Distributed Tracing in Microservice with Spring Cloud Sleuth & Zipkin

Sleuth and Zipkin Are totally different tools from Spring Cloud Family used for Distributed Log Tracing .

Sleuth we generate using trace id & span id for tracking purpose .

Zipkin **we use for distributed tracing in microservices** .

Use case : Suppose from an application I am consuming 4 microservices , then we start distributed tracing to find out the performance issues via distributed tracing .

**We can measure which component took how much time ?**

**Trace id , Span id**

A sampler can be installed just by creating a bean definition, e.g:

*@Bean*

**public** Sampler defaultSampler() {

**return** **new** AlwaysSampler();

}

A Span is the basic unit of work. For example, sending an RPC is a new span, as is sending a response to an RPC. Span’s are identified by a unique 64-bit ID for the span and another 64-bit ID for the trace the span is a part of. Spans also have other data, such as descriptions, key-value annotations, the ID of the span that caused them, and process ID’s (normally IP address). Spans are started and stopped, and they keep track of their timing information. Once you create a span, you must stop it at some point in the future. A set of spans forming a tree-like structure called a Trace. For example, if you are running a distributed big-data store, a trace might be formed by a put request.

**@EnableZipkinServer Annotation is used to label the application class that acts as Zipkin**

If spring-cloud-sleuth-zipkin is available then the app will generate and collect Zipkin-compatible traces via HTTP. By default it sends them to a Zipkin collector service on localhost (port 9411)