

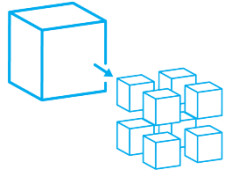
Microservicios – Arquitectura y Desarrollo

Por: Carlos Carreño

ccarrenovi@gmail.com

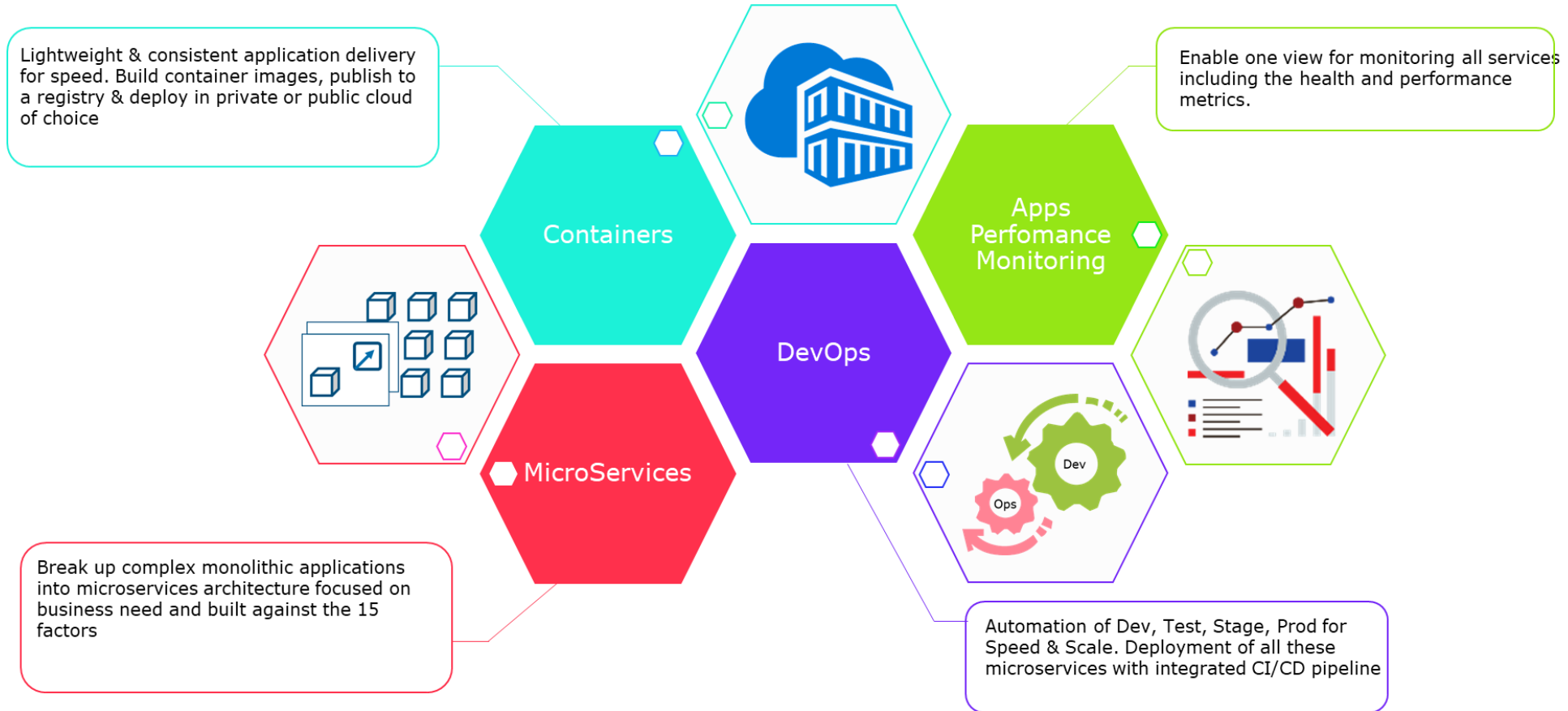
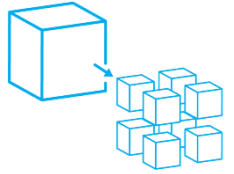
Noviembre, 2020

MÓDULO 9: Monitoreando Microservicios usando OpenZipkin, Spring Cloud Sleuth y Kibana Dashboard

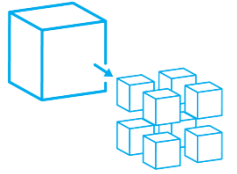


- Métricas y HealthChecks
- Trazas distribuidas
- Logs de aplicaciones y agregación de logs
- Monitoreo usando Dashboards

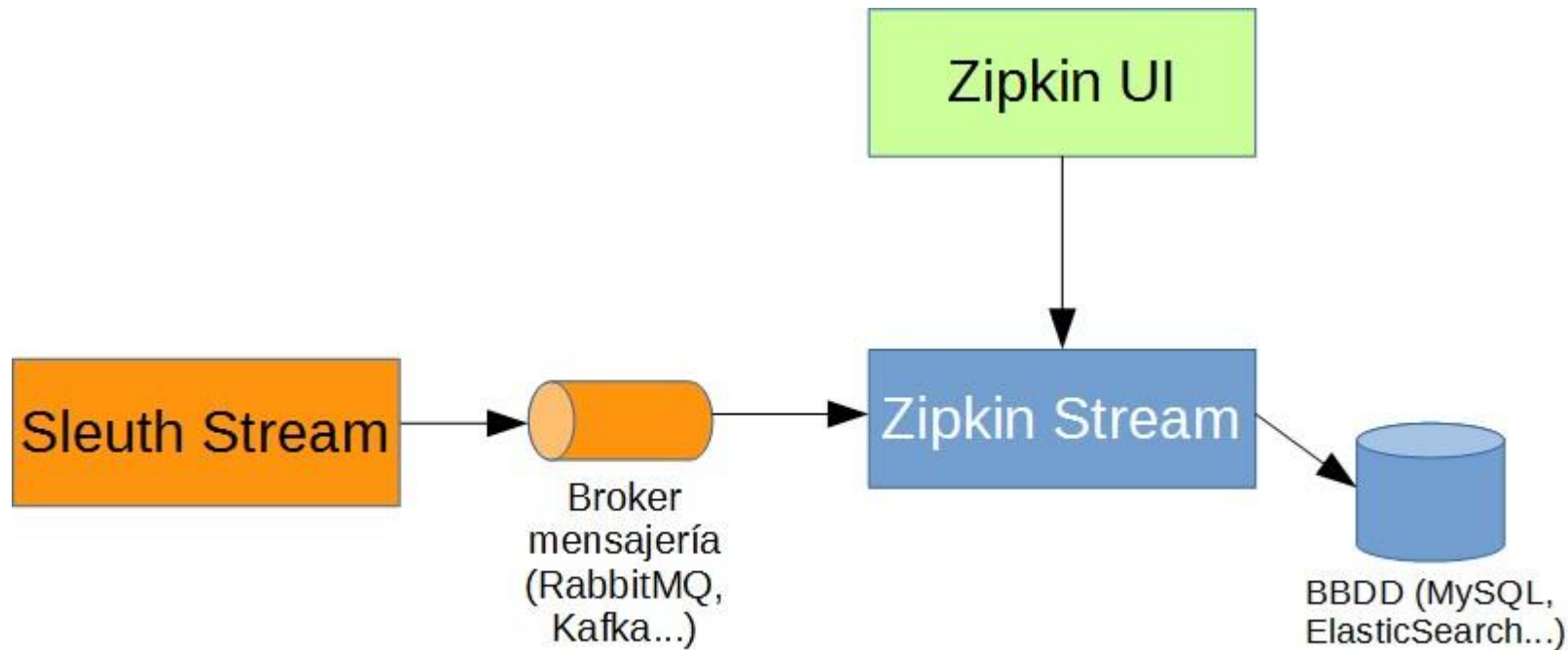
Métricas y HealthChecks



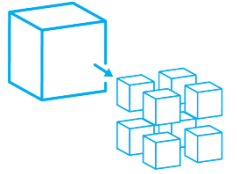
Trazas distribuidas: Arquitectura de la Trazabilidad



- La siguiente arquitectura de implementación se puede aplicar en Spring Cloud



Logs de aplicaciones y agregación de logs: Spring Cloud Sleuth



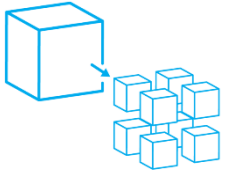
- Sleuth es una librería que permite el trazado distribuido.
- Mecanismo automático de identificación de peticiones.
- TraceID y SpanID

```
2016-11-15 15:12:25.977 INFO [sleuth-client,5501a4bacd8faaa8,44d49402e8b56e98,false] 9544 --- [nio-8999-exec-4] c.p.microservices.SleuthApplication : Handling home
```

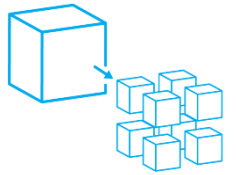
```
⌵ 🔍 "httpRequest.getHeaderNames()"= NamesEnumerator (id=142)
  ▶ headers= MimeHeaders (id=126)
  ▶ next= "accept" (id=144)
    ▶ pos= 1
    ▶ size= 9

=== MimeHeaders ===
accept = text/plain, application/json, application/*+json, */*
x-b3-traceid = 5501a4bacd8faaa8
x-b3-spanid = 1a424e54400f4782
x-b3-sampled = 0
x-span-name = http:/getName
x-b3-parentspanid = 44d49402e8b56e98
user-agent = Java/1.8.0_73
host = localhost:8089
connection = keep-alive
```

Sleuth Terminología

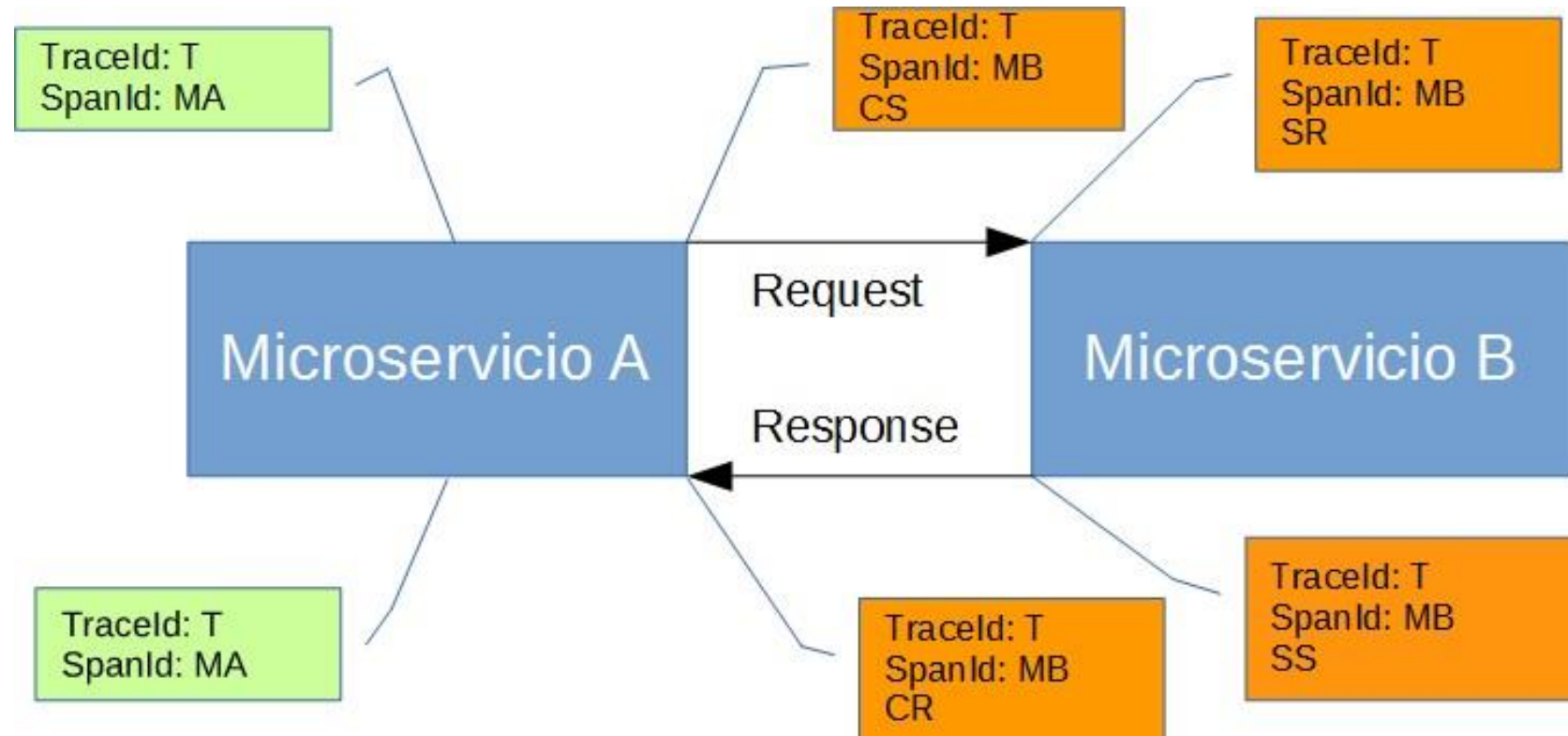


- ***Trace***: Un conjunto de spans que forman una estructura de árbol de llamadas, forma la traza de la petición.
- ***Span***: Es la unidad básica de trabajo.
 - Ejemplo una llamada a un servicio.
 - Se identifican con un id de span y un id de trace a la que pertenece dicho span.
 - Tienen inicio y fin, y con ello se consigue **trackear** el tiempo de respuesta entre peticiones.

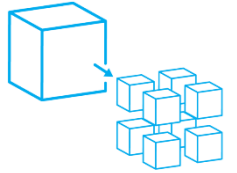


Sleuth Trace y Span

- Comportamiento de una petición HttpRequest



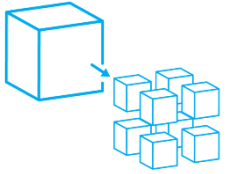
... continua



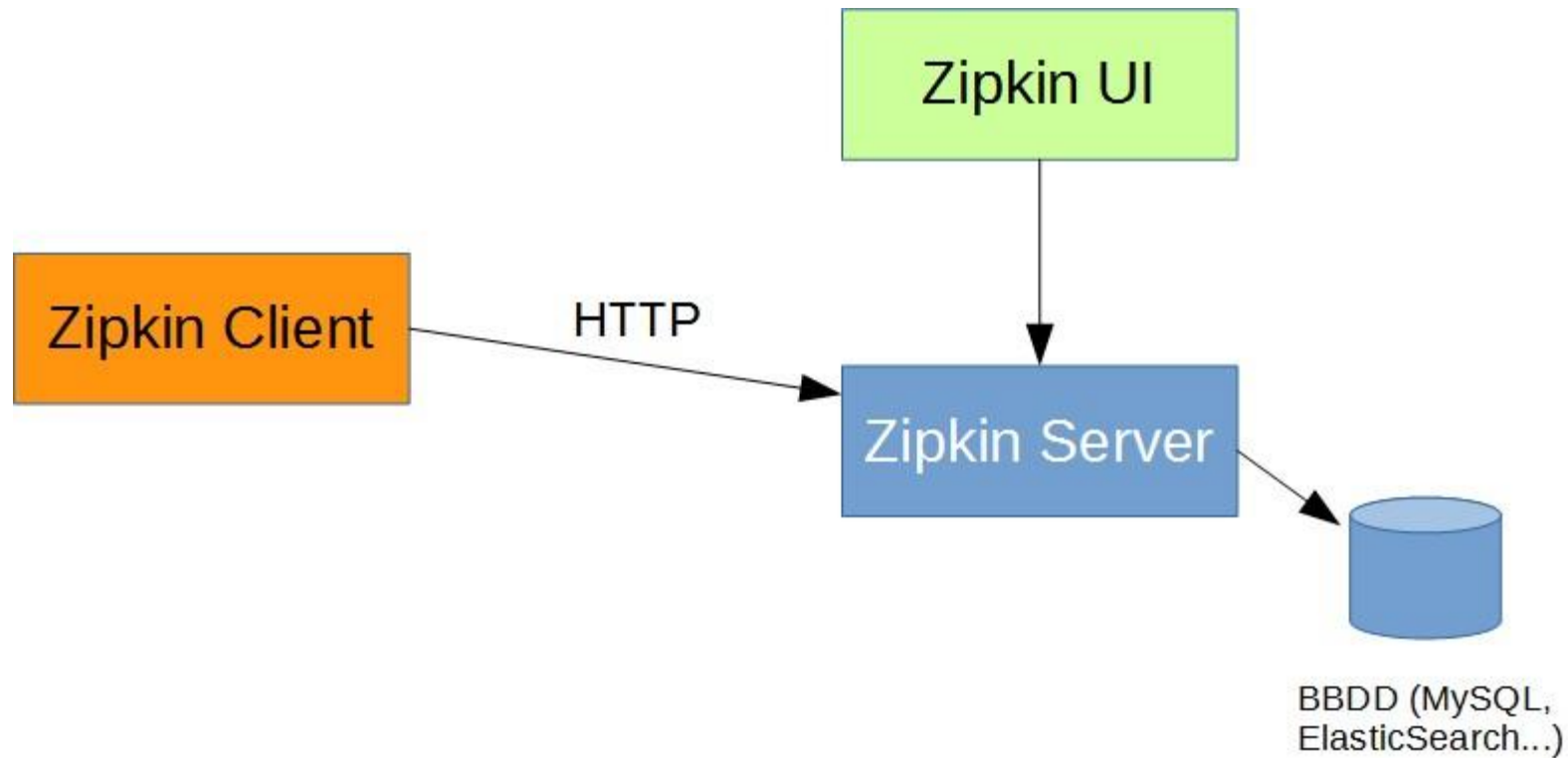
- **Annotation:** Se usa para grabar un evento en el tiempo. Las anotaciones más importantes que usa internamente son:
 - ***cs (Client Sent)***: El cliente ha hecho una petición. Inicio del span.
 - ***sr (Server Received)***: El servidor ha recibido la petición y empieza su procesado.
 $\text{timestamp}_{sr} - \text{timestamp}_{cs} = \text{latencia de red}$.
 - ***ss (Server Sent)***: Envío a cliente desde el servidor de la respuesta. $\text{timestamp}_{ss} - \text{timestamp}_{sr} = \text{tiempo de procesamiento de petición en servidor}$.
 - ***cr (Client Received)***: Fin del span. El cliente ha recibido correctamente la respuesta del servidor.* $\text{timestamp}_{cs} - \text{timestamp}_{cr} = \text{tiempo total de la petición*}$.

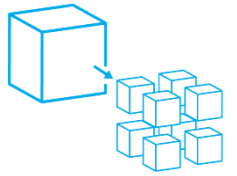
Tag: Par clave/valor que identifica cierta información en el span. No contiene timestamps, simplemente identifica

Zipkin



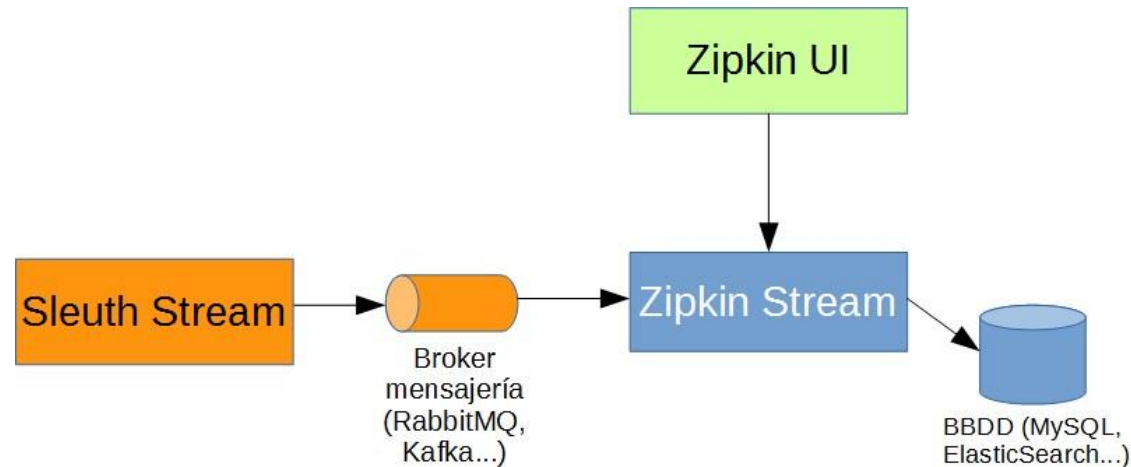
- Zipkin permite el almacenamiento, monitorización y visualización mediante un UI de las trazas de las peticiones.

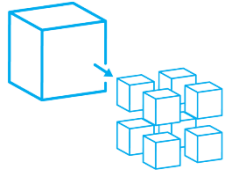




Zipkin Implementaciones

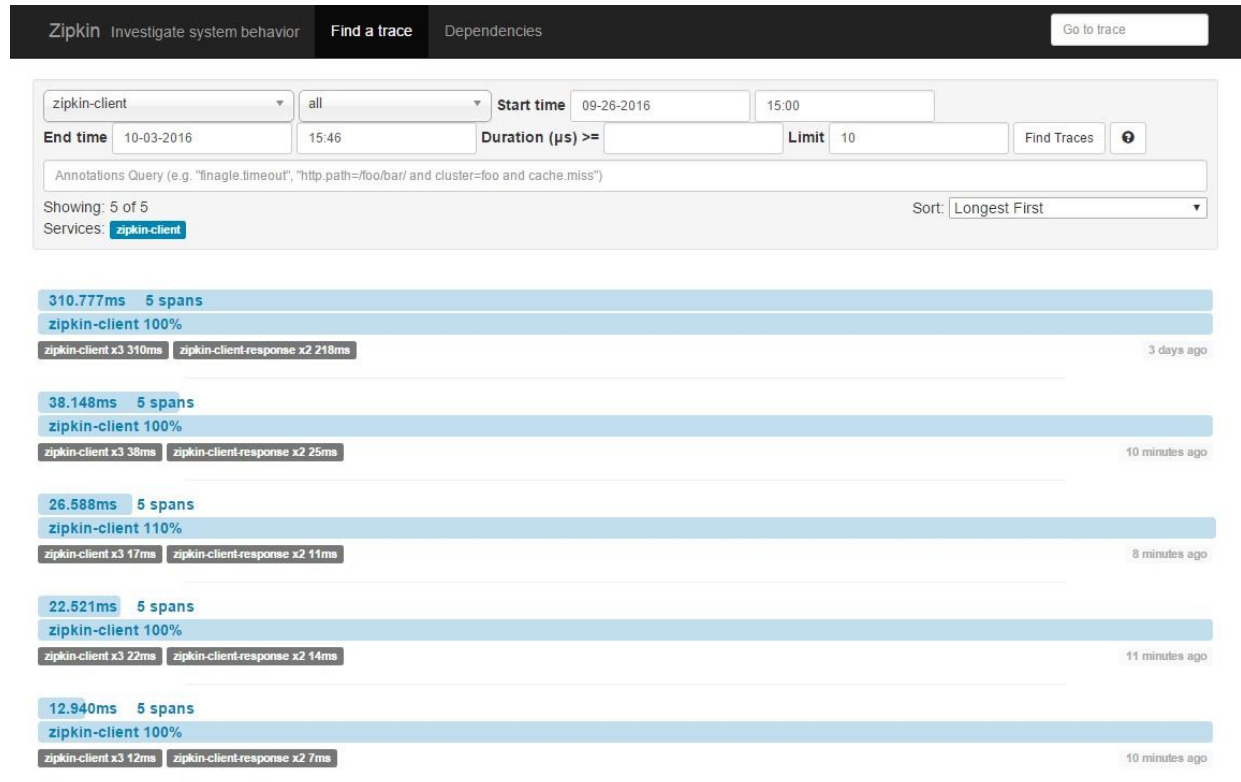
- Mediante Cabeceras Http.
 - Headers
- Usando Brokers de Mensajería.
 - RabbitMQ, Kafka



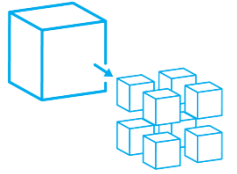


Monitoreo usando Dashboards: Zipkin UI

- Zipkin provee de una interfaz gráfica que agiliza las consultas a la BD del Zipkin Server y muestra la información recolectada.
- Zipkin UI por defecto usa el puerto: 9411



Obtención de Zipkin



- Java
- Docker Image
- Proyecto Maven del Código Fuente

The screenshot shows a web browser window with the address bar displaying `zipkin.io/pages/quickstart.html`. The page features a dark sidebar on the left with the Zipkin logo and navigation links: Home, Quickstart, Architecture, Tracers and Instrumentation, Server extensions and choices, Zipkin Community, Data Model, and Instrumenting a library. The main content area is titled "Docker" and explains that the Docker Zipkin project can build Docker images, provide scripts, and a `docker-compose.yml` file for launching pre-built images. It states that the quickest start is to run the latest image directly, followed by a code block showing the command: `docker run -d -p 9411:9411 openzipkin/zipkin`. Below this, the "Java" section explains that if Java 8 or higher is installed, the quickest way to get started is to fetch the latest release as a self-contained executable jar, followed by a code block showing the commands: `curl -sSL https://zipkin.io/quickstart.sh | sh` and `java -jar zipkin.jar`. The "Running from Source" section is partially visible at the bottom.

Preguntas

- Alguna pergunta?

