22. Metrics Monitoring and Alerting System

Real-life examples

- Datadog
- New Relic
- Prometheus
- Graphite

Requirements clarification

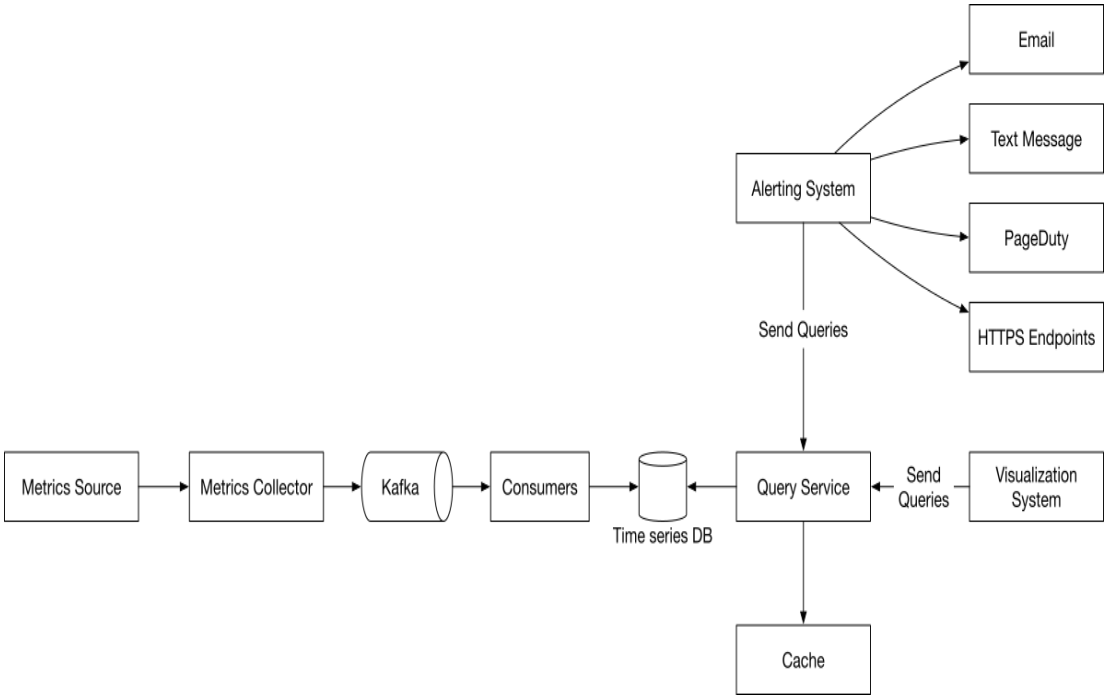
- **Functional requirements**
  - Collect a variety of metrics
    - ✓ CPU usage
    - ✓ Request count
    - ✓ Memory usage
    - ✓ Message count in message queues
  - Send alert notifications to various alerting destinations.
    - ✓ Email
    - ✓ Phone
    - ✓ PagerDuty
    - ✓ Webhooks
  - Support metrics visualization
  - Data retention policy:
    - ✓ Raw form for 7 days.
    - ✓ 1-minute resolution for 30 days.
    - ✓ 1-hour resolution for 1 year.
- **Non-functional requirements**
  - ✓ **Scalability**: The system should be scalable to accommodate growing metrics and alert volume.

- ✓ **Low latency**: The system needs to have low query latency for dashboards and alerts.
- ✓ **Reliability**: The system should be highly reliable to avoid missing critical alerts.
- ✓ **Flexibility**: Technology keeps changing, so the pipeline should be flexible enough to easily integrate new technologies in the future.

System interface definition

Data model definition

High-level design



- **Metrics Source**
  - ✓ Provides metrics, like application servers, database, message queues.
- **Metrics Collector**
  - ✓ Gathers metrics data and writes data into the time-series database.
- **Kafka**
  - ✓ Decouples the data collection and data processing services from each other.

- ✓ Prevents data loss when the database is unavailable, by retaining the data in Kafka.
- ✓ It could be entirely replaced by a competent time-series database (Facebook's Gorilla).

- **Time-series database**
  - ✓ Stores metrics data as time series.
  - ✓ Provides a custom query interface for analyzing and summarizing a large amount of time-series data.
  - ✓ Maintains indexes on labels to facilitate the fast lookup of time-series data by labels.
- **Query Service**
  - ✓ Makes it easy to query and retrieve data from the time-series database.
  - ✓ This should be a very thin wrapper if we choose a good time-series database. It could also be entirely replaced by the time-series database's own query interface.
- **Alerting system**
  - ✓ Sends alert notifications to various alerting destinations.
- **Cache**
  - ✓ Reduces the load of the time-series database and make query service more performant.

Detailed design

Metrics collection

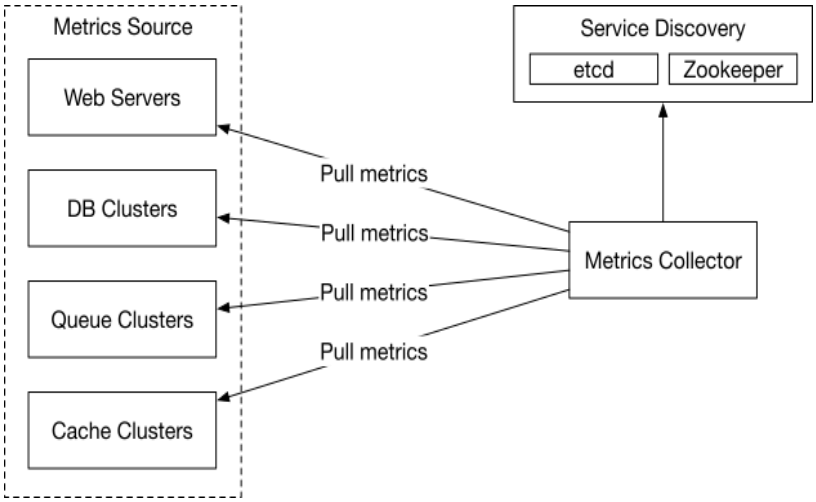
- Ways of collecting metrics

Pull model

There are dedicated metric collectors which pull metrics values from the running applications periodically via a pre-defined HTTP endpoint (for example, /metrics).

Metrics collectors will query Service Discovery to get the complete list of service endpoints to pull data from.

Use a consistent hash ring to decide each metrics collector needs to collect which set of servers.

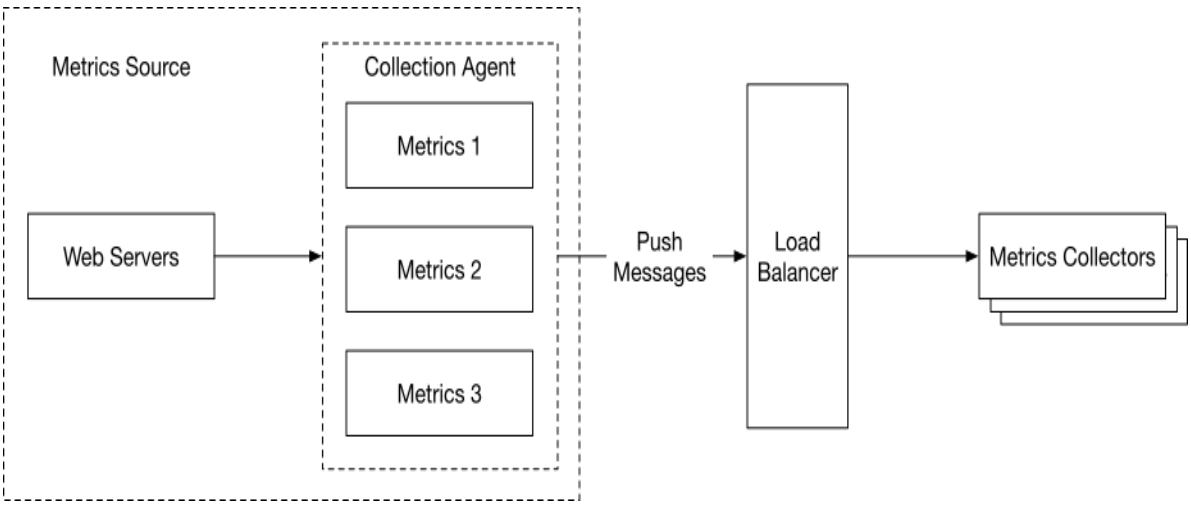


Push model

A collection agent

- Commonly installed on every server being monitored.
- Collects metrics and pushes those metrics periodically to the metrics collector.

The metrics collector should be in an auto-scaling cluster with a load balancer in front of it.

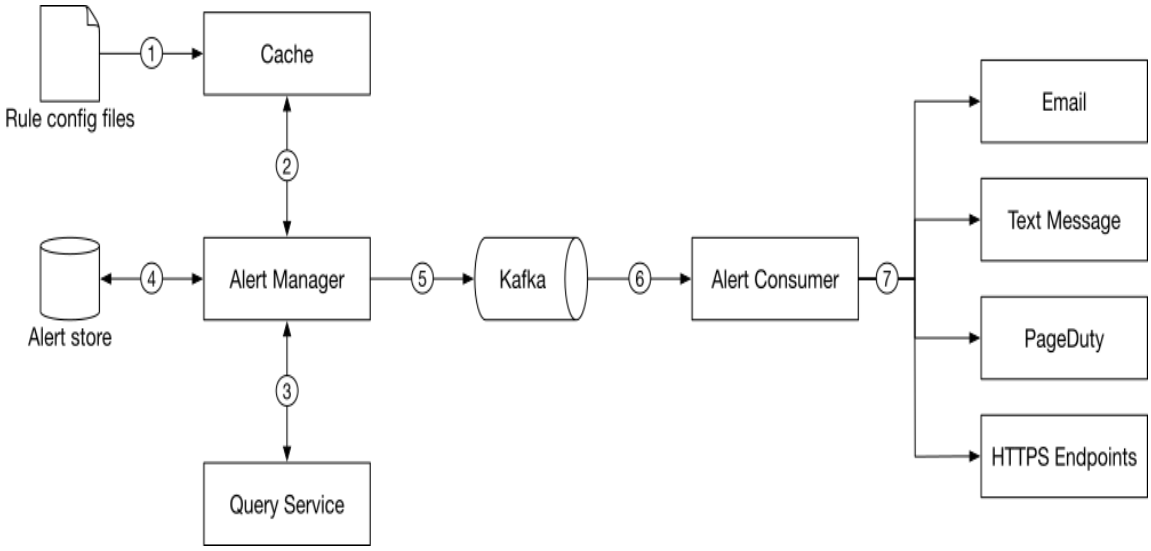


Scale the metrics transmission pipeline

Scale through Kafka

- Configure the number of partitions based on throughput requirements.
- Partition metrics data by metric names, so consumers can aggregate data by metrics names.
- Further partition metrics data with tags/labels.
- Categorize and prioritize metrics so that important metrics can be processed first.

Alerting system



Workflow

- Step 1: Load config files (rules) to cache servers.
- Step 2: The alert manager fetches alert configs from the cache.
- Step 3: Based on config rules, the alert manager calls the query service at a predefined interval.

If the value violates the threshold, an alert event is created.

- Step 4: The alert store is a key-value database and keeps the state (inactive, pending, firing, resolved) of all alerts. It ensures a notification is sent at least once.
- Step 5: Eligible alerts are inserted into Kafka.
- Step 6: Alert consumers pull alert events from Kafka.
- Step 7: Alert consumers process alert events from Kafka and send notifications over to different channels such as email, text message, PagerDuty, or HTTP endpoints.

Notes

- Consider to build or buy alerting system (PagerDuty).

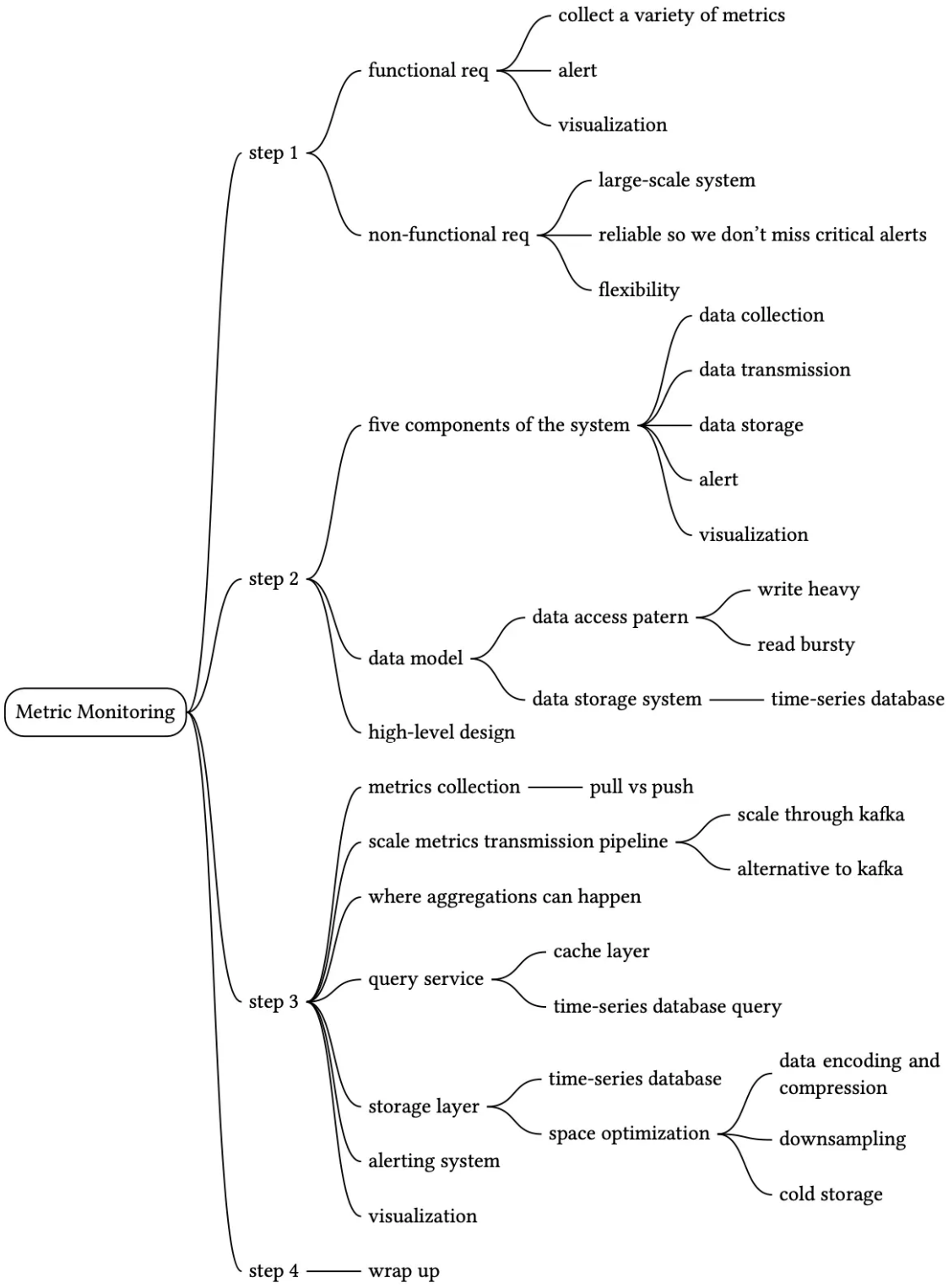
Storage

- Space optimization
  - Data encoding and compression
  - Downsampling (Convert high-resolution data to low-resolution)

Key points

- There are 2 ways to collect metrics: Pull model or Push model.
- The metrics collector cluster should be set up for auto-scaling, to ensure that there are an adequate number of collector instances to handle the demand.
- There are 3 ways to reduce disk usage: Encoding, compression and Downsampling
- Build vs buy options for alerting and visualization systems.

Summary


Metric Monitoring
step 1
functional req
collect a variety of metrics
alert
visualization
non-functional req
large-scale system
reliable so we don't miss critical alerts
flexibility
step 2
five components of the system
data collection
data transmission
data storage
alert
visualization
data model
data access patern
write heavy
read bursty
data storage system
time-series database
high-level design
step 3
metrics collection
pull vs push
scale metrics transmission pipeline
scale through kafka
alternative to kafka
where aggregations can happen
query service
cache layer
time-series database query
storage layer
time-series database
space optimization
data encoding and compression
downsampling
cold storage
alerting system
visualization
step 4
wrap up