Onepane

www.onepane.ai

Open Source Observability Stack

Make system Observable (easier to monitor , safer to update and simpler to repair)

Speaker

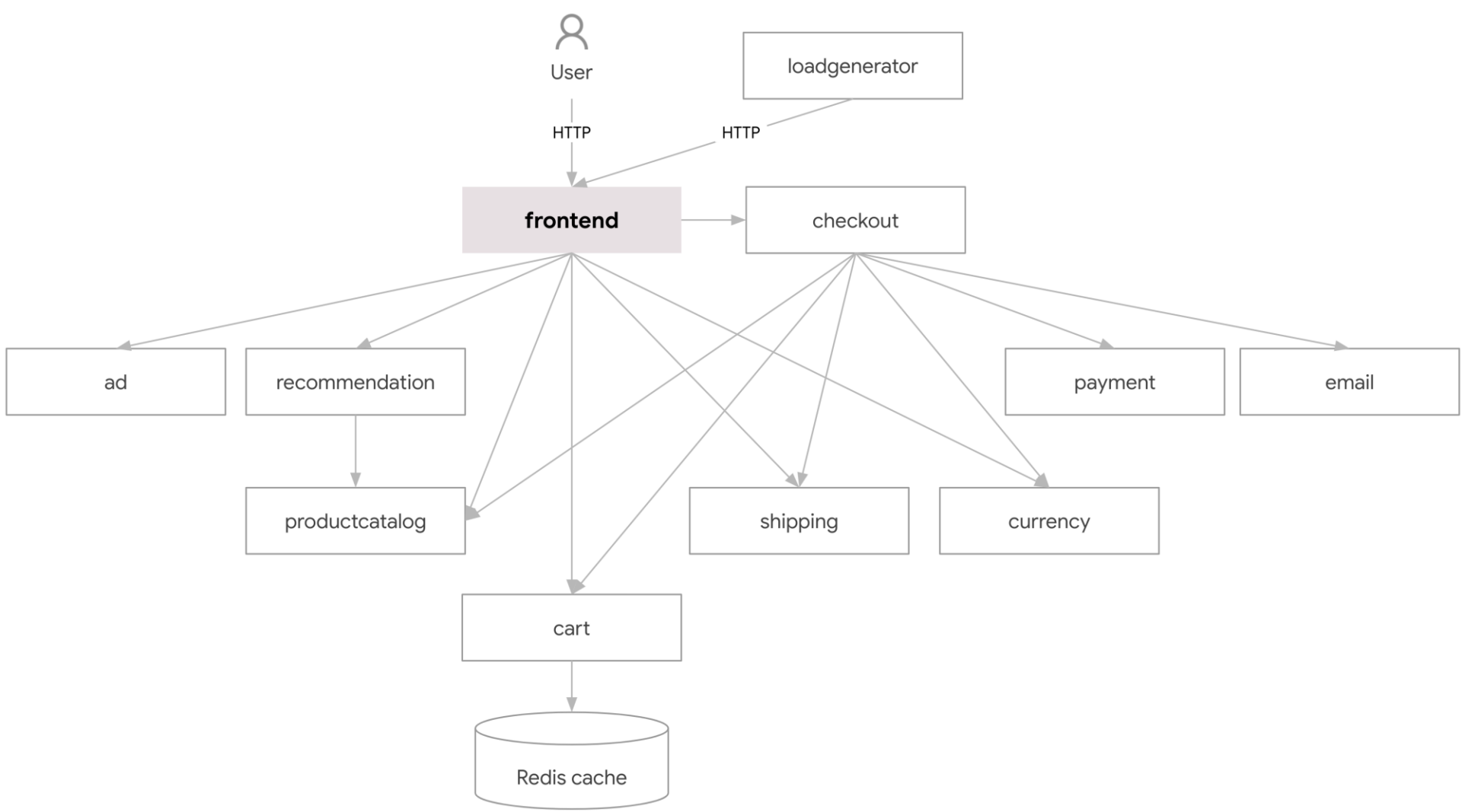
ASHMIL HUSSAIN

Contact

info@onepane.ai

Date

13 Jan 2024


User
loadgenerator
HTTP
HTTP
frontend
checkout
ad
recommendation
payment
email
productcatalog
shipping
currency
cart
Redis cache

What ?

In IT and cloud computing, observability is the ability to measure a system’s current state based on the data it generates

How ?

proactively collecting, visualizing, and applying intelligence to all of your metrics, events, logs, and traces—so you can understand the behavior of your complex digital system

Examples ?

proactively collecting, visualizing, and applying intelligence to all of your metrics, events, logs, and traces—so you can understand the behavior of your complex digital system

Phases ?

- Reactive

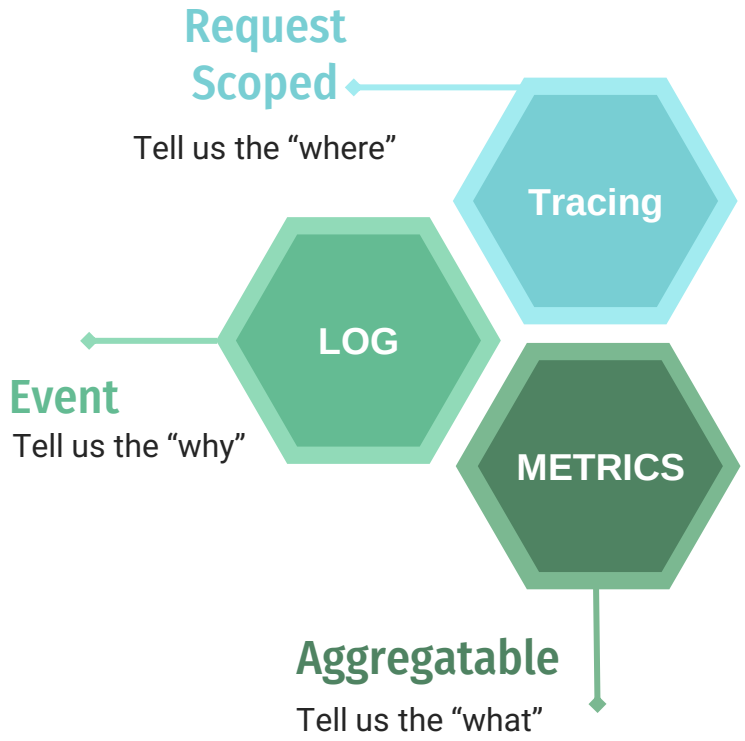
  (Investigation and Monitoring)
- Proactive

  (Alerts and Monitoring)
- Data-driven

  (Anomaly and Data driven)

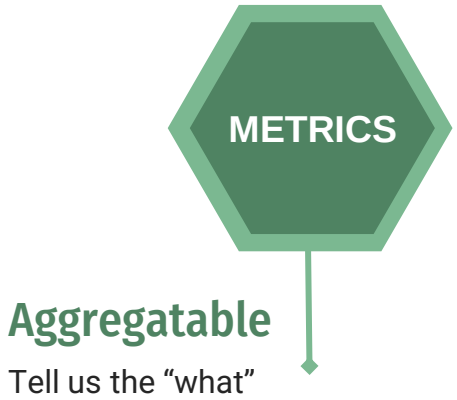
Three Pillars

Combining metrics, logs, and traces for observability is the only way to understand complex environments



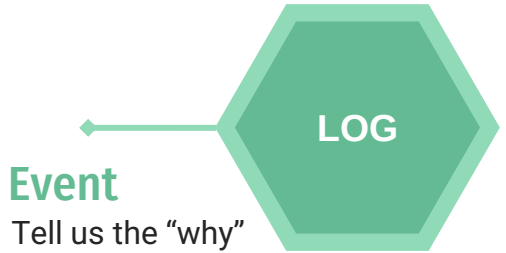
Metrics

Combining metrics, logs, and traces for observability is the only way to understand complex environments



Logs

Combining metrics, logs, and traces for observability is the only way to understand complex environments



Traces

Combining metrics, logs, and traces for observability is the only way to understand complex environments

Prometheus

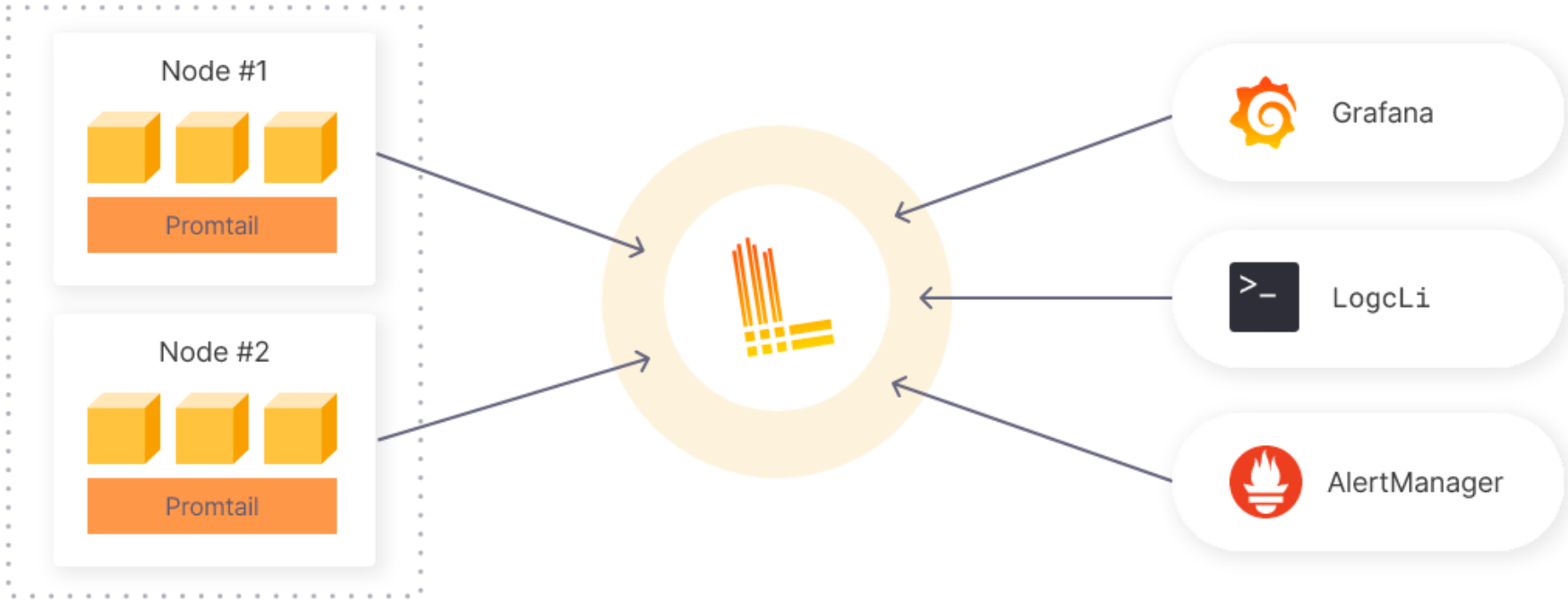
Prometheus is a free software application used for event monitoring and alerting. It records real-time metrics in a time series database built using a HTTP pull model, with flexible queries and real-time alerting.

Query language : PromQL

Create Alerts : Alertmanager

For pushing metrics remotely : Prometheus Push gateway

Loki


Node #1
Promtail
Node #2
Promtail
Grafana
LogcLi
AlertManager

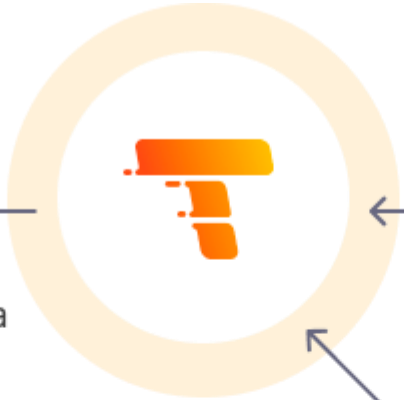
Jager

Grafana Tempo

Provides information about service dependencies

- **Request:** how microservices communicate each other.
- **Span:** work done by a single service with respect to time intervals.
- **Trace:** This implies the end-to-end user requests which consist of single or multiple spans.

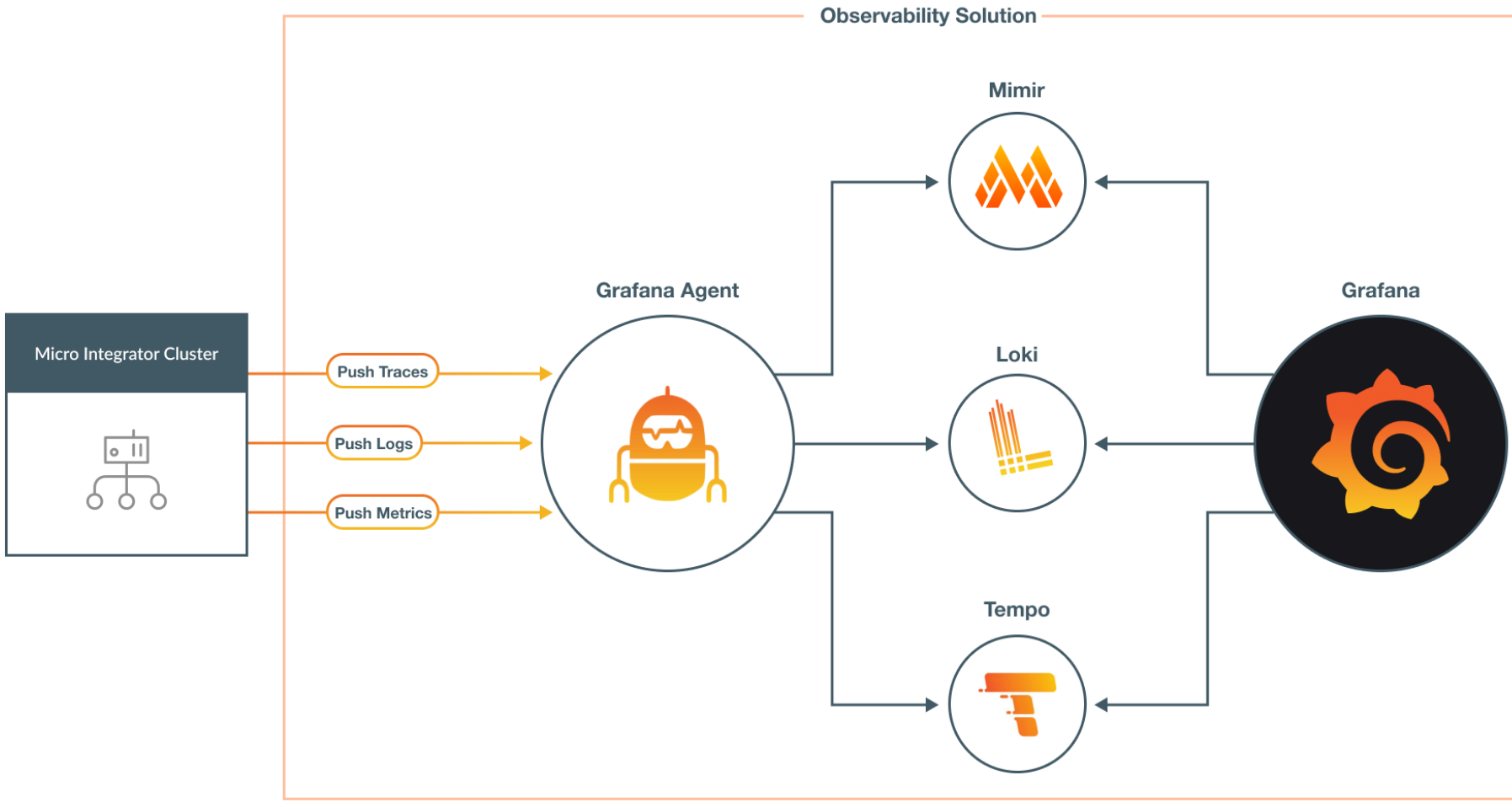
Jager

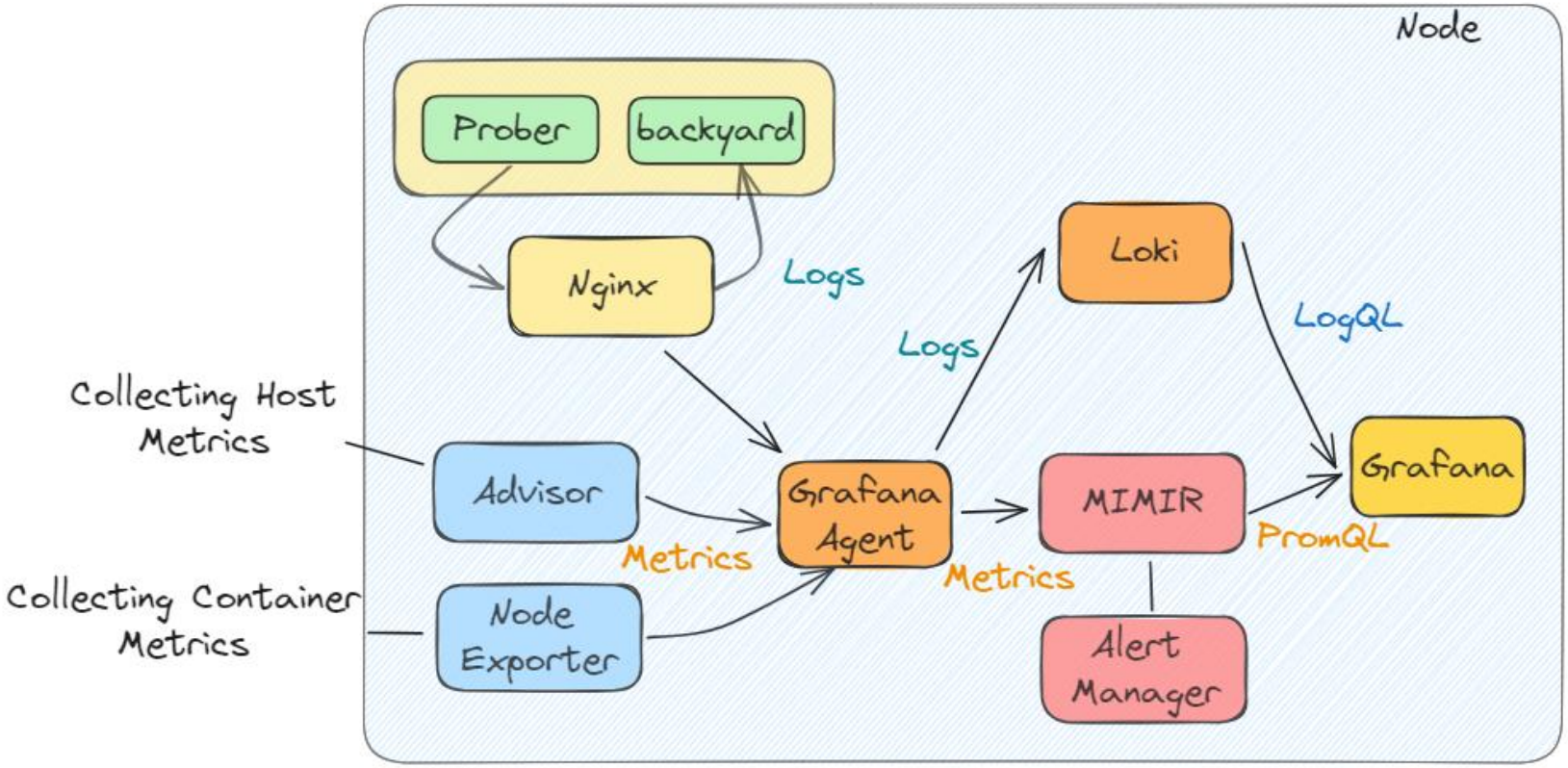


Grafana has a built-in Tempo data source that can be used to query Tempo and visualize traces.

Compatible with popular open source tracing protocols such as Zipkin, Jaeger, and OpenTelemetry.

With Tempo, the only dependency is object storage (GCS, S3, Azure Blob Storage), which makes it easy to hit massive scale while keeping costs low.


Observability Solution
Mimir
Grafana Agent
Grafana
Micro Integrator Cluster
Loki
Push Traces
Push Logs
Push Metrics
Tempo


Node
Prober
backyard
Nginx
Logs
Loki
LogQL
Logs
Collecting Host Metrics
Advisor
Grafana Agent
MIMIR
Grafana
PromQL
Metrics
Metrics
Collecting Container Metrics
Node Exporter
Alert Manager

Components

- Node Exporter
- CAdvisor
- Grafana agent
- Grafana
- Loki
- Mimir
- Nginx (Client / Server)

Prometheus

Mimir Configuration

config > ! mimir.yml > { } blocks_storage > { } tsdb

```yaml
# Do not use this configuration in production.
# It is for demonstration purposes only.
multitenancy_enabled: false

blocks_storage:
  backend: filesystem
  bucket_store:
    sync_dir: /tmp/mimir/tsdb-sync
  filesystem:
    dir: /tmp/mimir/data/tsdb
  tsdb:
    dir: /tmp/mimir/tsdb

compactor:
  data_dir: /tmp/mimir/compactor
  sharding_ring:
    kvstore:
      store: memberlist

distributor:
  ring:
    instance_addr: 127.0.0.1
    kvstore:
      store: memberlist
```

Grafana Agent

Metrics Scrap

```yaml
metrics:
  wal_directory: /tmp/grafana-agent/wal

  configs:
    - name: agent
      scrape_configs:
        - job_name: node-exporter
          static_configs:
            - targets: ['nodeexporter:9100']
              labels:
                job: grafana-agent
                __path__: /tmp/grafana-agent/wal
```

Prometheus Demo


Grafana

Grafana


apiVersion: 1
datasources:
- name: Mimir
type: prometheus
access: proxy
url: http://monitoring.mimir:9009/prometheus
basicAuth: false
isDefault: true
editable: true

PromQL

Type of Data

- **Scalar** - a simple numeric floating point value
- **String** - a simple string value
- **Instant vector** - a set of time series containing a single sample for each time series, all sharing the same timestamp.
- **Range vector** - a set of time series containing a range of data points over time for each time series.

PromQL

PromQL is functional query language used to select and aggregate time series data in real time. The result of an expression can be shown as a graph, viewed as tabular data. This Query Language in

PromQL Selectors

- **Instant vector :** filter the entire dataset (metrics)
  notation - {}
  logical operation - **=, !=, =~, !~,** |

  eg : *http_requests_total{job="prometheus",group!="canary"}*

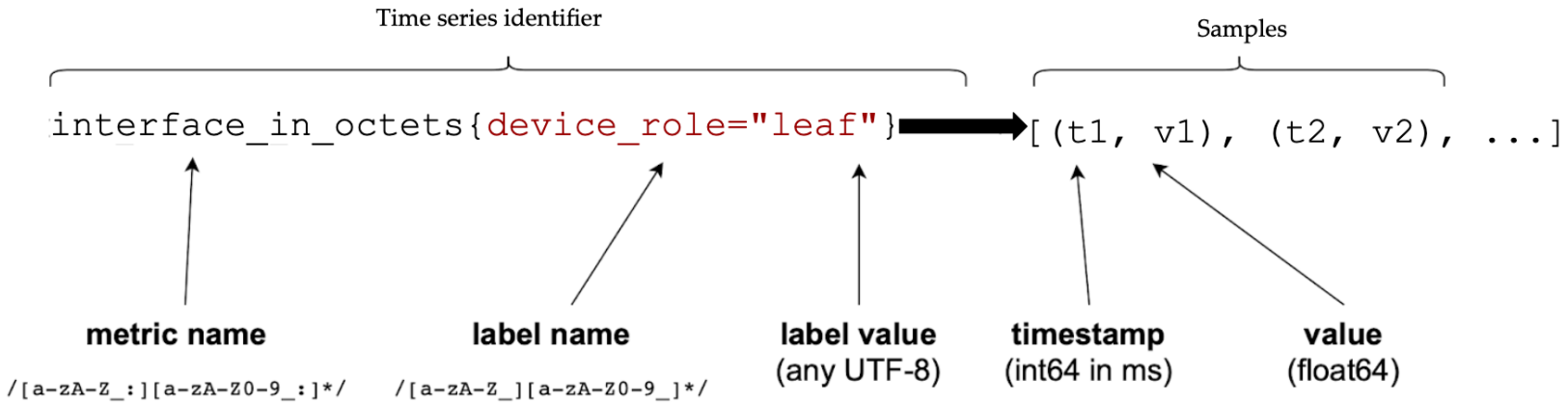
- **Range vector:** select a range of metrics data back from the current instant

  notation - '[ ]'
  Time duration - (ms, s, m, h, d, w, y)

  eg: http_requests_total{job="prometheus"}[5m]

- @ **modifier:** select the metrics data on specific timestamp.
  '@' notation is used for this.
  eg: sum(http_requests_total{method="GET"} @ 1609746000)

PromQL Example



PromQL Operators & Functions

- **abs** (*v instant-vector*) **-** returns the input vector their absolute value.
- **changes**(*v range-vector*) – returns the number of times its value has changed within the provided time range as an instant vector
- **ceil**(*v instant-vector*) - rounds the sample values of all elements up to the nearest integer.
- **floor***(v instant-vector*) - rounds the sample values of all elements down to the nearest integer.
- **vector**(*s scalar*) – return input scalar to vector

Grafana dashboard

Break

Loki

Loki

Loki is a horizontally scalable, highly available, multi-tenant log aggregation system inspired by Prometheus

Collector : Promtail
Query language : LogQL

Nginx configuration

```
http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    log_format vhost_time '$host $remote_addr - $remote_user [$time_local] '
                          '"$request" $status $body_bytes_sent '
                          '"$http_referer" "$http_user_agent" "$request_time"';

    access_log  /var/log/nginx/access.log  main;
```

Agent configuration

```
logs:
  configs:
  - name: default
    clients:
      - url: http://loki:3100/loki/api/v1/push
    positions:
      filename: /tmp/positions.yaml
    scrape_configs:
      - job_name: system
        static_configs:
          - targets: ['localhost']
            labels:
              job: docker
              __path__: /var/lib/docker/containers/*/*log
```

Log parsing

```
- match:
    selector: '{image_name="nginx.promtail.test"}'
    stages:
      - json:
          expressions:
            row: log
      - regex:
          expression: .+nginx.+\|.+\[0m(?P<virtual_host>[a-z_\.-]+) +(?P<nginx_log_row>.+)
          source: row
      - regex:
          source: nginx_log_row
          expression: ^(?P<ip>[\w\.]+) - (?P<user>[^ ]*) \[(?P<timestamp>[^ ]+).*\] "(?P<method>[^ ]*) (?P<request_url>
      - regex:
          source: request_url
          expression: ^.+\.(?P<static_type>jpg|jpeg|gif|png|ico|css|zip|tgz|gz|rar|bz2|pdf|txt|tar|wav|bmp|rtf|js|flv|s
      - regex:
          source: request_url
          expression: ^/photo/(?P<photo>[^/\?\.]+).*$
      - regex:
          source: request_url
          expression: ^/api/(?P<api_request>[^/\?\.]+).*$
      - template:
          source: request_type
          template: "{{if .photo}}photo{{else if .static_type}}static{{else if .api_request}}api{{else}}other{{end}}"
      - labels:
          api_request:
```

Grafana

```
datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://monitoring.prometheus:9090
    basicAuth: false
    isDefault: true
    editable: true
```

```
datasources:
  - name: Loki
    type: loki
    access: proxy
    url: http://monitoring.loki:3100
    basicAuth: false
    isDefault: false
    editable: true
```

```
datasources:
  - name: LokiAsPrometheus
    type: prometheus
    access: proxy
    url: http://monitoring.loki:3100/loki
    basicAuth: false
    isDefault: false
    editable: true
```

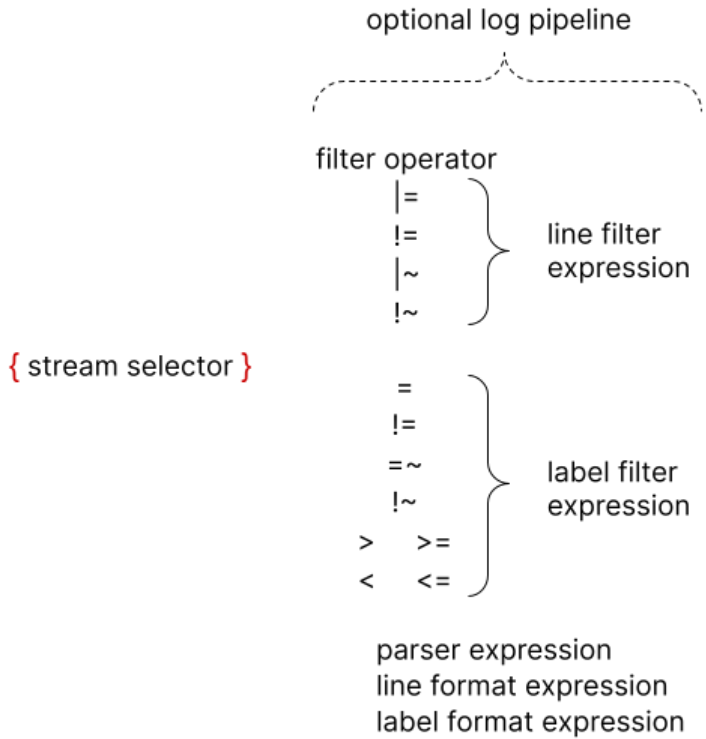
LogQL

An Query Language to query on log data, it is a PromQL like language, act as if they are a distributed grep to aggregate log sources. LogQL uses labels and operators for filtering. All LogQL queries contain a log stream selector.

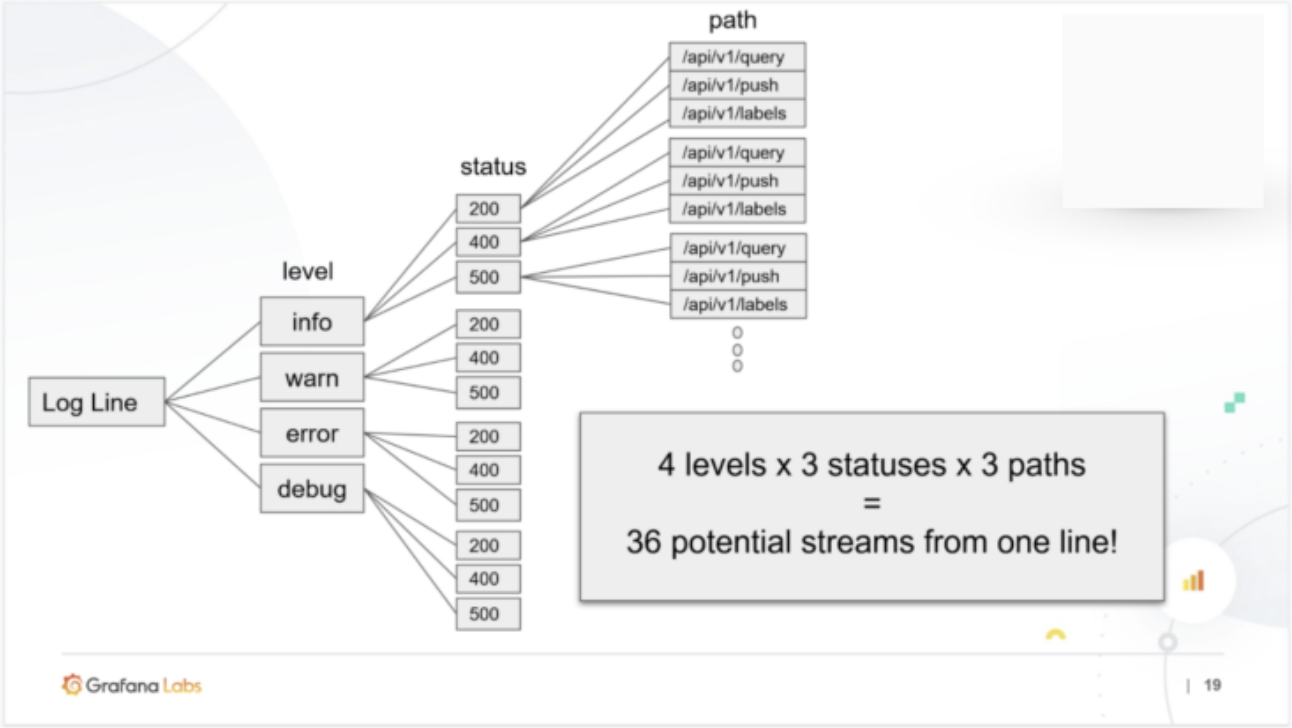
Types of LogQL queries

- Log Query : Log queries returning the contents of log lines as streams.
- Metrics Query : Metric queries that convert logs into value matrixes

LogQL Operators



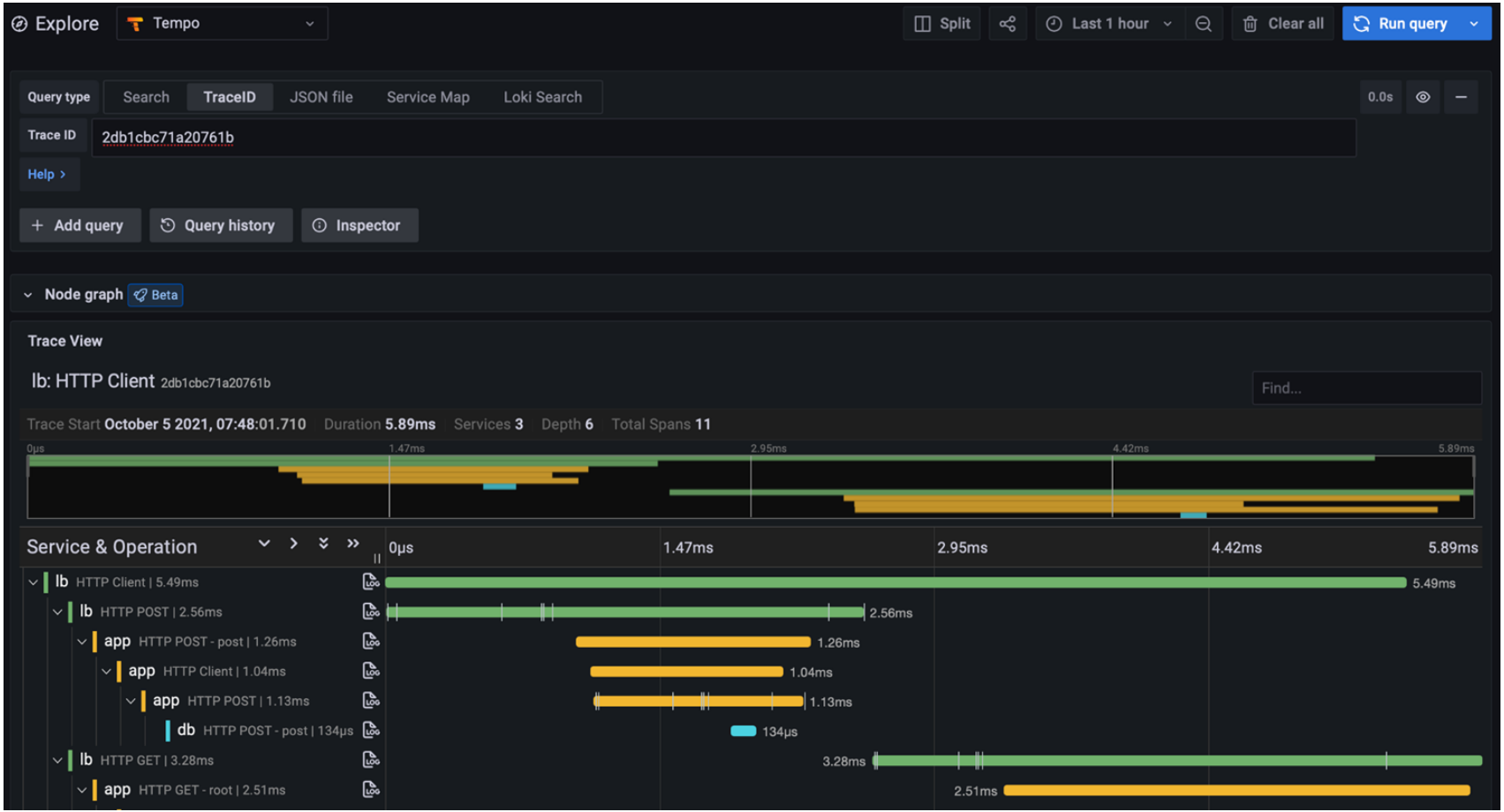
LogQL Example



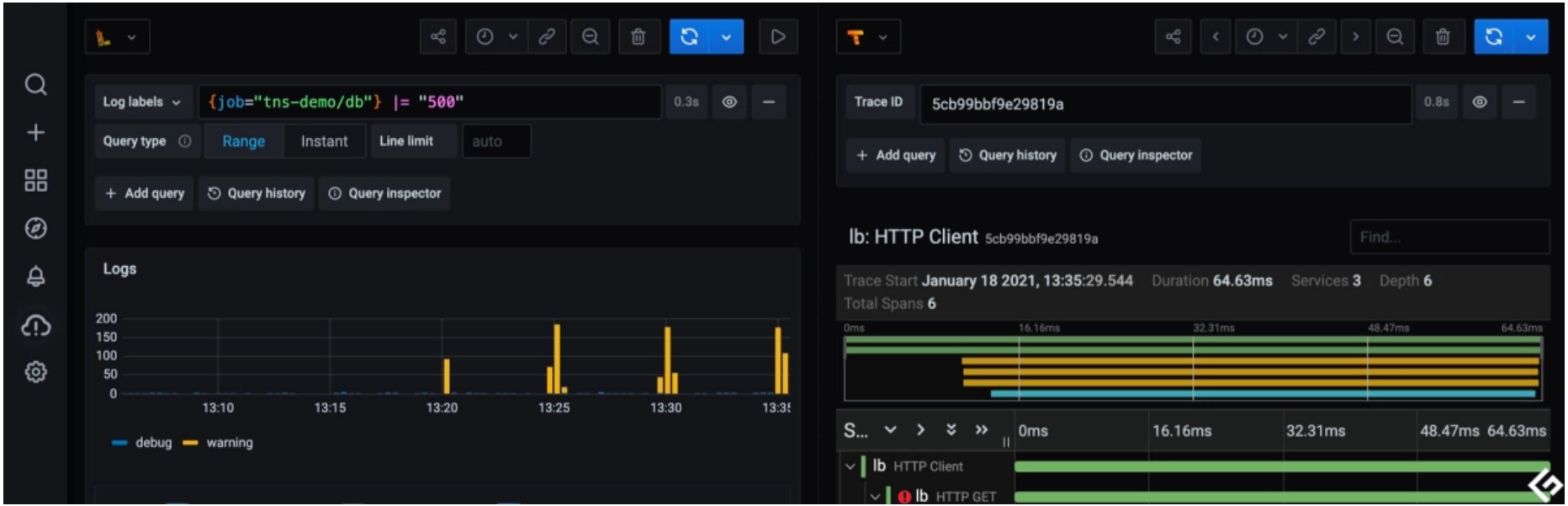

```
{cluster="OPS-cluster-1",namespace="loki-dev"} |= "level=debug" |= "status=200" |= "path=/api/v1/query"
```


Grafana Tempo

Tempo


Explore
Tempo
Split
Last 1 hour
Clear all
Run query
Query type
Search
TraceID
JSON file
Service Map
Loki Search
0.0s
Trace ID
2db1cbc71a20761b
Help
Add query
Query history
Inspector
Node graph
Beta
Trace View
lb: HTTP Client 2db1cbc71a20761b
Find...
Trace Start October 5 2021, 07:48:01.710
Duration 5.89ms
Services 3
Depth 6
Total Spans 11
0µs
1.47ms
2.95ms
4.42ms
5.89ms
Service & Operation
lb HTTP Client | 5.49ms
lb HTTP POST | 2.56ms
app HTTP POST - post | 1.26ms
app HTTP Client | 1.04ms
app HTTP POST | 1.13ms
db HTTP POST - post | 134µs
lb HTTP GET | 3.28ms
app HTTP GET - root | 2.51ms

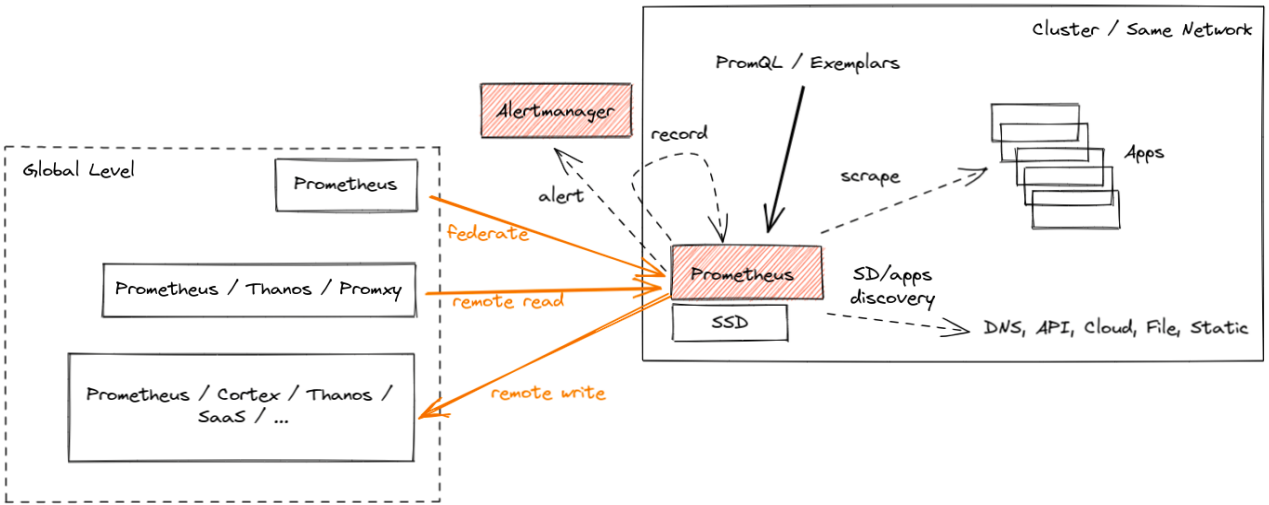
Tempo


Log labels
{job="tns-demo/db"} |= "500"
0.3s
Query type
Range
Instant
Line limit
auto
Add query
Query history
Query inspector
Logs
200
150
100
50
0
13:10
13:15
13:20
13:25
13:30
debug
warning
Trace ID
5cb99bbf9e29819a
0.8s
Add query
Query history
Query inspector
lb: HTTP Client 5cb99bbf9e29819a
Find...
Trace Start January 18 2021, 13:35:29.544
Duration 64.63ms
Services 3
Depth 6
Total Spans 6
0ms
16.16ms
32.31ms
48.47ms
64.63ms
lb HTTP Client
lb HTTP GET

Exploring More

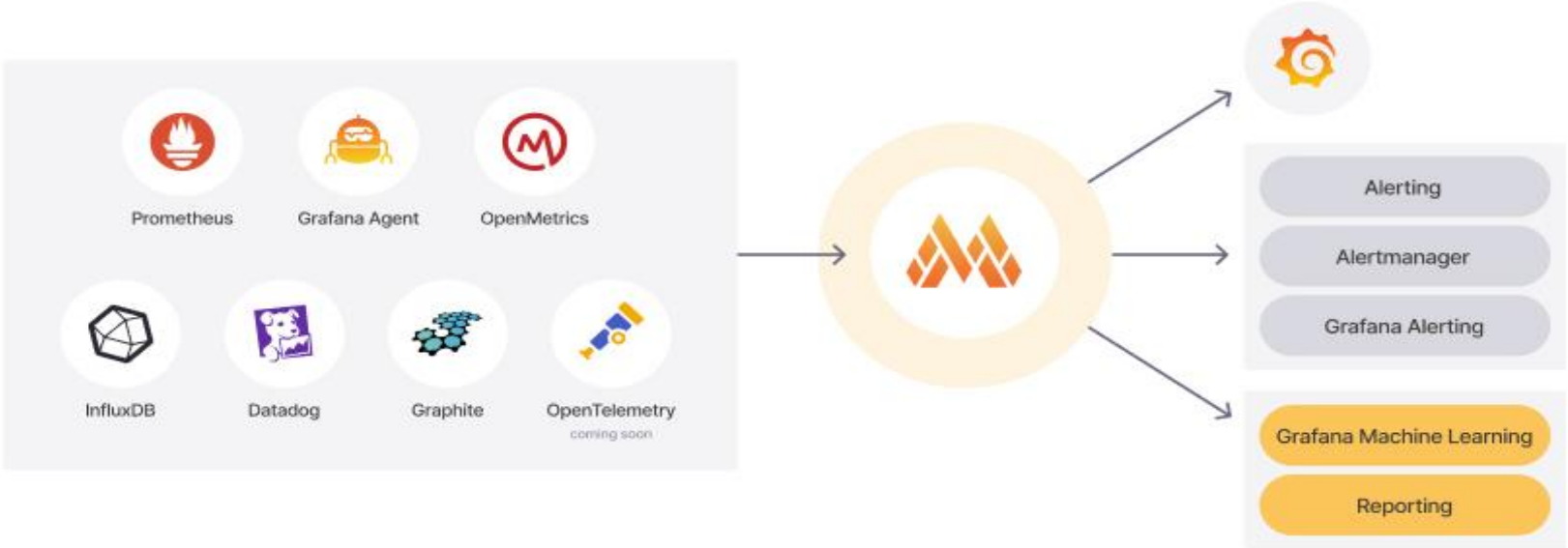
Global view concept

- monitoring data has to be somehow aggregated, presented to users and sometimes even stored on the *global* level. This is often called a **Global-View** feature.



Mimir

Grafana Mimir is a distributed, horizontally scalable, and highly available long-term storage for Prometheus.



Mimir Pro's and Con's

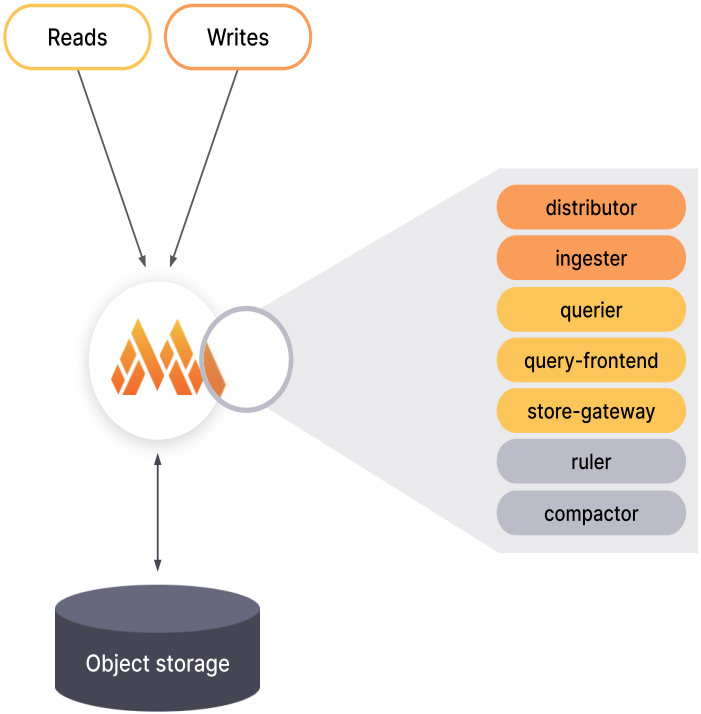
Pros

- Easy to install and maintain
- Horizontally scalable
- Baked in High Availability as all components are stateless
- Out of order ingestion support
- Cardinality API

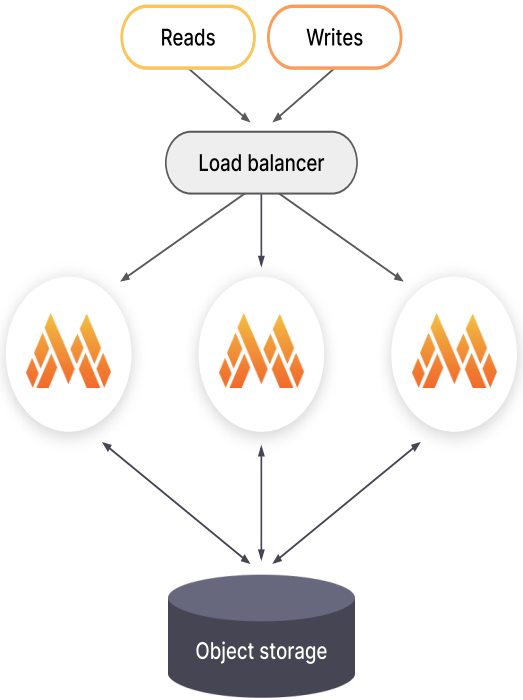
Cons

- Complicated setup with many components
- Lack of documentation
- Early adoption stage

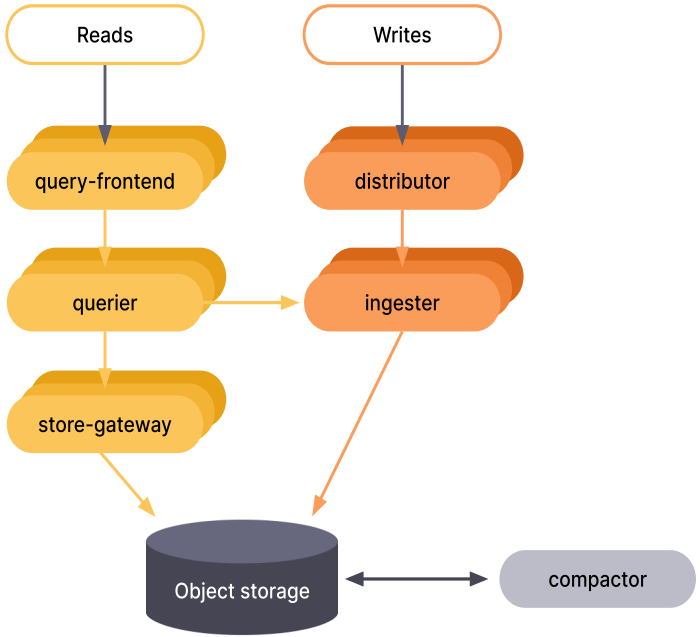
Mimir Deployment : Monolithic mode



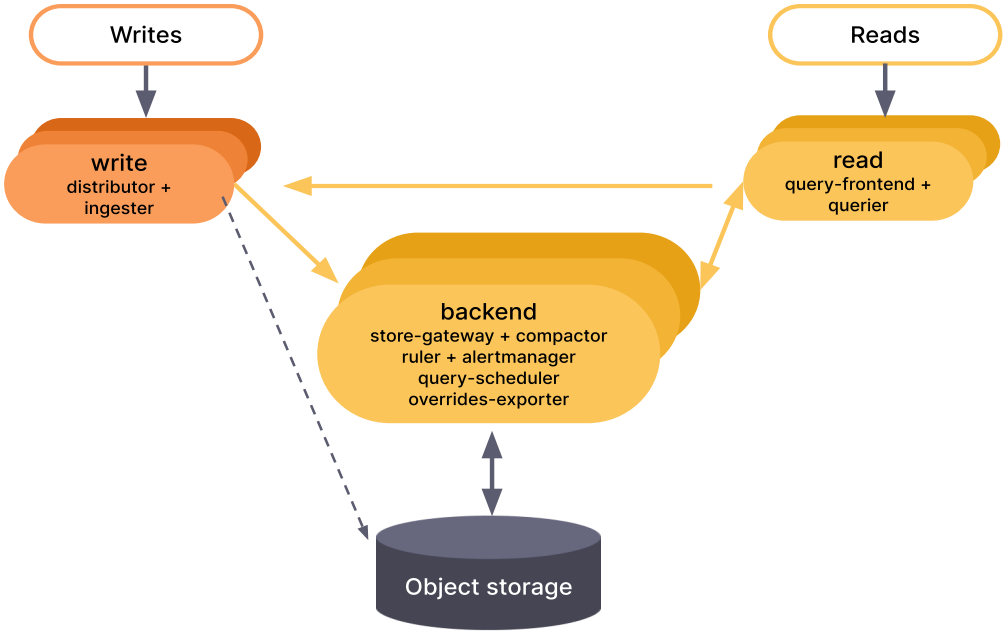
Mimir Deployment : Monolithic mode



Mimir Microservice mode



Read-Write mode



Installation options

Helm

Available at:
https://grafana.com/docs/helm-charts/mimir-distributed/latest/

Jsonnet

Available at:
https://github.com/grafana/mimir/tree/main/operations/mimir

Binaries, deb, .rpm

Available at:
https://github.com/grafana/mimir/releases

Mimir Capacity planning

Distributor
CPU: 1 core every 25,000 samples per second.
Memory: 1GB every 25,000 samples per second.

Ingester (SSD with high speed)
CPU: 1 core for every 300,000 series in memory
Memory: 2.5GB for every 300,000 series in memory
Disk space: 5GB for every 300,000 series in memory

Query-frontend(enable cache)
CPU: 1 core for every 250 queries per second
Memory: 1GB for every 250 queries per second

Querier (enable cache)
CPU: 1 core for every 10 queries per second
Memory: 1GB for every 10 queries per second

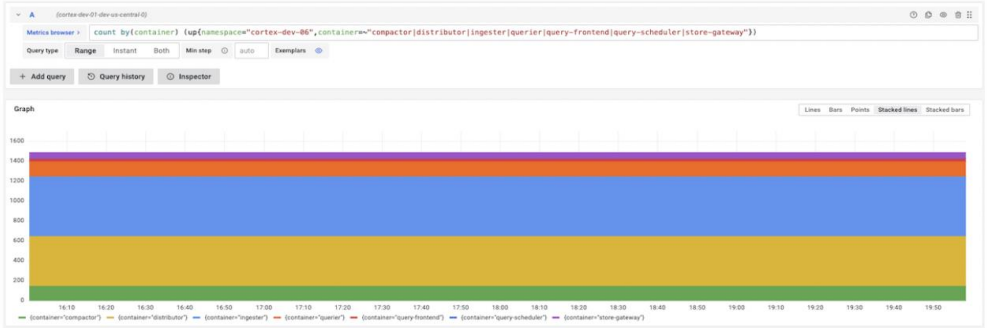
Compactor
Assuming you run one compactor instance every 20 million active series
CPU: 1 core
Memory: 4GB
Disk: 300GB

Store-gateway(enable cache)
CPU: 1 core every 10 queries per second
Memory: 1GB every 10 queries per second
Disk: 13GB every 1 million active series

https://grafana.com/docs/mimir/latest/manage/run-production-environment/planning-capacity/

Fun fact

Scaling Grafana Mimir to 1 billion active series



To run Mimir at this large scale, we deployed it in microservices mode with a distributor (500 replicas), ingester (600 replicas), query-frontend (30 replicas), query-scheduler (2 replicas), querier (150 replicas), store-gateway (60 replicas), compactor (150 replicas).

Approximately

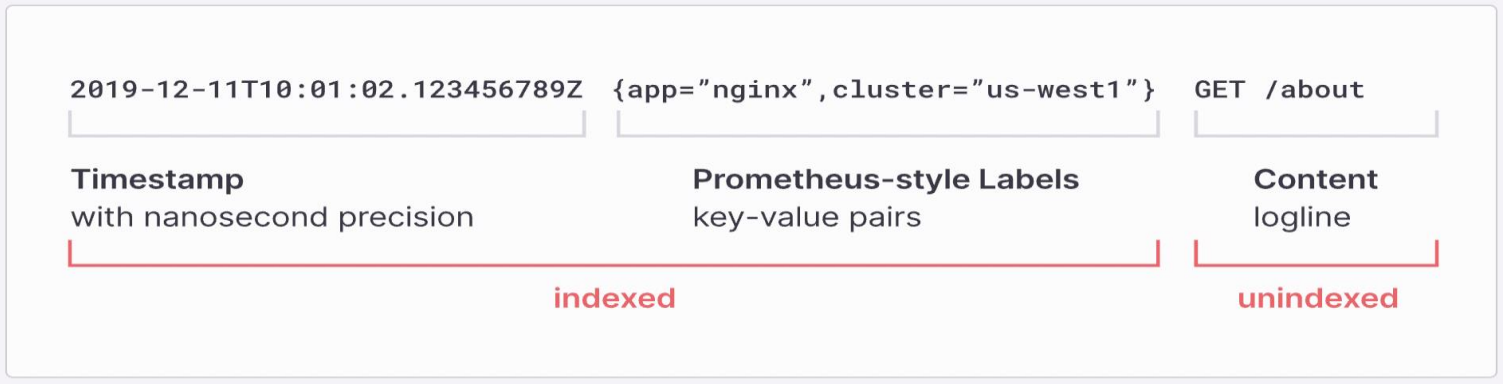
Replicas : 1500
RAM: 30TiB
CPU: 7000

MIMIR vs Thanos vs Federation

	Federation	Mimir	Thanos
One word	one access point for many prometheus	single point to write Prometheus metrics from multiple Prometheus	one access point for many prometheus
Deployment mode	Binary	Binary Microservice	Microservice
HA	no	yes	yes
Remote storage	no	yes	yes
Documentation	limited	limited	Good
Maturity	mature	Introductory stage	mature
HTTP API	yes	yes	no
Risk	Global Prometheus Management overhead	Ingesters failing and memory consumption	Compaction failing

Loki

- Grafana Loki is a set of components that can be composed into a fully featured logging stack.
- Loki does not index the contents of the logs, but only indexes metadata about your logs
- Log data is then compressed and stored in chunks in object storage and metadata is stored in indexes



Loki Cons and Pros

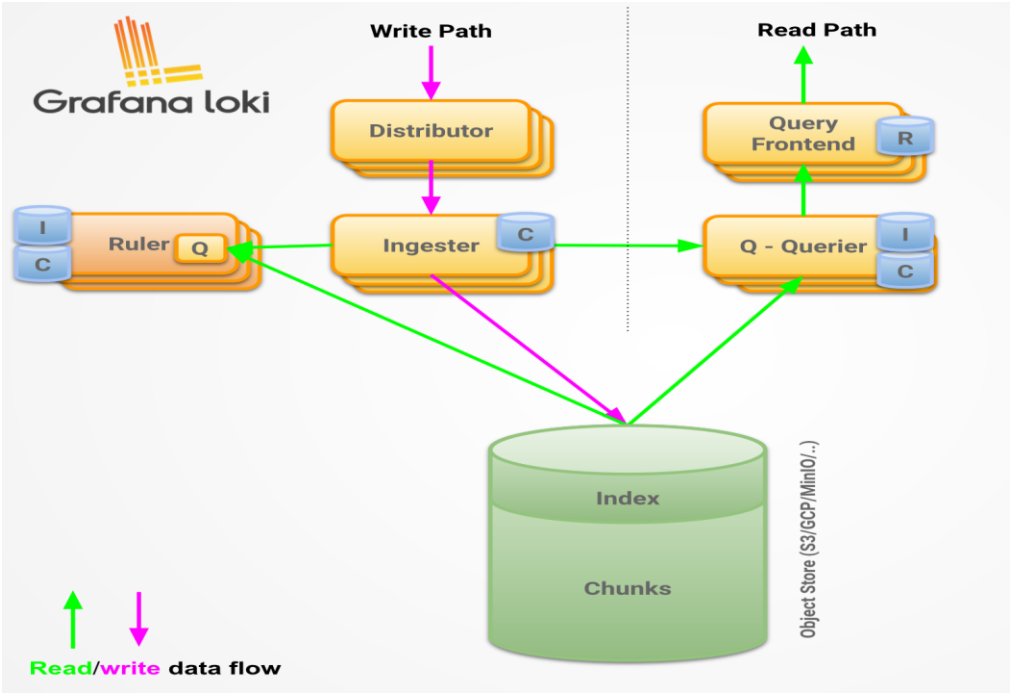
Pros

- Scalability
- Tiny indexes
- Log any and all formats
- Cut and slice your logs in dynamic ways

Cons

- Loki may experience performance challenges with high cardinality queries. High cardinality refers to datasets with a large number of unique labels or values.
- LogQL is not easy comparing to other log query engines

Loki architecture



Logs stream

- A log stream is a group of logs with same labels ordered by timestamp

```
2019-10-13T10:01:02.000Z  {app="nginx",instance="1.1.1.1"}  GET /about
2019-10-13T10:03:04.000Z  {app="nginx",instance="1.1.1.1"}  GET /
2019-10-13T10:05:06.000Z  {app="nginx",instance="1.1.1.1"}  GET /help
```

```
2019-10-13T10:01:02.000Z  {app="nginx",instance="2.2.2.2"}  GET /users/1
2019-10-13T10:03:04.000Z  {app="nginx",instance="2.2.2.2"}  GET /users/2
```

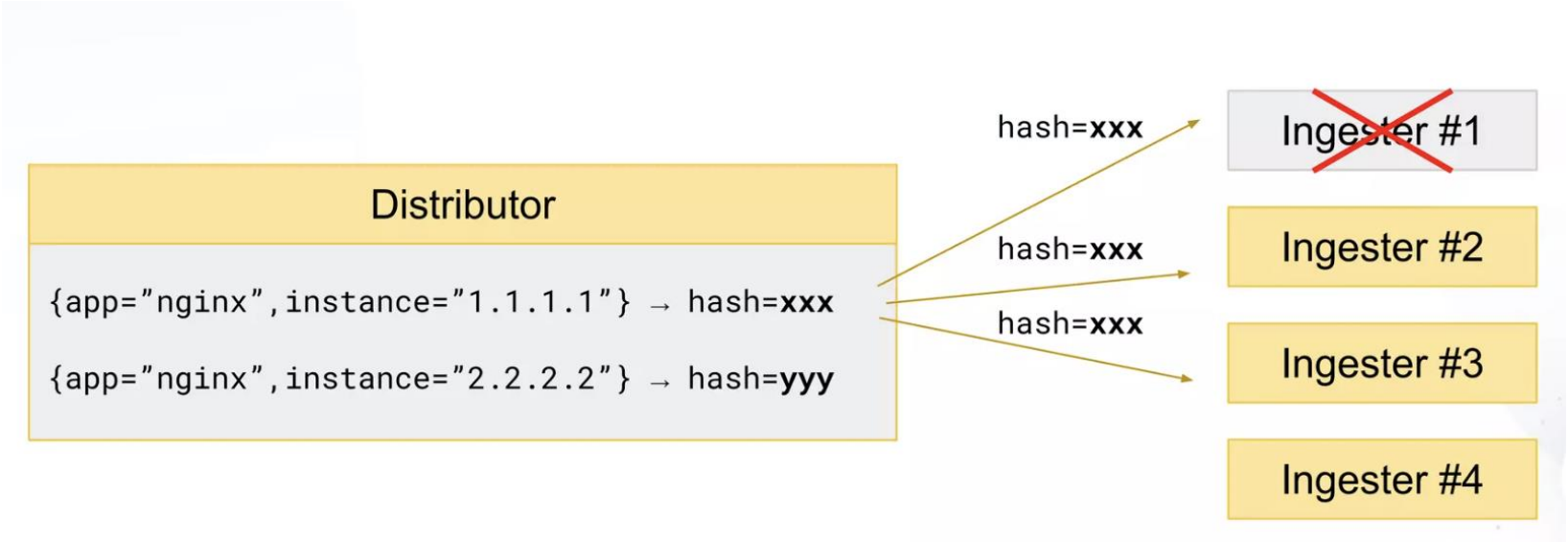
Storage chunks

- Each log stream stored into chunks of data for a particular time window
- Chunks are indexed by labels

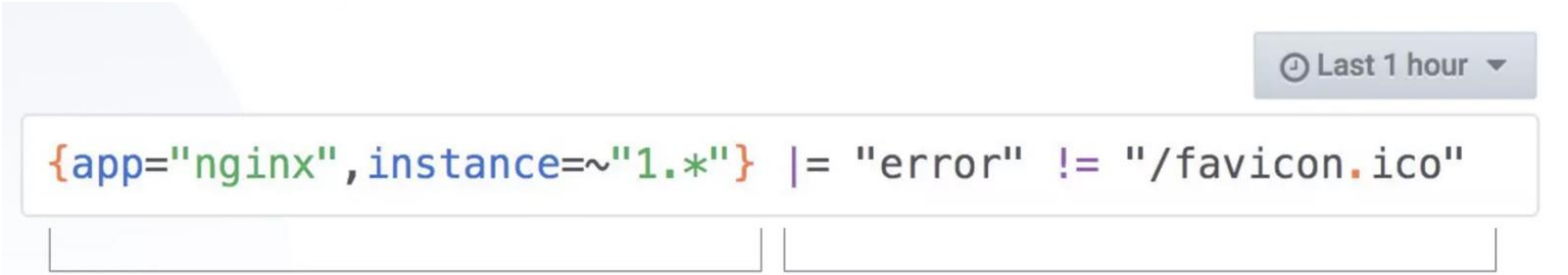
Labels set	From	To	Reference to
{app="nginx",instance="1.1.1.1"}	T1	T2	chunk #1
{app="nginx",instance="2.2.2.2"}	T1	T2	chunk #2
{app="nginx",instance="1.1.1.1"}	T2	T3	chunk #3

Shrading & replication

- Received Logs streams are hashed by labels and multiple copies and send into multiple ingesters for replication



Logs query



Log selector

Filter log streams by matching labels using an **index**

Filter expression

Given matching log streams, **scan and match** log entries (**unindexed**)

Supported storage systems

Index

- Amazon DynamoDB
- Google Bigtable
- Cassandra
- **Boltdb-shipper**

Chunks

- Amazon DynamoDB
- Google Bigtable
- Cassandra
- **S3**
- **GCS**
- FS

Sizing the cluster

https://grafana.com/docs/loki/latest/setup/size/

Loki vs E?K

Criteria	E?K	LGTM
Resources consumption	High	Low Resource and storage consumption
Data processing	Full text logs are indexed	Labels are indexed
Search	Can handle complex queries and filter using KQL	The performance of query execution depends on how many labels are selected to filter down log streams
Scalability	Highly configurable	Configurable microservice architecture
Access	Has authorization only in paid subscription	Grafana allows managing users and restricts access to logs exploration out of the box
Log-based alerting	Complex alerts configuration using third-party tools	Alert manager
Performance	Can make complex selections	Faster ingestion query may become slower based on labels used
Dashboards	Kibana can be used to build reach dashboards even with IP geolocation	Grafana

Thanks