Microservices II

Teaching Faculty: Umur INAN

Prepared by Umur INAN

SLEUTH

- Spring Cloud Sleuth provides Spring Boot auto-configuration for distributed tracing.
- Sleuth configures everything you need to get started.
  - Where trace data (spans) are reported to.
  - How many traces to keep (sampling).
  - If remote fields (baggage) are sent.

SLEUTH

- Spring Cloud Sleuth is able to trace your requests and messages so that you can correlate that communication to corresponding log entries.
- You can also export the tracing information to an external system to visualize latency.
- Spring Cloud Sleuth supports OpenZipkin compatible systems directly.

TERMINOLOGY

- Span
  - The basic unit of work.
  - sending an RPC is a new span.
  - Spans also have other data
    - Descriptions
    - Timestamped events
    - Key-value annotations (tags)
    - The ID of the span that caused them
    - Process IDs (normally IP addresses).

TERMINOLOGY

- Trace
  - A set of spans forming a tree-like structure.

- Annotation/Event
  - Used to record the existence of an event in time.

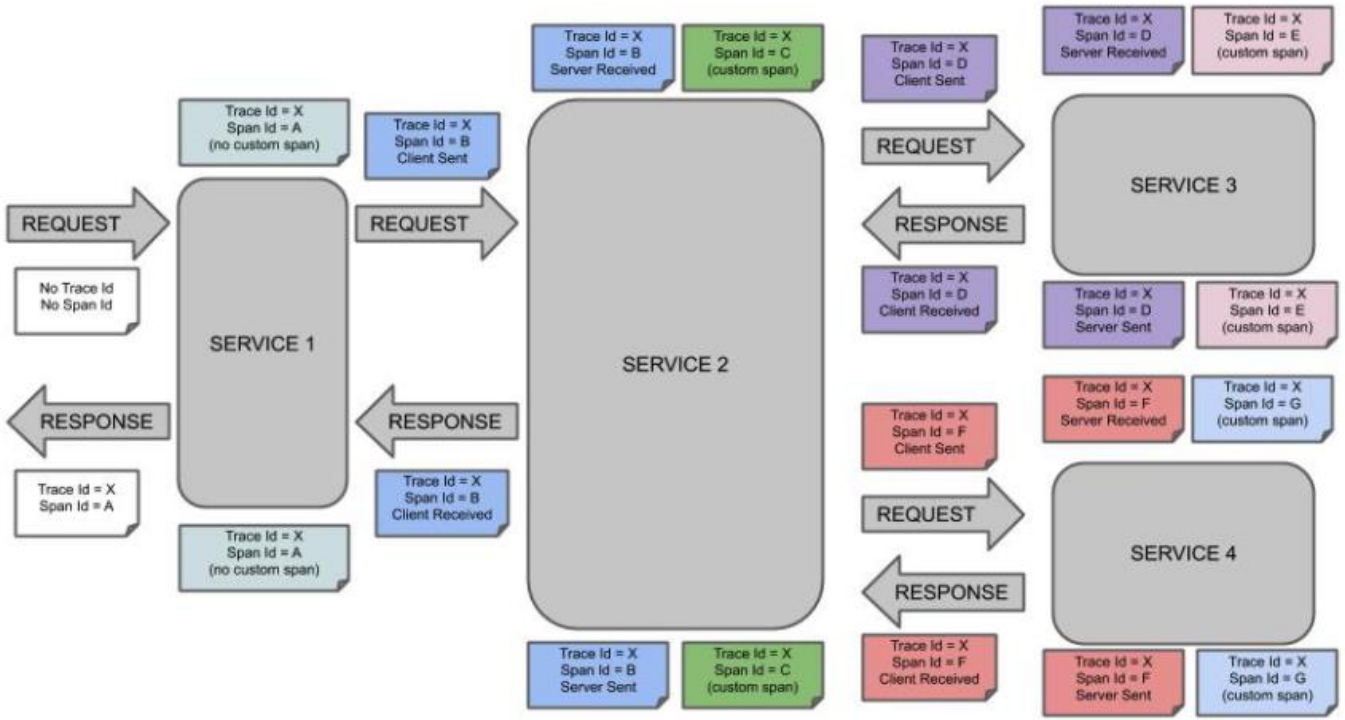
TERMINOLOGY

- cs
  - Client Sent. The client has made a request. This annotation indicates the start of the span.
- sr
  - Server Received: The server side got the request and started processing it. Subtracting the cs timestamp from this timestamp reveals the network latency.

TERMINOLOGY

- ss
  - Server Sent. Annotated upon completion of request processing (when the response got sent back to the client). Subtracting the sr timestamp from this timestamp reveals the time needed by the server side to process the request.
- cr
  - Client Received. Signifies the end of the span. The client has successfully received the response from the server side. Subtracting the cs timestamp from this timestamp reveals the whole time needed by the client to receive the response from the server.

Spans & traces



ZIPKIN

- Zipkin is a distributed tracing system.
- It helps gather timing data needed to troubleshoot latency problems in service architectures.
- Features include both the collection and lookup of this data.
- The Zipkin UI also presents a Dependency diagram showing how many traced requests went through each application.

Dependencies

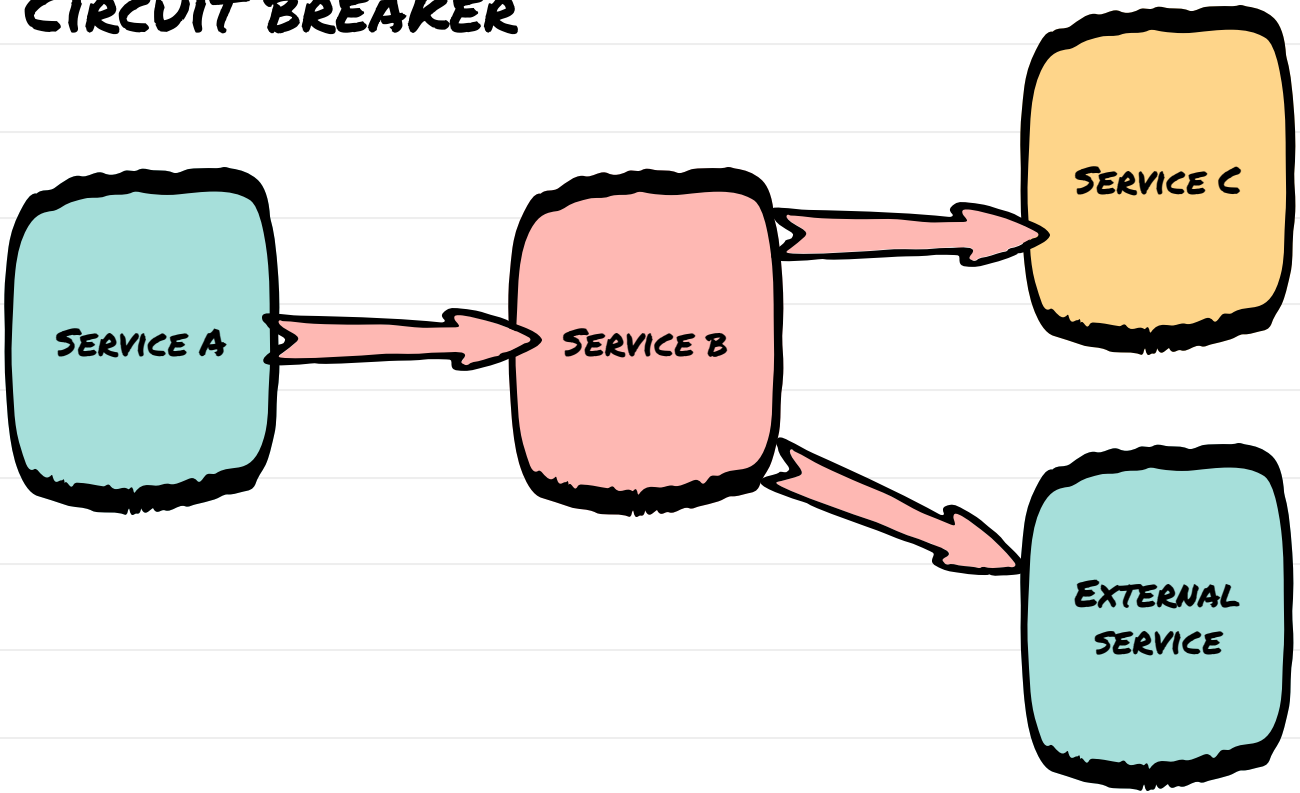
```
<dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>

<dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-sleuth-zipkin</artifactId>
</dependency>
```

Circuit Breaker

- In a microservice architecture, it's common for a service to call another service.
- There is always the possibility that the other service being called is unavailable or unable to respond.
- What to do?

Circuit Breaker



Circuit Breaker

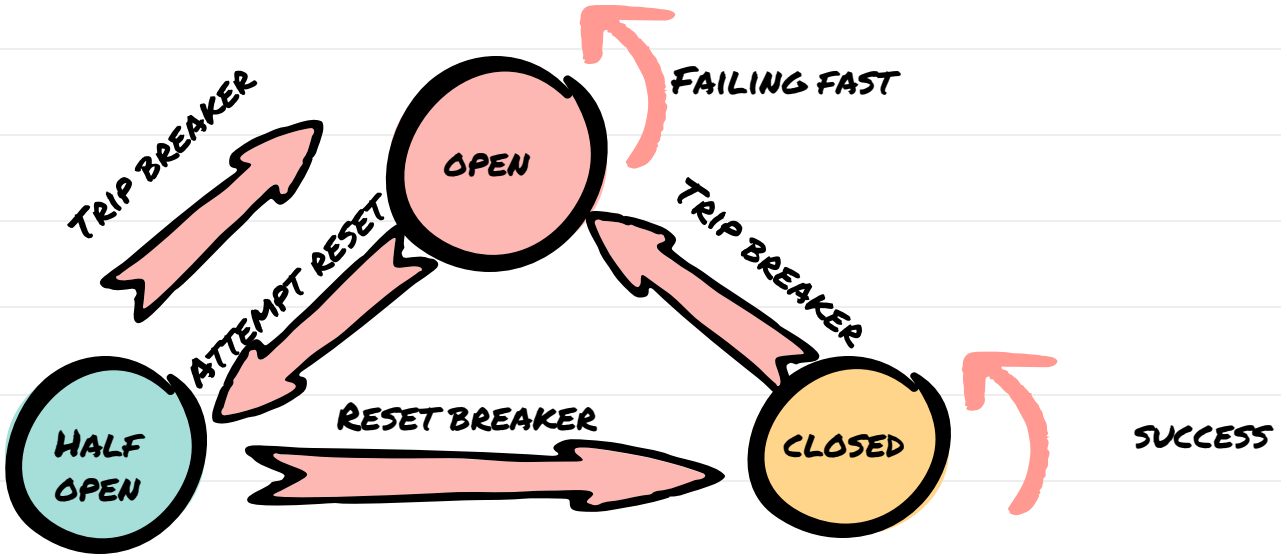
- When microservice B calls microservice C and microservice C unavailable or unable to respond, microservice B may have encountered an error.
- And if microservice A calls microservice B which is encountered an error, this causes microservice A is just wasting its resources

- When microservice B calls the external microservice and unfortunately the external microservice is down.
- Microservice B is just wasting its resources. This will also definitely affect microservice B when called by microservice A

Circuit Breaker

- The concept of a circuit breaker is to prevent calls to microservice when it's known the call may fail or time out.
- This is done so that clients don't waste their valuable resources handling requests that are likely to fail.
- Using this concept, you can give the server some spare time to recover.

Circuit Breaker State



STATES

- Closed
  - When everything is normal. Initially, the circuit breaker is in a Closed state.

STATES

- Open
  - When a failure occurs above predetermined criteria.
  - In this state, requests to other microservices will not be executed and fail-fast or fallback will be performed if available.
  - When this state has passed a certain time limit, it will automatically or according to certain criteria will be returned to the Half-Open state.

STATES

- Half-Open
  - Several requests will be executed to find out whether the microservices that we are calling are working normally.
  - If successful, the state will be returned to the Closed state.
  - However, if it still fails it will be returned to the Open state.

TYPES

- Count-Based
  - The circuit breaker switches from a closed state to an open state when the last N requests have failed or timeout.
- Time-Based
  - The circuit breaker switches from a closed state to an open state when the last N time unit has failed or timeout.

Count-based sliding window

- The count-based sliding window is implemented with a circular array of N measurements.
- If the count window size is 10, the circular array has always 10 measurements.
- The sliding window incrementally updates a total aggregation. The total aggregation is updated when a new call outcome is recorded. When the oldest measurement is evicted, the measurement is subtracted from the total aggregation and the bucket is reset. (Subtract-on-Evict)

Time-based sliding window

- The time-based sliding window is implemented with a circular array of N partial aggregations (buckets).
- If the time window size is 10 seconds, the circular array has always 10 partial aggregations (buckets).
- Every bucket aggregates the outcome of all calls which happen in a certain epoch second. (Partial aggregation)
- The head bucket of the circular array stores the call outcomes of the current epoch second.

Failure rate and slow call rate thresholds

- The state of the CircuitBreaker changes from CLOSED to OPEN when the failure rate is equal or greater than a configurable threshold. For example, when more than 50% of the recorded calls have failed.
- The CircuitBreaker also changes from CLOSED to OPEN when the percentage of slow calls is equal or greater than a configurable threshold. For example when more than 50% of the recorded calls took longer than 5 seconds.
    - This helps to reduce the load on an external system before it is actually unresponsive.

Spring Cloud Circuit Breaker

- It provides an abstraction layer across different circuit breaker implementations.
- It's a pluggable architecture.
- So, we can code against the provided abstraction/interface and switch to another implementation based on our needs.

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>
  <version>1.0.2.RELEASE</version>
</dependency>
```

Secret management

- Sensitive information management.
- Most applications need access to sensitive data in order to work properly.
  - API keys
  - Tokens
  - SSH
  - DB credentials

Secret management

- A simple solution is to store those credentials in a configuration file and read them at startup time.
  - Whoever has access to this file share the same database privileges the application have.

VAULT

- Vault is an identity-based secrets and encryption management system.
- Vault provides encryption services that are gated by authentication and authorization methods.
- Using Vault's UI, CLI, or HTTP API, access to secrets and other sensitive data can be securely stored and managed, tightly controlled (restricted), and auditable.

FEATURES

- Secure Secret Storage
  - Arbitrary key/value secrets can be stored in Vault.
  - Vault encrypts these secrets prior to writing them to persistent storage, so gaining access to the raw storage isn't enough to access the secrets.
  - Vault can write to disk, Consul, and more.

FEATURES

- Dynamic Secrets
  - Vault can generate secrets on-demand for some systems, such as AWS or SQL databases.
  - For example, when an application needs to access an S3 bucket, it asks Vault for credentials, and Vault will generate an AWS keypair with valid permissions on demand.
  - After creating these dynamic secrets, Vault will also automatically revoke them after the lease is up.

FEATURES

- Data Encryption
  - Vault can encrypt and decrypt data without storing it.
  - This allows security teams to define encryption parameters and developers to store encrypted data in a location such as a SQL database without having to design their own encryption methods.

FEATURES

- Leasing and Renewal
  - All secrets in Vault have a lease associated with them.
  - At the end of the lease, Vault will automatically revoke that secret.
  - Clients are able to renew leases via built-in renew APIs.

FEATURES

- Revocation
  - Vault has built-in support for secret revocation.
  - Vault can revoke not only single secrets, but a tree of secrets, for example all secrets read by a specific user, or all secrets of a particular type.
  - Revocation assists in key rolling as well as locking down systems in the case of an intrusion.

HCP Vault

- HCP Vault is a hosted version of Vault, which is operated by HashiCorp to allow organizations to get up and running quickly.
- HCP Vault uses the same binary as self-hosted Vault.

Spring Cloud Vault

- Allows applications to access secrets stored in a Vault instance in a transparent way.

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-vault-config</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-vault-config-databases</artifactId>
</dependency>
```