Introduction to Distributed Logging

Learn about why we need logging in a distributed system.

We'll cover the following

Logging in a distributed system

In today's world, an increasing number of designs are moving to microservice architecture instead of monolithic architecture. In microservice architecture, logs of each microservice are accumulated in the respective machine. If we want to know about a certain event that was processed by several microservices, it is difficult to go into every node, figure out the flow, and view error messages. But, it becomes handy if we can trace the log for any particular flow from end to end.

Moreover, it is also not necessary that a microservice is deployed on only one node. It can be deployed on thousands of nodes. Consider the following example, where hundreds of microservices are interdependent, and failure of one service can result in failures of other services. And if we do not have logs, we might not determine the root cause of failure. This emphasizes the need for logging.

Restrain the log size

The number of logs increases over time. At a time, perhaps hundreds of concurrent messages need to be logged. But the question is, are they all important enough to be logged? To solve this, logs have to be structured. We need to decide what to log into the system on the application or logging level.

Use sampling

We'll determine which messages we should log into the system in this approach. Consider a situation where we have lots of messages from the same set of events. For example, there are people commenting on a post, where Person X commented on Person Y's post, then Person Z commented on Person Y's post, and so on. Instead of logging all the information, we can use a **sampler service** that only logs a smaller set of messages from a larger chunk. This way, we can decide on the most important messages to be logged.

> **Note:** For large systems like Facebook, where billions of events happen per second, it is not viable to log them all. An appropriate sampling threshold and strategy are necessary to selectively pick a representative data set.

We can also categorize the types of messages and apply a filter that identifies the important messages and only logs them to the system.

Point to Ponder

Question

What is a scenario where the sampling approach will not work?

Show Answer

Use categorization

Let's look into the logging support provided by various programming languages. For example, there's log4j and logging in Python. The following severity levels are commonly used in logging:

- `DEBUG`
- `INFO`
- `WARNING`
- `ERROR`
- `FATAL/CRITICAL`

Usually, the production logs are set to print messages with the severity of `WARNING` and above. But for more detailed flow, the severity levels can be set to `DEBUG` and `INFO` levels too.

Click on the "Run" button to see the execution of an example that uses Python's logging library to print logs.

```
import logging as log
# set the logging level to DEBUG
log.basicConfig(level=log.DEBUG)
for i in range(6):
    if i == 0:
```

```
            log.debug("Debug level")
        elif i == 1:
            log.info("Info level")
        elif i == 2:
            log.warning("Warning level")
        elif i == 3:
            log.error("Error level")
        elif i == 4:
            log.critical("Critical level")
        elif i == 5:
            print("Uncomment the following to view a system generated erro
            #print(3/0)
```

Run Save Reset

> The output will be similar to:
>
> ```
> DEBUG:root:Debug level
> INFO:root:Info level
> WARNING:root:Warning level
> ERROR:root:Error level
> CRITICAL:root:Critical level
> ```

Let's uncomment **line 17** to view a system-generated error over the division of an integer by zero. Python itself logs the error to the console:

Structure the logs

Applications have the liberty to choose the structure of their log data. For example, an application is free to write to log as binary or text data, but it is often helpful to enforce some structure on the logs. The first benefit of structured logs is better interoperability between log writers and readers. Second, the structure can make the job of a log processir system easier.

> **Note:** The structuring of logs is a dense topic in itself. We refer the interested learners to the PhD thesis by Ryan Braud titled "Query-based debugging of distributed systems."

Points to consider while logging

We should be careful while logging. The logging information should only contain the relevant information and not breach security concerns. For secure data, we should log encrypted data. We should consider the following few points while logging:

- Avoid logging personally identifiable information (PII), such as names, addresses, emails, and so on.
- Avoid logging sensitive information like credit card numbers, passwords, and so on.
- Avoid excessive information. Logging all information is unnecessary. It only takes up more space and affects performance. Logging, being an I/O-heavy operation, has its performance penalties.
- The logging mechanism should be secure and not vulnerable because logs contain the application's flow, and an insecure logging mechanism is vulnerable to hackers.

Vulnerability in logging infrastructure

A **zero-day vulnerability** in Log4j, a famous logging framework for Java, has been identified recently. Log4j has contained the hidden vulnerability, Log4Shell (CVE-2021-44228), since 2013. Apache gave the highest available score, a CVSS severity rating of 10, to Log4Shell. The exploit is simple to execute and affects hundreds of millions of devices. Security experts are convinced that this vulnerability can allow devastating cyberattacks internationally because it can enable attackers to run malicious code and take control of the machine.