Zuul & Sleuth

What are they? Why use them?

Reverse proxy and tracing

Pivotal

Topics

- **Zuul**
- Sleuth

What is Zuul?

- Zuul is a JVM based router and server side load balancer by Netflix

- Typically used as an “edge-service”
  - The visible “front” of your microservices application
  - Routes requests to the right microservice

https://github.com/Netflix/zuul

Using Zuul

- Spring Boot style starter
  - `spring-cloud-starter-netflix-zuul`
- Enhance a Spring Boot application
  - `@EnableZuulProxy`

Zuul-Based Edge Service

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

@EnableZuulProxy
@SpringBootApplication
public class GatewayApplication {

  public static void main(String[] args) {
    SpringApplication.run(GatewayApplication.class, args);
  }

}
```

Pivotal™

Configuring Routing

- Set `zuul.routes.xxx.url` properties in `application.properties/yml`

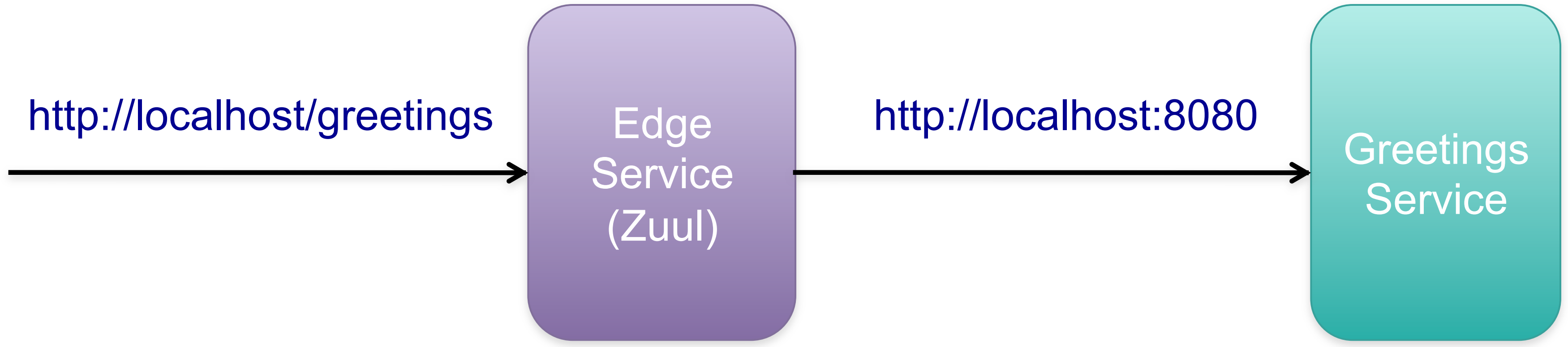
- For example, if the *greetings* service is on *localhost* on port 8080

```
zuul.routes.greetings.url=http://localhost:8080

ribbon.eureka.enabled=false

# The port this Zuul server is running on
server.port=80
```

Pivotal™

Edge Service Working

- So now http://localhost/greetings
  maps to http://localhost:8080



More Configuration

- Set mapped path explicitly
  – To map **`http://localhost/welcome`**

```
zuul.routes.greetings.url=http://localhost:8080
zuul.routes.greetings.path=welcome
```

- To integrate with Eureka
  – Maps to service registered as "*greetings-service*"

```
zuul.routes.greetings.serviceId=greetings-service
zuul.routes.greetings.path=greetings
```

Pivotal™

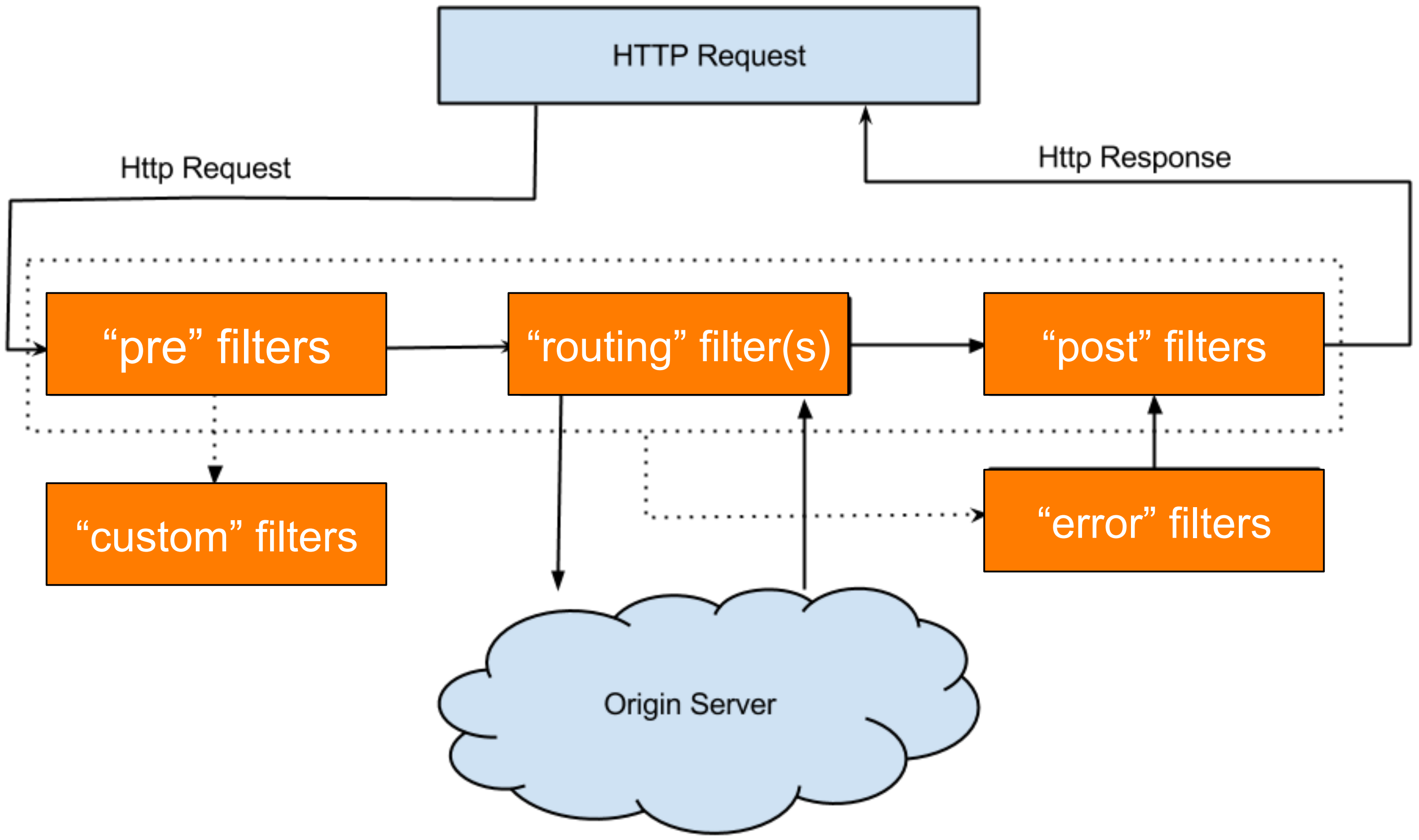
Filters

- Filters can be used to
  - Preprocess requests before routing
  - Postprocess requests
  - To implement custom routing
  - Handle errors
- Implement **`ZuulFilter`**
  - Declare as a Spring bean
  - Access request details using static method

```
RequestContext ctx = RequestContext.getCurrentContext();
```

Pivotal™

Zuul Routing and Filtering



Example “pre” Filter – 1

```
@Component
public class LoggingFilter extends ZuulFilter {

  private static Logger log =
          LoggerFactory.getLogger(LoggingFilter.class);

  @Override
  public String filterType() {
    return "pre";
  }

  @Override
  public int filterOrder() {
    return 1;
  }
                          // Continued on next slide ...
```

Pivotal™

Example “pre” Filter – 2

```
  @Override
  public boolean shouldFilter() {
    return true;   // Always used
  }

  @Override
  public Object run() {
    RequestContext ctx = RequestContext.getCurrentContext();
    HttpServletRequest request = ctx.getRequest();

    log.info(String.format("%s request to %s",
             request.getMethod(),
             request.getRequestURL().toString()));

    return null;
  }
}
```

What requests is this filter used for?

Setting request headers is a common use-case

Log each request

Topics

- Zuul
- **Sleuth**

Distributed Tracing

- **Visualize**
  - See full flow of a request through multiple microservices

- **Zipkin**
  – A distributed tracing system from Netflix OSS

- **Spring Cloud Sleuth**
  – Builds on Zipkin, and other projects
  – Distributed tracing for Spring applications

Pivotal

Spring Cloud Sleuth

- Adds trace and “span” ids to the Slf4J MDC
  - Extract all logs from a given trace/span in a log aggregator
- Provides abstraction over common distributed tracing data models
  - Traces, spans (forming a DAG), annotations, key-value annotations
  - Loosely based on HTrace, but Zipkin (Dapper) compatible
- Instruments common ingress and egress points from Spring applications
  - Servlet filter, REST template, scheduled actions, message channels, zuul filters, feign client ...

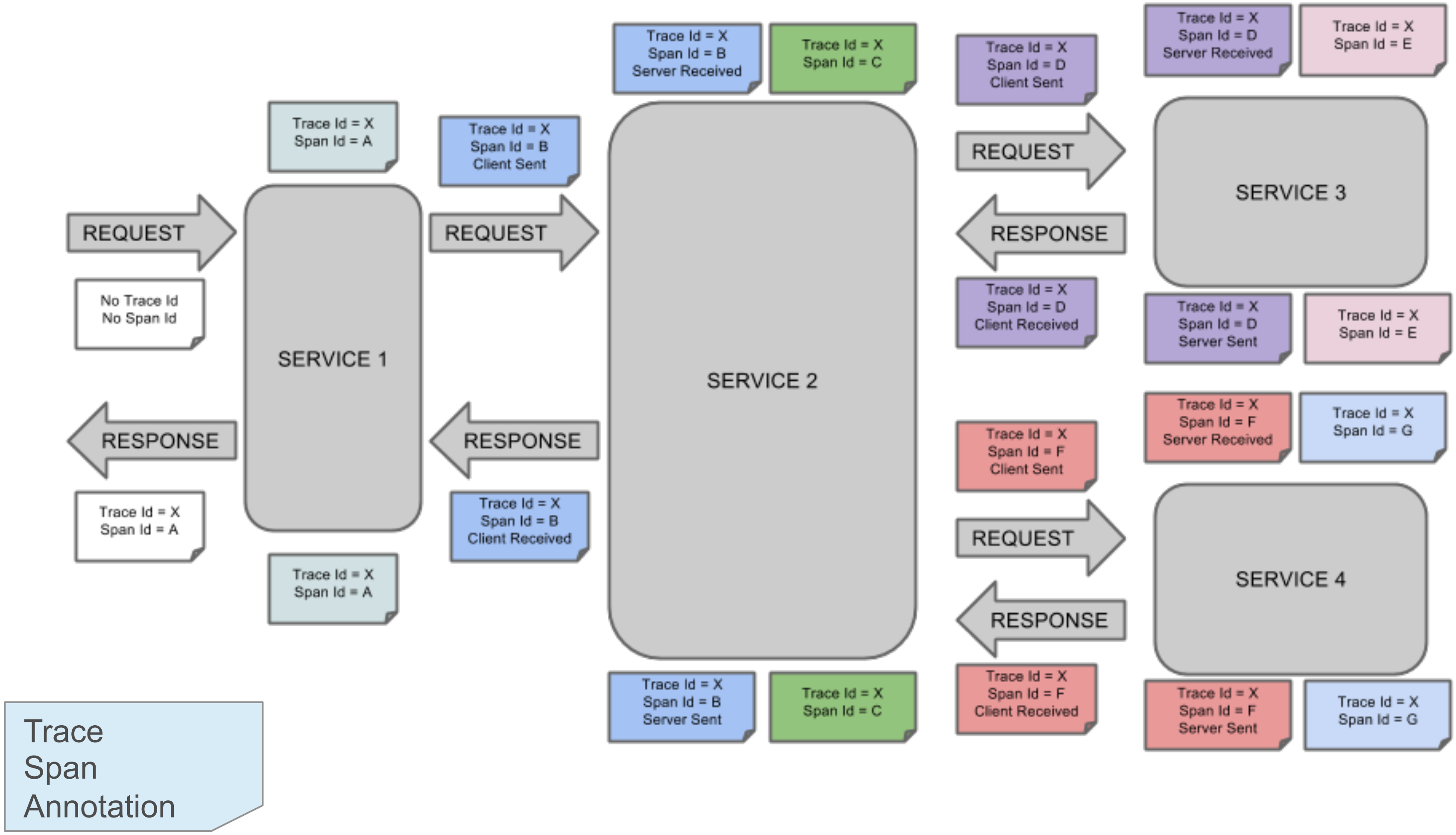
Spring Cloud Sleuth and Zipkin

- If `spring-cloud-sleuth-zipkin` is available
  - Application will generate and collect Zipkin-compatible traces via HTTP
  - By default it sends them to a Zipkin collector service
    - On localhost port 9411
  - Configure the location of the service using `spring.zipkin.baseUrl`

Terminology

- **Span**
  - A unit of work that is logged
  - Typically work done by a service or a communication with a service
- **Trace**
  - A complete interaction with the system
  - A *tree* of spans
- **Annotation**
  - Indicates what the span refers to

Example: A Single Trace → Trace X



To Use

- Add starter dependency to your services
  - Just sleuth (log correlation)
    - `spring-cloud-starter-sleuth`
  - Zipkin integration
    - `spring-cloud-starter-zipkin`
- Set this property
  - `spring.sleuth.sampler.percentage=1.0`
- Run your applications and exercise them
  - They will automatically start creating coordinated logs

Run Zipkin Server

- As Java application

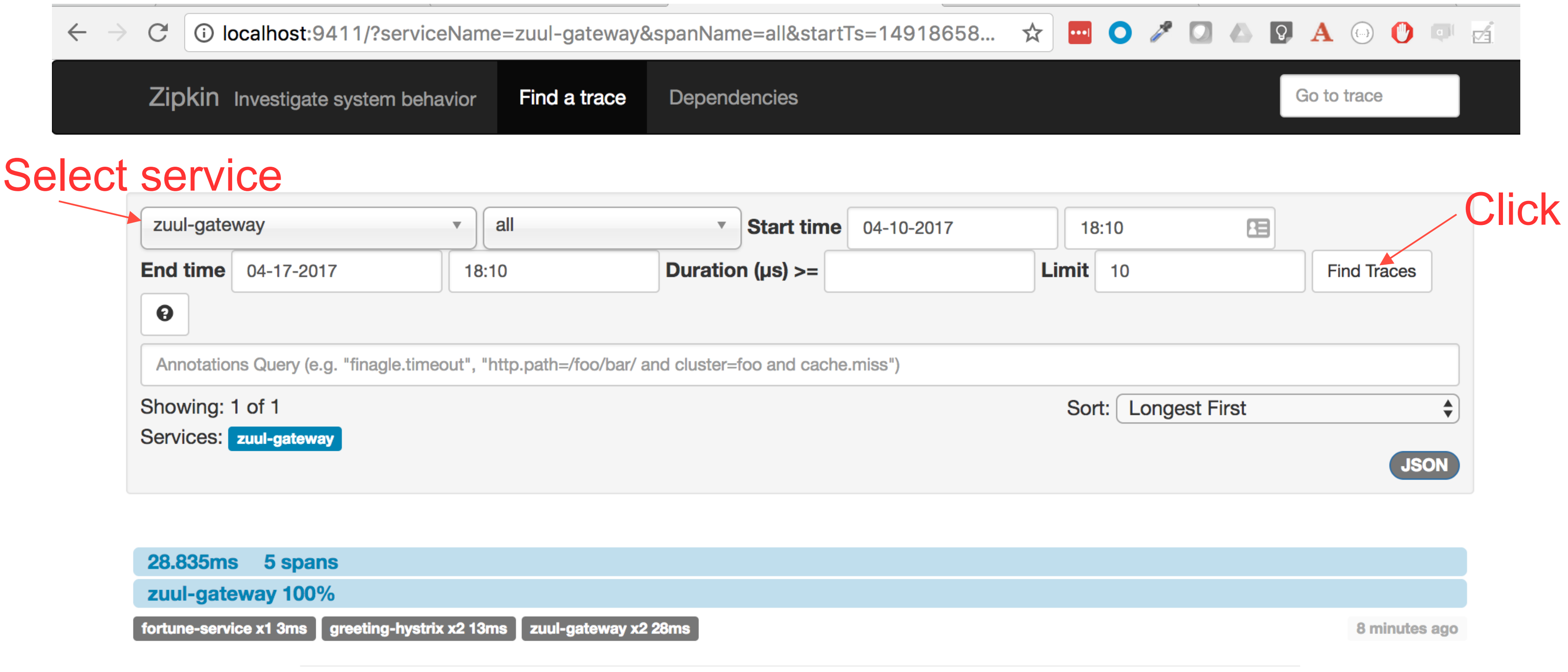
```
wget -O zipkin.jar 'https://search.maven.org/remote_content?
          g=io.zipkin.java&a=zipkin-server&v=LATEST&c=exec'
java -jar zipkin.jar
```

- Using Docker

```
docker run -d -p 9411:9411 openzipkin/zipkin
```

Pivotal™

View Traces in Zipkin: *localhost:9411*



localhost:9411/?serviceName=zuul-gateway&spanName=all&startTs=14918658...

zuul-gateway | all | **Start time** 04-10-2017 18:10

End time 04-17-2017 18:10 **Duration (μs) >=** **Limit** 10 Find Traces

Annotations Query (e.g. "finagle.timeout", "http.path=/foo/bar/ and cluster=foo and cache.miss")

Showing: 1 of 1

Services: zuul-gateway

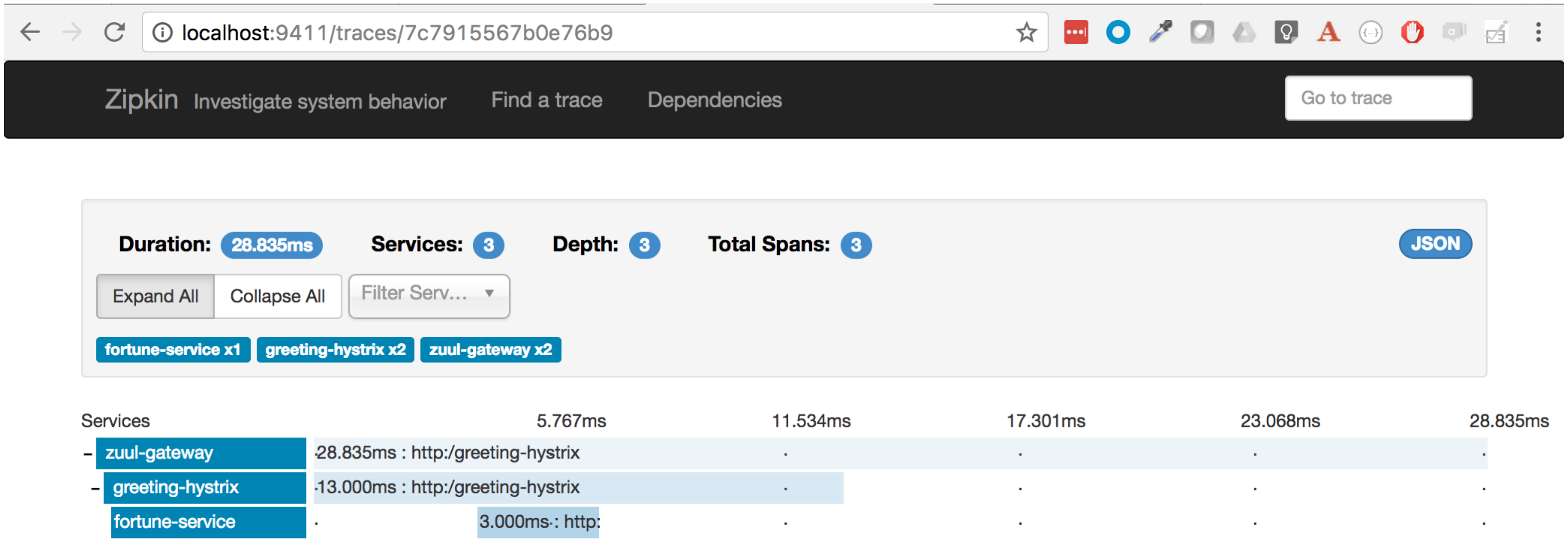
Sort: Longest First

28.835ms 5 spans

zuul-gateway 100%

fortune-service x1 3ms | greeting-hystrix x2 13ms | zuul-gateway x2 28ms | 8 minutes ago

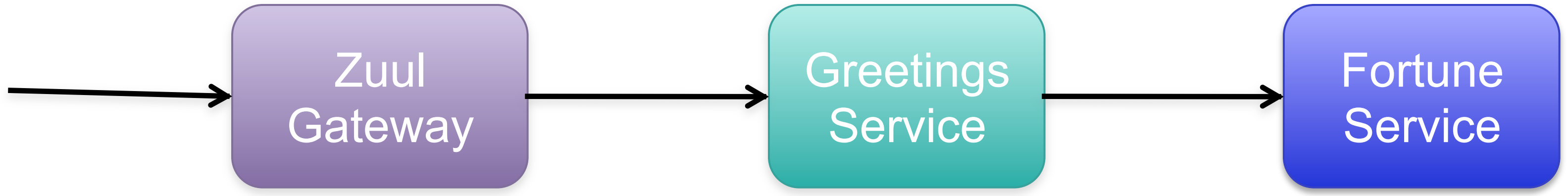
Drill Down



Duration: 28.835ms Services: 3 Depth: 3 Total Spans: 3

fortune-service x1 greeting-hystrix x2 zuul-gateway x2

```
Services          5.767ms       11.534ms      17.301ms      23.068ms      28.835ms
- zuul-gateway      28.835ms : http:/greeting-hystrix
 - greeting-hystrix 13.000ms : http:/greeting-hystrix
   fortune-service              3.000ms : http:
```



Define A Custom Span – 1

```
@Controller
public class GreetingController {

  private final FortuneService fortuneService;
  private final Tracer tracer;

  public GreetingController(FortuneService fortuneService,
                            Tracer tracer) {
    this.fortuneService = fortuneService;
    this.tracer = tracer;
  }

  @RequestMapping("/")
  String getGreeting(Model model) {
    model.addAttribute("msg", "Greetings!!!");
    model.addAttribute("fortune", fetchFortune());
    return "greeting";   // resolves to the greeting.ftl template
  }
                 // Continued on next slide ...
```

Inject a Tracer

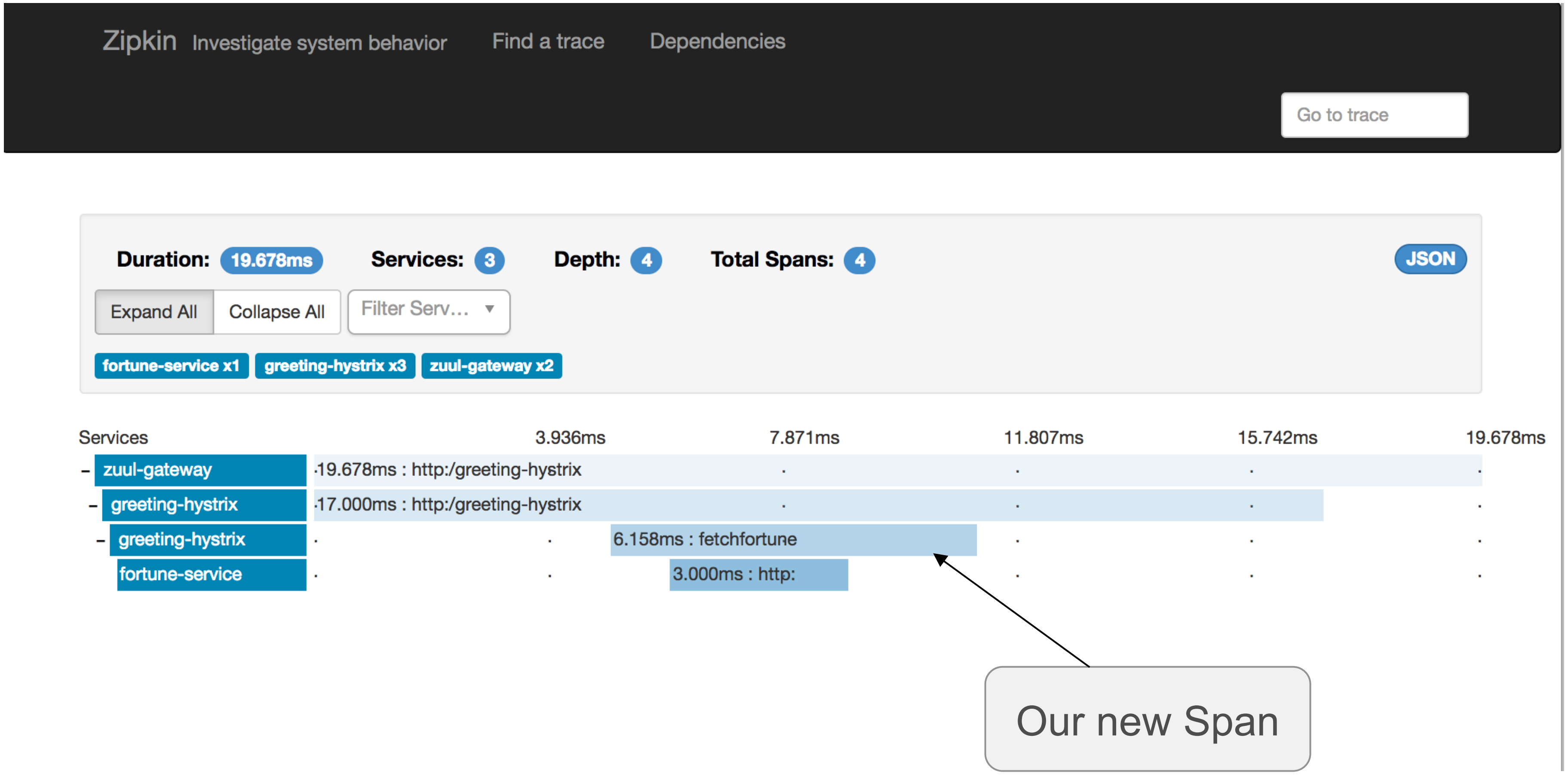
Define A Custom Span – 2

- Wrap your code in a *Span*
  - Good candidate for using AOP
  - Here done manually

```
  private String fetchFortune() {
    Span span = tracer.createSpan("fetchFortune");
    try {
      // Do the work of the method
      return fortuneService.getFortune();
    } finally {
      tracer.close(span);
    }
  }
```

Pivotal

View in Zipkin Server



Zipkin Investigate system behavior Find a trace Dependencies

Go to trace

Duration: 19.678ms **Services:** 3 **Depth:** 4 **Total Spans:** 4 JSON

Expand All | Collapse All | Filter Serv...

fortune-service x1 | greeting-hystrix x3 | zuul-gateway x2

```
Services            3.936ms        7.871ms        11.807ms        15.742ms        19.678ms
- zuul-gateway        19.678ms : http:/greeting-hystrix
 - greeting-hystrix   17.000ms : http:/greeting-hystrix
  - greeting-hystrix                 6.158ms : fetchfortune
     fortune-service                   3.000ms : http:
```

Topics Covered

- Zuul
- Sleuth