Scaling Observability with Loki at Dropbox

Alok Ranjan & Paul Shen

Agenda

01 Introduction & Context

02 Observability Challenges at Dropbox

03 Evaluation of Logging Solutions

04 Why Grafana Loki?

05 Deep Dive into Loki’s Architecture

06 Operational and Scaling Challenges

07 Integration with Dropbox Infrastructure & Cost Optimizations

08 Conclusion & Q&A

Alok Ranjan

Engineering Manager, Storage Platform

- Master's from Carnegie Mellon University
- Prior experience: Big Switch Networks, VMware, Cisco
- Focus: Storage systems, scalable infrastructure, Telemetry
- Interested in AI/ML infrastructure challenges

Paul Shen

Software Engineer, Logging Integration Tracing (LIT) Team

- Work: Deploying Grafana stack, Traces, Profiles
- Personal: ergonomic computer interfaces - terminal apps, LLM chat and voice-to-text, XR glasses

Dropbox

- Founded in 2007
- 700+ million users
- 18+ million paying users
- 1 T+ pieces of content
- Billions of files uploaded per day

Incident Triage Workflow

- **Detection & Alerting:** Automated alerts signal anomalies.
- **Dashboard Metrics:** Review key metrics for a high-level view.
- **Log Analysis:** Dig into logs for detailed error context.
- **Distributed Tracing:** Follow request flows to pinpoint root causes.
- **Resolution:** Implement fixes and validate the remediation.

Unstructured Logs

- **Raw Data:** Logs without a defined schema.
- **Sources:** From first-party code & third-party debug files (e.g., /var/log/dropbox).
- **Contrast:** Unlike structured logs (e.g., Hive records, traces)
- **Use Case:** Real-time troubleshooting.

Problem Statement

- Unstructured logs stored in /var/logs/
- ssh individual box
- Host rotated in 7 days
- Migration from standalone hosts to containers
- Containers are ephemeral

High-Level Requirements

- Provide a secure, ergonomic interface for analyzing unstructured logs
- Replace manual, on-host SSH log analysis for production service owners
- Ingest the complete firehose of DBX production logs without modifying application code
- Lay the groundwork for future integration with logs from acquisitions and corporate assets

Reliability Requirements

- **Retention:** Maintain logs for at least 1 week
- **Throughput:** Support up to 150TB/day of log data
- **Latency:** Achieve p99 ingestion latency within 30 seconds and query latency under 10 seconds
- **Availability:** Ensure 99% of log records are reliably stored and accessible

Security Requirements

- Enforce deny-by-default mTLS across all endpoints
- Implement strict access segmentation by service ownership with audited privilege delegation
- Encrypt log storage at rest using robust key management
- Integrate PII detection, filtering, and redaction at both ingress and query stages

Non Goals

- **Log Format**: Don’t mandate changes
- **Observability**: Not going to replace structured logging/tracing/metrics
- **Analytics**: Not for batch or historical analysis
- **Enforcement**: No mandated logging practices

Evaluation Metrics for Logging Solutions

- **Cost:** Total cost of ownership (OpEx/CapEx, contract risks)
- **Performance:** Ingestion rates, query latency & scalability
- **UX & Query:** Rich query engine, familiar Grafana integration
- **Integration:** Ease of connecting with existing observability tools
- **Security:** Data protection, sensitive data exposure risk

Do Nothing (Status Quo)

- Overview: Continue existing SSH-based log analysis
- Pros: No additional investment
- Cons: Manual, non-scalable, inefficient troubleshooting
- Outcome: Inadequate for modern observability needs

Evaluation of Logging Services

Solution	Overview	Pros	Cons	Outcome
Externally Managed SaaS	Fully managed logging service by a third party	Reduces in-house management overhead	High annual cost; potential security risks	Rejected due to cost and security concerns
Managed Cloud Logging	Managed search and logging on a cloud framework	Mature, scalable technology	Higher operational costs; complex configuration affecting UX	Not cost-effective; UX challenges
Self-Hosted Enterprise	Enterprise-grade log management on-premise	Rich feature set; robust vendor support	Expensive licensing and infrastructure demands	Too costly and cumbersome for our scale
Build Your Own Logging	Custom-developed solution	Full control; tailored features	High engineering effort; slow time-to-value	Not viable given rapid open-source advances

Grafana Loki

- **Cost-effective**: Open-source, low TCO
- **High-Performance**: Optimized for DBX-scale log ingestion and querying
- **Grafana Integration**: Native, unified observability interface
- **Scalable Architecture**: Distributed components

Why did we pick Loki: Evaluating Alternatives

- Build vs Buy: Building a new log analysis engine unlikely to make sense given mature observability sector

- SaaS vs self-hosted: SaaS was hugely disadvantaged by
  - Cost
  - Security
  - Integration Challenges

- Index complexity: full text index(esearchd) vs metadata index(loki) vs no index(BFS)

- Loki: a sweet spot in the cost vs UX tradeoff

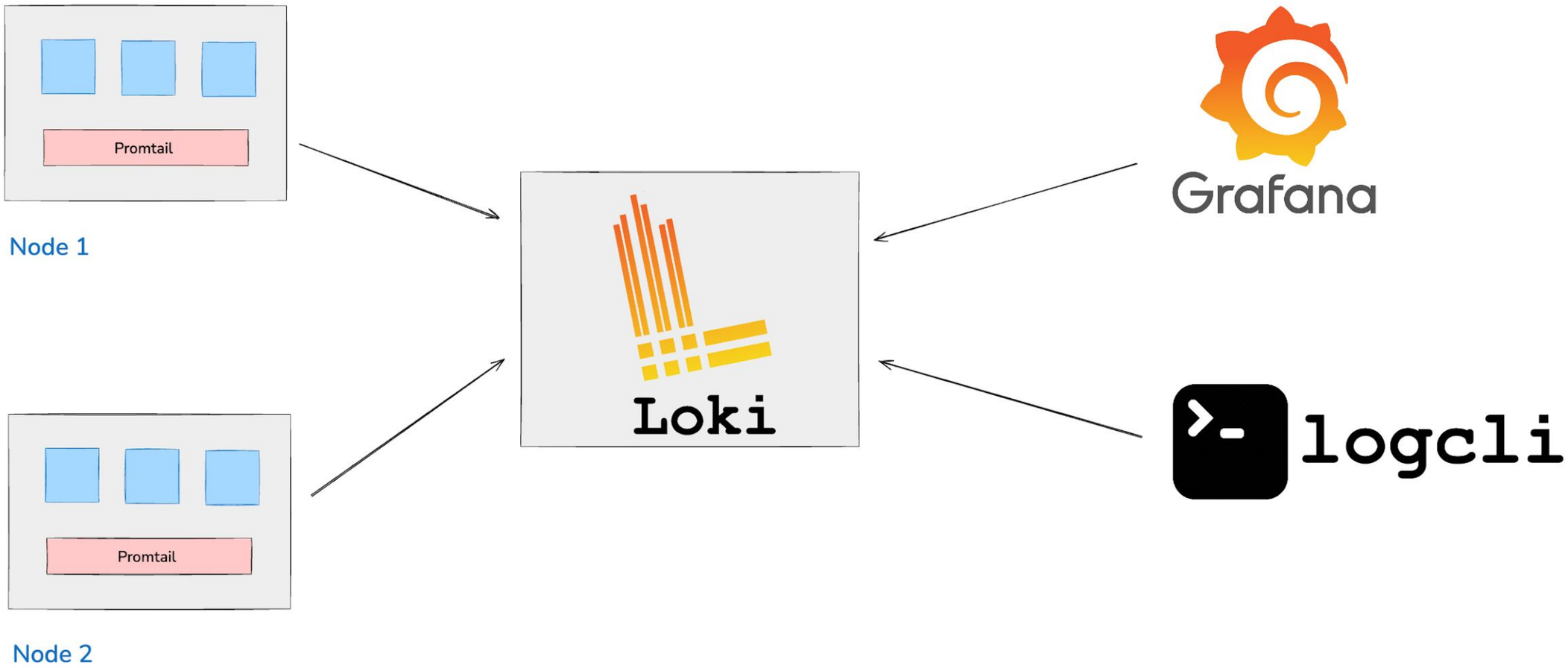
What is Loki?

- Apache 2.0 licensed
- Horizontally scalable
- Highly Available
- Multi tenant
- Prometheus inspired
- Log aggregation System

Why did we pick Loki: Other things we Liked

- Query model conceptually similar to our metrics query language V2
- Active community
- Implemented in golang so easy to patch
- Integrated/Core Grafana UX
- Backed by S3
- Scalable architecture

Architecture

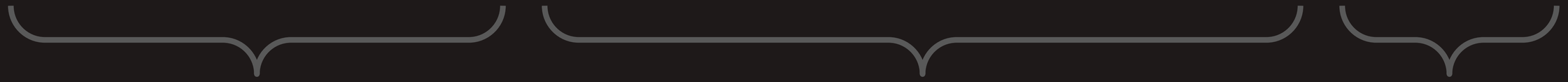


Loki Scalability

- Does not indexes the text of the log
- Loki indexes metadata
- It groups log entries into streams and indexes labels
- Faster ingestion and queries with minimal infrastructure

Logs

2025-02-01T09:02:03.123456789Z {service=”dummy_service”, node_id=”ex_node_1”} GET /about



Timestamp
With nanosecond precision

Prometheus-style **Labels**
Key-value pairs

Content
Log line

Logs - Stream

A **log stream** is stream of log entries with **exact same label set**

```
2025-02-01T09:02:03.000Z    {service=”dummy_service”, node_id=”ex_node_1”}    GET /about
2025-02-01T09:02:04.000Z    {service=”dummy_service”, node_id=”ex_node_1”}    GET /
2025-02-01T09:02:06.000Z    {service=”dummy_service”, node_id=”ex_node_1”}    GET /help
```

```
2025-02-01T09:02:03.000Z    {service=”dummy_service”, node_id=”ex_node_2”}    GET /files/1
2025-02-01T09:02:03.000Z    {service=”dummy_service”, node_id=”ex_node_2”}    GET /files/2
2025-02-01T09:02:03.000Z    {service=”dummy_service”, node_id=”ex_node_2”}    GET /files/1
```

Logs Storage - Chunks

- Streams are stored in separate chunks
- Sorted in timestamp order
- Chunks are filled till they reach a target size or timeout
- Once full, they’re compressed and flushed to Object Store

chunk #1 {service=”dummy_service”, node_id=”ex_node_1”}

2025-02-01T09:02:03.000Z GET /about
2025-02-01T09:02:04.000Z GET /
2025-02-01T09:02:06.000Z GET /help

Logs - Query

Log Stream

Chunks

{service=”dummy_service”, node_id=”ex_node_1”}

T1-T5
T6-T8
T9-T12

{service=”dummy_service”, node_id=”ex_node_2”}

T1-T3
T4-T6
T7-T12

Logs - Query

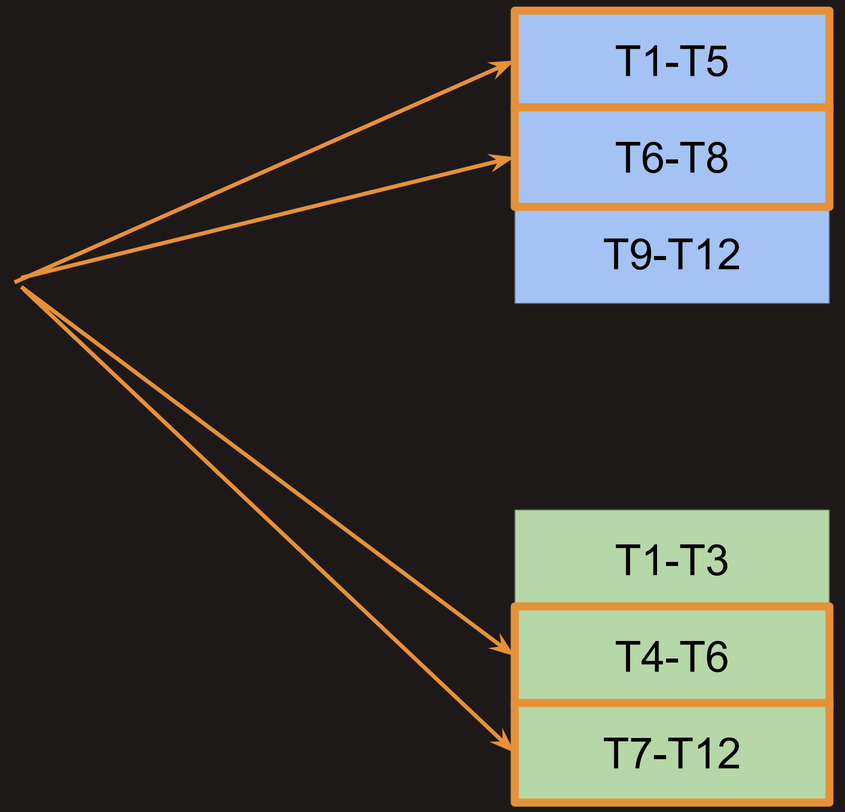
Log Stream

{service=”dummy_service”, node_id=”ex_node_1”}

Query: {service=”dummy_service”} start=T5 end=T7

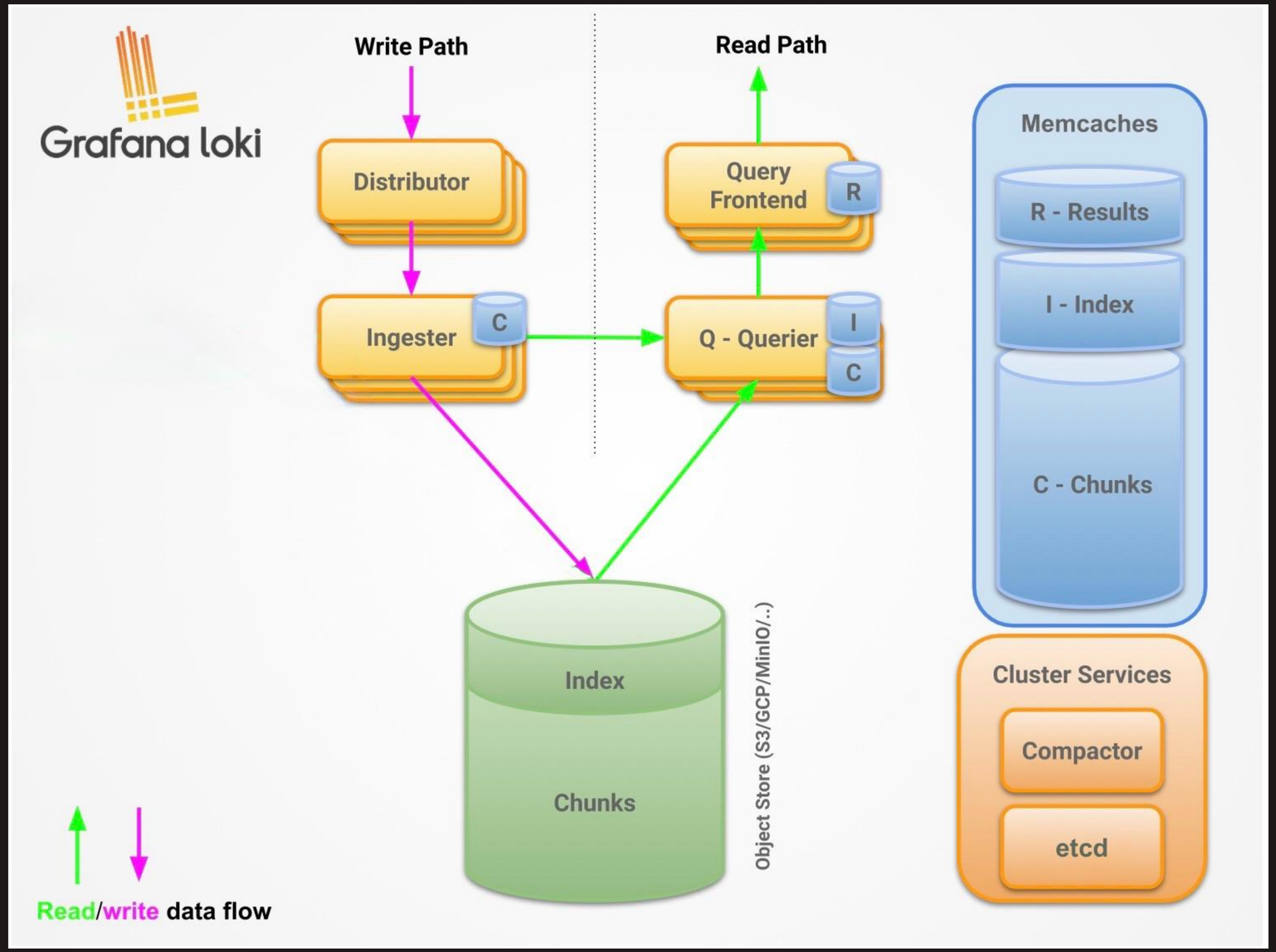
{service=”dummy_service”, node_id=”ex_node_2”}

Chunks



Dropbox-specific Loki

Loki Architecture



Loki at Dropbox: At a Glance

- ~10 GB/s logs processed
- 30 days of logs == ~10 petabytes stored in object storage
- ~1000 tenants
- <1 query per second

Multitenancy

- Loki isolates access and storage by tenant
- At Dropbox, tenant is a service (group of projects)
- One service with large log volume had to be split up by project

Auth: What?

- Before, engineers would use a production access permission for their service to SSH onto the service’s hosts to view logs
- Service == tenant aligns with previous permission model
- Some global services are accessible by everyone

Auth: Who?

- Use existing production access permission to authorize logs too
- New group permission to grant log access to a service
- Some teams have access to all logs

Auth: How?

- Custom query auth proxy that does permission lookup on user

- Grafana RBAC would’ve required a datasource per tenant/service and upgrade to enterprise

Auth: Sharing Challenges

- Team A wants to share access for their service’s logs to Team B
- Team B must request permission to the service logs for their group
- Because the permission is owned by the logging team, only we can approve
- During an incident, this delay can be costly

Auth: Breakglass

- Breakglass allows a user with a justified reason to gain temporary access to any service’s logs

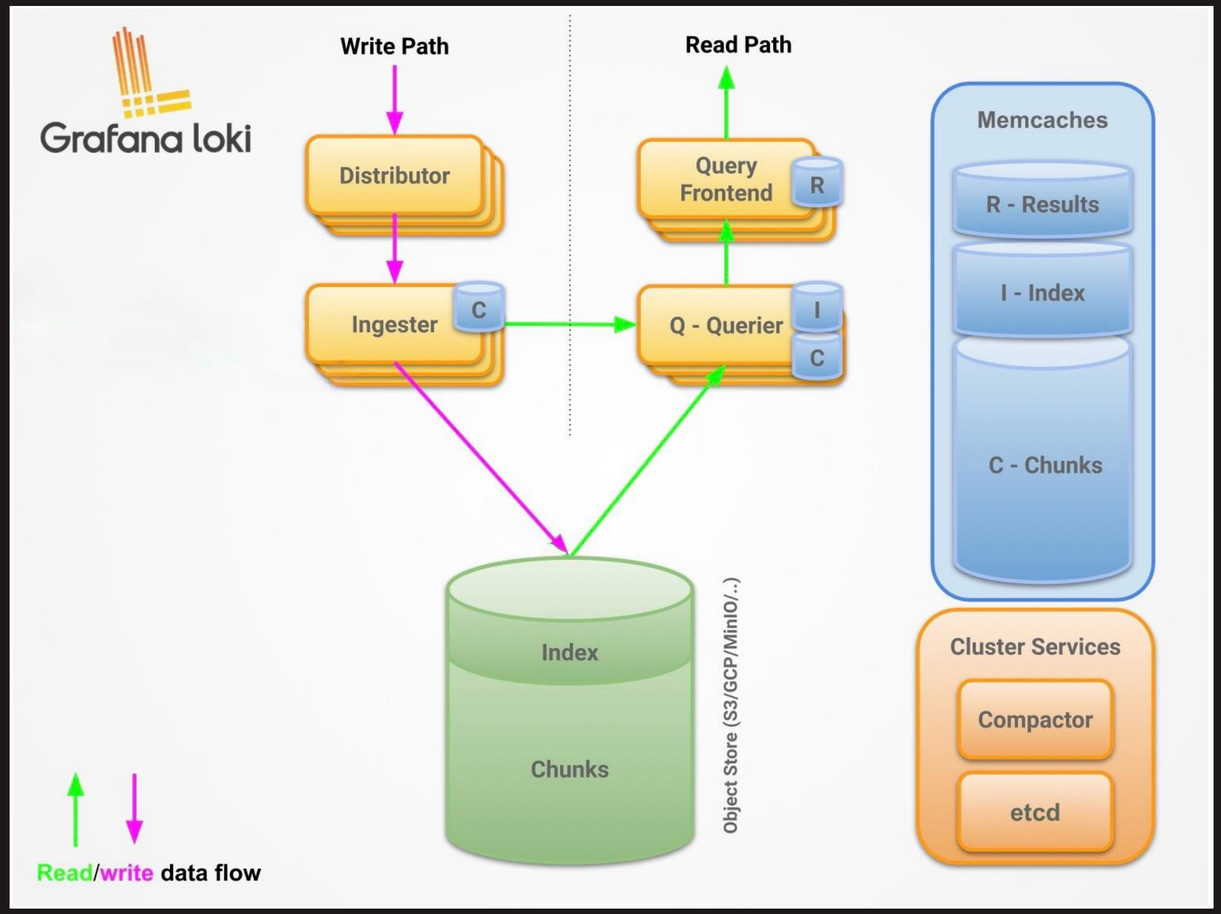
- Audit trail and safeguards in place

Multi-homing

- Run Loki in two data centers in separate geographic regions
- Same object storage is used in both regions
- Logs and queries are routed to the distributor and query frontend in the active region using a DNS server that caches Dropbox load balancer data

Multi-homing: Failover Steps

- Shift weights to the other region in load balancer global config
- After query traffic shifts to the new region, last hour of logs will be missing because they aren’t cached in new ingesters or flushed to object storage
- Restart old ingesters ASAP to flush logs to object storage
- Restart query results memcache cluster in the new region to pull in flushed logs
- Deactivate old compactor and activate compactor in new region



S3 Replacement

- Use internal object storage as drop-in replacement for S3 to save costs, especially data transfer cost
- Lower costs → Log retention increase from 1 to 4 weeks
- Performance characteristics are different
  - S3 gradually scales out reads vs. reserved capacity on-prem
  - Large index files still written to S3

Scaling Challenges

Ingester WAL

- Write Ahead Log stored on ingesters' disks
- Used to recover logs when ingester exits before flushing
- At Dropbox, disabled to prioritize availability over durability

Per-tenant Ingestion Rate Limits

- Set conservative default rate limits
- Alert notifies service owners when their service hit rate limits
- Allow tenants to override rate limits in a file
- Distribute file to Loki components using distributed KV store
- Reload Loki runtime config file with new rate limits

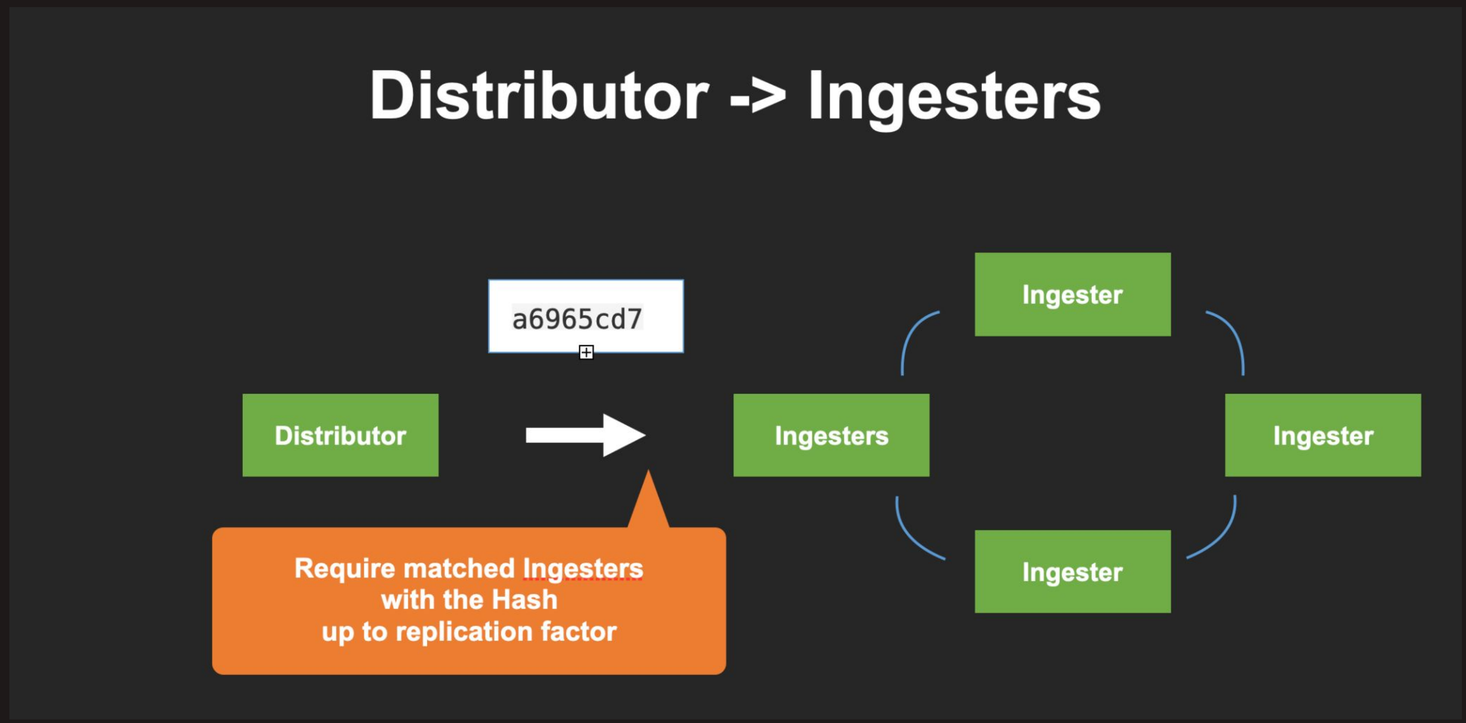
Hash Ring

- Ingesters shard log streams and own a range in the hash ring
- Ingester registers their range and health status in the ring stored in a distributed KV store
- Distributor uses ring to route log stream to ingester + replicate to other ingesters

Ingester Hash Ring Example

2025-02-01T09:02:03.000Z {service=”dummy_service”, node_id=”ex_node_1”} GET /about

a6965cd7



Hash Ring: etcd

- We original used etcd as backing KV store for Loki hash ring
- etcd: distributed, consistent KV store
- Often used for coordination and configuration, default for k8s
- Now widely used at Dropbox

Hash Ring: etcd Write Contention

- Each ingester sends a heartbeat every minute and updates the ring
- When an ingester joins/leaves the ring, it updates the ring
- etcd has a single key for the whole ring stored as a binary blob
- Each ring update is read + CAS (compare-and-swap) to replace the ring
- RF=3 and 67 ingesters for each replication factor is 201 total ingesters

Hash Ring: etcd Issues

- Deployments take hours, ingesters are pushed one at a time
- Availability alert would often trigger and fail the ingester pushes
- Single point of failure: etcd going down caused outages

Hash Ring: Migrate from etcd to memberlist

memberlist

- Now default in Loki and other Grafana projects
- Peer-to-peer gossip protocol
- Each update is only the delta, not the entire ring
- Eventually consistent

Hash Ring: memberlist Pros

- No write contention
- No single point of failure
- No issues in the last year using memberlist
- Able to scale out ingesters by 2x afterwards

Index: BoltDB → TSDB

- Log indexes determine query plan: how many log chunks to fetch
- Index format changed from BoltDB to TSDB
- TSDB based on Prometheus TSDB, ideal for labels
- Much better query performance after migration

Thank You

Alok Linkedin

Paul Linkedin