

WHITEPAPER

Running Authoritative DNS with PowerDNS and dnsmist

A reference design for primary and secondary nameservers across two sites, deployed and patched with Ansible, with the monitoring and lessons that come from running it in production.

PUBLISHED BY
ZSoftly Technologies Inc.

DATE
September 2026

VERSION
1.0

AUDIENCE
Platform, network and
operations engineers

CONTENTS

Executive summary	Page 2
Why run your own authoritative DNS	Page 3
Reference architecture	Page 4
Zones, caches, delegation and DNSSEC	Page 5
Deployment as code, patching and CVE response	Page 6
Monitoring and the alerts that matter	Page 7
Lessons from production	Page 8
Readiness checklist and how ZSoftly can help	Page 9
Sources	Page 10

PowerDNS, dnsmist and standards facts in this paper were checked against the vendor documentation and RFCs on 29 September 2026. The design and lessons come from authoritative DNS that ZSoftly runs for its own cloud platform. Names, addresses and values are left out on purpose.

EXECUTIVE SUMMARY

Two nameservers, two sites, one source of truth, and every step in code

Authoritative DNS is small, quiet and critical. When it fails, every service behind the zone fails with it, and the fault often shows up as a vague timeout far from the cause. This paper describes how we run authoritative DNS on PowerDNS with dnsmdist in front, and what we changed after it surprised us.

The problem

- Zones change through an API at any hour, and both nameservers must agree within seconds.
- Two caching layers sit between a change and the answer. Each can hold a wrong answer after the data is fixed.
- A secondary that reads its data across a site link inherits that link's latency and its outages.
- Security advisories often cover more than one product at once, and a package role that only installs never upgrades.

The approach

Run PowerDNS Authoritative on a database backend at a primary site. Run a second PowerDNS at another site as a true secondary with its own local database, fed by signed zone transfers. Put dnsmdist in front of both for rate limits, filtering, a short packet cache and metrics. Send every write to the primary through the PowerDNS HTTP API. Deploy and operate the lot with Ansible roles, with risky operations locked behind explicit tags and confirmations.

Primary and secondary

Each site answers from its own data. A site link fault delays updates, not answers.

API-first writes

One write path, serial bumps on every change, NOTIFY to the secondary.

Everything in Git

Roles, values, alert rules and operations scripts.

Patch one site first

The other nameserver answers while one restarts.

The outcomes to aim for

- New zones answer on both nameservers within one transfer cycle, with no manual step.
- Deletions, cache purges and zone moves run as tested playbook phases, never as typed commands.
- Alerts fire on what customers feel: no healthy backend, timeouts, SERVFAIL rate, latency and pending security upgrades.
- Registry delegation, glue and DNSSEC records are checked from outside before anyone relies on them.

Who should read this

Engineering leaders should read this page, the options on page 3 and the lessons on page 8.

Platform engineers will find the design on pages 4 and 5 and the deployment practice on page 6.

Operations and on-call teams should read the monitoring chapter on page 7. Anyone planning a build should work through the readiness checklist on page 9 first.

CHAPTER 1

Why run your own authoritative DNS

Most organisations should buy authoritative DNS as a service. It is cheap, it is someone else's pager, and large providers answer from many locations. Running your own makes sense in a narrower set of cases. Be honest about which one you are in before you build anything.

Good reasons to run it

- **You sell DNS.** A hosting or cloud platform that offers managed zones to its own customers needs a nameserver fleet it controls, with an API its portal can call.
- **DNS is part of your product workflow.** Zones are created and removed by automation many times a day, and you want that path to stay inside your own platform.
- **You need control of the data path.** You want to choose where zone data lives, who can read it and which software versions answer queries.
- **You already run the adjacent pieces.** Databases, monitoring, configuration management and on-call exist, so DNS adds one service, not a new discipline.

Reasons that are not enough

- Saving a small monthly fee. The engineering time to build, patch and watch two nameservers costs more.
- A wish for more record types or features. Check your current provider first. Many already offer them.
- Running a single nameserver. RFC 2182 asks for secondaries at geographically and topologically dispersed locations, so one failure cannot disable all of them [14]. One site is not a design.

Option	You operate	Fits when	Watch for
Managed DNS service	Zone content only	DNS supports your product but is not part of it	API limits, pricing per zone or query, lock-in of proprietary record types
Hidden primary, provider secondaries	The primary and its data	You want control of data and edits, and a provider's reach for answers	Transfer security, NOTIFY reachability, provider support for your signing model
Own primary and secondary	Everything, at two or more sites	You sell DNS or run it as part of a platform	Patching, abuse traffic, monitoring, registry and glue work

Table 1. Three ways to host authoritative DNS. This paper describes the third.

What running it asks of your team

- An owner who reads PowerDNS and dnstest advisories and patches within days, not months.
- On-call cover for the nameservers, with alerts routed to people who can act on them.
- Access to the registrar or registry account that holds your delegation and glue.

A middle path. A hidden primary that you run, with a provider's nameservers as public secondaries, gives you the API and the data while someone else absorbs query load. The zone lifecycle and caching guidance in chapter 3 still applies.

CHAPTER 2

Reference architecture

Two sites, each with one DNS virtual machine. Each machine runs dnsmist on public port 53 and PowerDNS Authoritative behind it on loopback. The primary site's PowerDNS reads and writes a replicated database. The secondary site's PowerDNS reads only a local database on the same machine, filled by zone transfers. Each public nameserver address comes from a different upstream provider.

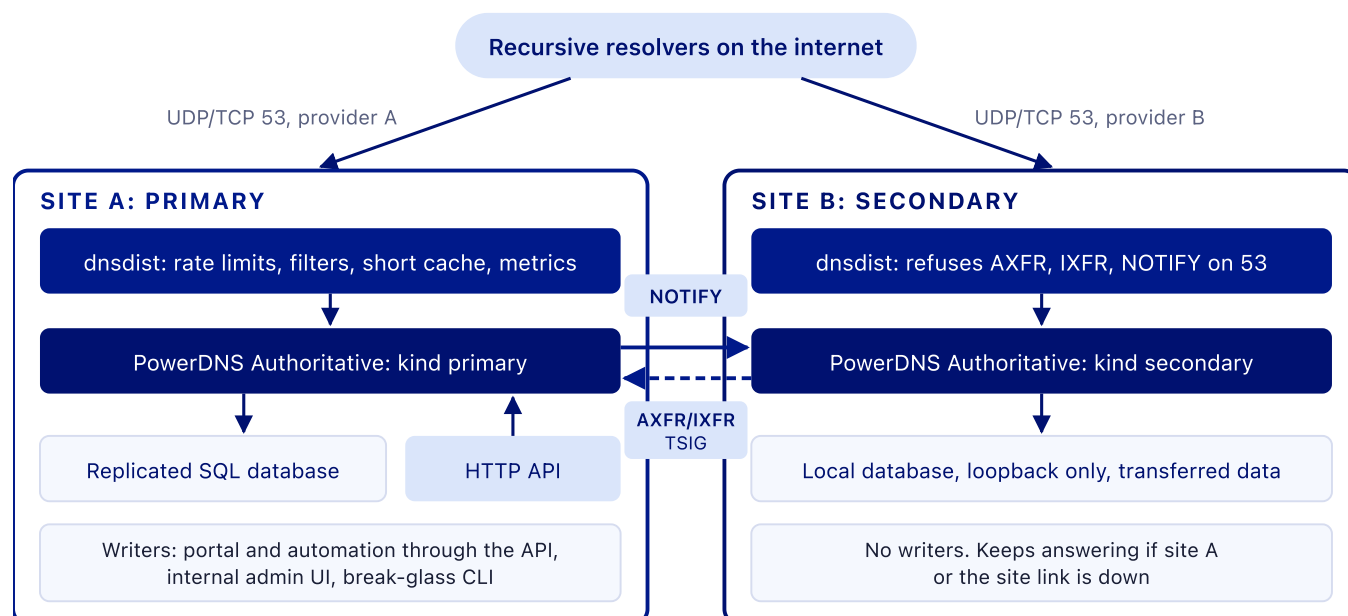


Figure 1. Query path in dark blue boxes, transfer path between the sites. Transfers use a direct PowerDNS listener that only the peer can reach, and never pass through the public dnsmist.

Flow	Path	Control
Public query	Resolver to dnsmist on 53, then PowerDNS on loopback	dnsmist rate and response-code limits, ANY refused, short packet cache
Zone write	Portal or automation to the primary's HTTP API	API key, internal network only, <code>SOA-EDIT-API</code> raises the serial on each change [3]
Change signal	Primary sends NOTIFY to the secondary's direct listener [1]	Firewall opens the port to the peer address only
Zone transfer	Secondary pulls AXFR or IXFR from the primary	TSIG required per zone. A signed request passes even outside <code>allow-axfr-ips</code> , so guard the key [4]
New zone	NOTIFY for an unknown zone from a trusted autopriamry	The secondary provisions the zone itself [1]

Table 2. Traffic flows and the control on each. PowerDNS behaviour cited from its documentation.

Why not one shared database

A simpler design points both PowerDNS servers at one replicated database. We ran it that way first. Every cache miss at the far site then crossed the site link, and new zones reached that site only on its periodic refresh. A true secondary keeps answers local and turns the site link into an update path. If the link fails, the secondary serves the last data it received until the SOA expire timer runs out.

CHAPTER 3

Zones, caches, delegation and DNSSEC

One write path

Send every write to the primary. The PowerDNS HTTP API creates zones with a POST and changes record sets with a PATCH [5]. Set `SOA-EDIT-API` on every zone so each API change raises the serial [3]. The primary then sends NOTIFY and the secondary transfers the change [1]. Writes made straight into the database skip that metadata, so check serial parity after any edit from an admin UI that writes SQL.

Zone lifecycle

Create

- 1 Create the zone on the primary through the API, with SOA and NS records at the apex.
- 2 Autosecondary provisions it on the secondary from the first NOTIFY [1].
- 3 A follow-up phase pins the transfer to the direct port and turns on TSIG for the new zone.
- 4 Validate: zone kind on both sides, TSIG set, SOA serials equal.

Delete

- 1 Delete on the primary first. A secondary copy removed first comes back with the next NOTIFY.
- 2 Delete on the secondary. PowerDNS has no way to signal a removal, so autosecondary never deletes [1].
- 3 Purge both cache layers on both nodes.
- 4 Wait until both nodes answer REFUSED for the zone.

Two caches, purged together

PowerDNS keeps a packet cache and a query cache. The packet cache TTL defaults to 20 seconds [2]. dnsmdist keeps its own packet cache per pool, and by default gives failure and REFUSED answers a 60-second TTL [8]. Keep both short. A normal change then needs no purge. After a manual correction, purge PowerDNS with `pdns_control purge` [2], then empty dnsmdist, and check that the dnsmdist cache reports zero entries. Clearing only one layer leaves the other serving the old answer.

Registry delegation and glue

If your nameserver names sit inside a domain you delegate, the parent needs glue addresses for them. In the registry protocol, a nameserver is a host object subordinate to its parent domain, and addresses are required only to produce glue [15]. Create the host objects under the parent domain at your registrar before you point any zone at them. Some registries also expect a distinct address for each nameserver. Give each nameserver its own address from a different upstream, which RFC 2182 asks for anyway [14]. Confirm the result from outside with `dig +trace`, not from a machine inside your network.

DNSSEC is a decision, not a checkbox

PowerDNS can sign online. RRSIG records are computed on the fly and cached, and a signing primary sends the secondary everything it needs to serve the zone without the keys [6]. We sign with algorithm 13, ECDSA P-256 with SHA-256, which RFC 8624 lists as MUST for signing and validation [16]. Before you publish a DS record at the parent, prove the chain from the root with a validating tool such as `de1v`. A signature on the answer is not proof. If the result is bogus, remove the DS record first and investigate second.

CHAPTER 4

Deployment as code, patching and CVE response

Every DNS machine is built, configured and operated from one Ansible repository. The PowerDNS project publishes official roles for PowerDNS Authoritative and for dnsmist [12][13]. We vendor both and wrap them in numbered playbooks, so the build order reads from the file names.

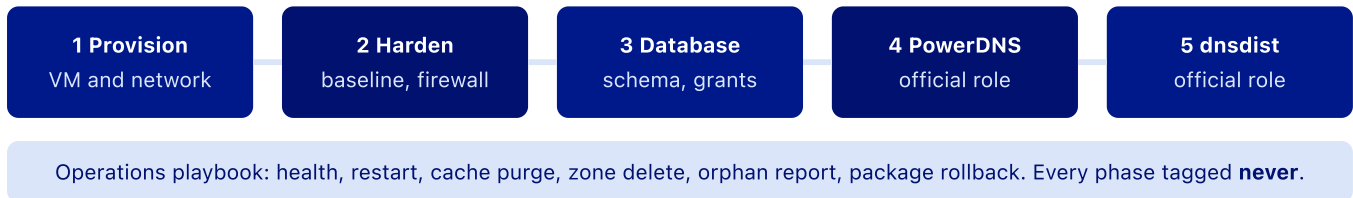


Figure 2. Build order as numbered playbooks, plus one operations playbook whose phases run only when named.

Rules that keep DNS operations safe

- **Risky phases are opt-in.** Ansible skips a task tagged `never` unless you ask for it by tag [17]. Phases that change DNS data also need a confirmation variable that names the phase, and an explicit zone list.
- **Every phase supports check mode.** Engineers preview a deletion or a zone conversion before it runs.
- **One node per run.** Deploys that restart services pass a limit of one host. An unscoped run could restart both nameservers at once.
- **Restart in dependency order.** Restart PowerDNS, wait for its listeners, then restart dnsmist so its health checks find a live backend.
- **Scripts are templates.** Health, purge, parity and TSIG audit scripts are Jinja templates in a role, deployed to each node. Nobody pastes a shell loop from a wiki.

Patching and CVE response

PowerDNS products poll a DNS name for their own security status. Status 2 means an upgrade is recommended and status 3 means it is mandatory [7]. Alert on both. The status is based on version, so a configuration workaround never clears it. Advisories can cover several products at once. PowerDNS Security Advisory 2026-11 named the Authoritative Server, Recursor and dnsmist together [11]. When one product's alert fires, check the others.

- 1 Run the health phase. Confirm the candidate package is the fixed release.
- 2 Upgrade the secondary site first, dnsmist then PowerDNS. Both official roles default their package state to `present`, which never upgrades an installed package [12][13]. Pass `latest` for the patch run only.
- 3 Run health again. Confirm versions, both services active and answers on the public address.
- 4 Repeat on the primary. Record one row per upgrade in a version log.
- 5 Roll back with a pinned package version if needed. An in-branch release rarely changes configuration.

Changing the design under live traffic

Moving from a shared database to primary and secondary is a one-time change with its own gates. Take one public nameserver out of service first, so resolvers use the other while its data is rebuilt. Convert the zones, validate kinds, TSIG and serials on every zone, then cut the site link and prove the secondary answers from local data alone. Keep a rollback per zone and a rollback for the whole change, and never mix the two.

CHAPTER 5

Monitoring and the alerts that matter

Scrape three targets on each DNS node: PowerDNS, dnsmist and the host. dnsmist serves Prometheus metrics at `/metrics` from its built-in web server. By default that server accepts connections only from loopback and requires authentication for metrics [10]. Widen the ACL to your monitoring network only, and decide deliberately whether the metrics endpoint needs credentials. If two Prometheus servers watch every target, let one own the DNS rule set so each alert fires once.

Alert	Signal	Why it matters
No healthy backend	dnsmist queries with no policy, or backend down	Always critical. dnsmist has nowhere to send queries
Backend timeouts	Downstream timeouts rising	PowerDNS is slow or its database is. Clients see delays before errors
High latency	dnsmist moving average latency	Set the threshold per site. A site with a slower backend path needs its own line
SERVFAIL rate	SERVFAIL share of answers	Broken zone data or an unreachable database
Dynamic blocks active	Sources currently blocked	An attack, or a legitimate resolver caught by a threshold
Security status	secpoll status 2 or 3 on either product [7]	A known vulnerability in the running version
No queries	Zero query rate after start-up	The node is healthy but unreachable from outside, often NAT or glue
Secondary database down	Local database exporter reports down	The secondary will stop answering new data

Table 3. The DNS alerts we keep. Thresholds depend on your traffic and are left out.

Cache behaviour needs a baseline first

Cache hit ratio alerts look useful and mislead early. dnsmist answers repeat queries before PowerDNS counts them, so the PowerDNS hit ratio drops when you add the dnsmist cache, and nothing is wrong. Collect a few weeks of data after any cache change. Only then set hit ratio alerts, and keep them at warning level. The same applies to DNSSEC signing rates. Cached and presigned answers create no new signatures, so prove DNSSEC health with a validating query, not a counter.

Dynamic blocks: choose the action

dnsmist dynamic blocks are short-lived rules inserted when a source crosses a threshold, such as a query rate, a response-code rate or a query-type rate [9]. The default action drops queries. We answer REFUSED instead. A resolver that gets REFUSED moves to the other nameserver at once. A drop makes it wait for a timeout, and truncate pushes it to TCP on the same node. Exempt your monitoring and internal resolvers, or a noisy probe will block itself.

Probe from both sides. An internal check proves the service runs. Only a check from outside your network proves resolvers can reach it through the NAT, the firewall and the registry glue.

CHAPTER 6

Lessons from production

Each row below is a real event from our own DNS service, written up in a postmortem and fixed in code. We removed the names and numbers. The patterns repeat in any PowerDNS estate.

What we saw	Cause	What we changed
A new zone answered REFUSED for SOA and NS on one nameserver only, for up to an hour	Both nodes read one shared database. The remote node learned new zones only on its periodic zone refresh, answered REFUSED in the meantime, and its packet cache held that REFUSED for a long TTL. The cache key includes EDNS options, so only some query shapes failed	Moved to primary and secondary with NOTIFY, so the secondary learns each zone at once. Kept cache TTLs short. Added a purge that clears both layers
Latency alert on one site with an idle server	The remote node read its database across the site link. Cache hits were fast, each miss paid several round trips, and one alert threshold covered both sites	Gave the secondary a local database fed by transfers. Made latency thresholds site-aware
A new zone answered REFUSED on both nameservers while the infrastructure-as-code plan showed no changes	The zone object existed with serial 0 and no SOA or NS records. A zone with no SOA is not authoritative. The plan tracked only that the object existed	Check the serial first when a zone refuses. Validate the apex after every zone create. An API success is not enough
Zones deleted in the portal kept answering on the secondary	Autosecondary never deletes. There is no signal for a removal [1]	A delete playbook that runs primary then secondary, purges caches and waits for REFUSED, plus a report of orphan zones
A security alert cleared on one product, the same advisory still applied to the other	The advisory covered dnsmist and the Authoritative Server together. The install roles default to <code>present</code> and never upgrade	Patch both products in one window, secondary site first, with <code>latest</code> passed for the run
The registrar rejected a nameserver name during delegation	The nameserver needed a host object under its parent domain, with glue, before any zone could use it	Create host objects and glue first, a distinct address per nameserver, and verify with a trace from outside

Table 4. Anonymised production lessons. Each fix is now a role, a playbook phase or an alert.

What the lessons share

- **Caches remember failures.** Any cache that stores negative or refused answers turns a short gap into a long outage. Keep TTLs short and purge every layer.
- **Remote reads hide in the query path.** A design that answers from data at another site works until the link is slow. Keep each nameserver's answers local.
- **Green in one tool is not green.** An API success, a clean plan or a healthy local check does not prove resolvers get the right answer. Query the public address.
- **Deletion is its own feature.** Create paths get tested every day. Delete paths get tested when a customer leaves. Build and test both.

How we turn an incident into code

Every postmortem ends with follow-ups that land in the repository: a role change, a new playbook phase, an alert rule or a runbook step. A follow-up that exists only in a ticket tends to be forgotten until the same fault returns. We close the postmortem only when the fix is merged and has run once in production.

WORKING WITH US

Readiness checklist and how ZSoftly can help

Check your readiness

- | | |
|--|--|
| ☐ Two sites, each with its own upstream and a distinct public address per nameserver. | ☐ Alerts for no backend, timeouts, SERVFAIL, latency and security status. |
| ☐ One write path to the primary, with serial bumps on every change. | ☐ An external probe that queries each nameserver from outside. |
| ☐ A secondary that answers from local data, not across a site link. | ☐ Host objects and glue at the registry, verified with a trace. |
| ☐ TSIG on every transfer, and transfers kept off the public listener. | ☐ A written DNSSEC decision, and a validation step before any DS record. |
| ☐ Short cache TTLs, and a purge that clears every layer. | ☐ A patch runbook that upgrades one site at a time. |
| ☐ A tested delete path that removes the zone from every node. | ☐ Everything above in Git, run by playbooks, not typed by hand. |

If more than three boxes are empty, start with an assessment before you move any zone.

How we work

ZSoftly Technologies Inc. is a cloud engineering company. We design, build and run cloud platforms for clients in Canada, the United States and the United Kingdom. The engineers who design your DNS are the ones who build it and hand it over.

Assess

Review your zones, providers, registry setup and failure history.

Design

Agree sites, write paths, transfer security and DNSSEC in writing.

Build

Deliver the roles, playbooks and alerts as code in your repository.

Operate

Run the service with you through cutover and the first patch cycle.

Hand over

Runbooks and access move to your team, which runs a drill unaided.

Where we usually start

- 1 Discovery call.** We review your current DNS and goals. Your first hour with our engineers is free.
- 2 Assessment.** A written report on delegation, glue, caching, transfer security and monitoring gaps.
- 3 Pilot.** A primary and secondary pair serving a low-risk zone, with alerts live.
- 4 Cutover.** Zones move in batches with a validation gate and a rollback for each batch.

What you keep

- All roles, playbooks, alert rules and dashboards, in your own repository.
- Runbooks for zone operations, patching, cache purges and rollback.
- Registry, server and key access that belongs to you from the first day.

Talk to an engineer. Book a discovery call at zcp.zsoftly.ca/contact or call +1 (647) 771-1065.

REFERENCES

Sources

All sources were checked on 29 September 2026.

1. PowerDNS. Authoritative Server: DNS modes of operation (primary, secondary, autoprimary).
<https://doc.powerdns.com/authoritative/modes-of-operation.html>
2. PowerDNS. Authoritative Server: performance and caching.
<https://doc.powerdns.com/authoritative/performance.html>
3. PowerDNS. Authoritative Server: per-zone metadata (SOA-EDIT-API, TSIG-ALLOW-AXFR, AXFR-MASTER-TSIG).
<https://doc.powerdns.com/authoritative/domainmetadata.html>
4. PowerDNS. Authoritative Server: TSIG.
<https://doc.powerdns.com/authoritative/tsig.html>
5. PowerDNS. Authoritative Server HTTP API: zones.
<https://doc.powerdns.com/authoritative/http-api/zone.html>
6. PowerDNS. Authoritative Server: DNSSEC modes of operation.
<https://doc.powerdns.com/authoritative/dnssec/modes-of-operation.html>
7. PowerDNS. Authoritative Server: security polling.
<https://doc.powerdns.com/authoritative/security.html>
8. dnsmdist. Caching responses.
<https://www.dnsmdist.org/guides/cache.html>
9. dnsmdist. Dynamic rule generation (dynamic blocks).
<https://www.dnsmdist.org/guides/dynblocks.html>
10. dnsmdist. Built-in webserver.
<https://www.dnsmdist.org/guides/webserver.html>
11. dnsmdist. Security advisories, including PowerDNS Security Advisory 2026-11.
<https://www.dnsmdist.org/security-advisories/index.html>
12. PowerDNS. Ansible role for PowerDNS Authoritative (pdns-ansible).
<https://github.com/PowerDNS/pdns-ansible>
13. PowerDNS. Ansible role for dnsmdist (dnsmdist-ansible).
<https://github.com/PowerDNS/dnsmdist-ansible>
14. IETF. RFC 2182, Selection and Operation of Secondary DNS Servers.
<https://www.rfc-editor.org/rfc/rfc2182.html>
15. IETF. RFC 5732, Extensible Provisioning Protocol (EPP) Host Mapping.
<https://www.rfc-editor.org/rfc/rfc5732.html>
16. IETF. RFC 8624, Algorithm Implementation Requirements and Usage Guidance for DNSSEC.
<https://www.rfc-editor.org/rfc/rfc8624.html>
17. Ansible. Tags, including the special tags always and never.
https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_tags.html

Notices

This paper is for general information. It does not create a warranty or a commitment from ZSoftly Technologies Inc. The lessons describe our own systems with identifying details removed. PowerDNS and dnsmdist are open-source projects under their own licences, and PowerDNS is a trademark of its owner. Other names may be trademarks of their respective owners.

© 2026 ZSoftly Technologies Inc. Version 1.0, September 2026.