ebay

How To Take Prometheus Planet Scale - Massively Large Metrics Installations

Vijay Samuel - Architect, Observability

Agenda

Observability @ eBay

Where It Began

How We Solved It

Lessons Learnt

What’s Next

Questions

Observability @ eBay

1

Instrument

Developers through open source libraries shipped with managed frameworks instruments metrics, logs, traces.

2

Onboard

Through eBay's Cloud Console, end users let the platform know that their application needs to be monitored.

3

Harvest

Agents deployed across eBay's data centers collect user instrumented data and ship them into Sherlock.io - eBay's observability platform.

4

Alert

Users can set up alerts (threshold based and anomaly detection model based) to be notified of issues in their applications.

5

Visualize

Sherlock.io's console allows users to visualize their data.

Scale

Endpoints Scraped	Ingest Per Second	Active Time Series	Queries Per Second	Raw Metric Retention
1.25 Million	41 Million	2 Billion	8000	1 year

Where it began…

The year… **2018**. Cloud Native is catching up within eBay along with Prometheus becoming more and more popular.

Stand alone Prometheus are now used by individual teams with a growing ask to support Prometheus as a centralized Observability platform offering.

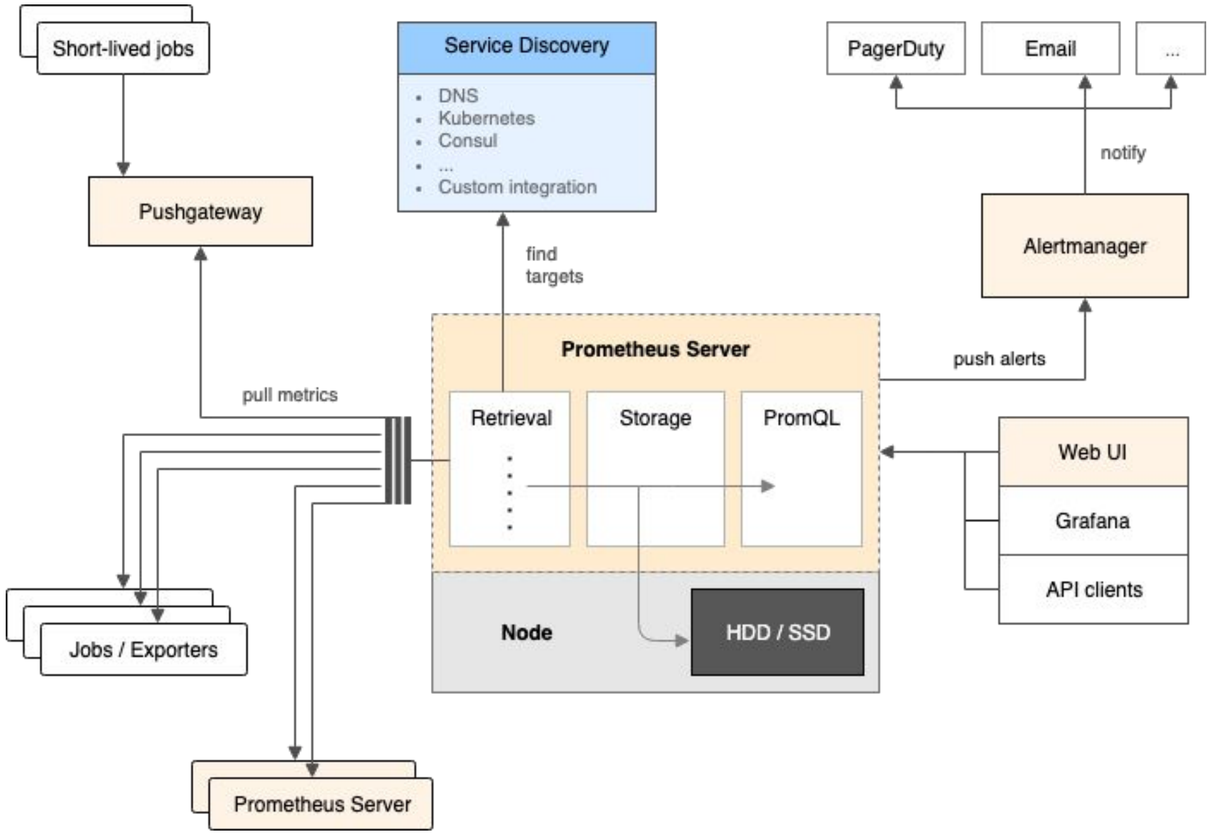
Legacy System

- Modified version of OpenTSDB
- Custom protocols, DR inefficiencies, data inconsistencies
- HBase scaling, availability challenges
- Excessive tribal knowledge

The Need

- A system that scales a lot better
- Support community focused ingest
- Richer ad-hoc query support
- Support cloud native monitoring

Prometheus



Highlights

- Open Source CNCF backed TSDB
- Offers PromQL as a query language
- Simple to operate - single server
- Offers instrumentation clients
- Standard exposition format to announce metrics
- Efficient storage of series with delta-delta encoding

Challenges With Adopting Prometheus

No Support for HTTP Push (at the time)

Necessary for feature parity with legacy platform.

Prometheus didn’t support remote write.

Scaling Is Hard

More targets to monitor requires vertical scaling of Prometheus.

Deploying multiple Prometheus to get around vertical limits has operational overheads.

Federated View of Data Was Complicated

Setting up federation over 100s of Prometheus instances is tedious.

Initial Goals

Keep It Simple

Solving replication is hard. Instead do fanout writes.

Do deduplication at query time.

Use Prometheus TSDB as-is.

Integrate Tightly With Kubernetes

Discover shards using the Kube API Server. An Operator can manage the clusters.

Alerting and Recording

End users must be able to onboard recording and alerting rules.

Simple Tenanting

Route “namespaces” to dedicated clusters of Prometheus TSDBs

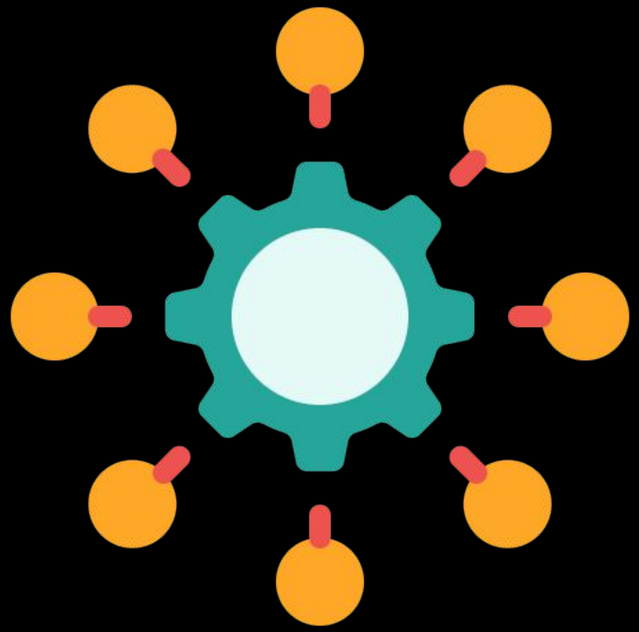
Support PromQL and Grafana

End users must be able to fully leverage Grafana to build out their user experiences.

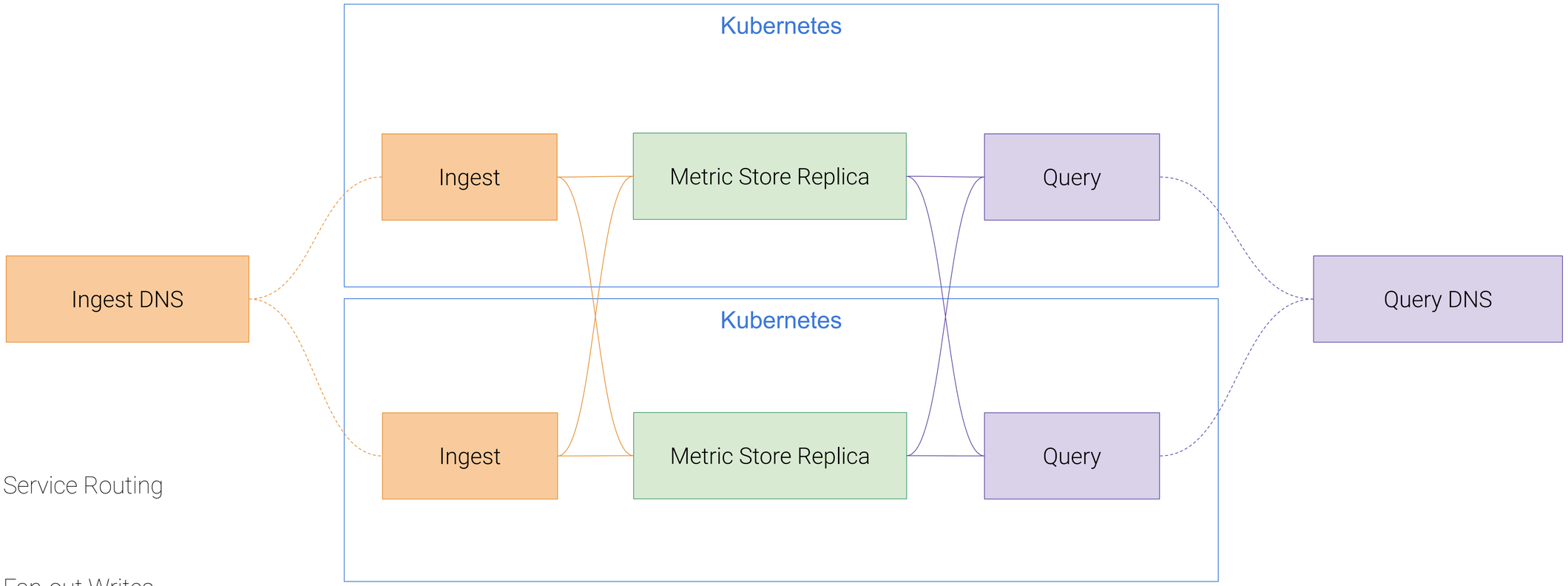
Support Twice The Cardinality Limits Of Legacy Platform

Boy! Were we surprised with the outcome :)

The First Pass - Centralized



Data Flow & Availability



Service Routing

Fan-out Writes

Aggregated Reads

Reads / Writes are routed based on tenant

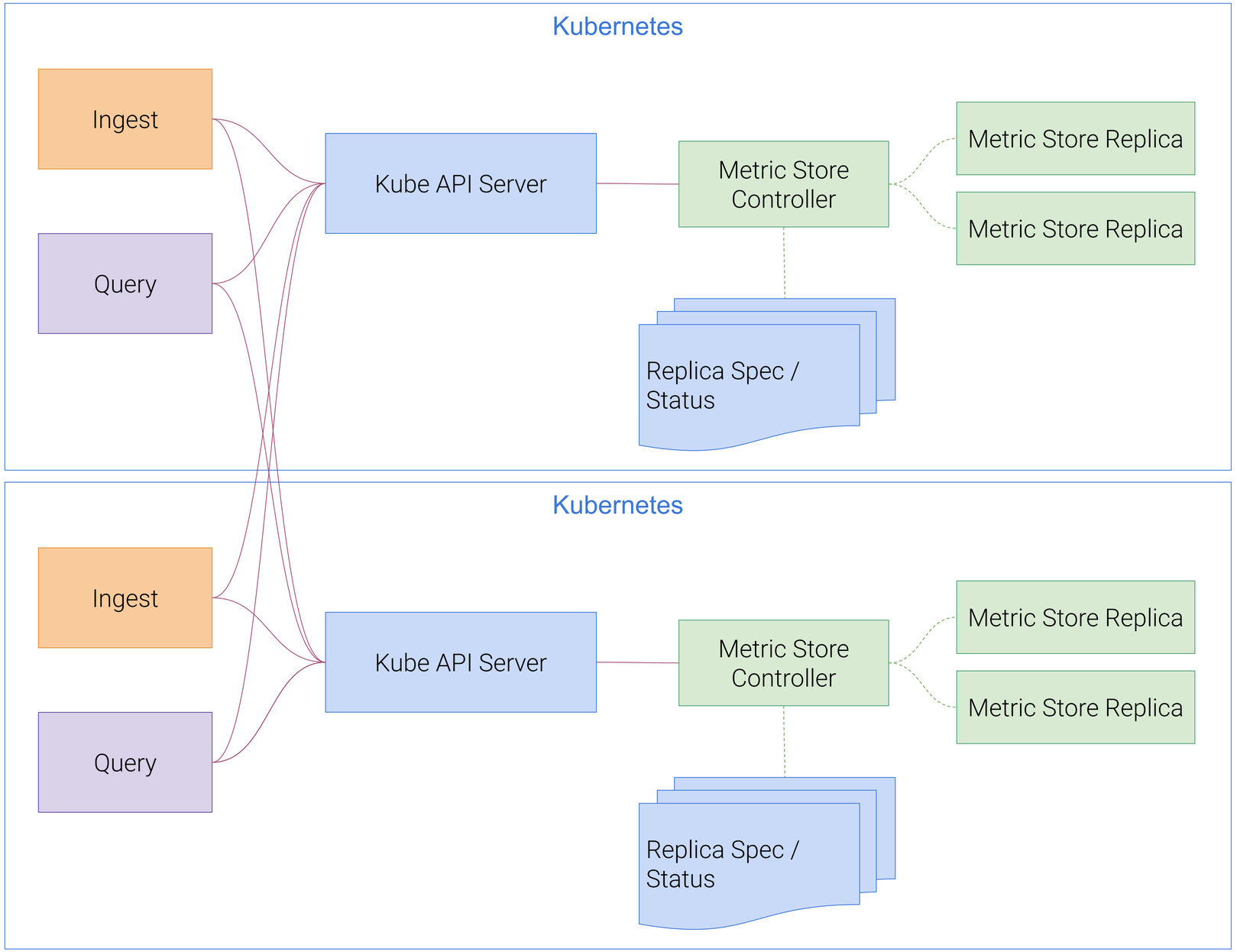
Adjust tenant availability by controlling number of deployed replicas across Kubernetes clusters

Storage Discovery & Management

Watch Kube Resources

Manage Kube Resources

Replica resources contain deployment, partitioning, and tenanting details



Challenges Still…

Can’t Keep Up With New Use Case Onboarding

Platform grew 100% YoY many years in a row.

2 → 4 → 8 →16 →24 → 40M/sec ingest

Hard To Support High Cardinality

A single TSDB cluster has horizontal limits before deteriorating query performance

Tenant Based Query Routing Is Flawed

Reserved _***namespace***_ keyword always required.

Prevents supporting things like Labels API

Too Many Dependencies Involved

Load balancers, cross region networks involved in monitoring path.

Introducing Aggregators…

Drop Labels Before Series Hits Storage

Stream based aggregation of raw series.

Raw Labels Aren’t Always Necessary

Some labels like containerID churn a lot and aren’t queried as much.

High Cost To Store Raw Series

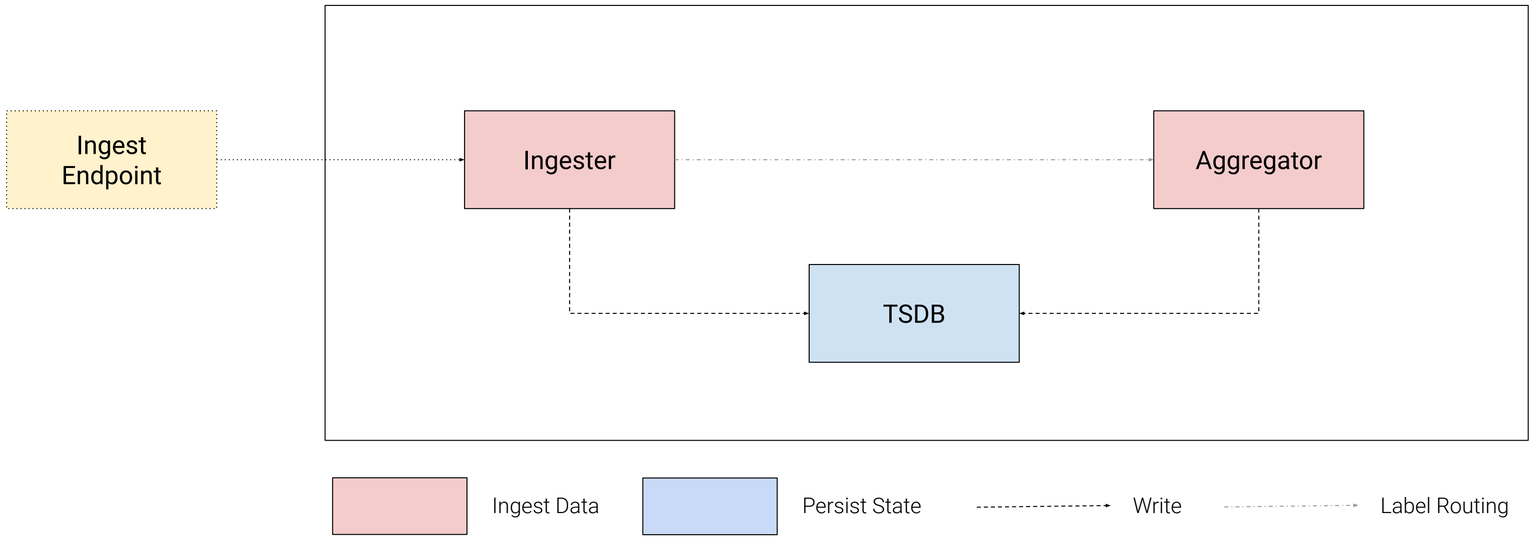
Each series on Prometheus Head block has a cost that adds up as we store more raw series.

Higher the cardinality, greater the storage cost.

Massive Savings!

60% CPU, 80% Memory savings as compared to doing recording rules to drop cardinality

Ingest Routing - Aggregation



Let’s Take Things Planet Scale…

Unveiling Of Monarch

Google published a white paper on Monarch; it's "planet scale time series database"

Like every curious engineer we went - "is it possible to replicate this inside of eBay"

Stating The Obvious…

- We don't have the same awesome tech that Google has :)
  - Substitutions need to be made with Open Source technology
- Spec management is critical
  - GitOps
- The white paper is all that we had to go with
  - Some aspects weren't fully talked about.

Anatomy Of A Planet Scale Installation

Data is closer to the source

- Deployed per Region / AZ / Kubernetes cluster
- Number of tiers are composable based on scale requirements

Rollups Get Bubbled Up

- Leaf - Stores raw data
- Zone - Rolled up data for the given region
- Root - Higher level roll ups (user queries)

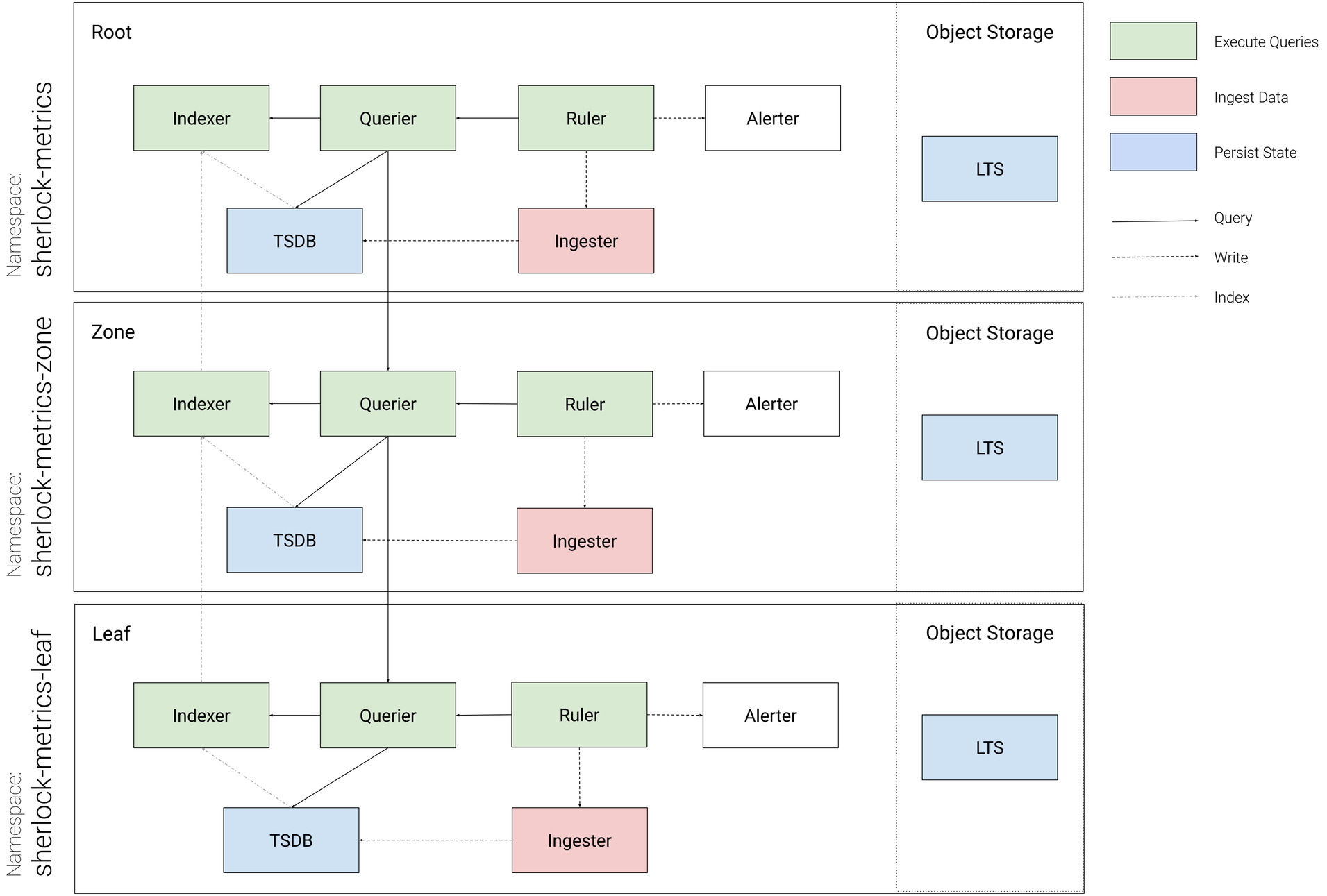
Queries are federated

- Field Hint Indices provide knowledge on where a series might reside
- Scatter queries to only matched targets.
- User is oblivious to the data layout

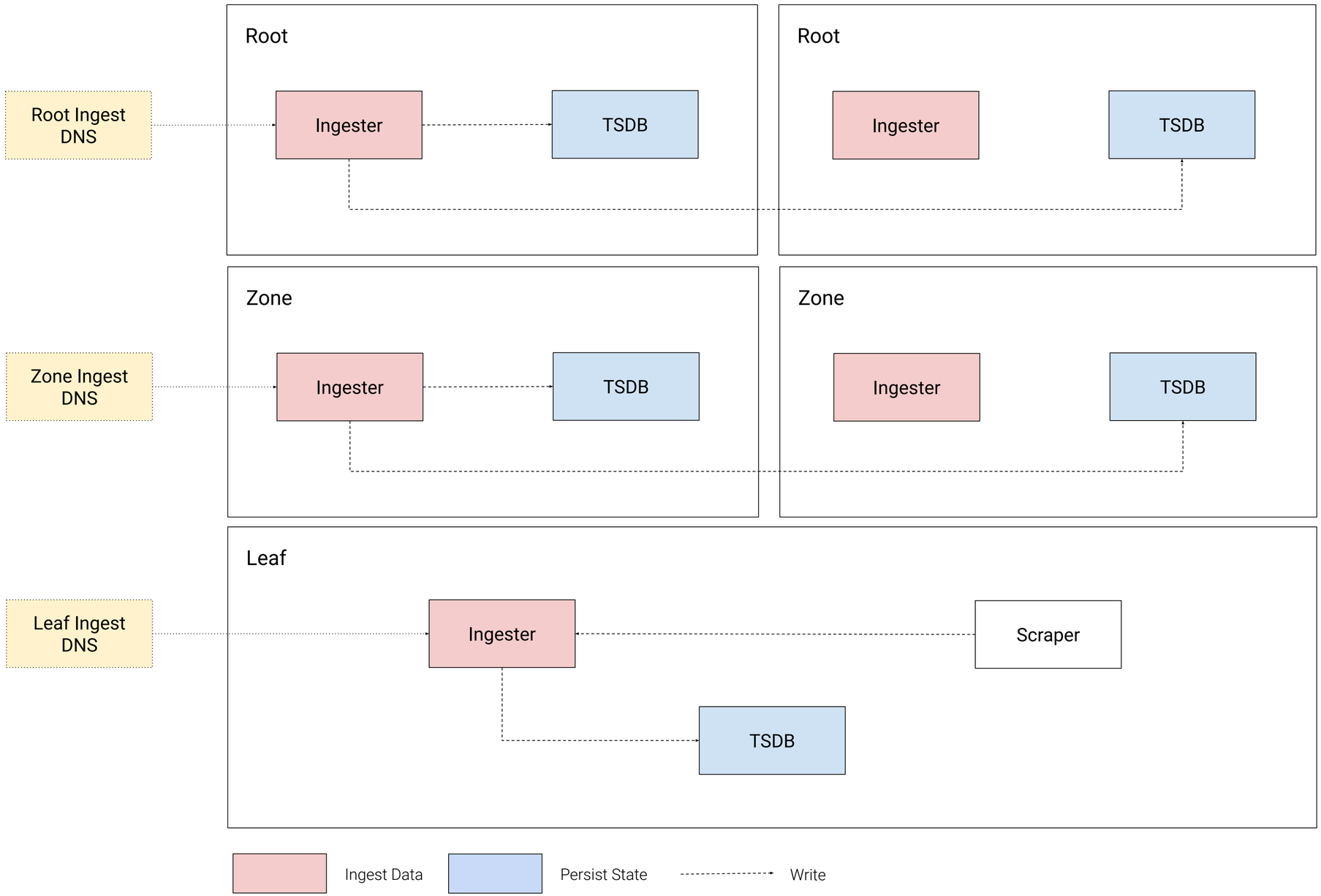
Delivery through GitOps

- Manage specs of leaves/zones/root on Git
- Control loops watch over Git and deliver the changes into the cluster

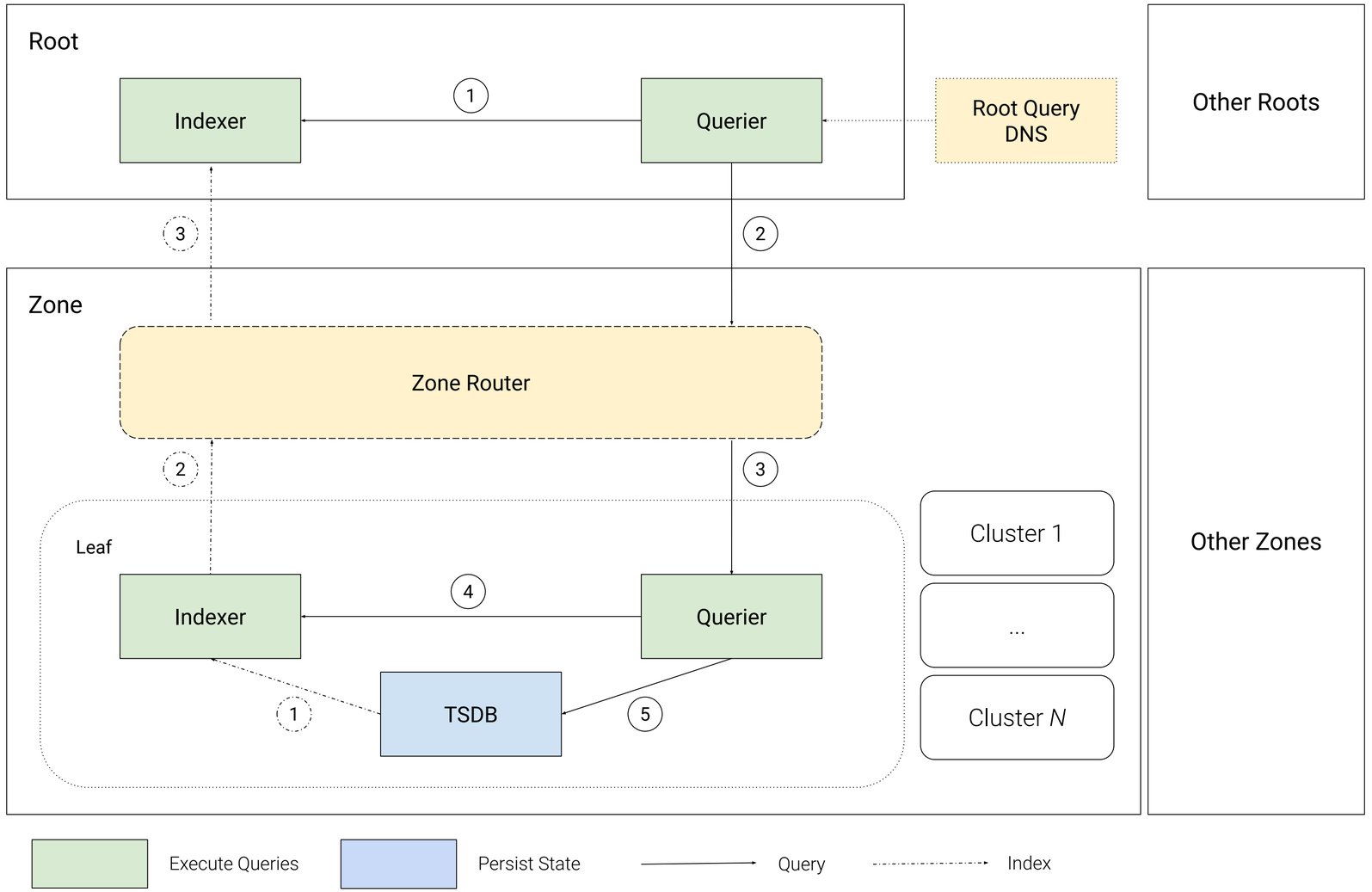
Deployment Architecture



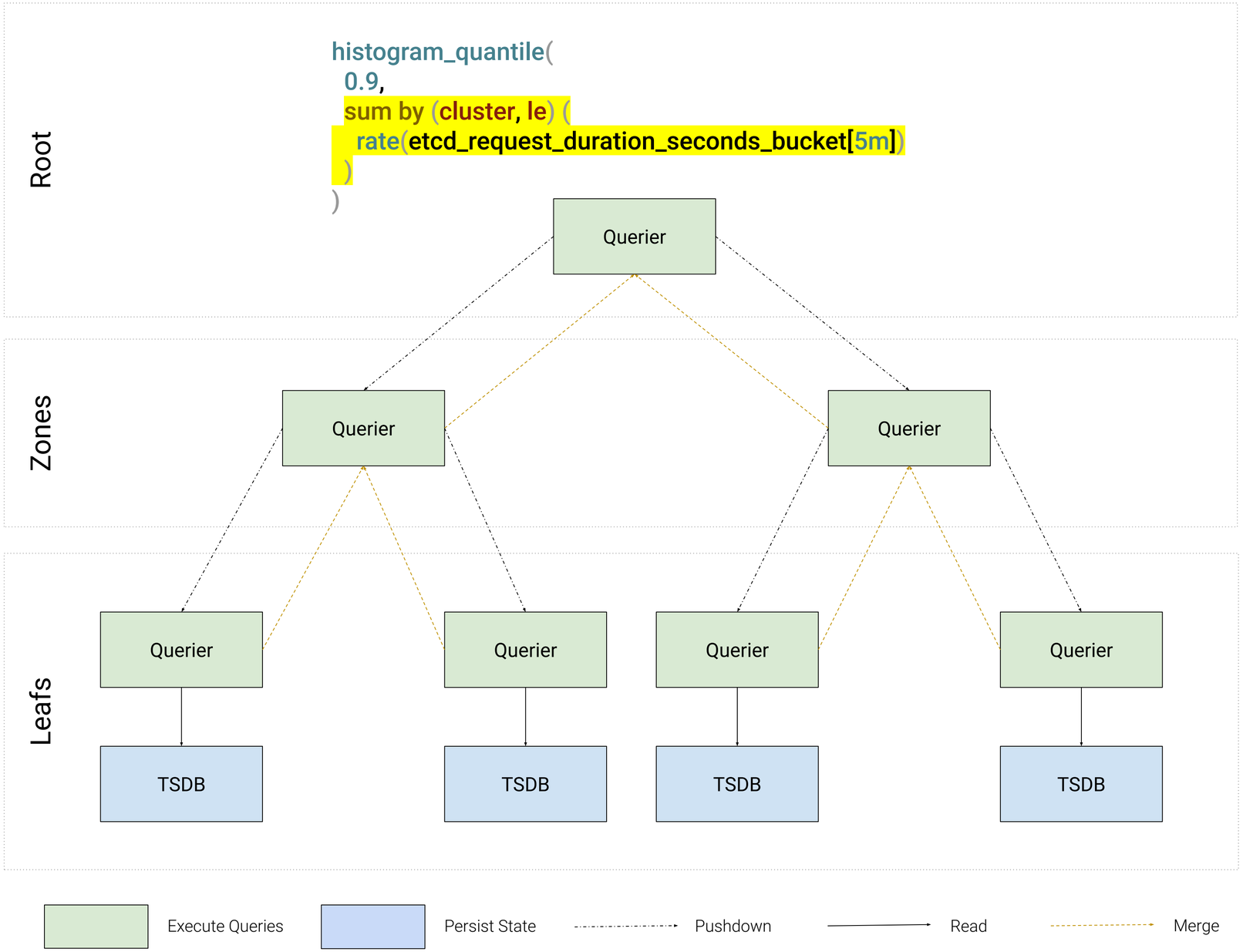
Ingest Routing - Overview



Query Routing - Overview



Query Pushdown & Projection



What Did We Get Out Of This…

Better Scaling Capabilities For High Cardinality

Leafs per Kubernetes cluster or AZ can break down the cardinality into more manageable chunks.

Zones have rolled up data that is easier to query.

More runway for growing needs of end users.

Better Ingest/Query Performance

Internal gRPC based optimized ingest and query APIs cheaper on network.

Query pushdown + projection ensures the root only sees rolled up data with required set of labels.

Independently Usable Metricstores

Each leaf / zone is now a fully functional metricstore than can be independently used for visualization and alerting.

Fixed Shard Sizing

Horizontal scaling based on zone/leaf installations allows fixed size shards.

Easier to operate.

Lessons Learned

Independently Usable Metricstores

Each leaf / zone is now a fully functional metricstore than can be independently used for visualization and alerting.

GitOps Makes Life Easy!

Ensures changes are reviewed, automated and easy to roll back.

A spec can never be lost.

A massively distributed system can’t succeed without proper spec management!

Internal APIs Aren’t a Bad Thing

Reusing the Prometheus contract end-to-end wasn’t required.

Internal gRPC based API had massive cost savings.

Enabled query push-down and projection.

Continuous Observation And Evolution Is Key

Learning new tricks goes a long way!

Sometimes that rewrite we shy away from is probably necessary ;)

What’s next

Automatic Rollups

Metrics being ingested at the leaf can be automatically rolled up and stored in the zone.

Drop high cardinality labels like pod, node, etc.

Queries can be intelligent to pull data from the right tier.

Support Native Histograms

How would query push down work?

How much would we save in cardinality?

Open source… Maybe?

If you think that this is interesting… let’s chat :)

Questions?

Thank you