheapest check we could think of.

Job	Check	Result
`pg_dump`, wrong database	Expected table data in the archive listing. A size floor.	Both passed on the intended dump and failed with exit 1 on the wrong one
`pg_dump` piped into `gzip`	`set -o pipefail`	The failing pipeline exited 1. No good-side run recorded
Empty mount point	A canary file on the mount. File count against the previous run.	Both failed with exit 1 on the empty mount. No good-side run recorded
Retention after empty backups	The file-count check, run before retention	Not tested

Two checks were run against both a good and a bad artifact and the others only against the bad one, so the false-alarm rate of those is untested [1][M]. The thresholds are parameters, with defaults of 100,000 bytes and a file ratio of 0.5 in our script. They suit our test data and nobody else's. Set yours from your own last good run. The last row is the dangerous one: retention turns a few bad nights into a lost archive and reports success while doing it[O].

GitLab's 2017 postmortem has the same shape: the cron emails were rejected by the receiver, so no signal reached a person [2][S].

7.2 Three levels of verified §

Trust a backup only as far as the last drill that restored it[O]. The SRE book puts the same point this way: "you only know that you can recover your recent state if you actually do so" [3][S]. Verification has three levels, each costlier than the last and closer to the real question.

Level 1, the repository reads back. The stored bytes are the bytes the tool wrote. It can notice damage in parts of the repository you are not about to restore. It cannot tell you the backup contained the right thing. The 1,030-byte dump is an intact file.

Level 2, the data restores. Restore onto a scratch host and compare the result with something you know independently: file counts, row counts, a size floor, a canary. A restore compared against known counts shows the wrong database (0 tables) and the empty mount (0 of 1,030 files)[M].

Level 3, the application works. Log in as the application's own database user, read, write, and run the query the product depends on. It catches what a data restore cannot, as the fresh-host test in chapter 5 shows.

The levels are the checks a drill runs, and they map onto the gates of chapter 1. Level 1 is evidence for gate 2, Levels 2 and 3 are evidence for gate 3, and the stopwatch around them is gate 4[O]. The size floor and the canary are the same checks as in the table above, applied to the restored data at Level 2 instead of to the artifact at gate 1.

7.3 What damage does to each level §

We took a clean backup, flipped a few bytes in one stored file, and asked each tool what it could see. The restic and borg runs flipped 16 bytes (8 in the restic header case) in one stored file of a repository holding a 50 MB dataset. The PostgreSQL runs flipped 4 bytes of valid text in one heap page, so the damage is semantic, not structural. We did not test truncated files, deleted packs, a damaged index or several flips at once.

Four results in that table matter for drills.

Backups kept succeeding against damaged repositories. A new restic or borg backup exited 0 even for a byte-identical copy of the data[M]. We did not test why, but our reading for restic is that a new snapshot deduplicates

Damage	Next backup or dump	Default check	Deep check	Restore
restic, middle of a data pack	exit 0 (also for a byte-identical copy)	`check` exit 0, "no errors were found"	`--read-data` exit 1	exit 1, 1 of 1,029 files differ or missing, and a file with wrong content still written
restic, pack header	exit 0	`check` exit 0	`--read-data` exit 1	exit 0, 0 of 1,029 files differ
borg, one segment	`create` exit 0 (also for a copy)	`check` exit 1	`--verify-data` exit 1, stopped at the repository stage	`extract` exit 2, 23 of 1,029 files extracted
PostgreSQL 14.18, checksums off	`pg_dump` exit 0, `pg_basebackup` exit 0	`pg_checksums` cannot run	none run	exit 0, 1 row changed in the restored database
PostgreSQL 14.18, checksums on	`pg_dump` exit 1 (partial 4,032-byte file left behind), `pg_basebackup` exit 1	`pg_checksums --check` exit 1	none run	`pg_restore` exit 1 (from the partial dump). On the damaged cluster, `count(*)` exit 0 and full-scan `sum` exit 1

Table 7.1. One flipped region per run, one laptop, restic 0.19.1, borg 1.4.5, PostgreSQL 14.18 [1][M]. Exit codes are the signal. Identical outcomes in both runs of the suite.

against the damaged blob without reading it, so the snapshot would rest on damaged data with no message°. The default `restic check` gave false comfort, and the docs say why: it does not verify pack files on disk. Of the two restic checks, only `--read-data` failed[M]. Reading the whole repository takes time that grows with its size, and we only timed it on a 50 MB repository (0.89 s). The subset option spreads the cost: `--read-data-subset=n/t` reads a fraction per run, so a rotation over t runs covers the whole repository [22][S]. We did not run one.

Level 2 passed while Level 1 failed. Same repository, two answers: with 8 bytes flipped in a restic pack header, the restore of the first snapshot exited 0 and every file was identical, while `restic check --read-data` exited 1[M]. A drill that only restores would have called that repository healthy. The restore and the deep check answer different questions, and the drill needs both°.

A smoke query can pass on a damaged table. With checksums on, `select count(*)` still succeeded because the primary-key index answered it, and only a query that scans the heap failed. With checksums off, nothing in the backup or restore path failed: `pg_dump`, `pg_basebackup` and `pg_restore` all exited 0 and the restored database held the wrong value[M]. Write the Level 3 query so it reads the data, and check which PostgreSQL release you are on before assuming checksums.

7.4 What each tool's check reads §

The damage results above come from tools that answer different questions when you say "check". The table below is what each one's documentation says, as read on 2026-11-11, and the thing to look for is what each check leaves out. The restic, borg and pgBackRest pages are the current stable ones and may describe newer versions than the restic 0.19.1 and borg 1.4.5 we ran. The PostgreSQL entries come from the PostgreSQL 18 documentation, not 14.18.

Check	Level	Documented behavior and what it leaves out
`restic check`	1	Verifies structural consistency and integrity (snapshots, trees and pack files). By default it does not verify that the pack files on disk are unmodified [22][S].
`restic check --read-data`	1	Also verifies pack file integrity, at the cost of reading all of them. `--read-data-subset` reads only part n/t, a random percentage or a random size per run [22][S].
`borg check`	1	Read-only by default. The repository part covers segment headers and the metadata and data of all objects. `--verify-data` is a complete cryptographic verification and "very time consuming", and cannot be combined with `--repository-only` [23][S].
`pgbackrest verify`	1	Determines whether the backups and archives in a repository are valid. With `--set` it verifies all database and archive files of that backup set [24][S]. We did not confirm from the pages we read what it checks beyond that.
`pgbackrest --checksum-page`	at backup	Validates data page checksums while backing up a cluster. Failures do not abort the backup. The list of invalid pages is stored in the backup manifest [25][S].
`pg_verifybackup`	1	Checks a `pg_basebackup` backup against its `backup_manifest`: missing and extra files, file checksums, and with `pg_waldump` that the WAL needed for recovery is present and parseable. WAL checks work only for plain-format backups and are version-specific. The docs say it cannot include every check a running server makes, and that you should still perform test restores [26][S].
PostgreSQL data checksums	on read	A per-page checksum, verified on every read, covering data pages only. Default in PostgreSQL 18, set per cluster, changed offline with `pg_checksums` [27][S]. Our 14.18 clusters had them off unless we asked.

7.5 Alerting on absence §

A missing success signal is a failure°. A drill that fails loudly is the easy case. The dangerous drill is the one that stops running, because a scheduler was disabled, a host was retired or the alert path broke. So the drill reports success by sending a heartbeat, and a dead-man switch pages the owner when the heartbeat is late. This is our design, not a quotation from any source. In the figure, both failure paths end at the same named person. The timer needs a threshold, derived from your drill schedule plus a grace period, and it needs to live outside the host that runs the drill°.

GitLab's rejected emails show the weakness of a failure signal that depends on someone receiving it: if the channel breaks, no signal arrives°. The same principle appears inside PostgreSQL. The PostgreSQL 18 documentation says an archive command must return a zero exit status if and only if it succeeds, and that you should set up and

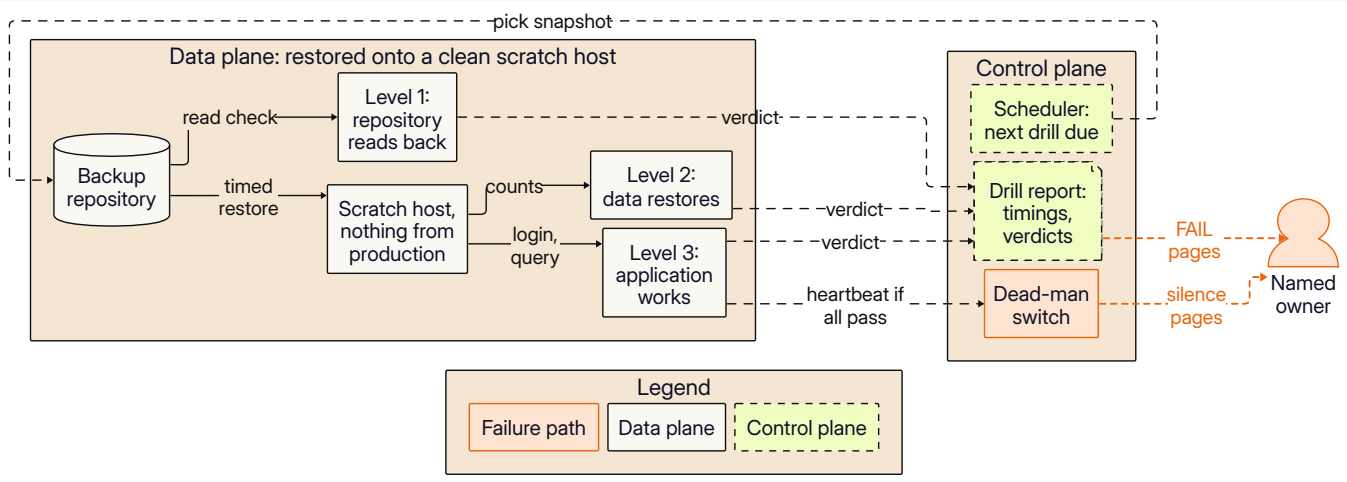


Figure 7.1. A restore drill pages its owner in two ways: a failed level, and a missing report. Only a full pass sends the heartbeat.

test WAL archiving before taking your first base backup [16][s]. A script that returns zero without archiving breaks that contract[o].

The SRE book says: “Continuously test the recovery process as part of your normal operations.” It also says to automate recovery tests whenever possible and then run them continuously. It credits regular dress rehearsals and dry runs for the February 2011 Gmail restore from GTape, the first large-scale use of that backup system to restore live customer data [3][s].

7.6 A drill you can run unattended §

A scheduler restores the newest snapshot and an old one onto a scratch host, runs Levels 2 and 3 against the restored copy and Level 1 against the repository, and writes a report. The newest tests recency and an old one tests whether older data still restores[o]. Appendix A specifies the inputs, steps and pass criteria.

WHY THE SCRATCH HOST MUST BE CLEAN

A drill on the production host can pass by accident, because everything the backup forgot is still sitting there[o]. Our fresh-host test shows how much that can hide. The dump restored with 12 ignored errors, a row-count comparison would have passed, and only the application login showed that the roles, grants and settings were gone (chapter 5) [1][m].

WHAT THE STOPWATCH MEASURES

Stop the clock at the first successful application query, not when the restore command returns. A physical backup taken under write load needed several times longer from start to first query than an idle one, because the server replayed WAL and ran an end-of-recovery checkpoint first, and in the logical path about 27% of the time went to something other than loading rows, with `ANALYZE` left to run by hand (chapter 4)[m]. Size does not extrapolate either, and the same suite run twice gave different times. Keep the range. One number hides it.

> **WHO HEARS WHEN THE DRILL DIES?**
> Force a failure on purpose: point the drill at an empty directory, or disable its scheduler. Time how long until a named human is told, and write the minutes next to your declared RTO.

CHAPTER 8

Restore at scale and under pressure §

The question: why can giving back one customer take so much longer than copying their bytes, and how do you choose the unit of restore before the bad day?

In the model of chapter 1: gate 4 at the size of one part.

Backups are written for the whole and restored for the part. A nightly job copies a database, a bucket or a volume as one stream. The bad day rarely wants all of it. It wants one customer's rows, one table somebody dropped, one bucket a script emptied. If the layout cannot hand back that part without restoring everything around it or sorting it out by hand, your real RTO is the slowest manual path between the backup and the part°.

Call the smallest part you promise to hand back on its own the *unit of restore*: a tenant (one customer of a shared system), a database, a table, a bucket prefix. Three jobs sit between the backup and a usable unit, and any of them can be the slow one. Locate it, meaning find where it lives inside the backup. Extract it, meaning get it out without the rest, or restore the rest somewhere safe and carve it out. Reconcile it, meaning give it back with what it depends on and check it works before its owner sees it. Extraction is the job that depends on bytes. Locating and reconciling depend on how many units there are and how many people must act°.

NIST SP 800-34 Rev. 1 says the recovery time objective defines the maximum amount of time that a system resource can remain unavailable before there is an unacceptable impact on other system resources, supported mission/business processes, and the maximum tolerable downtime [15]ˢ. The definition is per system resource, so nothing stops you declaring one per unit. A single RTO for the whole says nothing about the part°.

8.1 Case study: one store, many customers §

The interest in Atlassian's April 2022 incident is the mechanism, not the blame. Atlassian's account of the trigger is a script run with the wrong execution mode and the wrong list of IDs: the team supplied the IDs of entire cloud sites instead of the IDs of the app being deactivated [7]ˢ. Its post-incident review of 29 April 2022 counts 775 affected customers and 883 deleted sites [8]ˢ. It also says no customer lost more than five minutes of data [8]ˢ, and finishing the restore took about two weeks (derived below)ᴰ. The 13 April update had said approximately 400 customers, a mid-incident estimate that the review replaces [7]ˢ.

The storage layout then set the shape of the restore. Atlassian says each data store contains data from multiple customers, so it had to manually extract and restore individual pieces from backups, that it restored customers in batches of up to 60 tenants at a time, and that it takes between 4 and 5 elapsed days to hand a site back to a customer [7]ˢ. It also says it maintains immutable backups designed to be resilient against data corruption events [7]ˢ. The 60-tenant batch figure appears in the 13 April update, not in the review.

A small recovery point and a long recovery time fit in one incident.

The review says no customer lost more than five minutes of data [8]ˢ. The update says that as of 18 April 2022, 23:57 UTC, all impacted customers had been restored [7]ˢ. The two pages date the start differently: the update gives Monday 4 April 2022 in Pacific Time, the review gives Tuesday 5 April 2022 at 07:38 UTC. We derive the duration from the review's start only. From the review's start to the update's last restore is 13 days, 16 hours and 19 minutesᴰ, about two weeks. How often you copy sets the recovery point. What you must do to give a part back sets the recovery time°.

8.2 The part has a periphery §

The unit's dependencies sit outside the unit, so a backup of the unit alone is not a backup of what the unit needs. Our fresh-host run in chapter 5 is the small version: one database was backed up and restored, and the roles, grants and settings it depended on were not in the dump. That run covered PostgreSQL roles, grants and settings. We did not test configuration files, secrets or DNS [1]ᴹ.

8.3 The range is the number §

We timed whole datasets and never timed giving back one table, so this chapter has no figure for a part. Dividing a full-restore time by the number of tenants is not an estimate of one tenant's restore, and we have no measurement that says what the right estimate is°. Chapter 4 shows how little a whole-dataset timing transfers across sizes and across runs. In an incident, plan with the range, not the median°.

8.4 Restore under pressure is several jobs §

In its 24 May 2024 post about a customer's deleted Private Cloud, Google Cloud says the customer and Google teams worked 24x7 over several days to recover the Private Cloud, restore network and security configurations and applications, and recover data [5]ˢ. Data is the last item in that list. A restore under pressure is a list of jobs, and only one of them is a copy of data°.

Pressure also removes the chance to learn the path. Krebs on Security reports experts tying victims' payments to the fact that nobody tested in advance how long restoring would take (chapter 4) [17]ˢ, and the SRE book credits dry runs for a large restore that worked (chapter 7) [3]ˢ. None of the three sources gives a timing for a part restore, so measure yours before the incident.

8.5 Choose the unit on purpose §

Choosing a unit means deciding what it is and writing down what it needs. These are recommendations, not findings.

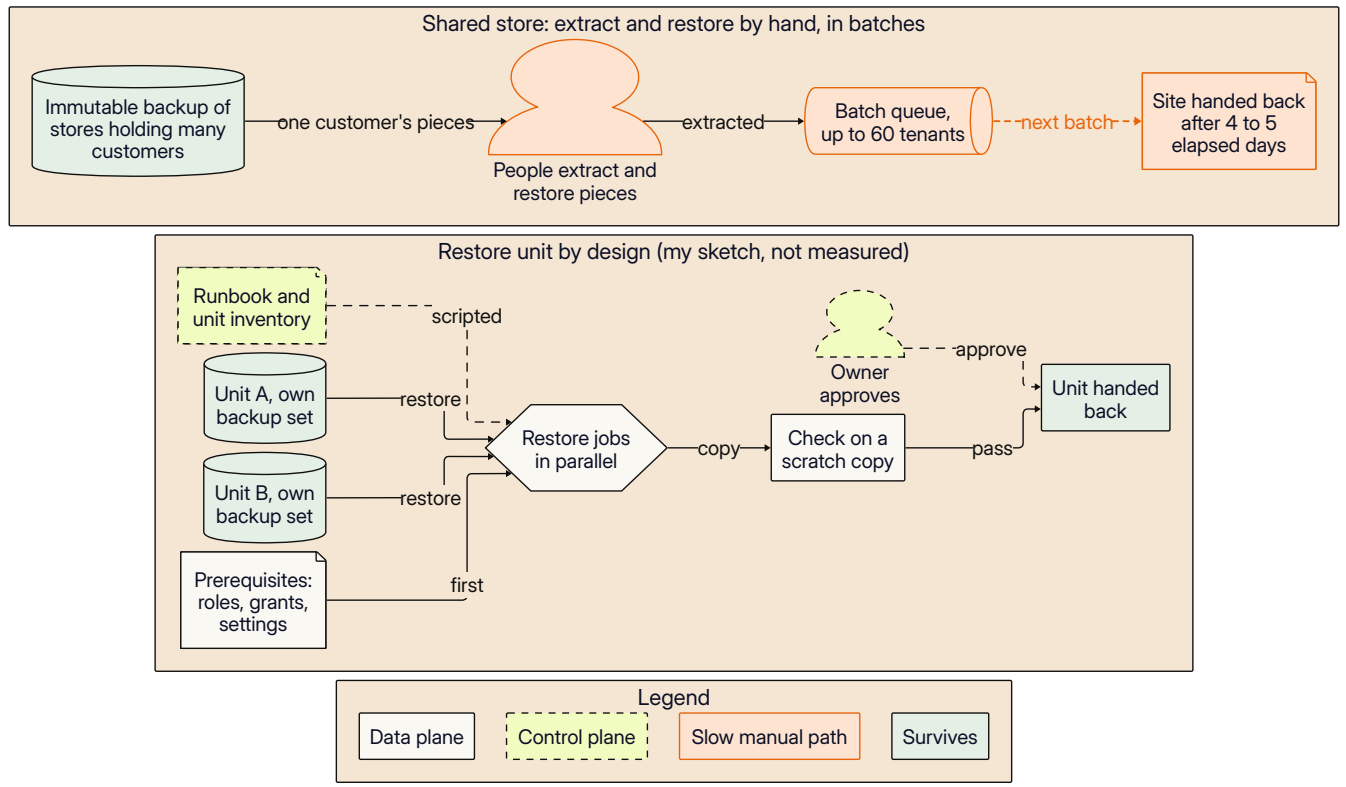


Figure 8.1. Same bytes, different unit of restore. Top: a shared store where extract and restore are hand work, fed through batches (figures from Atlassian's 13 April 2022 update). Bottom: our design sketch, where each unit has its own backup set, prerequisites restore first, jobs run in parallel and the owner approves handback. No numbers are claimed for the sketch.

Decide	Evidence to keep
What is the unit?	One sentence naming it: a tenant, a database, a bucket prefix. If you cannot name it, you cannot promise it back.
How do you find it?	The command that lists the unit inside the backup without restoring the rest, run once, with its output saved.
What does it need outside itself?	Roles, grants, settings, configuration, secrets, DNS, and the order to restore them in.
Where does it restore to?	A scratch target that is not production, and the name of the person who can create it at night.
Who checks it, who approves it?	Two named people, a smoke test, and the elapsed time.
How many at once?	The shared bottleneck: one repository, one person, one approval queue.

The cost of running a finer unit is out of scope here. Measure it on your stack before you split anything.

> **HOW LONG DOES ONE PART TAKE?**
> Pick one small tenant, table or bucket. With a stopwatch, restore only that from the backup onto a scratch host, to the point where its owner could use it. Write down the time, the number of manual steps and the number of people who had to act. Put it beside your full-restore number. If the part is not much faster than the whole, your unit is the whole.

CHAPTER 9

Runbook, ownership and the evidence checklist §

The question: who owns the backups, and what is the least evidence that owner should be able to show today?

In the model of chapter 1: all four gates, held by a person.

Someone has to own all of this, and that person needs evidence to show.

GitLab's postmortem of its January 2017 database outage contains a sentence that reads like a root cause: "Because there was no ownership, as a result nobody was responsible for testing this procedure" [2][s]. The same postmortem lists the gaps that chapter 1 walks through. We read the ownership sentence as the reason those gaps could stay open. A gap that nobody owns has no deadline[o].

9.1 Owned means paged §

A backup system is owned when one named person is paged by its silence and a drill report exists for the last quarter[o]. That is our rule, not a standard. The Google SRE book describes runbooks as playbook entries describing how to respond to a validation failure alert, and it splits ownership: a central infrastructure team maintains the validation framework, while product teams own the business logic of their validators. It does not state a rule that one named person must own backups [3][s]. We add the rule because GitLab's sentence shows what its absence looks like.

The rule needs three names. The owner is paged first and signs the drill reports. The deputy is paged when the owner does not answer, and is the person who proves the runbook works without its author. The approver can accept a gap between the declared and the measured recovery time, or pay to close it[o]. One person on holiday is the same as no owner, so a team of one still needs a second human on the page.

Evidence beats intent. "We back up nightly" is intent. A dated report with a snapshot id and a count of minutes is evidence. The exit code of the job is intent too. In our scripted cases four useless jobs all exited 0 (chapter 1) [1][M], so somebody has to own the checks that catch them.

9.2 The runbook is written from a drill §

A runbook is the page the owner opens when the incident is declared. We give it five verbs, and the one-page template is in Appendix E[o].

1. *Declare.* Who may call it, what triggers the call, and the two names.
2. *Decide.* Restore to which point and onto which host, and who approves overwriting production.
3. *Restore.* The exact commands in order, with tool versions, copied from a drill and not from memory.
4. *Verify.* The check that the data is right, which is a different thing from the command exiting 0.
5. *Communicate.* Who is told, by whom, and how often.

The third verb is where memory fails: restoring one database onto a clean cluster needed four rounds of fixes before an application login worked (chapter 5)[M].

9.3 Three questions, three artifacts §

Chapter 1 asked whether a system can be restored. Figure 9.1 asks it of a person. Each of its three questions has a yes that leaves an artifact on file and a no that leaves a named gap.

The left column applies chapter 4 to a person. If you cannot say how many minutes the last restore took, the RTO you declared is a guess. Write "assumed" beside it, and where there is no number at all, write "unknown". It is the gap that the experts in Krebs's article name (chapter 4).

The middle column is about the deputy: a runbook is evidence only after someone who did not write it has followed it, and each question they ask is a line missing from the page. The right column is the dead-man switch of chapter 7 seen from the owner's chair.

One number is not a report. A drill report records the date, the data size, the host load and the range over repeated runs, and we would declare an RTO from the slow end of that range, not the median[o].

9.4 Five incidents, one question §

Table 9.1 reads this paper's five incidents (OVHcloud counts as one row for two decisions) with the same question: what had nobody produced evidence for? The gate numbers are the four from chapter 1 (1 exists, 2 readable, 3 restorable, 4 fast enough) and the domains are those of chapter 3. The gate and control columns are our reading[o]. The facts in each row come from the cited sources[s].

9.5 The evidence checklist §

Appendix F is the printable page: ten rows an owner should be able to put on the table today. It makes no compliance claim[o]. One distinction does the work. A row is *measured* when someone ran it and kept the output, and *declared* when it was written down and never exercised. The rows, in order: owner and deputy named, a timed restore, declared against measured RPO and RTO, a restore run by the deputy, a forced failure, who can delete the newest backup, the per-copy table, config restored without production, one unit restored alone, and the verification level run.

> WHO IS THE DEPUTY?
> Write down the names of your owner and of the deputy who covers when they are away. Ask the deputy to restore something to a clean host using only the runbook. Time it and write down every question they asked. If you cannot name both people, that is the finding.

Every row needs a date and a name. Evidence expires: a new server, a new tool version, a new credential or a much larger dataset turns a measured row back into a declared one[o].

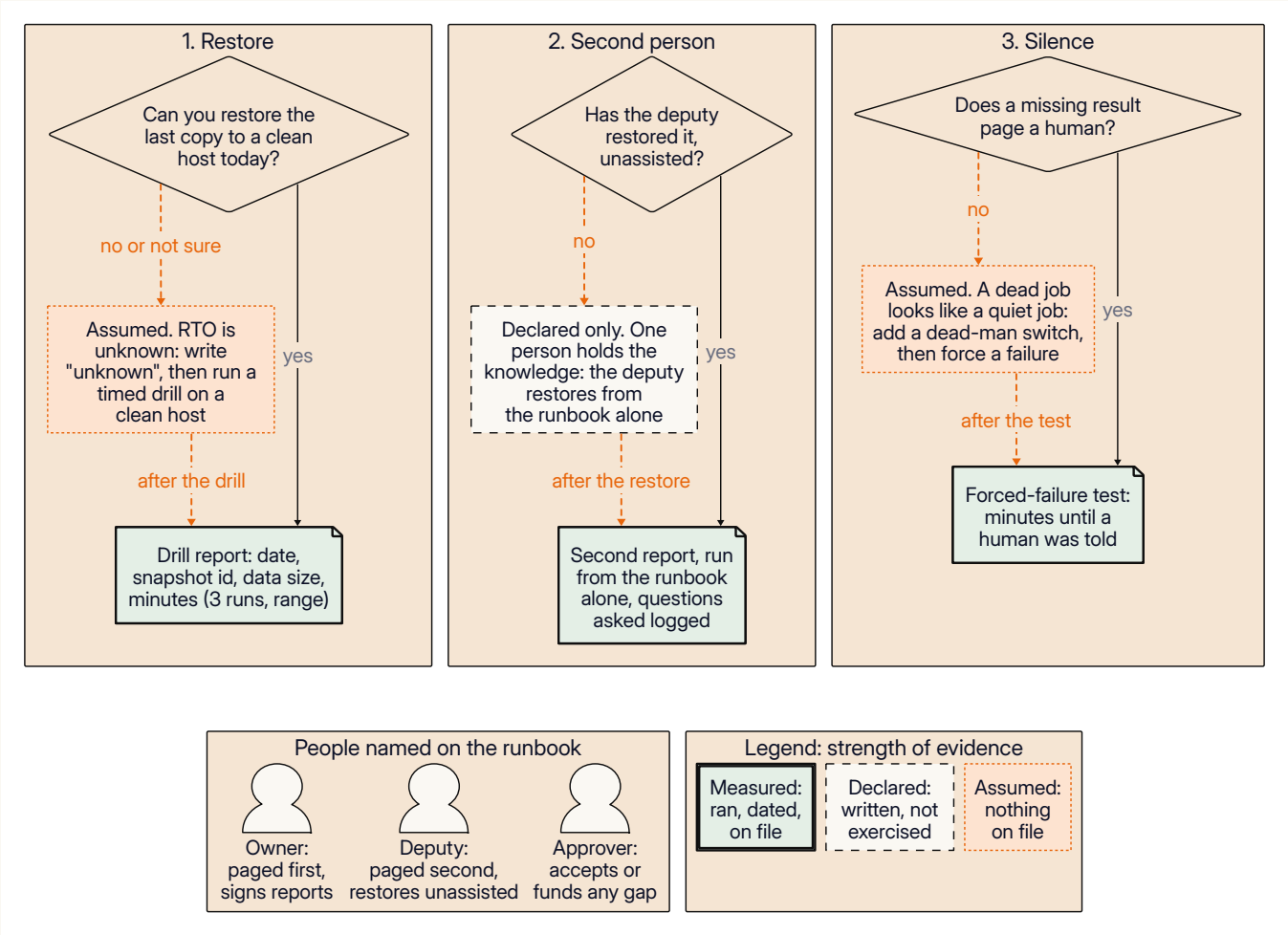


Figure 9.1. Three questions an owner can answer today. A yes is only worth what it leaves on file, and a no is a gap with a next action.

Incident	Gate not proven	Shared domain	Limiting control	Source and its limits
GitLab, Jan 2017	Gates 1 and 3: recovery used a copy of the staging database, and nobody was responsible for testing the procedure [2].	Software version: `pg_dump` 9.2 against PostgreSQL 9.6.	A named owner of the restore test, and failure emails that reach a person.	First-party postmortem. Loss figures are GitLab's estimate.
Code Spaces, Jun 2014	Gate 1: the company said most of its data, backups, machine configurations and offsite backups were partially or completely deleted [9].	Credentials (chapter 3's reading): an attacker reached the EC2 control panel, and the company said he removed EBS snapshots and S3 buckets. How access was gained is not stated [9].	A copy that the production credentials cannot delete (chapter 6).	Press report of the company's statement. Entry method, MFA and account layout are not stated.
OVHcloud, cases reported Mar 2023	Gate 1: awards of EUR 153,837 to Bluepad and EUR 101,102 to Bati Courtage over lost backup data are reported [6].	Facility. Bluepad: production in SBG1, backup in SBG2. Bati Courtage: the backup was in the same building as the SBG2 VPS, though the contract called it physically isolated.	Check where the copy sits instead of reading what the contract says (chapter 3).	Trade press, 23 March 2023. OVH had appealed one decision and reportedly intended to appeal the other, as of that date.
Atlassian, Apr 2022	Gate 4: restores ran in batches of up to 60 tenants, 4 to 5 elapsed days to hand a site back. All customers restored as of 18 April 2022, 23:57 UTC [7].	None shown. Atlassian says it maintains immutable backups. Each data store held data from several customers, so pieces had to be extracted and restored manually [7].	A tested path for restoring one tenant, not only a whole store (chapter 8).	First-party update and review. 775 customers per the review, about 400 in the earlier estimate [8].
UniSuper, May 2024	Gate 1: the copies sat inside the deleted scope, since UniSuper says the deletion crossed both of its geographies [4]. Gate 4: no restore duration is stated, only that teams worked 24×7 over several days [5].	Account: a misconfiguration deleted the Private Cloud subscription. UniSuper says it crossed both of its geographies [4].	A copy outside the deleted subscription. UniSuper cites backups with an additional provider, Google cites Cloud Storage backups in the same region that were not impacted.	Two first-party statements that differ in framing and on which backups were used. Google describes one region.

Table 9.1. The five incidents against the evidence question. Every row names something a person could have been asked to produce beforehand.

APPENDIX A

Restore-drill harness specification §

This is the tool-neutral specification behind the drill figure in chapter 7. It says what a drill needs, what it does and what it must report. It names no product, and it is our own design°. The experiment scripts published with this paper run some of these steps for the cases we tested. They send no alerts.

A.1 Inputs §

Input	What it holds
Repository and credential	Where the backups live, and an identity that can read them but not delete them.
Snapshot rule	The newest snapshot and one old one. The newest tests recency. The old one tests whether older data still restores.
Scratch host	A clean machine that holds nothing from production, so nothing the backup forgot can hide there.
Expected values	Known counts from the last good run: files, rows, a size floor, a canary file or row.
Thresholds	Read from the environment or a config file, never edited into the script. Set them from your own last good run.
Application probe	Logs in as the application's own user, reads, writes inside a transaction it rolls back, and checks one setting. Reports the first step that fails.
Schedule and grace period	How often the drill runs, how late its success signal may be, and the name that is paged.
Declared RPO and RTO	The two numbers the drill is meant to confirm or contradict.

A.2 Steps §

1. Record the start instant, and the tool versions on the writing side and on the scratch host.
2. Select the snapshots by the rule above.
3. Level 1: run a check that reads the stored data, not only its metadata. In our restic test only the deep check failed on a damaged pack[M].
4. Provision the scratch host from nothing.
5. Restore everything the application needs, not just the data: roles, grants and settings, configuration and secrets, in the order the runbook gives.
6. Level 2: compare the restored data with the expected values.
7. Level 3: run the application probe. Stop the clock at the first successful application query, not when the restore command returns.
8. When sizing an RTO, repeat at least three times and keep the median and the worst.
9. Write the drill record (Appendix B).
10. Send the heartbeat only if every level passed and the record exists.
11. Destroy the scratch host.

A.3 Pass criteria §

Check	Passes when	Fails when
Level 1	The deep check exits 0 on the repository.	Any error, or the check did not run.
Level 2	Counts match the expected values within the thresholds and the canary is present.	Size below the floor, a file count below the allowed ratio of the previous run, a missing canary.
Level 3	Login, read, write and setting all pass.	Any step fails. The report names the first one.
RTO	The worst measured run is at or under the declared RTO.	It is over. Record the gap, then fix the system or change the declared number.
RPO	The age of the newest surviving marker row at the failure instant is at or under the declared RPO.	It is over.
Report	A record exists within the schedule plus the grace period.	No record. A missing result is a failure.

An exit code from the restore command is never a pass criterion on its own.

A.4 Alerts and timing fields §

Two routes end at the same named person: a level that failed, and a heartbeat that is late. The timer that watches for it lives outside the host that runs the drill (chapter 7). Time each stage of chapter 4 and mark whether the number came from a machine or from a person: detect, decide, access, transfer, load, make usable, verify. The human stages need a game day with an operator who did not write the runbook.

APPENDIX B

Restore-drill record §

One filled copy per drill. Keep it with the output it summarizes: logs, the probe result and the command lines used. A drill with no record did not happen.

Date and start time		**System and data size**	
Snapshot id and its age		**Restore host and its load**	
Writing tool and version		**Restore host tool version**	
Declared RPO and RTO		**Run number (at least three)**	
Run by		**Second person who followed the runbook**	

STAGES (CHAPTER 4)

Stage	Machine or person	Minutes	Started	Notes and questions asked
Detect				
Decide				
Access				
Transfer				
Load				
Make usable				
Verify				

RESULT

Item	Pass or fail	Evidence kept (file, value)
Level 1 repository check		
Level 2 counts and canary		
Level 3 application probe		
Measured RPO		
Measured RTO, median and worst		
Gap to declared		
Heartbeat sent		

FOLLOW-UPS

Each question the second person asked is a line missing from the runbook	Owner	Due

Owner signature and date: _____

APPENDIX C

Failure-domain audit §

Chapter 3 explains the method. For each copy, write the value in each of the four domains and circle every cell that equals production's. The count of circled cells is how many of the four domains one event could use to take both. It is a screen and not a risk model.

Copy	Account	Region or facility	Credential that can delete it	Software version: writer and restore host	Shared (of 4)	Immutable, against which credential	Last restore drill
Production					n/a		n/a
Copy 1							
Copy 2							
Copy 3							
Copy 4							
Copy 5							

C.1 Evidence for each cell §

Chapter 3 lists the evidence worth writing down for each of the four domains. Attach it to the cell, not to the page.

C.2 Rules for the count §

- A cell is shared when it equals production's.
- Snapshots in one backup repository share its packs, so count them as one copy[D].
- "Immutable" is a claim about a credential. Write which credential cannot delete the copy, and who holds it.
- Add a column when a copy needs one, for example where the key that decrypts it is kept. A drill has the last word, not this table[O].

APPENDIX D

Threat by copy matrix §

A blank copy of the matrix from chapter 2. Write the kind of each copy you actually have in the header, then mark each cell **Yes** (the copy is outside the threat's reach), **If** (only when a condition holds, which you write in the cell) or **No** (it dies with the original). Assume retention reaches back before the damage began. Every cell is a claim to prove with a restore drill°.

Threat	Copy kind 1:	Copy kind 2:	Copy kind 3:	Copy kind 4:	Copy kind 5:
Disk or host failure					
Fire or loss of the site					
Mistaken delete or script					
Bad data written (bug, corruption)					
Provider deletes or closes the account					
Stolen credentials					
Ransomware					
Backup job fails or writes nothing					

D.1 Reading it §

Chapter 2 explains how to read the matrix, by rows. Count the rows with a **No** in every cell you have and write the number here: _____.

APPENDIX E

Runbook skeleton §

The page the owner opens when an incident is declared. Chapter 9 gives the reasoning. Fill it from a drill, not from memory, and have someone who did not write it follow it once.

System		**Version and date**	
Owner (paged first)		**Deputy**	
Approver		**Last drill record**	

Verb	Write here
Declare. Who may call it, and what triggers the call.	
Decide. Restore to which point and onto which host. Who approves overwriting production.	
Restore. The exact commands in order, with tool versions, copied from a drill. Include everything the application needs besides the data: roles, grants, settings, configuration, secrets.	
Verify. The check that the data is right, which is a different thing from the command exiting 0. Name the application probe and the counts it is compared with.	
Communicate. Who is told, by whom, and how often.	

Chapter 9 explains the three names. If you cannot name the owner and the deputy, you have no owner.

APPENDIX F

Evidence checklist §

What an owner should be able to show today. Chapter 9 explains the rows and the strength column. Fill the date and name columns first: a blank row is a gap with a next action.

Evidence	Strength	Ch.	Date	Who signed	Where it is kept
Owner and deputy by name, on the runbook, with the date each last read it	declared	9			
Date, snapshot id, data size and minutes of the last full restore to a clean host, over repeated runs	measured	4, 7			
Declared RPO and RTO next to the measured values, with the gap	measured	4			
The same restore run by the deputy from the runbook alone, with the questions asked	measured	9			
A forced failure, and the minutes until a human was told	measured	7			
Which credentials can delete the newest backup, and whether any also deploys production	declared	6			
Per-copy table with account, region, credentials and tool version, shared cells circled	declared	3			
Config and secrets restored to a clean host without production, and what was missing	measured	5			
One tenant, table or bucket restored alone, timed against the full restore	measured	8			
The verification level the drill ran: repository, restored data or application check	measured	7			

How to verify this paper

Everything here is meant to be checked. If a number does not reproduce, or a source no longer says what is printed, report it using the errata address below and it will be handled in public.

Artifact repository	https://github.com/glincker/wp-2026-101-backup-restore-testing
Release tag	Not set
Rerun command	`EXP_SCRATCH=/path/to/empty/dir ./experiments/run-all.sh`
Expected result and tolerance	Every pass or fail outcome and exit code should match the published ones (69 outcomes in the corruption, fresh-host and empty-backup tests). Timings differed by 1.0 to 9.6 times between two runs on the same laptop, so compare orders of magnitude, not exact values.
Time and cost to rerun	About 3 GB of free disk. Needs restic, borg, zstd and the PostgreSQL 14 server and client tools.
Data DOI	Not set
Archive snapshot	Not set
Errata and corrections	https://glinr.com/research/2026-101-backups-you-can-restore#corrections
Response time	Acknowledged within 7 days, fixed or rejected with a reason within 30 days
Last verified	November 2026
Re-verify by	May 2027
Review status	none

Corrections are never silent: a corrected claim is struck through in the changelog with the old and new wording and the version that changed it.

CHANGELOG

Versions follow semantic versioning: a major version changes conclusions or recommendations, a minor version adds material or experiments, a patch corrects errors and typos. The report ID never changes.

Version	Date	What changed
0.1.0	2026-11-11	First draft.

Glossary

3-2-1 rule Three copies of the data on two media types, with at least one copy off-site, as Veeam states it [12].

3-2-1-1-0 rule Veeam's extension of the rule: it adds one immutable copy and results in zero recovery errors [12]. The term is vendor language. We keep it as a checklist and recast it in failure-domain terms.

Append-only mode A repository setting that allows new backups and prevents deleting or changing existing ones, as the restic rest-server README describes it [19]. In Borg 1.4.5, `borg delete` and `borg prune` still run but free no space [20].

Archive A copy kept for the record. The SRE book says the most important difference between backups and archives is that backups can be loaded back into an application [3].

Backup In this paper, a copy of data from which a service can be restored. It is trustworthy only after it passes the four gates.

Base backup A file-level copy of a PostgreSQL cluster. With an unbroken sequence of archived WAL it supports point-in-time recovery [16].

Canary file A small file whose presence shows that the path or mount being backed up is the one you meant.

Checksum A value computed from stored bytes so that later damage can be noticed when they are read. PostgreSQL 18 protects data pages with checksums by default. Older releases do not, and our 14.18 clusters had them off unless we asked [27].

Dead-man switch An alarm that fires when an expected signal stops arriving. The drill sends a heartbeat on success and an outside timer pages the owner when it is late. The design is our own.

Failure domain A set of things that fail together for one cause. Chapter 3 uses four: account, region or facility, credentials and software version. The list is our own.

Gate One of four tests a copy must pass: it exists, it is readable, it restores, and it restores fast enough. The framing is our own, not a standard.

Heartbeat The success signal a drill sends to its dead-man switch. The timestamp written into the data to measure the RPO is called a marker row.

Immutable copy A copy that cannot be changed or deleted for a set period. Always ask: by which credential, held by whom.

Logical and physical backup A logical backup exports the data, as `pg_dump` does. A physical backup copies the files. The PostgreSQL documentation says `pg_dump` and `pg_dumpall` do not produce file-system-level backups and cannot be used as part of a continuous-archiving solution [16].

MTD Maximum tolerable downtime. NIST defines it as the total time the system owner is willing to accept for an outage. Unlike the RTO, the RPO is not part of it [15].

Object Lock An Amazon S3 feature using a write-once-read-many model. It needs versioning and protects one object version. Compliance mode stops every user, root included, from deleting a locked version, and governance mode can be bypassed by a principal with the right permission [13].

Point-in-time recovery Restoring a database to its state at any time since a base backup. PostgreSQL's continuous archiving supports it and needs an unbroken run of archived WAL files back to the start of that backup [16].

Pull and push backup In a push design the production host holds a credential for the backup store. In a pull design a backup host in another zone reaches into production with a read-only credential, and production holds nothing that points at the backup.

Replica A live copy kept current from production. It follows a mistaken delete as it follows data, and the SRE book says replication and redundancy are not recoverability [3].

Repository The store a backup tool writes to. Snapshots in one repository share its packs, so we count them as one copy.

Restore drill A timed restore onto a clean host, followed by verification, that ends in a written record.

Retention The rule that decides which backups are deleted and when. It is the identity that deletes backups.

RPO Recovery point objective. NIST defines it as the point in time, before a disruption, to which data can be recovered, given the most recent backup copy [15]. We measure it as the age of the newest restorable copy at the moment of failure.

RTO Recovery time objective. NIST defines it as the maximum time a system resource can remain unavailable before there is an unacceptable impact [15]. We start the clock at the failure, not at the restore command.

Runbook The page the owner opens when an incident is declared: declare, decide, restore, verify, communicate.

Scratch host A clean machine that holds nothing from production, used for drills.

Silent corruption A change to stored bytes that the commands you run do not report.

Snapshot A point-in-time copy. The word covers platform disk snapshots and the per-run records some backup tools keep, restic among them.

Tenant One customer of a shared system.

Unit of restore The smallest part you promise to hand back on its own, such as a tenant, a database, a table or a bucket prefix.

Version skew A mismatch between the tool or server version that wrote a copy and the one that must read it. GitLab's postmortem describes `pg_dump` 9.2 used against PostgreSQL 9.6 [2].

WAL PostgreSQL's write-ahead log. An archive command must return a zero exit status if and only if it succeeds [16].

References

[1] GLINR Studios, "GLINR-WP-2026-101 experiments: scripts, raw records and results." 2026.

[2] GitLab, "Postmortem of database outage of January 31." Accessed: Nov. 11, 2026. [Online]. Available: https://about.gitlab.com/blog/postmortem-of-database-outage-of-january-31/

[3] R. Blum and R. Singh, "Data Integrity: What You Read Is What You Wrote." Accessed: Nov. 11, 2026. [Online]. Available: https://sre.google/sre-book/data-integrity/

[4] UniSuper and Google Cloud, "A joint statement from UniSuper CEO and Google Cloud CEO." Accessed: Nov. 11, 2026. [Online]. Available: https://www.unisuper.com.au/about-us/media-centre/2024/a-joint-statement-from-unisuper-and-google-cloud

[5] Google Cloud, "Sharing details on a recent incident impacting one of our customers." Accessed: Nov. 11, 2026. [Online]. Available: https://cloud.google.com/blog/products/infrastructure/details-of-google-cloud-gcve-incident

[6] "OVH Cloud must pay damages for lost backup data." Accessed: Nov. 11, 2026. [Online]. Available: https://blocksandfiles.com/2023/03/23/ovh-cloud-must-pay-damages-for-lost-backup-data/

[7] Atlassian Engineering, "April 2022 outage update." Accessed: Nov. 11, 2026. [Online]. Available: https://www.atlassian.com/engineering/april-2022-outage-update

[8] Atlassian Engineering, "Post-Incident Review on the Atlassian April 2022 outage." Accessed: Nov. 11, 2026. [Online]. Available: https://www.atlassian.com/blog/atlassian-engineering/post-incident-review-april-2022-outage

[9] The Register, "Code Spaces goes titsup FOREVER after attacker NUKES its Amazon-hosted data." Accessed: Nov. 11, 2026. [Online]. Available: https://www.theregister.com/2014/06/18/code_spaces_destroyed/

[10] L. N. Bairavasundaram, G. Goodson, B. Schroeder, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "An Analysis of Data Corruption in the Storage Stack," in *6th USENIX Conference on File and Storage Technologies (FAST '08)*, 2008, pp. 223–238. Accessed: Nov. 11, 2026. [Online]. Available: https://www.usenix.org/legacy/events/fast08/tech/bairavasundaram.html

[11] The Register, "I was told to make backups, not test them. Why does that make you look so worried?." Accessed: Nov. 11, 2026. [Online]. Available: https://www.theregister.com/2025/02/07/on_call/

[12] "How to follow the 3-2-1 backup rule with Veeam Backup & Replication." Accessed: Nov. 11, 2026. [Online]. Available: https://www.veeam.com/blog/how-to-follow-the-3-2-1-backup-rule-with-veeam-backup-replication.html

[13] "Amazon S3 User Guide: Locking objects with Object Lock." Accessed: Nov. 11, 2026. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/object-lock.html

[14] "PostgreSQL 18 Documentation: pg_dump." Accessed: Nov. 11, 2026. [Online]. Available: https://www.postgresql.org/docs/current/app-pgdump.html

[15] "Contingency Planning Guide for Federal Information Systems," NIST Special Publication 800-34 Rev. 1, 2010. Accessed: Nov. 11, 2026. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-34r1.pdf

[16] PostgreSQL Global Development Group, "PostgreSQL 18 Documentation, 25.3 Continuous Archiving and Point-in-Time Recovery (PITR)." Accessed: Nov. 11, 2026. [Online]. Available: https://www.postgresql.org/docs/current/continuous-archiving.html

[17] Krebs on Security, "Don't Wanna Pay Ransom Gangs? Test Your Backups.." Accessed: Nov. 11, 2026. [Online]. Available: https://krebsonsecurity.com/2021/07/dont-wanna-pay-ransom-gangs-test-your-backups/

[18] "pgBackRest User Guide (Debian and Ubuntu), version 2.60.0." Accessed: Nov. 11, 2026. [Online]. Available: https://pgbackrest.org/user-guide.html

[19] "rest-server README." Accessed: Nov. 11, 2026. [Online]. Available: https://github.com/restic/rest-server/blob/master/README.md

[20] "Borg 1.4.5 documentation: Additional Notes (append-only mode)." Accessed: Nov. 11, 2026. [Online]. Available: https://borgbackup.readthedocs.io/en/stable/usage/notes.html

[21] Backblaze, "Backblaze Docs: Object Lock." Accessed: Nov. 11, 2026. [Online]. Available: https://www.backblaze.com/docs/cloud-storage-object-lock

[22] restic project, "restic documentation: Working with repositories." Accessed: Nov. 11, 2026. [Online]. Available: https://restic.readthedocs.io/en/stable/045_working_with_repos.html

[23] BorgBackup project, "BorgBackup documentation: borg check." Accessed: Nov. 11, 2026. [Online]. Available: https://borgbackup.readthedocs.io/en/stable/usage/check.html

[24] pgBackRest project, "pgBackRest Command Reference: Repository Verify." Accessed: Nov. 11, 2026. [Online]. Available: https://pgbackrest.org/command.html

[25] pgBackRest project, "pgBackRest Configuration Reference." Accessed: Nov. 11, 2026. [Online]. Available: https://pgbackrest.org/configuration.html

[26] PostgreSQL Global Development Group, "PostgreSQL documentation: pg_verifybackup." Accessed: Nov. 11, 2026. [Online]. Available: https://www.postgresql.org/docs/current/app-pgverifybackup.html

[27] PostgreSQL Global Development Group, "PostgreSQL 18 Documentation: Data Checksums." Accessed: Nov. 11, 2026. [Online]. Available: https://www.postgresql.org/docs/current/checksums.html

About this paper and license

Publisher

This paper is written and published by GLINR Studios (glinr.com). It is an independent technical write-up: claims are sourced, experiments are described with their limits, and the studio names its own interests in the disclosure under the abstract so that readers can weigh them. No sponsor shaped the content, and no vendor reviewed it.

Cite this paper

```
@techreport{glinr2026wp101,
  author = {{GLINR Studios}},
  title = {Restore Is the Product: Backups you can
prove: measuring RPO and RTO, mapping threats to
surviving copies, and testing restores before you
need them},
  institution = {GLINR Studios},
  type = {Working paper},
  number = {GLINR-WP-2026-101},
  year = {2026},
  version = {0.1.0},
  url = {https://glinr.com/research/2026-101-backups-
you-can-restore}
}
```

BibTeX, RIS and Markdown versions are linked from the web page.

Corrections

Corrections are welcome and handled in public. Send them to glinr.com/research/2026-101-backups-you-can-restore#corrections or to support@glinr.com. A correction that changes a claim is recorded in the revision history with the old and new wording. This paper has not been externally reviewed.

License and reuse

This paper is licensed under Creative Commons Attribution 4.0 International (CC BY 4.0). You may share and adapt it, including commercially, as long as you give appropriate credit, link to the license and indicate whether changes were made. Full text: creativecommons.org/licenses/by/4.0.

Figures and tables may be reproduced with the credit line “GLINR Studios, *Restore Is the Product*, GLINR-WP-2026-101, 2026, glinr.com/research/2026-101-backups-you-can-restore”.

Tool and product names belong to their owners. They appear only to name what was tested or quoted, and imply no endorsement. Provided as is, without warranty. Test every procedure in your own environment before relying on it. This paper is not professional advice.

Colophon

Typeset with Typst. Text is set in Newsreader, headings in Stack Sans Headline, labels in Plus Jakarta Sans and code in JetBrains Mono. Diagrams are drawn as code in D2.