From Fragmented Logs to Unified Insights

Alok Ranjan

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

Dropbox

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

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

Requirements

Reliability	Security
Retention: 1-week log storage	**mTLS**: Deny-by-default enforcement
Throughput: 150TB/day	**Access Control**: Service-based segmentation
Latency: p99 ingestion <30s, queries <10s	**Encryption**: Secure storage with key management
Availability: 99% log durability & access	**PII Protection**: Detection, filtering & redaction

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

- **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

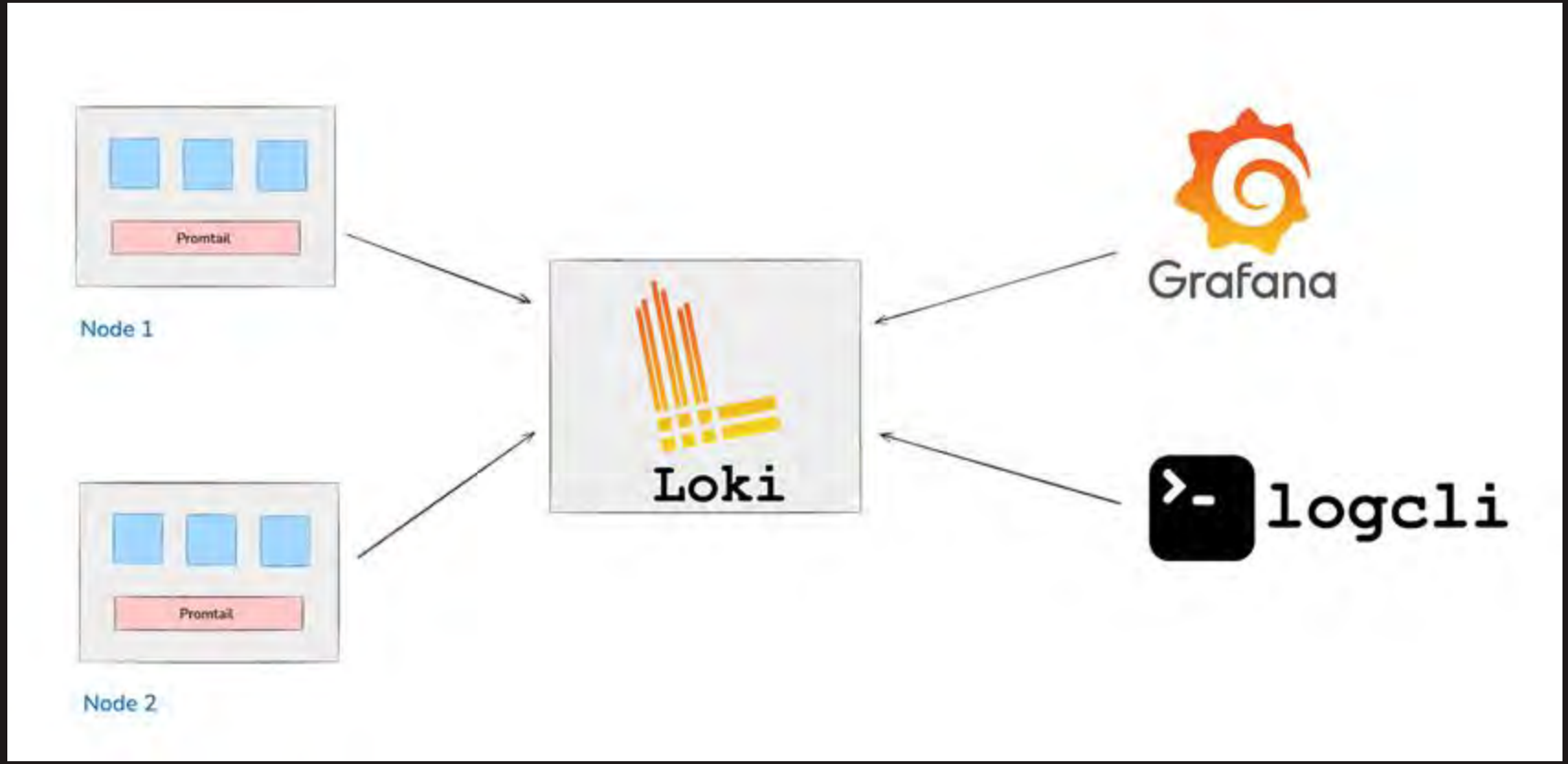
Grafana Loki

- **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

What is Loki?

- Open source
- Horizontally scalable
- Highly Available
- Multi tenant
- Prometheus inspired
- Log aggregation System

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
```

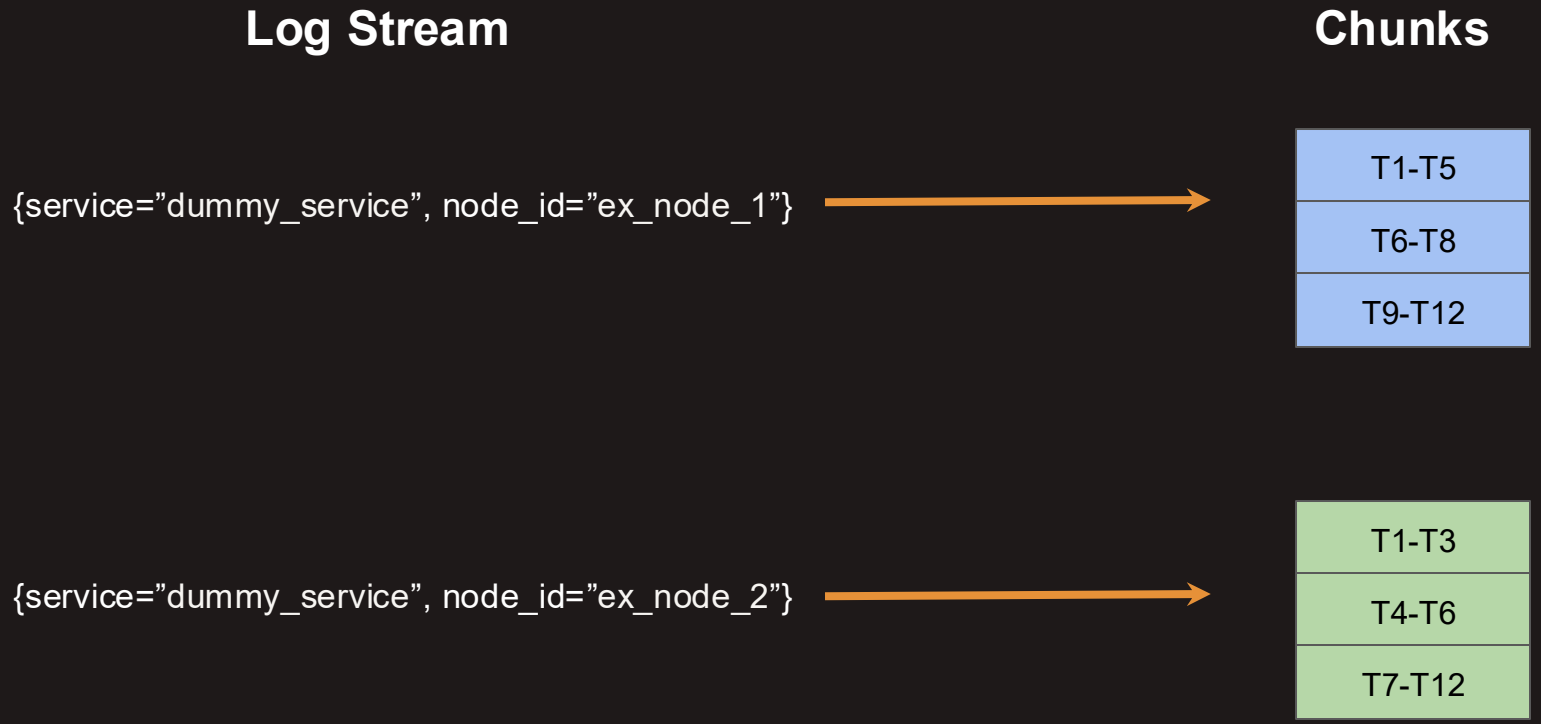
Logs Storage - Chunks

- 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

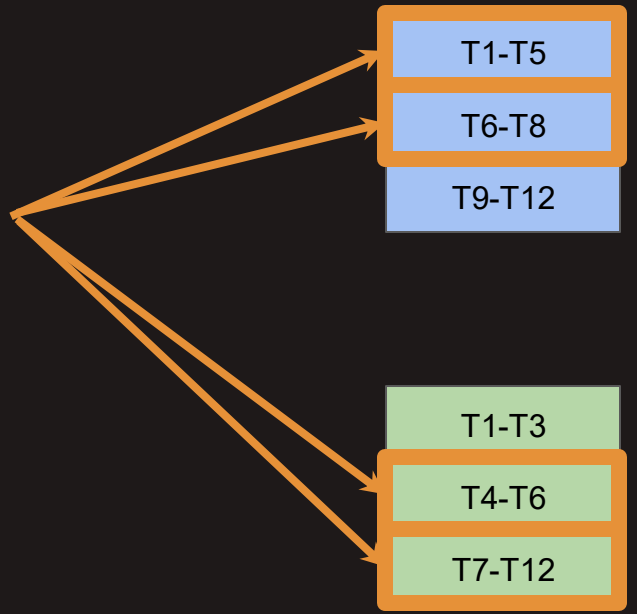
Log Stream

{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

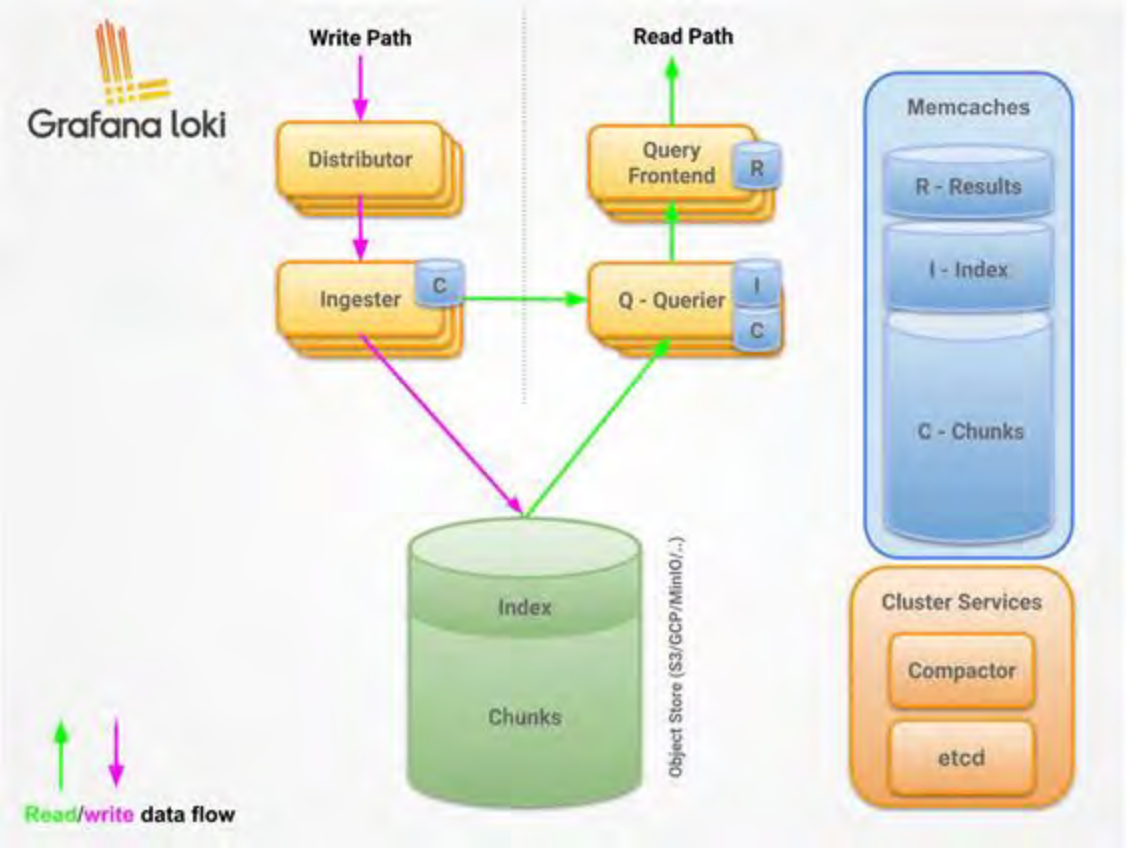


Label Cardinality

- Be mindful of label selection
- Avoid high-cardinality labels like trace_id, user_id, path, and status.
- Favor low-cardinality labels such as cluster, app, and filename for efficient indexing
- Using log level (4), status (3), and path (3) yields 36 unique streams

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

S3 Replacement

- Replaced S3 with internal storage to reduce costs (esp. data transfer)
- Lower costs → Increased log retention (1 week → 4 weeks)
- Performance differences:
  - S3 scales gradually; large queries cause timeouts
  - Internal storage has reserved capacity; avoids scaling issues
- Large index files still stored in S3

Multitenancy

- Loki isolates access/storage per tenant
- Dropbox: Tenant = service
- High-volume service split by project

Auth: What, Who & How?

- Previously: Production SSH access → log viewing
- Tenant (service) matches existing permission model
- Some global services are accessible by everyone
- Custom query auth proxy handles permissions (avoids Grafana RBAC complexity)

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 active region using DNS

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

- Conservative default limits set
- Alerts notify owners on breaches
- Tenants can override via config file
- Config distributed via KV store
- Loki reloads settings dynamically

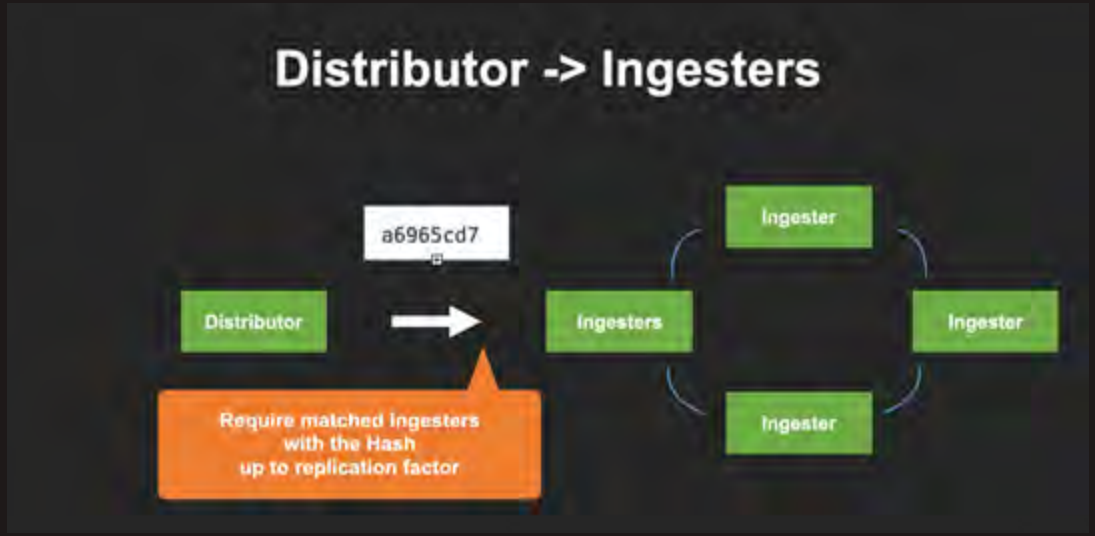
Hash Ring

- 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

- Ingester sends heartbeat & updates ring every minute
- Ring updates occur on join/leave events
- etcd stores the ring as a single binary blob; updates use read + CAS
- RF=3 with 67 ingesters per factor = 201 total ingesters

Hash Ring: etcd Challenges

- Deployments take hours, ingesters are pushed sequentially
- Frequent availability alerts during ingester pushes
- Single point of failure: etcd outages cause disruptions

Hash Ring: etcd → 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

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

Summary

- **Goal:** Scale and enhance observability at Dropbox
- **Challenges:** Manual SSH analysis of unstructured logs
- **Approach:** Evaluate multiple logging solutions
- **Solution:** Adopt Grafana Loki for cost-effective, high-performance logging
- **Result:** Improved retention, reduced costs, and efficient multi-tenant access

Thank You