Video 31. Distributed Logging & Metrics System Design | Microservices Logging

Lets see all the non-functional and functional requirements

Logger Service

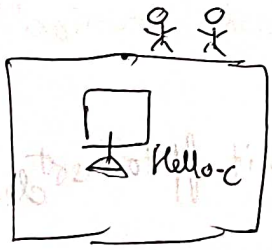
Lets say to begin with we take a simple C file

→ printf("Helloworld"); → this should get logged.

Hello-c

gcc Hello-c →

Now lets say we uploaded our Application on a server.

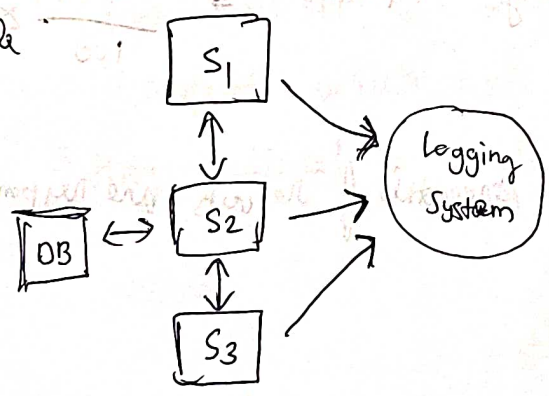


Server.

1) Now I want to log how many users are using my application.
2) How many requests are getting served.
3) What different types of Errors are there.

} For this Logging System is Required.

Lets say we broke our Application to multiple parts and multiple services.



(Functional Requirements)

① Centralised Logging Repasitory [Splunk, CloudWatch].

② Metrics Publishing → the hosts we have / pods we want to see their →{ Graph plotting Capability, + Setting up Alerts }

1) disk Usage
2) CPU Utilisation
3) Heap Utilisation.
4). API hits/ errors → (5xx, 4xx Errors)

③ Capability of Searching logs [like in splunk].

(Non-Functional Requirements)

① Scalability → For now we only have 3 microservices whose logging we are handling. In future if I have 100 services. I should be able to log it

② Availability → Logger Service should not be down anyday and we do not get to store logs of that particular day

③ Latency → Minimum Latency. No defined Number.

④ Eventual Consistency is fine.

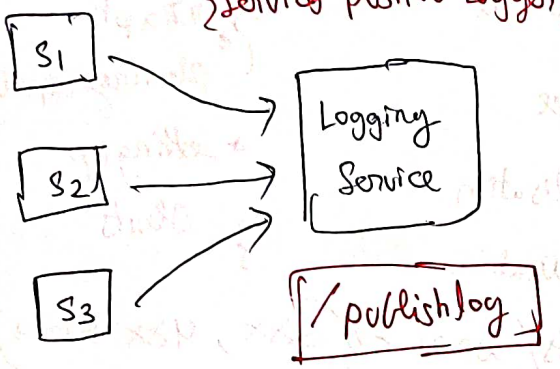
⑤ Durability / Retention → Data is Persisted.

Now Lets move to actual designing.

Two ways to design it.

Push Mechanism

{Services push to Logger}



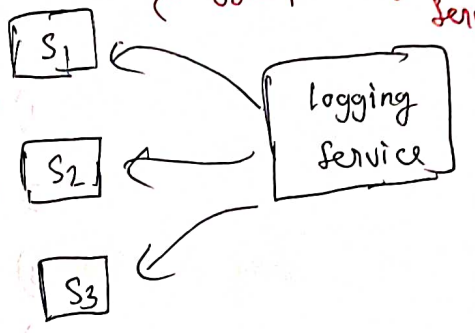
API exposed.

* In this case as soon as logs are produced it pushes to the service, by this the latency will be less

But a problem here is that if new microservices come and push to Logging Service, and Service is not that scaled up to handle the load.

Pull Mechanism

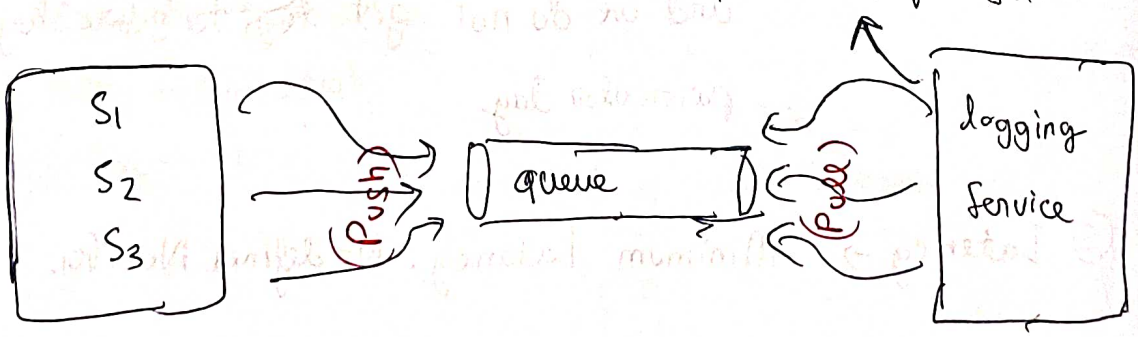
{Logger pulls from services}



* If we introduce pull service then in some later stage a new microservice will get added and we have to configure our Logging Service Accordingly.

[Mixture of Push-Pull]

As per scalability of service it pulls messages from service



this queue will be a distributed ~~topic~~ queue basically.

Lets say a Kafka cluster. The cluster will have 2 topics.

1) Application Log → to store Application Logs
2) metrics Log. → to store metrics

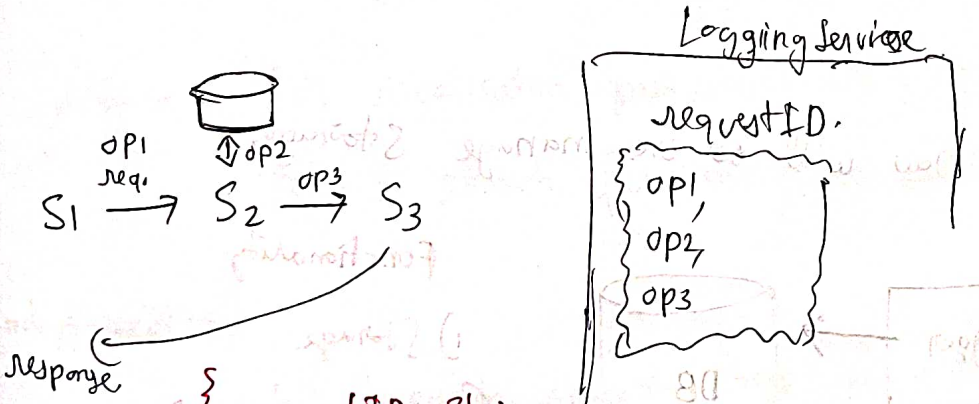
(Make the publishing of messages a Async Process.
And on the consumer side, configure the Brokers and
have consumer group separately for each consumer.)

Lets now focus on the storage or storing of ~~App~~ the logs.

For this we need to first see the structure of the logs.

(**) If you Remember we want the search functionality of the logs

1) Application Logs → Could be searched by requestID



{ requestID: String
timestamp:
metadata: { hostName:
WorkerNode:
ip Address:
⋮ }
logMessage: { }
}

(Structure of Application Log.)

Structure of

Metric Log → CPU Utilisation

{ request id

timestamp : { 12:01 pm - 12:02 pm }

metric → CPU Utilisation

}

this is required as we publish CPU utilisation of 1 minute timestamp range

(Memory Estimation)

Application log → 1KB String

Requests/second

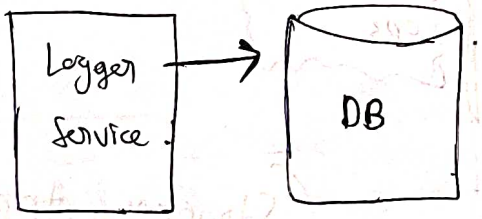
leading to logs/~~second~~ → 10K logs/second

Retention Period → 1 year.

Memory Required → $(1KB \times 10{,}000) \times 60 \times 60 \times 24 \times 365$

$\boxed{\text{Storage}}$

Now lets see how will ~~we~~ we manage storage



Functionality

1) Storage

(2) Search by logs.)

(~~3~~)

Elastic Search is the best option to search for logs and it even allows us the storage facility.
Provides indexing.

{ELK feature widely popular.}

logstash → Elastic Search → Kibana.

logstash → Works as the consumer which pulls the logs from the queue (topic) and does some pre processing.
Transformation, into storable format.
(Like Hiding sensetive data).

Elastic Search → Creates Indexes on the data received from Logstash

Kibana → For visualisation purposes. Its a dev tool which gives us search functionality as well

(2nd Alternative)
We use a File System to store the logs. But we would need the search capability as well.

Choose (HDFS with MapReduce and Spark for searching) [We will read in detail in Data Intensive Applications. Chapter 10]

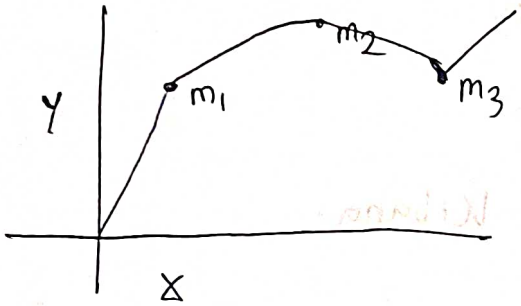
② Metric Log

⇓

{Plotting Graphs}

Lets say we are receiving events as per timestamps.

m① { } m② { }



→ to be able to do this plotting we will be using Time Series Database

we will Read about this in-detail in Future and make Notes.

(Examples of Time Series DB)

Influx DB

Prometheous

[Read More]

Final Architecture

1. Improvements we can do here is instead of pushing single log file into queue, Batch fully together and send → (Save CPU cycles)

S_1 S_2 S_3

D_1 D_2 D_3

(Push)

Distributed queue
Two topics

(Pull)

logger Service

Option 2 → HDFS

(Option 1 : Elastic Search)

Elastic Search → Kibana

Frontend Service Exposed to User

(User)

(timestamp)
(Search String)
(Returns Response)

Search Service
Runs Map-Reduce on HDFS

2. Logs Stored wherever can be compressed to save Storage