Metrics Monitoring and Alerting System

12/04/2022

In house system vs Sass

Four golden metrics

- **延迟(Latency)**:服务请求所需耗时,例如HTTP请求平均延迟。需要区分成功请求和失败请求,因为失败请求可能会以非常低的延迟返回错误结果。
- **流量(Traffic)**:衡量服务容量需求(针对系统而言),例如每秒处理的HTTP请求数或者数据库系统的事务数量。
- **错误(Errors)**:请求失败的速率,用于衡量错误发生的情况,例如HTTP 500错误数等显式失败,返回错误内容或无效内容等隐式失败,以及由策略原因导致的失败(比如强制要求响应时间超过30ms的请求为错误)。
- **饱和度(Saturation)**:衡量资源的使用情况,例如内存、CPU、I/O、磁盘使用量(即将饱和的部分,比如正在快速填充的磁盘)。

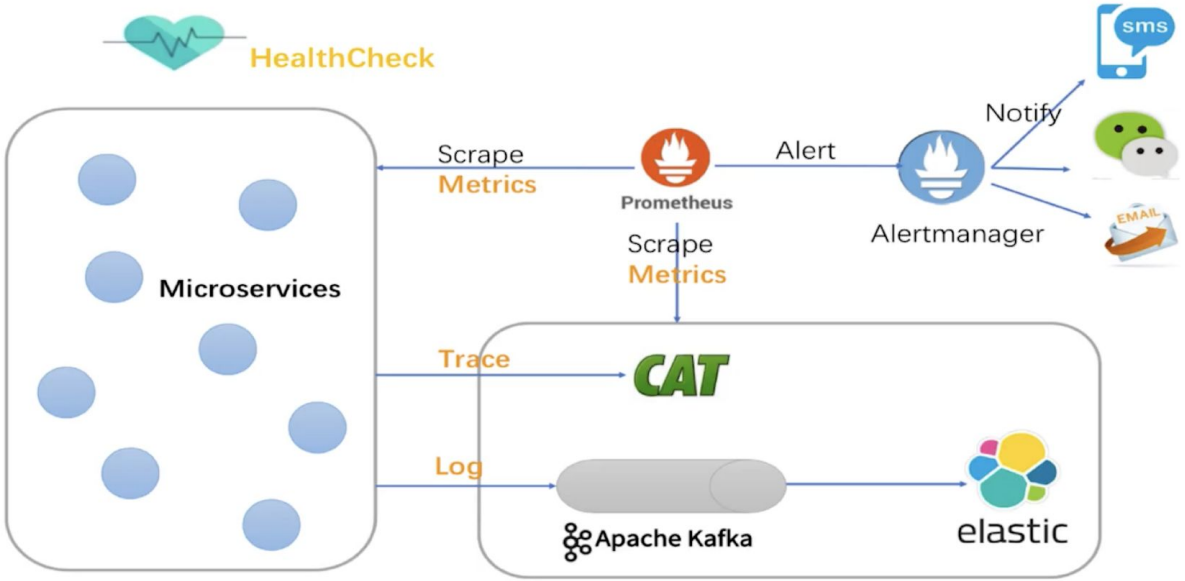
Netflix的USE方法:
使用率Utilization:关注系统资源的使用情况。这里的资源主要包括但不限于CPU、内存、网络、磁盘等。100%的使用率通常是系统性能瓶颈的标志。Weave Cloud的RED方法
饱和度Saturation:例如CPU的平均运行排队长度,这里主要是针对资源的饱和度(注意,不同于四大黄金指标)。任何资源在某种程度上的饱和都可能导致系统性能的下降。
错误Error:错误数。例如,网卡在数据包传输过程中检测到以太网络冲突了14次

Weave Cloud的RED方法 (Microservices, Cloud Native):
(Request)Rate:每秒接收的请求数。
(Request)Errors:每秒失败的请求数。
(Request)Duration:每个请求所花费的时间,用时间间隔表示。

组件	延迟	通讯量	错误	饱和度	其它
Web服务	响应时间	请求数	4xx/5xx	Token bucket, 线程池等	
数据库	查询时间	请求量	请求超时	连接数	主从延迟
缓存	请求时间	请求量	请求超时	连接数 使用百分比	命中率
消息队列	请求时间	请求量	请求超时		消息堆积
第三方服务	请求时间	请求量	错误码 请求超时		

Not

- Log Monitoring
- Distributed System Tracing



Monitoring/Alerting System 5 Fundamental Components

- Data Collection
- Data Transmission
- Data Storage
- Alerting
- Visualization

Understand Time Series and Metrics (line protocol)

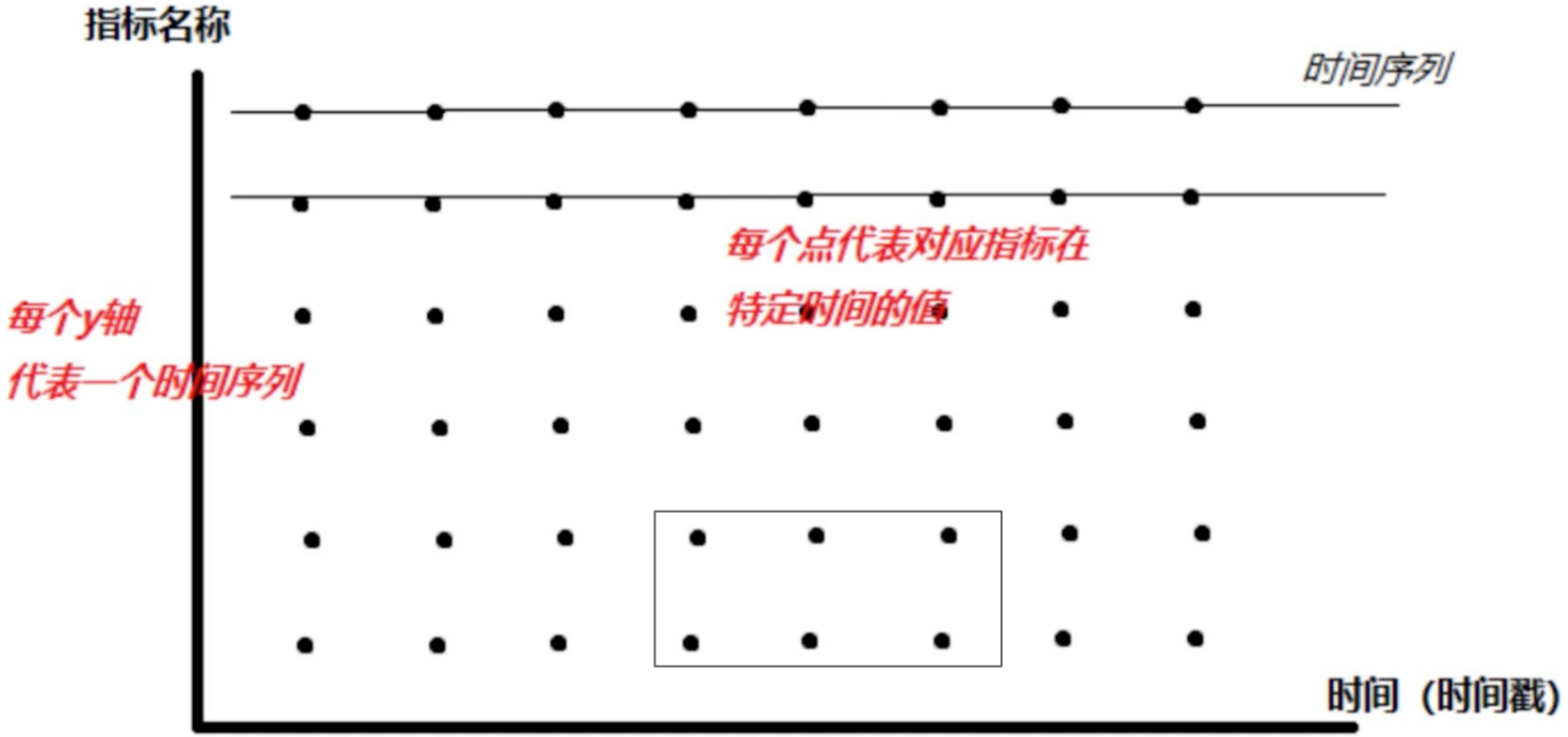
metric_name	cpu.load
labels	host:i631, env:prod
timestamp	1613707265
value	0.09

Name	Type
A metrics name	String
A set of tags/labels	List of <kv> paris
An array of values and their timestamps	An array of <KV> pairs

```
CPU.load host=webserver01,region=us-west-2 1613707265 50
CPU.load host=webserver01,region=us-west-2 1613707265 62
CPU.load host=webserver02,region=us-west-2 1613707265 43
CPU.load host=webserver02,region=us-west-2 1613707265 53
...

CPU.load host=webserver01,region=us-west-2 1613707265 76
CPU.load host=webserver01,region=us-west-2 1613707265 83
```

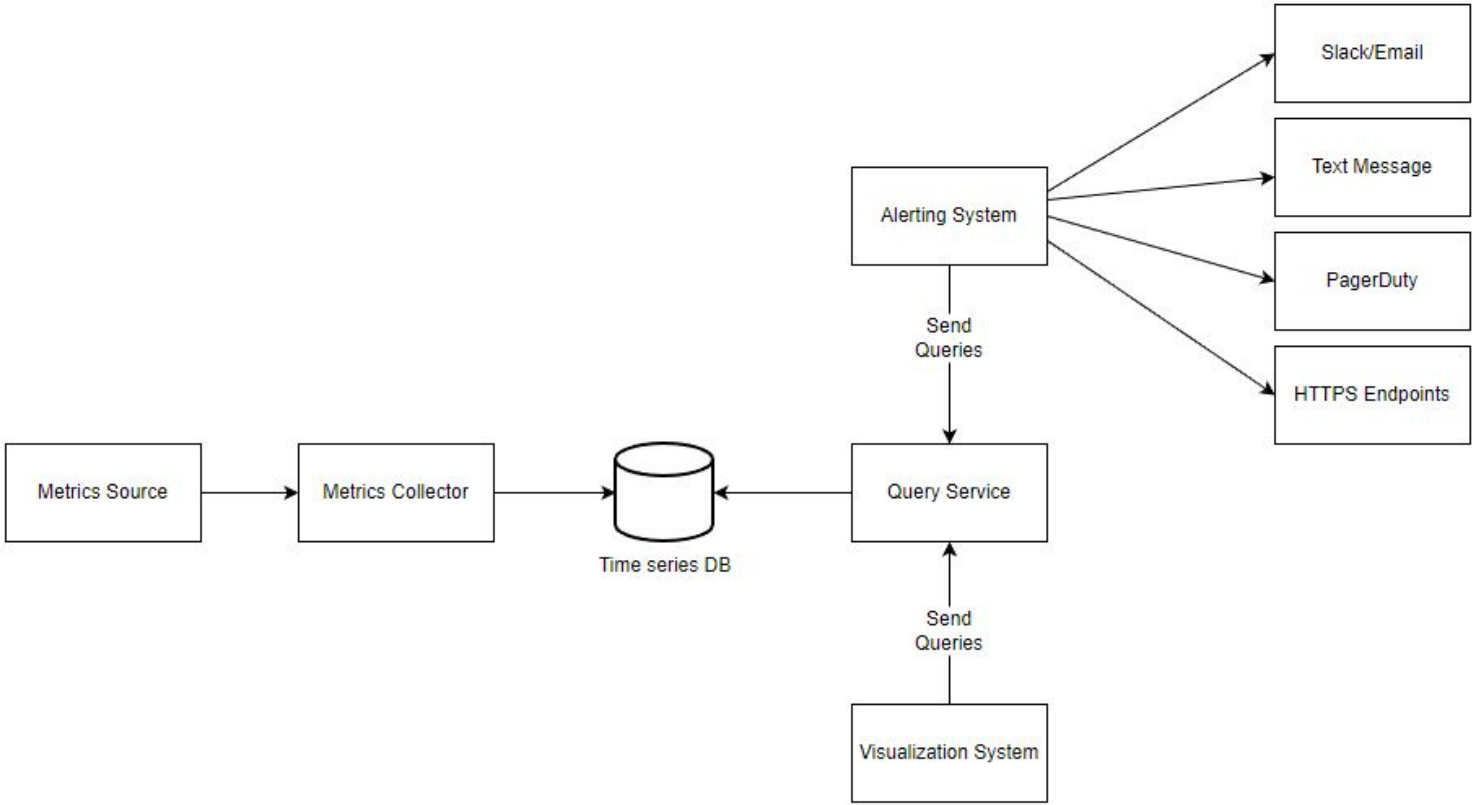
Understand Time Series and Metrics



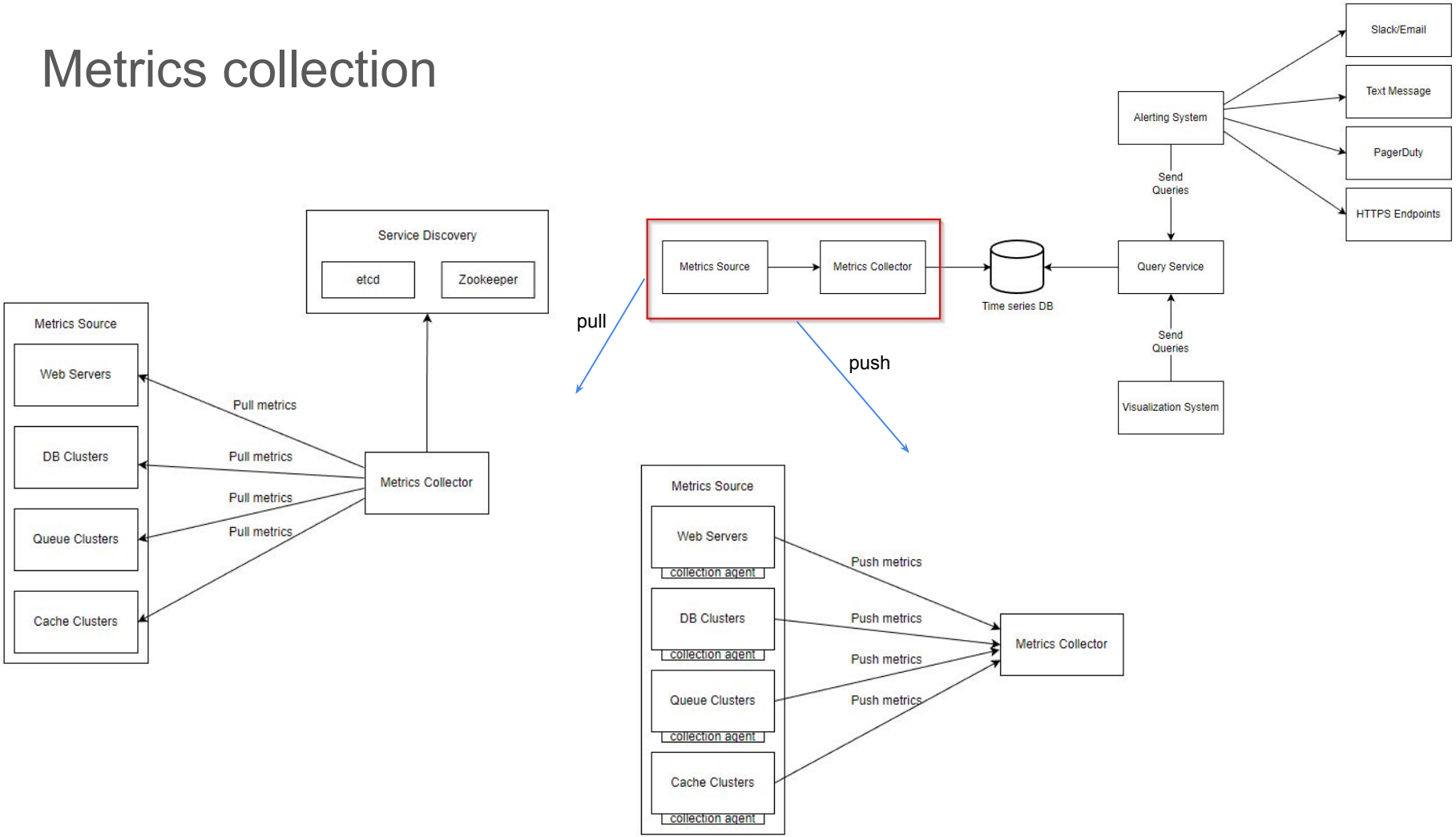
Traffic and Data Storage

- Write - constant heavy
- Read spiky
- SQL vs NoSQL
- Time Series DB

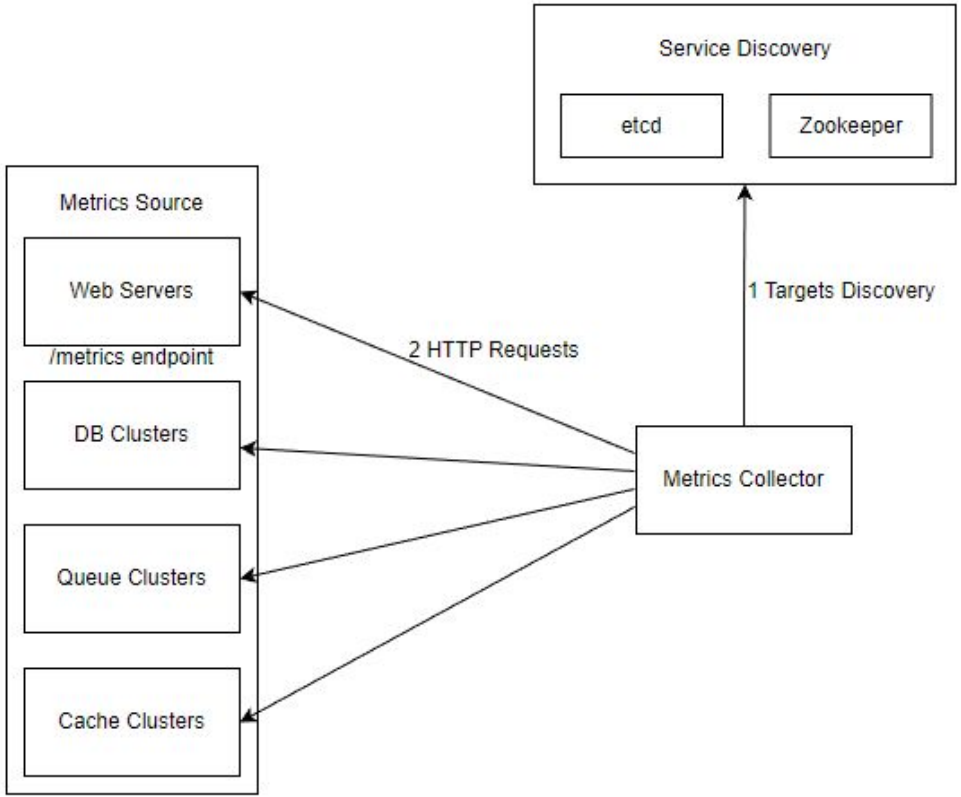
High Level Design



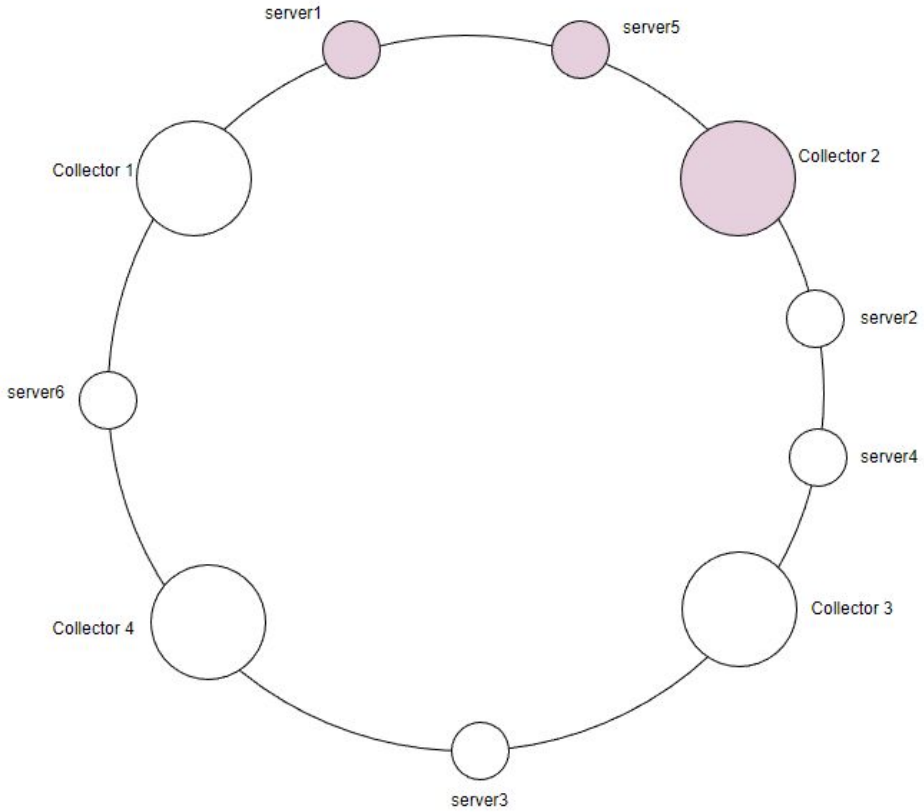
Metrics collection



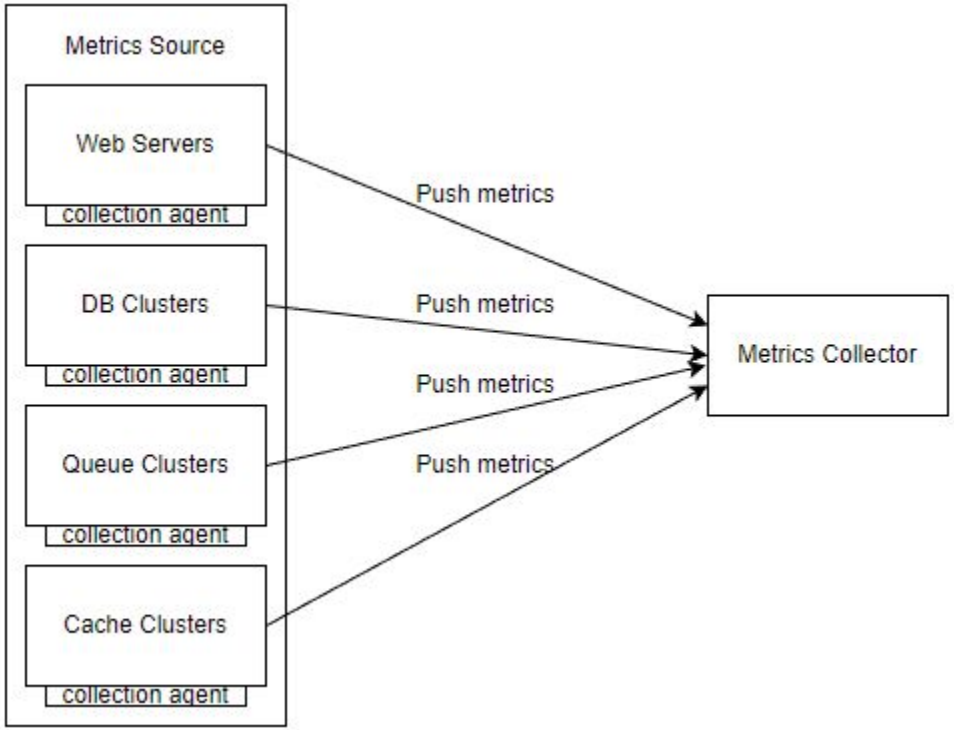
Metrics collection - Pull model



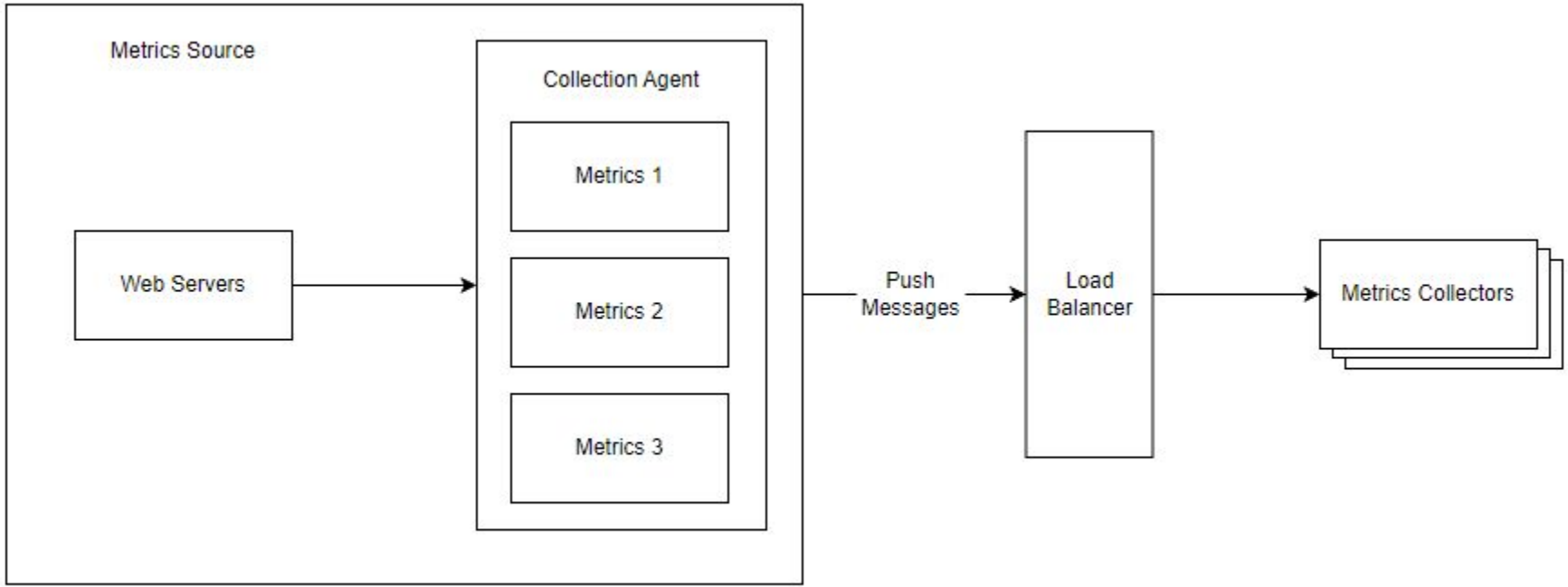
Metrics collection - Pull model on Auto-scaling



Metrics collection - Push model



Metrics collection - Push models Auto-scaling



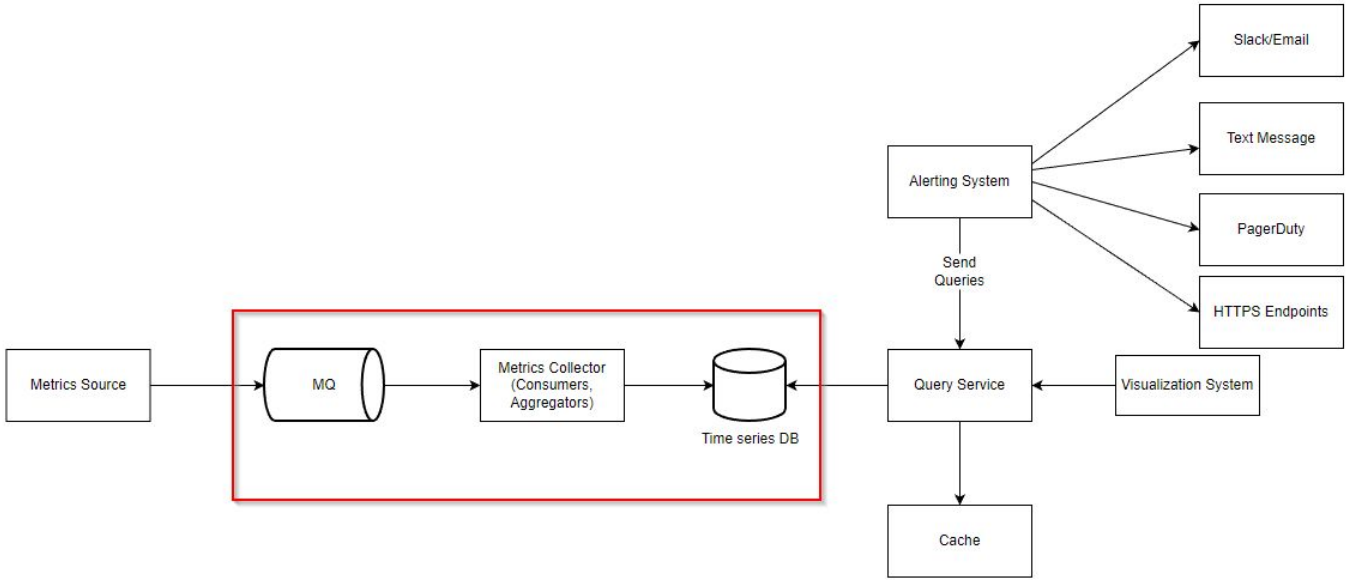
Pull vs Push (sometime both)

	Pull	Push
Easy debugging	The /metrics endpoint on application servers used for pulling metrics can be used to view metrics at any time. Even on your laptop. **Pull wins**	
Health check	If an application server doesn't respond to the pull, can quickly figure out if an application server is down. **Pull wins**.	If the metrics collector doesn't receive metrics, the problem might be caused by network issues.
Short-lived jobs		Some of the batch might be short-lived and don't last long enough to be pulled. **Push wins**. This can be fixed by introducing push gateways for the pull models.
Firewall or complicated network setups	Having servers pulling metrics requires all metric endpoints to be reachable. This is potentially problematic in multiple data center setups. It might require a more elaborate network infrastructure.	If the metrics collector is set up with a LB and an auto-scaling group, it is possible to receive data from anywhere. **Push wins**.

	Pull	Push
Performance	Pull methods typically use TCP.	Push methods typically use UDP. This means the push method provides lower-latency transports of metrics. The counterargument here is that the effort of establishing a TCP connection is small compared to sending the metrics payload.
Data authenticity	App servers to collect metrics from are defined in config files in advance. Metrics gathered from those servers are guaranteed to be authentic.	Any kind of client can push metrics to the metrics collector. This can be fixed by whitelisting servers from which to accept metrics, or by requiring authentication.
Fault Tolerance	Asyn	Syn

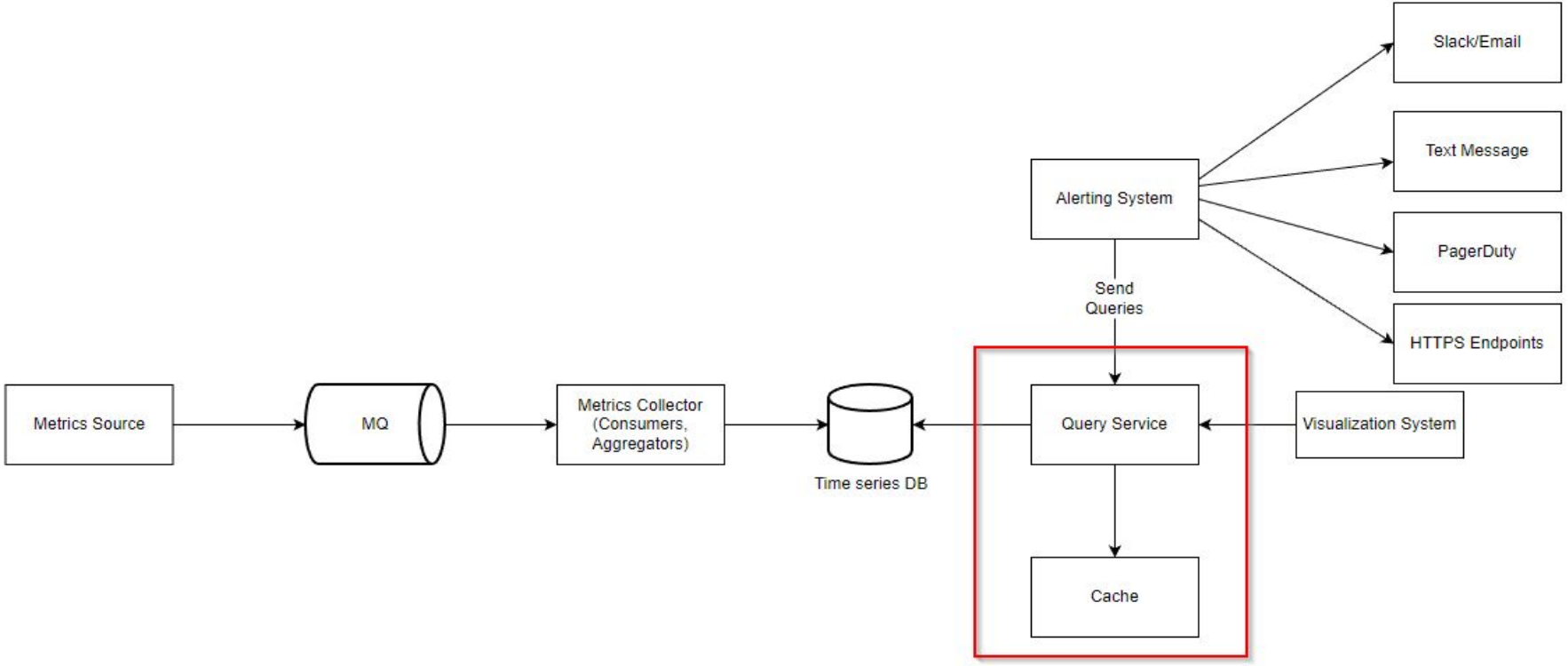
Metrics Transmission

- MQ
- Aggregation (Collection agent, Ingestion pipeline, Query side)



Query Service

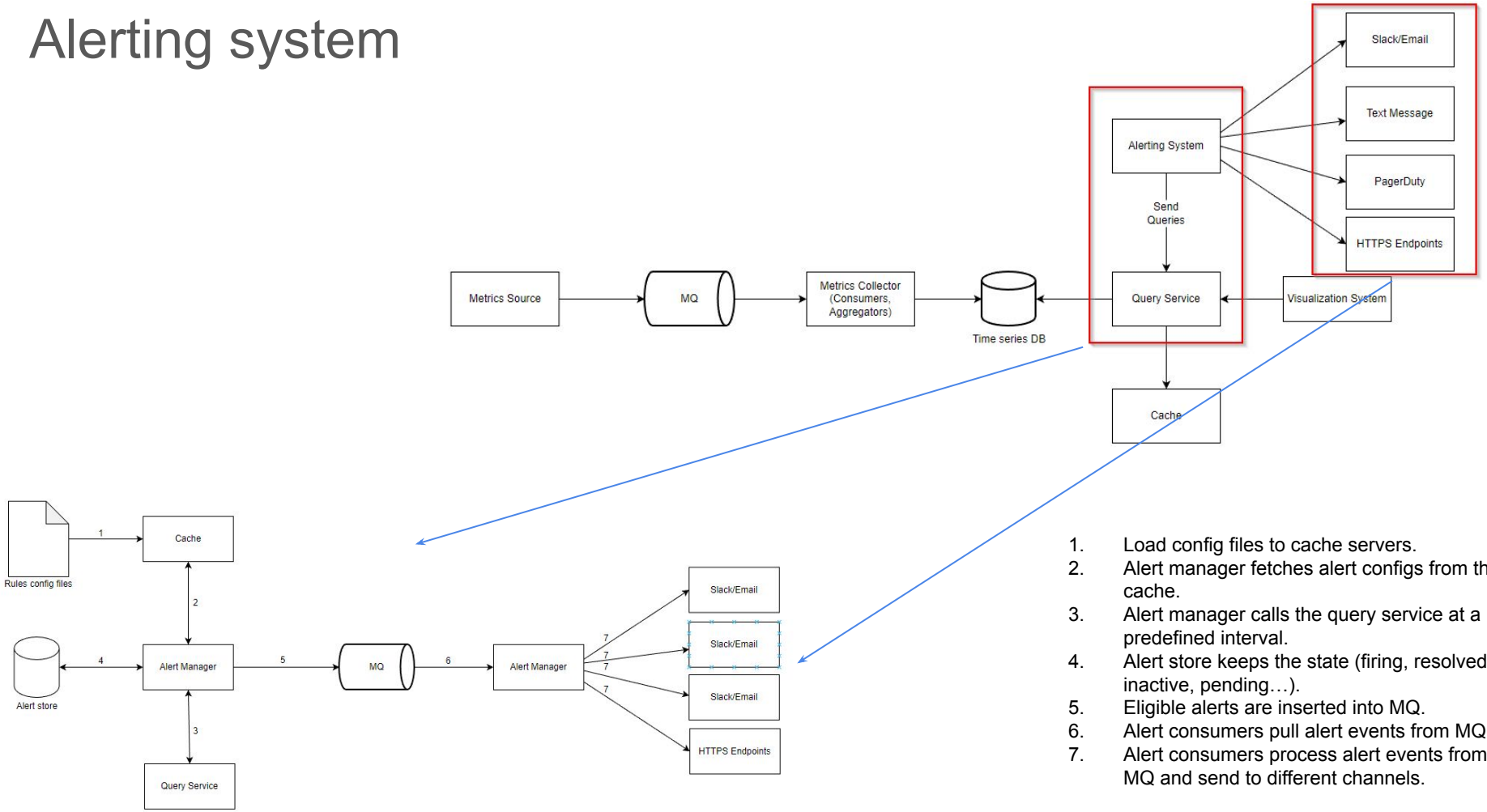
- Dedicated cluster of query servers
- Cache Layer



Space optimization

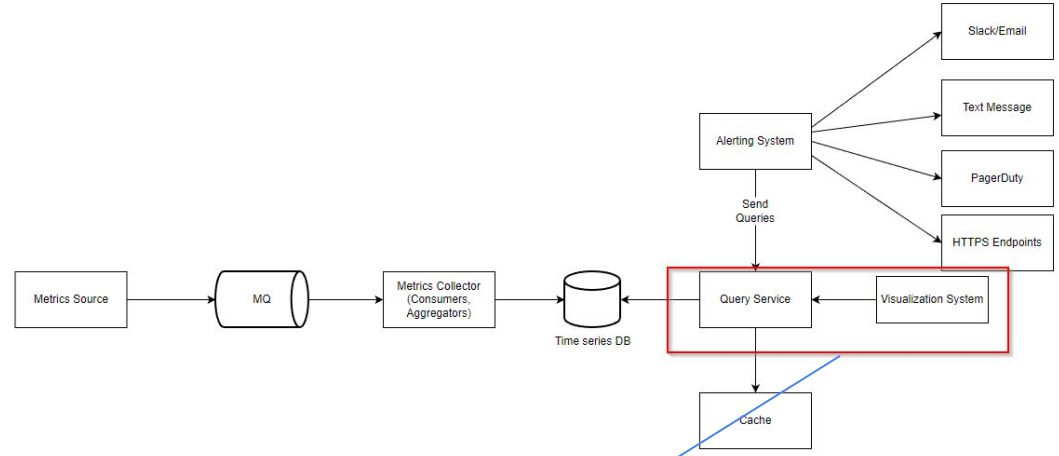
- Data encoding and compression - delta
- Downsampling (special aggregation by time)
- Cold storage (Glacier, Coldline)

Alerting system

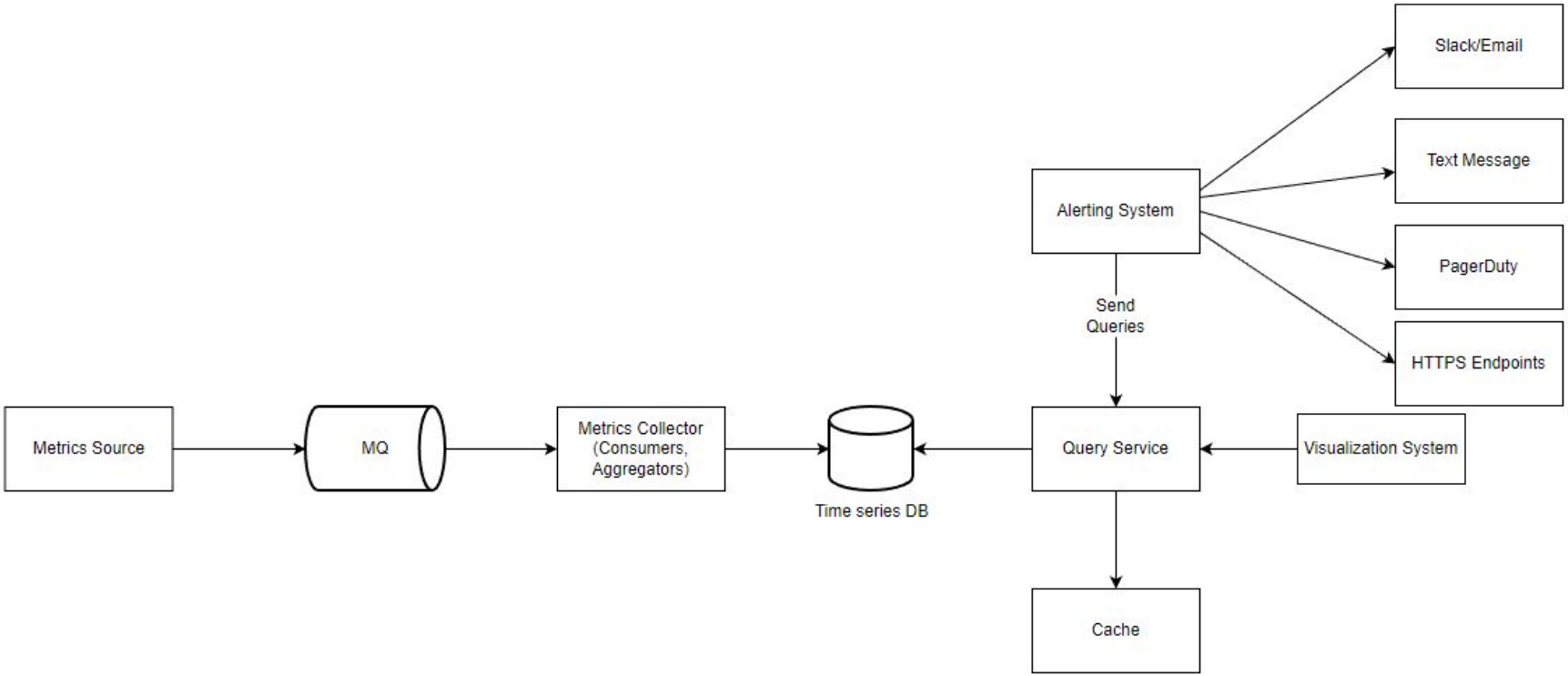


1. Load config files to cache servers.
2. Alert manager fetches alert configs from the cache.
3. Alert manager calls the query service at a predefined interval.
4. Alert store keeps the state (firing, resolved, inactive, pending…).
5. Eligible alerts are inserted into MQ.
6. Alert consumers pull alert events from MQ.
7. Alert consumers process alert events from MQ and send to different channels.

Visualization system



Final Design



Furthermore

.rrd

References

- https://sre.google/books/
- https://prometheus.io/docs/alerting/latest/alertmanager/
- https://prometheus.io/docs/prometheus/latest/configuration/recording_rules/
- https://ahelpme.com/software/grafana/create-graph-for-linux-cpu-usage-using-grafana-influxdb-and-collectd/
- https://grafana.com/oss/phlare/
- https://www.researchgate.net/publication/320607363_Large-scale_time_series_data_down-sampling_based_on_Map-Reduce_programming_mode