Modern zone replication with LMDB and Lightning Stream

@kevin@km6g.us (Fediverse)
@kevin:km6g.us (Matrix)

Kevin P. Fleming – FOSDEM 2025

What does an authoritative server need?

- **Zone Data** – resource record sets (RRsets)
- **Zone List** – the names of zones for which the server will respond with **AA=1**
- **Zone Metadata** – additional information the server uses to manage zones
  - signing keys if online signing is used
  - servers to **NOTIFY** about **SOA** changes
  - **TSIG** keys used for **UPDATE** or other operations
  - access controls (**AXFR**, **UPDATE**)

Why is replication needed?

- Fault tolerance – hardware/software/network failures (BGP!)
- Response time – locating authoritative servers closer to resolvers
- Load sharing – high query volumes may overload single servers

In the beginning...

- (where ‘beginning’ is ‘before 1987’, when RFC 1034/5 were published)
- ‘master files’ contained **Zone Data**
- the **Zone List** was literally the contents of the directory containing the master files
- there was no **Zone Metadata** of any significance
- ‘primary (master)’ and ‘secondary (slave)’ concepts didn’t exist yet; all servers functioned identically, and the **Zone Data** was read-only
- **Zone Data** was replicated between servers by copying master files

RFC 1034/5 introduce replication

- RFCs 1034 and 1035 introduced the concept of replication, using the **AXFR** query type
- As with copying master files, this only replicated **Zone Data**
- This meant that a **Zone List,** and **Zone Metadata,** had to be configured on each server which offered a replica of the **Zone Data**
- Such servers would issue periodic **SOA** queries to the configured ‘source’ server to see if the serial number had changed
- When the response indicated a change, the server would use **AXFR** to obtain a fresh copy of the **Zone Data** and persist it in the filesystem

RFC 1996 introduces NOTIFY

- Since polling for **SOA** changes could be costly, or slow, or both, a mechanism was standardized which allowed the source server to *notify* the replica servers that the zone contents may have changed
- This RFC introduced replication concepts:
  - ‘primary master’ servers (the root of the dependency graph)
  - ‘master’ (now ‘primary’) servers (sources of **Zone Data**)
  - ‘slave’ (now ‘secondary’) servers (sinks of **Zone Data**)
  - ‘stealth’ servers (unlisted servers which could act as both ‘master’ and ‘slave’)

RFC 1996 introduces NOTIFY (continued)

- For the first time, **Zone Metadata** was required on the ‘primary’ server in some cases, as it needed to know where to send **NOTIFY** queries (if any **NOTIFY** destinations were not included in the **NS** RRset of the zone, e.g. ‘stealth’ servers)
- This introduced *tight coupling* between the servers, as they both had to know of the others’ existence and location
- This also introduced *eventual consistency* to the system, as the exact moments when all the replica servers would have an updated copy of the **Zone Data** were not synchronized

That’s all we need, right?

- In many cases, yes!
- If the set of servers responding to queries for the zones is relatively static, *tight coupling* is not an issue and can be addressed using configuration management tools.
- If the TTLs of the RRs in the zones are relatively long (60 seconds or longer), *eventual consistency* is not an issue as there is no expectation of ‘rapid updates’
- The combination of **AXFR** and **NOTIFY** can be used to build graphs deeper than two levels, if desired
- There is no requirement that all the servers in the graph be using the same software, since the replication is performed using the DNS protocol itself

Why would we need anything else?

- Transferring **Zone Data** using **AXFR** can be slow and costly; if the RDATA for a single RR in the zone is modified, the entire contents of the zone must be transferred, which could be tens (or hundreds) of thousands of RRs (**IXFR** can be used to mitigate this, but places additional burdens on the ‘primary’ servers, and may not be possible in graphs deeper than 2 levels)
- Some applications, especially ‘cloud native’ deployments, involve dynamic creation and deletion of RRs which must be available in all authoritative servers nearly simultaneously (*strong consistency*)
- Some applications, again including ‘cloud native’ deployments, involve *ephemeral* server instances, so *tight coupling* is an obstacle

Let’s use a database!

- Many authoritative server implementations support popular databases, and some of those databases provide native replication (‘clustering’)
- Depending on the configuration of the database, both *eventual* and *strong* consistency models may be available
- Using a database for replication eliminates any need for **Zone Metadata** related to replication, so there is no *tight coupling* between the authoritative servers
- However… there will be *tight coupling* between the database nodes, in most cases

Is that the answer?

- For you, it might be!
- If your organization already has a well-managed clustered database solution, operated by some team separate from the DNS team, leveraging it may be the right choice
- If your organization doesn't already have that… your goal of providing a geographically distributed and redundant database (DNS itself) was satisfied by adding another geographically distributed and redundant database underneath it
- Additionally, the commonly-available databases which serve this need (MySQL/MariaDB, PostgreSQL) aren't a great fit for DNS data; they have 100x more capability than is required

Using a clustered database

- The database will contain, and replicate, the **Zone Data**, **Zone List**, and **Zone Metadata**
- All DNS servers using the same database will serve the same role (primary, secondary, etc.)
- Dynamic changes to RRs can be handled by one, or more, servers in the cluster, if the database manages conflicts

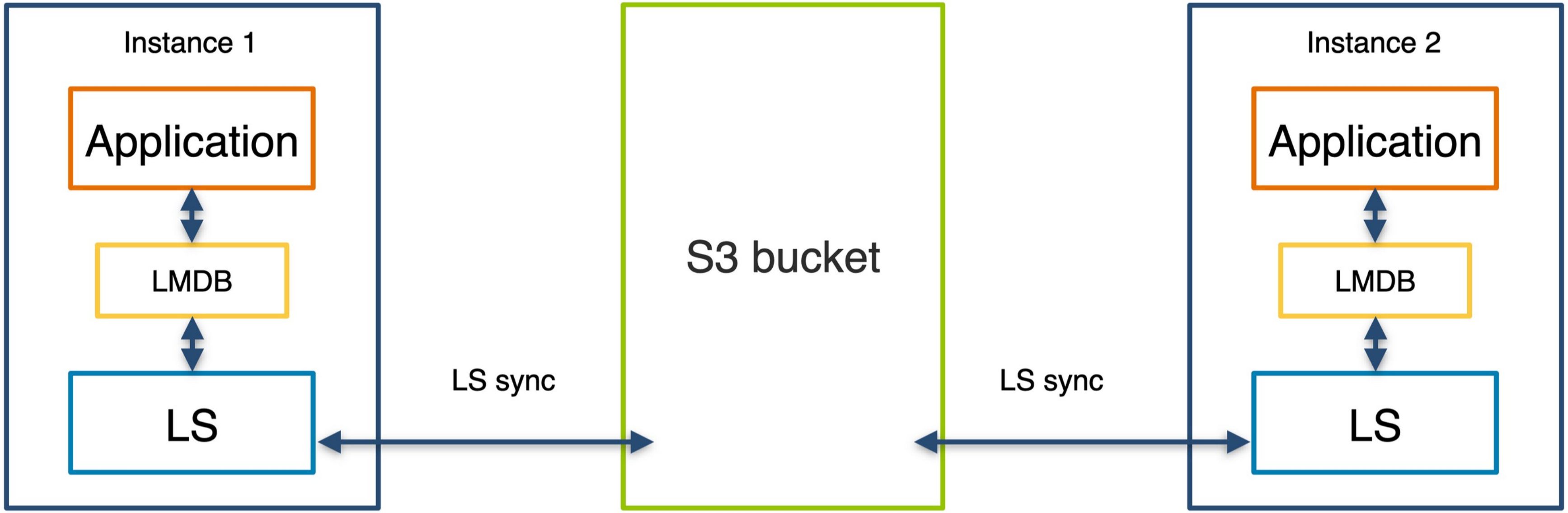
Another option: LMDB

- Lightning Memory-Mapped Database
- Single server (non-clustered) high performance database, part of the OpenLDAP project
- Supported by PowerDNS Authoritative Server
- Holds **Zone Data**, **Zone List**, and **Zone Metadata**

LMDB replication: Lightning Stream

- Replication tool for LMDB
- One LS instance per LMDB instance (one per authoritative server)
- LS serializes LMDB contents into compressed ‘snapshot’ files, which are stored into an S3-compatible object storage bucket
- LS instances monitor the bucket for new ‘snapshot’ files, download them, and apply them to the local LMDB
- As with other replication options, this provides *eventual consistency*
- Unlike other replication options, there is no *tight coupling* as the servers have no knowledge of each other

LMDB replication: Lightning Stream



Benefits of using LMDB+LS

- All servers in the dependency graph are independent; their ‘meeting point’ is the object storage bucket
- Servers can be added or removed from the dependency graph at any time
- Bandwidth requirements for replication are quite low… but since the servers poll for changes, they can be higher than using **NOTIFY**+**AXFR**
- ‘Multiple master’ models are easily supported, since LS uses the timestamp of each RR to decide which version of the RR is ‘current’

Risks of using LMDB+LS

- The object storage bucket is a single point of failure; choose the storage provider wisely!
- Troubleshooting replication failures can be more difficult, as the only tool available for inspecting the contents of snapshots is LS itself
- If your DNS deployment needs to support rapid distribution of dynamic changes, the LS polling frequency will need to be quite low
- Replication is proprietary to PowerDNS Authoritative Server

Using LMDB+LS

- With PowerDNS Authoritative Server 4.8 (or later) installed, configure the LMDB backend (see the notes on the LS website for details of the required configuration) – don’t allow the LMDB to be created before the configuration is in place
- Build LS from source (no packages available yet), install it into a suitable location, and use your preferred mechanism for running it as a service
- Configure LS to connect to the object storage bucket
- Run ‘lightningstream sync’ on the source server
- Run ‘lightningstream receive’ on the replica servers

Summary

- Provides many of the benefits of using a clustered database, with dramatically lower resource consumption
- Support *ephemeral* server deployments
- Does not require administration of a complex database system

Questions Welcome!