PowerDNS: DNS Without the Archaeology

August 13, 2026 / Gavin Jackson

powerdns poweradmin dns bind samba security linux open-source networking

A friend at work recently built a PowerDNS server and put Poweradmin in front of it. He sent me the URL and I had one of those mildly irritating moments where a tool makes you question why the thing you use every day still has to be so awkward.

I am used to DNS administration meaning one of two things. On Linux, it is usually BIND configuration and zone files, followed by the familiar routine of checking the file, reloading it and hoping I remembered the serial. On Windows, it is the DNS Manager console: capable enough, but not a place I enjoy spending time.

Poweradmin looked refreshingly normal. I could search zones and records, see the owner of a zone, edit records in a table, add several records at once and work with forward and reverse zones from the same interface. It was immediately obvious how I could give one team access to its zones without giving it access to everyone else's.

That sent me down a rabbit hole. We have several security domains that need a common set of internal DNS zones. I want those records maintained in the management domain and copied out to local DNS servers in each downstream domain. The downstream servers should keep answering if management is offline, but they should never become another place where people make changes.

PowerDNS looks very well suited to that job.

Before getting into it, PowerDNS is a possible replacement for the **DNS part** of a BIND or Samba deployment. It is not a replacement for Samba as a whole. It does not provide an Active Directory domain controller, Kerberos, LDAP or file sharing. If the brief is "replace Samba", PowerDNS only answers the DNS portion of it.

Who actually makes PowerDNS?

Development started in 1999, initially to support fast DNS-based geographical load balancing. Bert Hubert registered PowerDNS.com B.V. in 2000, and the code was released under the GPL in 2002. PowerDNS merged with **Open-Xchange** in 2015.

Open-Xchange is now the commercial home of PowerDNS. It sells support and a range of enterprise and carrier products, while the Authoritative Server, Recursor and dnsdist remain open source. The software is packaged for the usual Linux and BSD platforms, and paid support is available if a community mailing list is not an acceptable escalation path during an outage.

The commercial side did not bother me when I looked through the product. The useful DNS server is not a crippled teaser for an enterprise edition. The open-source components are the same ones people have been running at serious scale for years.

PowerDNS is really a family of three familiar tools:

- **PowerDNS Authoritative Server** answers from zones it hosts.
- **PowerDNS Recursor** looks up names elsewhere and caches the results.
- **dnsdist** handles DNS-aware load balancing, routing and filtering in front of other DNS servers.

They are separate programs. The authoritative server does not quietly become a recursive resolver because one setting was overlooked, and the recursor does not hold the writable copy of an authoritative zone. The PowerDNS documentation treats them as separate products and deployments can use one without the others.

I prefer this to asking one large daemon to do every DNS job. The server answering for an internal zone has different access rules and failure modes from the resolver accepting queries from thousands of clients. Running them separately makes those boundaries much harder to blur by accident.

The backend is the interesting bit

The Authoritative Server speaks DNS at the front and fetches records from a backend. That backend can be PostgreSQL, MariaDB/MySQL, SQLite, LMDB, BIND-format files, LDAP, a remote service or one of several other modules. A server can load more than one backend.

This is where PowerDNS feels different from BIND. A BIND deployment generally revolves around `named.conf` and zone files. PowerDNS can use those too, but its SQL backends turn zones and records into ordinary structured data. They can be searched properly, updated over HTTP, backed up with existing database tools and presented in a web interface without another program having to parse and rewrite text files.

It still speaks normal DNS to other servers. A zone can be a **primary**, a **secondary** or **native**. A primary has the writable copy and notifies its secondaries when the zone changes. A secondary compares the SOA serial and transfers a newer copy from the primary. A native zone assumes that something outside DNS, such as database replication, is keeping the data in sync.

Across security boundaries, I would stick to primary and secondary zones. Replicating the SQL database into every domain would couple those domains to the PowerDNS schema, database credentials and database replication. It would also move more than DNS records across the boundary. AXFR has a much smaller job: copy this zone to this authorised server.

```
%%{init: {"themeVariables": {"lineColor": "#94a3b8"}}}%%
flowchart TB
 subgraph M["Management security domain"]
 direction LR
 I["OIDC / SAML / LDAP<br/>plus MFA"] --> P["Poweradmin<br/>UI and API"]
 P -->|"HTTPS API"| H["PowerDNS Authoritative<br/>hidden primary"]
 H --> D[("SQL backend")]
 end

 H --> X["Zone distribution<br/>NOTIFY and AXFR<br/>TSIG authenticated<br/>one key per domain"]

 subgraph A["Security domain A"]
 direction LR
 RA["Resolver and clients"] -->|"local DNS"| SA["PowerDNS secondary<br/>read-only zone copy"]
 end

 subgraph B["Security domain B"]
 direction LR
 RB["Resolver and clients"] -->|"local DNS"| SB["PowerDNS secondary<br/>read-only zone copy"]
 end

 subgraph C["Security domain C"]
 direction LR
 RC["Resolver and clients"] -->|"local DNS"| SC["PowerDNS secondary<br/>read-only zone copy"]
 end

 X --> SA
 X --> SB
 X --> SC

 SA ~~~ SB
 SB ~~~ SC

 linkStyle default stroke:#94a3b8,stroke-width:2px;
```

The "single source" in this design is logical. It does not have to mean one lonely VM with one disk. The writable service can be made redundant inside the management domain. Each other domain receives a local read-only copy and answers its own clients without crossing back into management for every

query.

If the primary goes offline, the secondaries carry on serving their last copy. How long they do that is controlled by the SOA expiry value. That timer belongs in monitoring; discovering it when an otherwise healthy secondary starts returning SERVFAIL would be a particularly bad day.

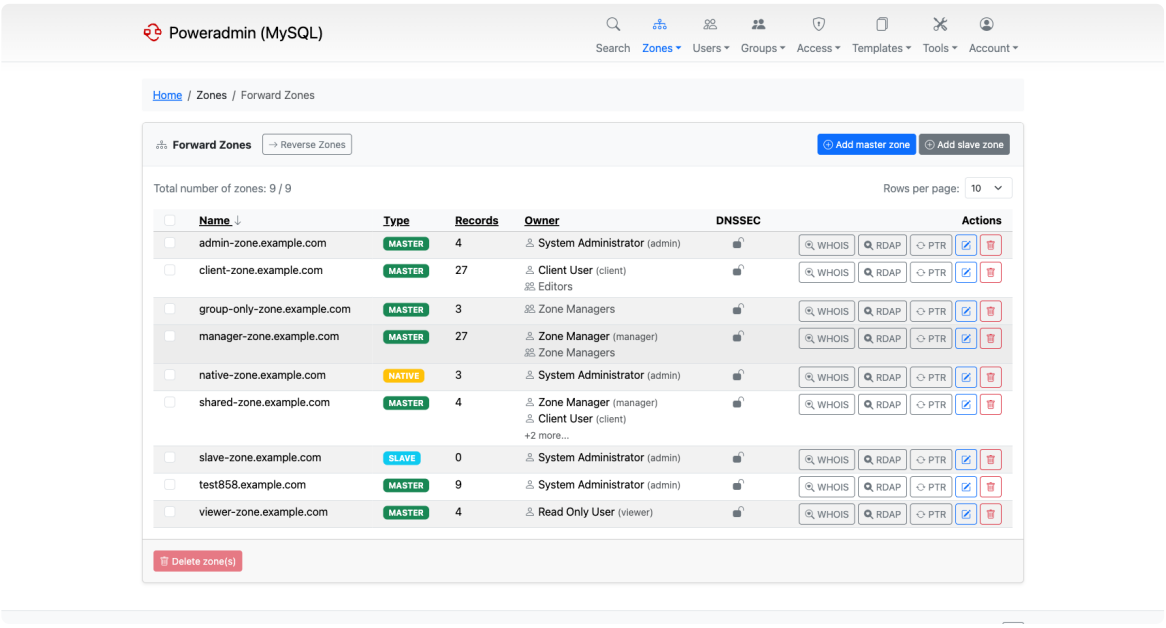
Poweradmin is not a PowerDNS product

The naming is confusing. **Poweradmin is an independent community project**, not a PowerDNS or Open-Xchange product. It is GPLv3 software and is primarily maintained by Edmondas. PowerDNS sells its own commercial management tools; Poweradmin is a separate web application built to manage the open-source Authoritative Server.

During an assessment, I would treat them as two suppliers. The DNS server has a company and commercial support behind it. Poweradmin has a much smaller maintainer base. They need separate upgrade plans, security monitoring and recovery tests even when they are installed together.

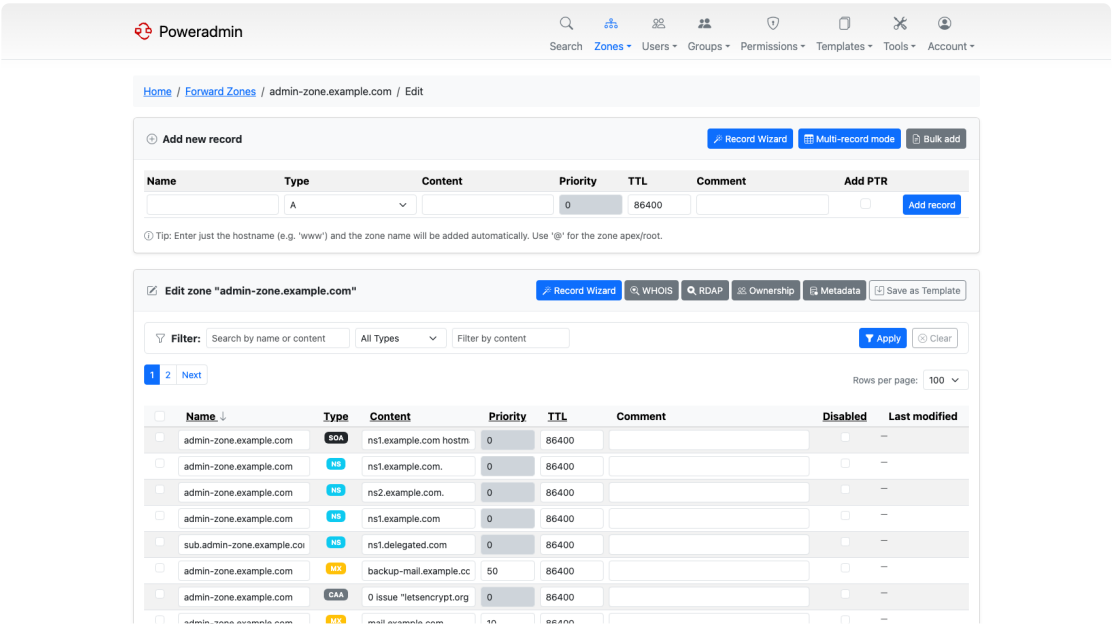
Poweradmin can connect to PowerDNS in two ways. The traditional setup gives it direct access to the PowerDNS SQL database, with the HTTP API used for some operations such as DNSSEC. Since version 4.3 it can instead run in **API backend mode**, where all DNS changes go through the PowerDNS REST API. It still needs its own database for users, groups, permissions and zone ownership.

API mode is my pick for the management-domain build. It reduces the connection between the web application and the DNS database to an HTTPS API path. It does not make the credential low risk: the Poweradmin documentation warns that the PowerDNS API key grants full control of the server. The API stays on a management address, with restricted source IPs, HTTPS and the sort of rotation schedule given to other privileged service credentials.



Forward zones in Poweradmin. Zone type, owners, groups and DNSSEC state are visible in the list. Screenshot from the [Poweradmin documentation](#).

The zone list was the screen that sold me on the interface. Ownership is right there beside the record count and zone type. A shared zone can belong to a group, a user or both. The person making a change does not need to understand how the application stores that relationship to see who is responsible for it.



The zone editor, with inline record entry, filtering, ownership and metadata controls. Screenshot from the [Poweradmin documentation](#).

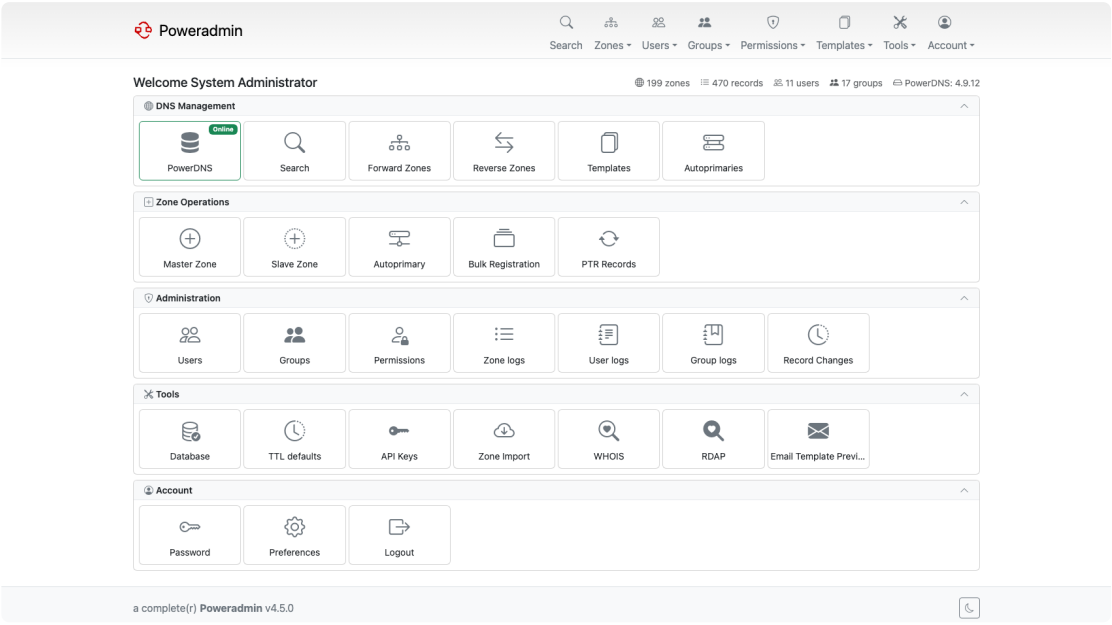
The editor is much closer to how I think about DNS records than the Windows DNS console. Records are rows. I can filter by name, content or type, edit them inline, add comments and add several at once. There are tools for PTR records, WHOIS, RDAP, metadata and templates without burying every action in a different dialogue.

The access model has had quite a lot of work too. Users can authenticate locally or through LDAP, SAML or OIDC. TOTP MFA is available. Permissions are collected into templates and cover individual operations such as viewing a zone, editing records, creating a primary or secondary zone, deleting zones, managing templates and reading logs.

Groups own zones and have their own permission template. OIDC or SAML claims can populate those groups at login, so an existing `dns-editors` identity group can become the source of access instead of another manually maintained user list. Poweradmin can also enforce group-only zone ownership if individual ownership is not wanted.

There are a few traps in the [permission documentation](#). The uberuser permission bypasses everything else. Anyone who can create users or edit permission templates may be able to grant themselves wider access. Permissions from personal templates and group membership are combined, so a stray group membership adds rights; another group cannot take them away.

The roles can stay boring: administrator, zone editor, viewer and auditor. Very few people should be able to edit the templates themselves. Audit logs should go somewhere the DNS administrators cannot alter.



Poweradmin's dashboard brings DNS operations, permissions, logs and API keys into one place. Screenshot from the [Poweradmin documentation](#).

This dashboard should be nowhere near the ordinary DNS clients. It belongs behind the management network, TLS, SSO and MFA. The Authoritative Server's raw API should listen on localhost or a dedicated management interface and use `webserver-allow-from`. If Poweradmin and PowerDNS are on different hosts, a tightly restricted reverse proxy with mutual TLS would be reasonable.

How the zones would move

The management domain would host a **hidden primary**. This is the only writable copy. Ordinary clients would not query it, and it would not need to be listed in the zone's NS records.

Each security domain would run one or two authoritative secondaries for the shared zones. The local resolver—whether that is PowerDNS Recursor, BIND, Unbound, Windows DNS or Samba—would send queries for those zones to the local secondaries.

A change would take this path:

1. An operator or automation updates a record through Poweradmin.
2. Poweradmin sends the change to the Authoritative Server API.
3. The zone serial changes and the primary sends NOTIFY to its secondaries.
4. Each secondary checks the SOA and transfers the new zone.
5. Clients continue using DNS inside their own domain.

Every security domain gets its own **TSIG** key, probably HMAC-SHA256. A key should be allowed to transfer only the zones that domain is permitted to receive. Source address rules would still be present, but an IP allow-list should not be the only protection for a full copy of an internal zone.

The firewall rules are quite narrow. The primary sends NOTIFY to known secondaries. The secondaries can query and transfer from the primary over port 53; AXFR uses TCP. Nothing in a downstream domain needs the Poweradmin interface, SQL database or PowerDNS HTTP API.

With a small number of zones, explicit configuration is fine. When the list starts changing often, PowerDNS supports RFC 9432 catalog zones. A secondary consuming the catalog can add and remove member zones automatically. PowerDNS also has **autoprimary** provisioning, previously called supermaster, but a zone created that way is not removed automatically. Catalog zones handle that lifecycle better.

AXFR copies the entire zone. It cannot send one set of records to domain A and hide those records from domain B. If the domains have different need-to-know rules, they need separate zones, delegated subzones or separate authoritative instances. Internal DNS is often a very good inventory of systems, locations and naming conventions. The contents need classifying before anyone waves a transfer through as infrastructure plumbing.

What else could do the job?

There is no shortage of good open-source DNS servers. The differences are mostly about the surrounding operating model.

BIND

BIND can do this design. It supports primary and secondary zones, TSIG, NOTIFY, views and catalog zones. If an organisation already has BIND configuration in source control, automated checks and a reliable deployment pipeline, replacing it may achieve very little.

The difference is how the work reaches the DNS server. PowerDNS expects structured backends and has an HTTP API. Put Poweradmin in front and the control plane already has search, ownership, role-based access and audit logs. Building the same experience around BIND is possible, but it is a separate project.

PowerDNS is not automatically safer because it has an API, nor is BIND unsafe because it has text files. A bad Poweradmin permission template can do damage much faster than a carefully reviewed zone-file change. I am interested in PowerDNS because it matches how we want people and automation to manage records, not because BIND has somehow stopped being a good DNS server.

Samba and Windows DNS

Samba's internal DNS server is there to support Active Directory. It stores its zones with the directory and accepts Kerberos-secured dynamic updates from domain members. It deliberately does not provide caching recursion, shared-key TSIG, stub zones or zone transfers. Queries outside the AD

zones are forwarded to another resolver.

Samba can use **BIND9_DLZ** when the AD environment needs more of BIND's features. The DLZ plugin reads the Samba database directly and must run on the domain controller. Samba's own documentation recommends an external DNS server in front for high-traffic deployments rather than treating BIND9_DLZ as the primary service for every query.

Windows AD-integrated DNS also has features that should not be casually discarded. Any suitable domain controller can accept an update, zones replicate through Active Directory, and secure dynamic updates use the identity and ACL of the computer or account changing a record. That behaviour is important for workstation registration, DHCP and domain-controller discovery.

My split would be to leave the AD namespace on Windows DNS or Samba. Shared infrastructure zones would live in PowerDNS. AD DNS could conditionally forward those shared zones to the local PowerDNS secondaries, or the namespace could use normal DNS delegation. That avoids trying to make PowerDNS impersonate Active Directory DNS.

Knot, NSD, CoreDNS and Unbound

Knot DNS and NSD are strong authoritative servers. Both are particularly attractive on a serving tier where speed, stability and a small attack surface are more important than a friendly control plane. They could consume transfers from a PowerDNS primary perfectly well. Poweradmin would still manage the primary, while Knot or NSD answered in the downstream domains.

CoreDNS is the odd one in the comparison. Its plugin model makes it excellent for Kubernetes and custom service discovery. I would choose it for that plugin ecosystem, not as the most direct replacement for a traditional internal DNS management system.

Unbound belongs on the recursive side. It is a lean, validating resolver and would pair well with any of the authoritative servers above. Comparing it directly with PowerDNS Authoritative is like comparing a librarian with the publisher: both handle the same material, but their jobs are different.

The test deployment

My trial would put Poweradmin and a PostgreSQL-backed PowerDNS hidden primary in the management domain. Poweradmin would use API backend mode and the existing identity provider. The API would be reachable only from Poweradmin.

Each test security domain gets a local secondary, a unique TSIG key and an allow-list covering only that server. Critical zones get two secondaries per domain. Monitoring compares the SOA serials and alerts on transfer failures or a secondary approaching expiry.

Then I would break it on purpose: block NOTIFY, block TCP 53, use the wrong TSIG key, stop the primary for a day, restore the databases, roll back a bad record, then add and remove a catalog member. Watching those failures is a better test than watching one successful A-record change appear in three places.

Backups need both sides of Poweradmin. The PowerDNS database holds the zones. Poweradmin's database holds users, groups, permissions, ownership and its own audit data. Restoring only the zones produces a working DNS server, but not the management system that was controlling it.

Permission to edit records should be separate from permission to create or delete zones. Changing an A record can cause an outage. Creating or removing a zone changes an authority boundary and can redirect a whole part of the namespace. Poweradmin has separate permissions for these jobs, so there is no reason to hand them out together.

Where I landed

I came away impressed. The Authoritative Server uses ordinary DNS protocols, has sensible storage options and does not force recursion into the same process. Poweradmin puts a genuinely good operator interface on top, including the ownership and access controls that are usually missing from small DNS tools.

I would not replace a healthy BIND deployment just to avoid looking at a zone file. AD-integrated zones can stay where they are too. For new shared infrastructure zones, though, PowerDNS is now high on my list.

The design for our security domains is almost boring: make changes in one protected place, transfer complete zones with a separate key for each domain, and answer locally from read-only secondaries. Boring is a fine quality in DNS.

References

- PowerDNS company history
- PowerDNS product documentation
- PowerDNS Authoritative Server and backends
- PowerDNS primary, secondary and autoprimary operation
- PowerDNS TSIG
- PowerDNS catalog zones
- PowerDNS HTTP API security
- Poweradmin source and screenshots
- Poweradmin users, roles and ownership
- Poweradmin groups
- Poweradmin API backend mode
- BIND 9 primary and secondary architecture
- Samba internal DNS backend
- Samba BIND9_DLZ backend
- Microsoft Active Directory-integrated DNS zones
- Knot DNS

- NSD
- CoreDNS manual
- Unbound