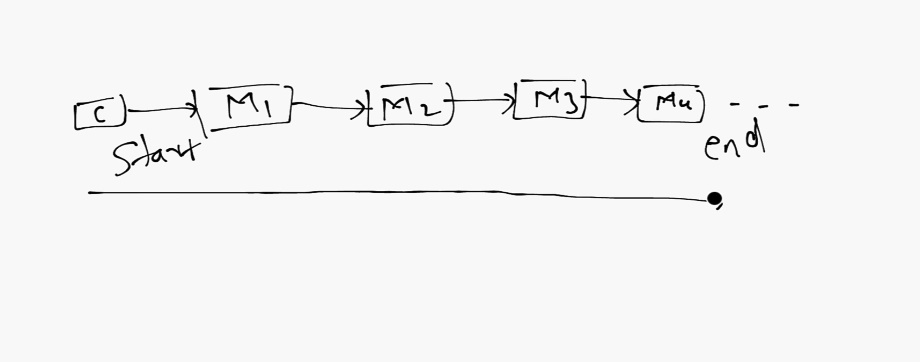
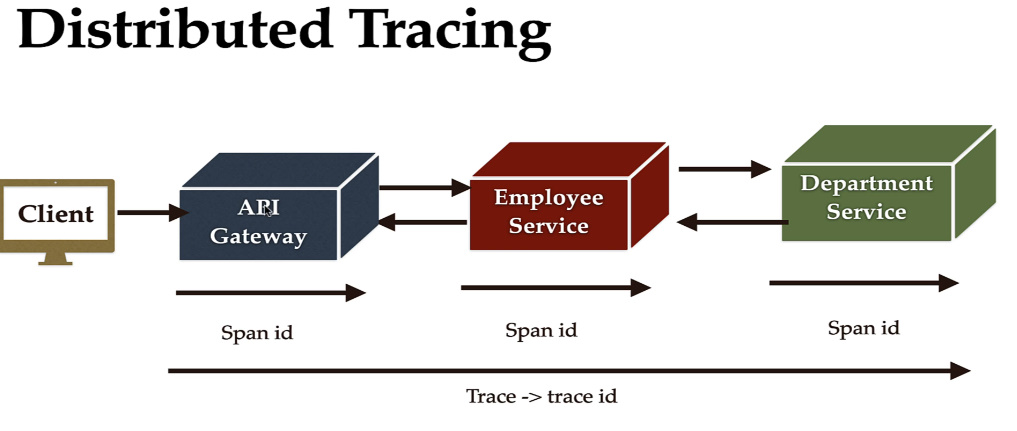
* O implementam prin intermediul la:

- Micrometer – pentru distributed tracing

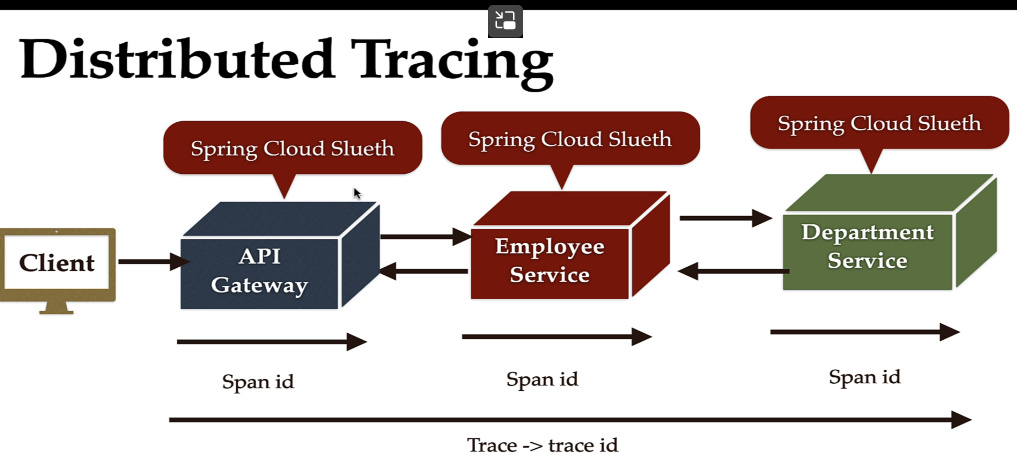
- Zipkin – pentru interfatag grafica

* O provocare in microservices este de a da debug la issues
* Cand un user da un request, el va trece prin multe microservicii, si e foarte greu de a face debug. Nu prea vom putea afla unde anume e problema daca apare vreo eroare cu un request
* Anume, distributed tracing va oferi posibilitatea de a urmari aceasta cale la un request, sa vedem prin ce microservices trece
* 

Deci, vom putea vedea tot acest chain la un request si chiar vom putea vedea care microservice a luat prea mult timp sa fie executat



* **Spring Cloud Sleuth e depreciat si nu mai merge in Spring Boot 3! Alternativa lui e Micrometer**
* **Micrometer** colecteaza date despre timpul de executie la un request per service, pe unde se duce request, errors etc.





****



**app name –** numele la service in care apare logul, luat din properties

**trace id –** este un id unic oferit unui request. Asa, putem vedea prin care microservicii gasim acest id in loguri si asa aflam la sigur ca request a trecut pe acolo

**span id –** este un id unic oferit la request per service. Deci, in acelasi service acest request are acelasi id, in alte service va avea alte ids.

**Implementare**

1. **Adaugam dependenta Micrometer in gateway si microservicii**

<dependency>

<groupId>io.micrometer</groupId>

<artifactId>micrometer-tracing-bridge-brave</artifactId>

</dependency>

<dependency>

<groupId>io.github.openfeign</groupId>

<artifactId>feign-micrometer</artifactId>

</dependency>

**micrometer-tracing-bridge-brave** dependency will allow us to trace our spring boot application.Deci, vede requestul pe unde se duce si gata

**feign-micrometer** dependency was added because I am using feign in my microservices to call other APIs. This dependency will configure micrometer to work with feign.

1. **Adaugam in properties la gateway si microservicii:**

logging.level.org.springframework.web.servlet.DispatcherServlet=DEBUG  
logging.pattern.level=%5p [${spring.application.name:},%X{traceId:-},%X{spanId:-}]  
management.tracing.sampling.probability=0.1

**logging.level.org.springframework.web.servlet.DispatcherServlet** – e necesar de a-l pune, caci requests facute nu sunt aratate in consola in mod default, dar asa vor fi deja

**logging.pattern.level** – aici stabilim cum va fi afisat mesajul langa log

**management.tracing.sampling.probability** – arata care sunt sansele ca anumite date sa fie colectate de micrometer. 0.1 inseamna ca doar 10% din date vor fi colectate, deci 90% vor fi evitate. Asta e bun pentru performance, dar va evita multe requesturi. Default e 0.1

Acum, fiecare request si log va arata asa:

2023-08-05T17:26:02.172+03:00 DEBUG [EMPLOYEE-SERVICE,64ce5bf9a2e823a533b3a4f065720d89,33b3a4f065720d89] 5772 --- [nio-8081-exec-2] o.s.web.servlet.DispatcherServlet : Completed 200 OK

Observam ca e scris [service,trace id, span id]

**Zipkin**

* Cu Zipkin putem vedea trace in interfata grafica
* Zipkin poate fi rulat in docker sau ca jar

[Quickstart · OpenZipkin](https://zipkin.io/pages/quickstart.html)

1. **Instalam zipkin jar si il punem in project la noi**
2. **Rulam acest jar cu**

java -jar nume.jar din consola la Inellij

si acum ne putem conecta cu url in el din consola

1. **Adaugam dependentele in gateway si microservicii:**

<dependency>

<groupId>io.zipkin.reporter2</groupId>

<artifactId>zipkin-reporter-brave</artifactId>

</dependency>

**Aduaugam si asta, caci ea sa trimita datele catre zipkin**

<dependency>  
 <groupId>io.micrometer</groupId>  
 <artifactId>micrometer-observation</artifactId>  
</dependency>

**micrometer-observation –** aduna date despre errros, pe unde se duce request, timpul de executie per microservice etc.

**Atentie! Actuator e necesar**

1. **Punem sansele ca datele sa fie colectate mai mari, 1.0, adica 100%**

Daca vom lasa

management.tracing.sampling.probability=0.1

doar 10% din toate requeturile si datele despre ele vor fi colectate de micrometer si trimise la zipkin. Deci, 90% din requesturi nu vor fi inregistrate, si asta e bun pentru performance. Daca punem

management.tracing.sampling.probability=1.0

toate requesturile mereu vor fi insregistrate!

1. **Acum legam gateway si microservices de zipkin**

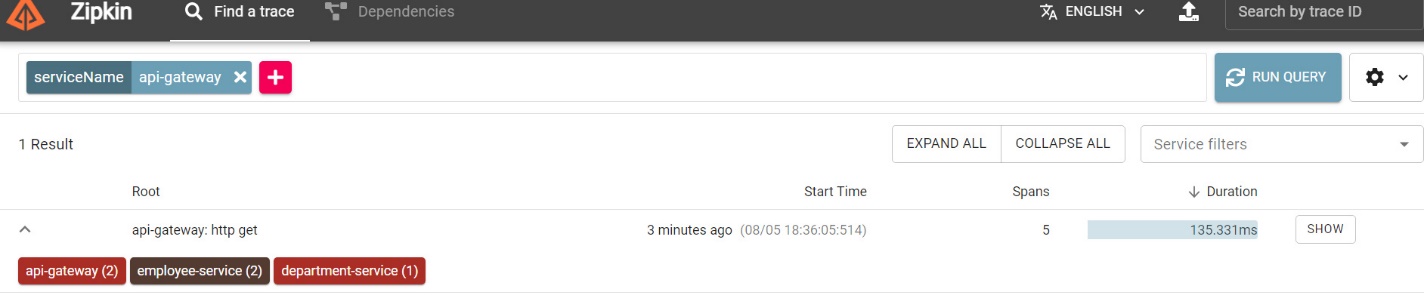
management.zipkin.tracing.endpoint=http://localhost:9411/api/v2/spans

Si gata. Acum, cand intram in el, putem face urmatoarele lucruri:

- apasam pe + de la find a trace, si acolo putem gasi dupa nume, max time de executie etc.

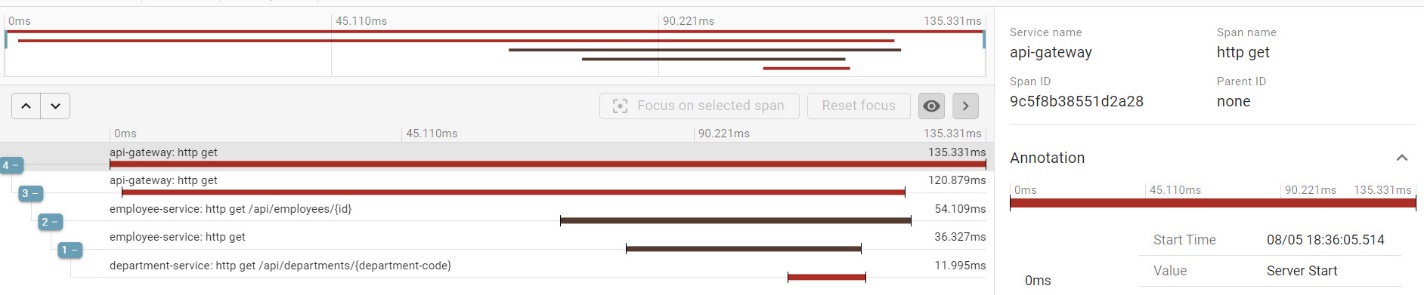
- luam dupa serviceName si ne apare lista de services disponibile, si deobicei luam dupa gateway si apasam Run Query

- Acum vedem lista de requests care corespund la api gateway facute si info despre ele



vedem si cate spans sunt in fiecare service, adica de cate ori s-a trecut prin ele ca sa se execute request

- daca apasam pe SHOW, vedem info mai detaliat si timpul de executie pe service



- putem apasa acum pe dependecies, de pe pagina de baza ca sa vedem care services de care sunt dependente

