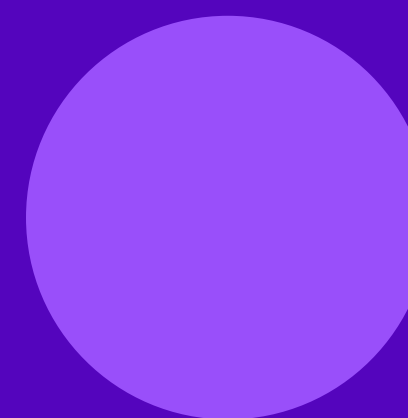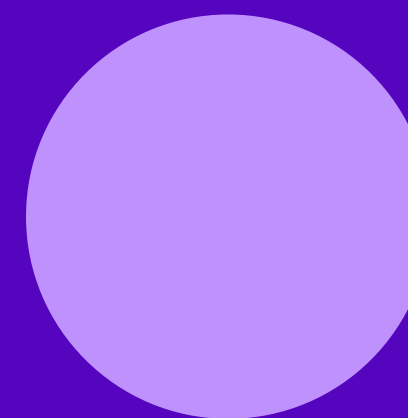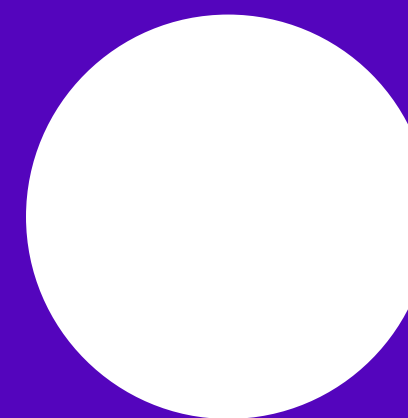# Spring Cloud

What is **Spring cloud?**

- Is a framework for building robust cloud applications.

- Common patterns in distributed systems.
    - Configuration management, service discovery, circuit breakers, intelligent routing,

      micro-proxy, control bus.

- The easiest way to get started is visit start.spring.io.
    - Select your Spring Boot version and the Spring Cloud projects you want to use.
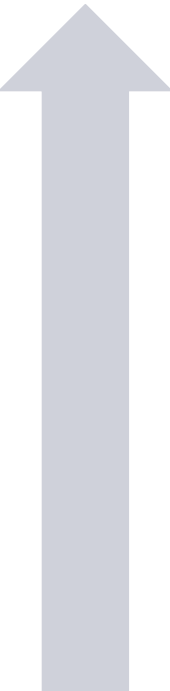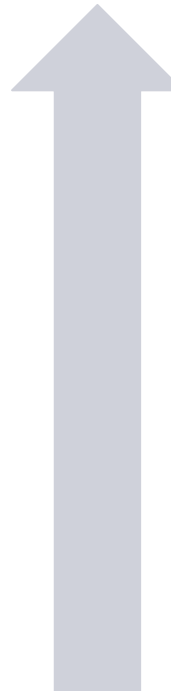
- Cloud Foundry, AWS, etc

Agenda

- Configuration Server

- Service Discovery

- Declarative Rest Client

- Client Load Balance

- Api Gateway

- Distributed Trace

Cloud Configuration Server

talkdesk®

Bootiful Configuration

- Spring boot improves configuration things considerably:

- Automatically load properties from a well-known hierarchy:
  - PropertySource order designed to allow sensible overriding of values, <u>properties are considered in the following order</u>:

    1. **Command line arguments (--server.port=9000)**
    2. **Java system properties (System.getProperties)**
    3. **OS environment variables**
    4. **@PropertySource annotation on your @Configuration classes,**
    5. **Application Properties outside of you package jar**
    6. **Application Properties inside your jar.**
    7. **Default properties (specified using SpringApplication.setDefaultProperties)**

Bootiful Configuration

- Profiles:
  - ```spring.application.profiles.active=aws,prod,etc```
  - ```@org.springframework.context.annotation.Profile({"aws", "prod", "etc"})```
- Annotations:
  - @Configuration
  - @ConfigurationProperties
  - @Value
  - @ConditionalOnX
  - @PropertySource, etc...

- Actuator
  - ```curl -X GET 'http://127.0.0.1:8081/actuator/configprops' | jq .```

Missing answers

- **Server Downtown**

  - *Changes to an application's configuration, as configured, would require a restart.*

- **There is no traceability**

  - How do we determine what changes were introduced into production and, if necessary, rollback those changes?

- **Configuration is decentralized**

  - Is not immediately understandable where to go to change what.

- **Encryption and Decryption**

  - There is no out-of-the-box support for encryption and decryption for security.

  - We may have a common set of properties for each of your microservices. For example, database properties (username, password)..

Solution: <u>Cloud Config Server</u>

- **Rest API**, which our client clients will connect to draw their configurations

- It manages a **version-controlled repository**

- It sits in **between our clients and the repository** of configuration

  - Is in an enviable position to interpose security on the communication from clients to the service and on the communication from the service to the version-controlled repository of configuration.

⚠ *Technologies spring cloud config server are important, but they will add cost to your operational overhead. Ideally, this competency should be managed by the platform, and automated.* ***If you are using Cloud Foundry, there is a Config Server service, in the service catalog****.*

:talkdesk®

Implementation key Points

- **<u>Configuration Server</u>**

  – Github integration

  – Local system

  – Cloud Bus

- **<u>Downstream service</u>**

  – Bootstrap

  – @RefreshScope

  – Actuator

<< show example >>

Service Register &
Service Discovery

Service Register

🤔 We want a logical mapping from a service's **ID** to the **Hosts** and **Ports** on which that service is available.

🚀 A Service Register is a good fit.

- Spring Cloud **DiscoverClient** abstraction:
  - Make it easy for clients to work with different types of service registries:
    - Apache Zookeeper
    - Hashicorp
    - Consul
    - Netflix Eureka

Eureka Server

- Using [spring initializr](#), create Eureka Server
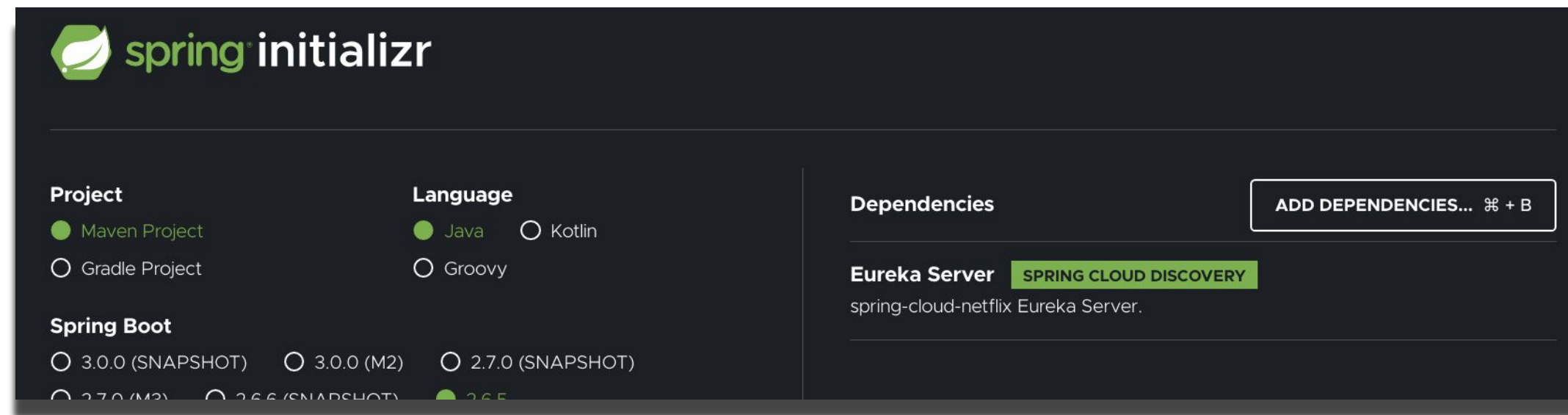
- Apply the annotation @EnableEurekaServer

- Configuration

Talking to the Registry

- Dependencies:
  - Netflix-eureka-client
  - Actuator

- Configuration
  - eureka.client.register-with-eureka=true
  - eureka.client.fetch-registry=true
  - eureka.client.service-url.defaultZone=http://127.0.0.1:8761/eureka
  - eureka.client.healthcheck.enabled=true

- Enable DiscoveryClient abstraction
  - @EnableDiscoveryClient

- Links
  - curl -L -X GET http://localhost:8761/eureka/apps
  - http://localhost:8761

<< show example >>

Declarative Rest Client

:talkdesk®

OpenFeign

- Is a REST Client to make rest call from our spring application.
  - Microservice
  - Third-part service

- Clean alternative to **RestTemplate** and **WebClient** with some advantage:
  - Providing a declarative way.
  - Make our life very easy
  - Cloud CircuitBreaker Support
  - OAuth2 Support

<< *show example* >>

Client Load balance

:talkdesk®

Client Load Balancer

`org.springframework.cloud:spring-cloud-starter-loadbalancer`

Client Load Balancer - Configuration

```java
@Bean
@LoadBalanced
public RestTemplate restTemplate(RestTemplateBuilder restTemplateBuilder) {
    return restTemplateBuilder.build();
}
```

```java
@Bean
@LoadBalanced
public WebClient.Builder webClient() {
    return WebClient.builder();
}
```

Client Load Balancer - Configuration

```java
@LoadBalancerClients({
        @LoadBalancerClient(value = "routingms")
})
public class CargoRoutingOpenFeignLoadBalancerConfig {

    @Bean
    @LoadBalanced
    public Feign.Builder feignBuilder() { return Feign.builder(); }

}
```

```java
@FeignClient(
        name = "routingms",
        contextId="routingmsOpenfeign",
        decode404 = true)
public interface CargoRoutingOpenfeign {

    @GetMapping(path = "/routing", produces = MediaType.APPLICATION_JSON_VALUE)
    TransitPath findOptimalRoute();
}
```

<< *show example* >>

API Gateway

:talkdesk®

API Gateway

📚 [Spring cloud api gateway documentation](#)

- Top-Level context for request into a system

  - Geolocation
  - Cookie propagation
  - Token decryption
  - Authentication
  - CORS (Cross-Origin Resource Sharing)

- Ideal place to address routing from the client to downstream services

API Gateway

- **Netflix Zuul - @Deprecated**

    *The spring team has stopped supporting **Netflix Zuul**, this project has been replaced in Spring-cloud framework by the new **Spring-api-gateway** implementation.*

- **Spring Cloud Api-gateway**

    - **Communicate with Eureka**
    - **Redirect the HTTP requests to the right services**
    - **Load balance**
        - **Load balance can be made in the api-gateway instead of in the downstream microservices.**
    - **Rate-Lime**
    - **Authorization**

<< show example >>

Distributed Trace

:talkdesk®

Distributed Tracing

- Is a method of observing requests as they propagate through distributed cloud environments

- Distributed tracing follows an interaction by tagging it with a unique identifier.

  - This identifier stays with the transaction as it interacts with microservices, containers, and infrastructure.

  - The unique identifier offers real-time visibility into user experience, from the top of the stack to the application layer and the infrastructure beneath.

Problem

- When we work with microservice by default any microservice store each log files. So we could have a requisition passing in multiple microservices, but we are not able to trace the flow between the microservices.

- How to trace the flow of one requisition?
  - Consulting every log file of each service a each replicas.

- How to identify high service latence?

Benefits

- helps teams get to the bottom of application performance issues faster.

- Upon discovering an issue, the organization can rapidly identify the root cause and address it.

- Improves developer productivity and output:
  - by making it easy to understand the behavior of distributed systems. Distributed tracing tools are easy to setup and they will help you and your team drastically reduce time spent debugging and troubleshooting issues with your systems.

- works across multiple applications, programming languages, and transports.

- Most importantly, distributed tracing facilitates excellent cross-team communication and cooperation. It eliminates costly data silos that could otherwise hinder developers' ability to quickly locate and fix sources of error.

Spring Cloud Sleuth

- Auto Configuration

- Integrates with Open Zipkin Brave

- Trace Request with unique identifier

- Export the tracing information to an external system to visualize latency.

    Spring Cloud Sleuth supports OpenZipkin compatible systems directly.

Terminology

📚 [Official documentation](#)

- **Trace-Id**
  - Transaction-id
  - Propagated for each downstream microservice

- **Span**
  - Unique over each microservice request
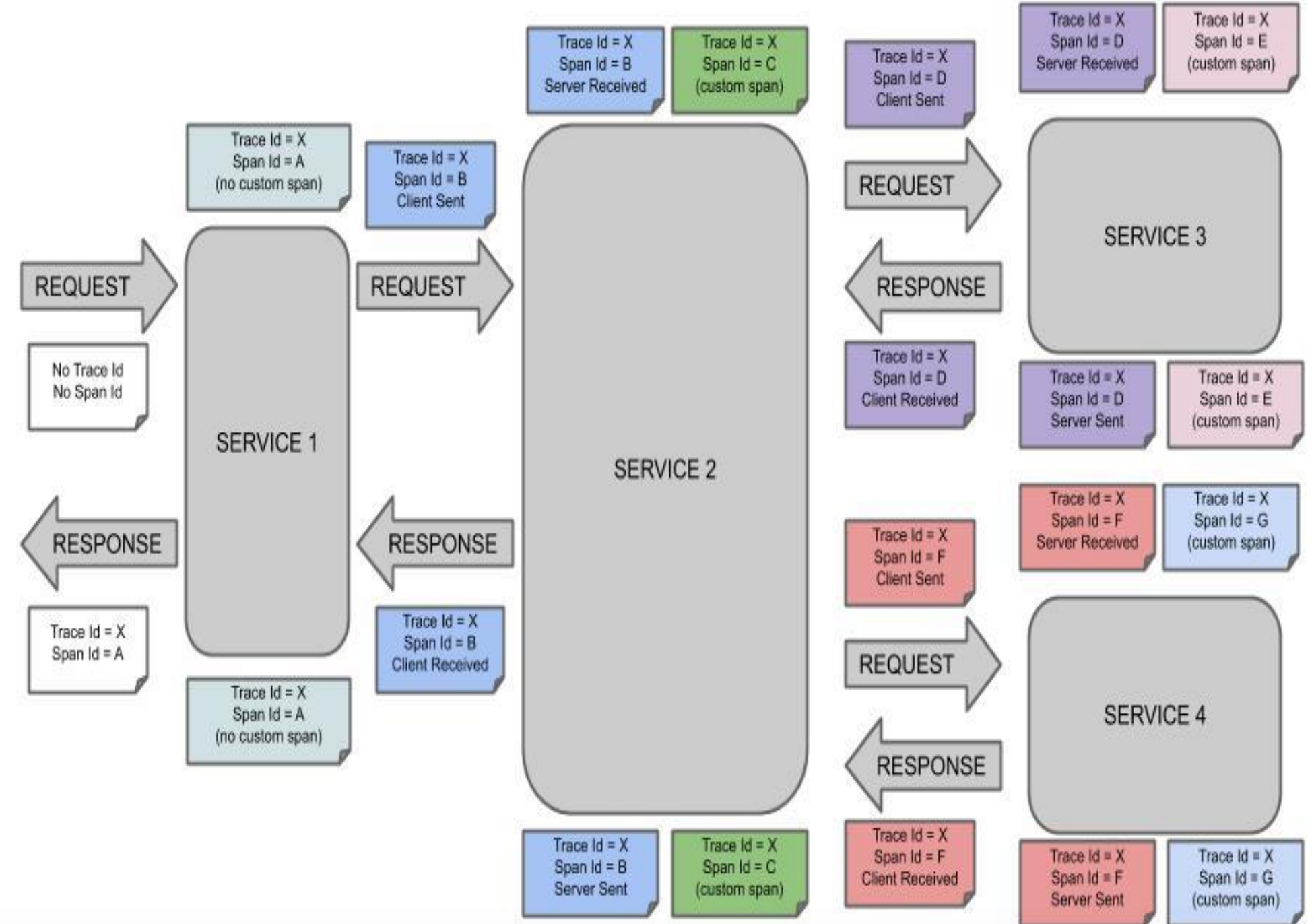
DEBUG [**bookingms**,**223d69674c7f0fea**,**5b6b0d0702387430**] 36284 --- [nio-8081-exec-1] POST "/booking/ef622f7a/route", parameters={}

DEBUG [**routingms**,**223d69674c7f0fea**,**3d33319c40aec234**] 36288 --- [nio-8082-exec-1] GET "/routing?origin=ABC&destination=DEF&deadline=2022-04-01T21%3A23%3A43.598697Z"

DEBUG [**bookingms**,**223d69674c7f0fea**,**5b6b0d0702387430**] 36284 --- [nio-8081-exec-1] Completed 200 OK

Distributed Tracing - zipkin
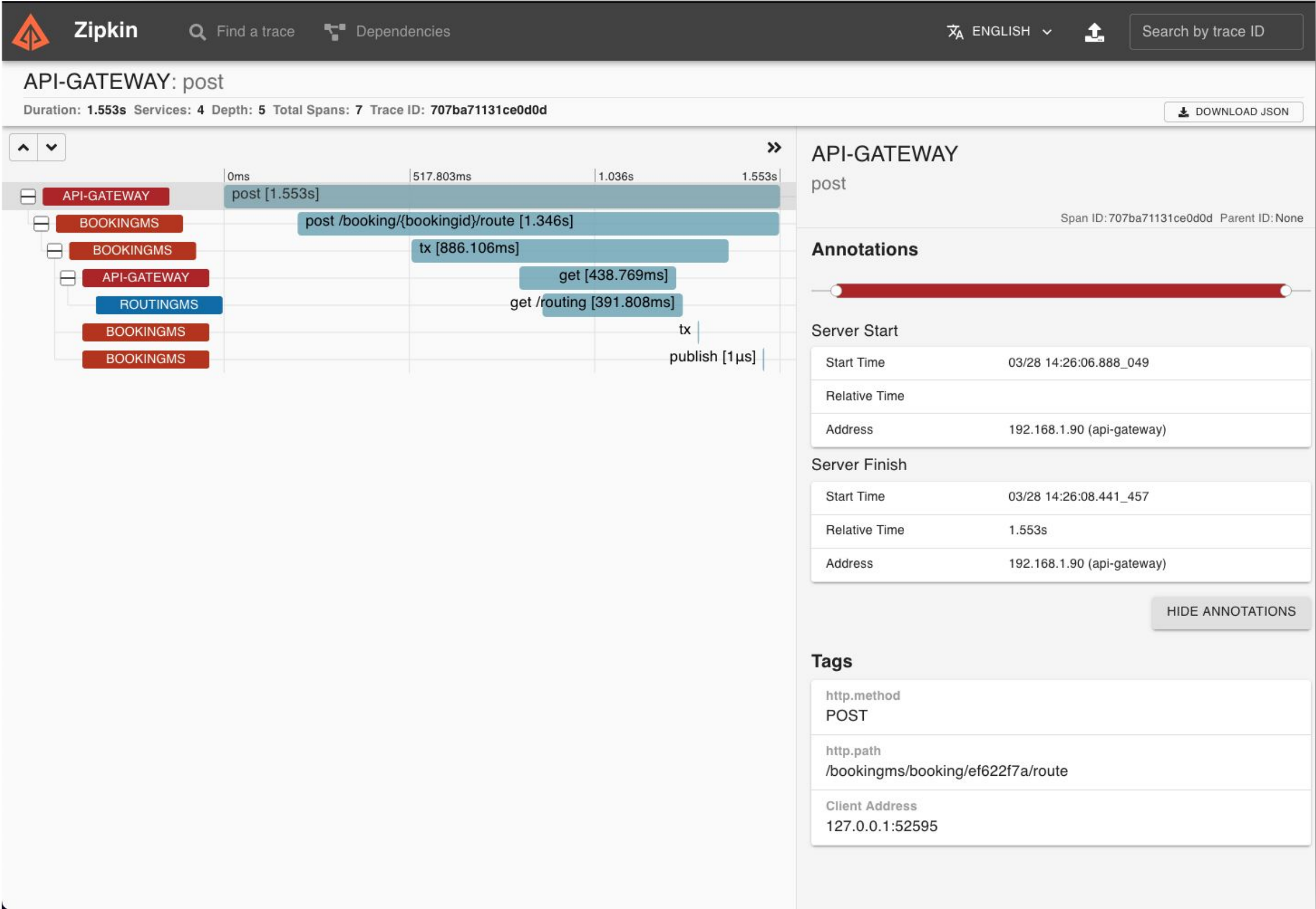
Distributed Tracing - centralized logger

<< show example >>

:talkdesk®

Thank you!