Mapping Services Using Intelligent Routing

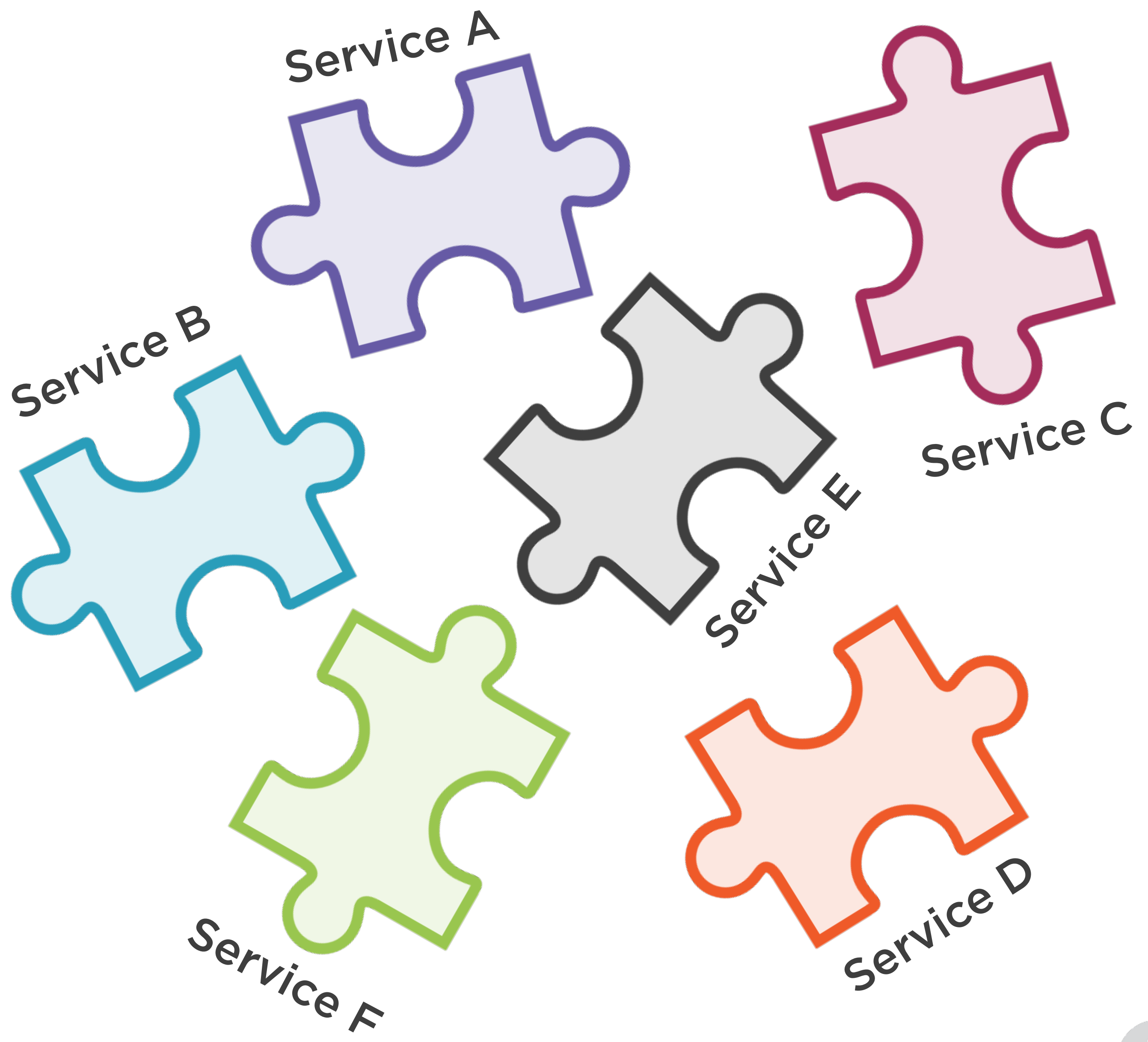
Outline

Routing in cloud native apps

Netflix Zuul

- Proxy server
- Setting up routes
- Setting up filters

Individually Deployable Services



Challenges with Individual Services

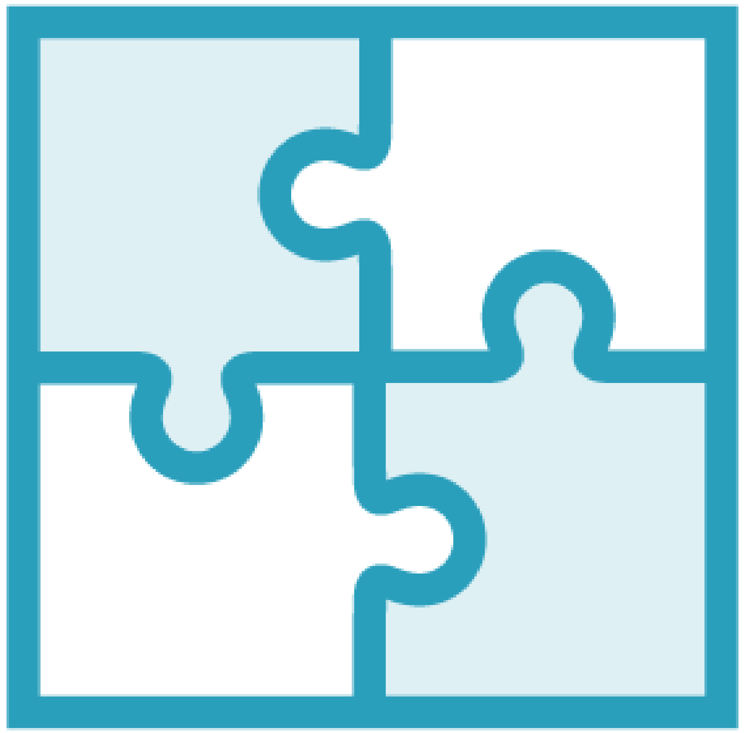
Different ports

Different addresses

Different APIs & paths

As a client (such as mobile or web), interacting with ***each individual*** service would be a disaster

The Solution: Intelligent Routing



Appears as a whole but still individual pieces

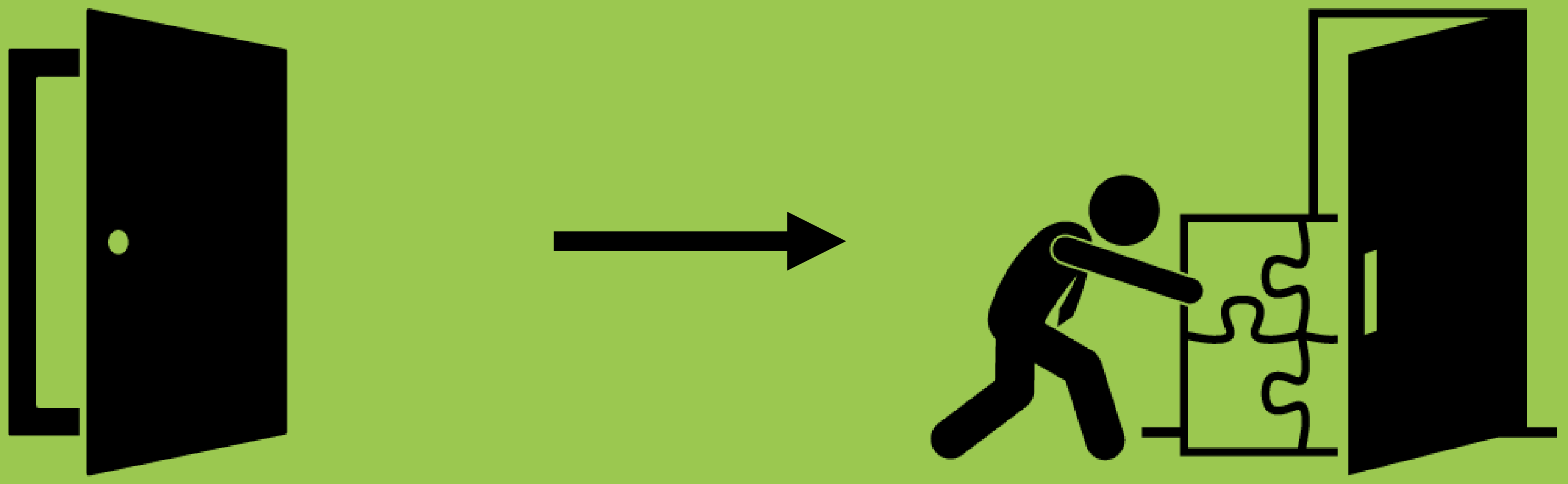
Intelligent Routing via a Gateway Service

API Gateway

… a single entry point for all clients …

- Chris Richardson

Gateway Service: The Front Door



The front door, edge service, the gateway to services

Services are placed behind the edge service

A Gateway Service Provides

Dynamic Routing & Delivery

Security & Filtering

Auditing & Logging

Request Enhancement

Load Balancing

Different APIs for different clients

Intelligent Routing with Spring Cloud & Netflix Zuul

Netflix Zuul

Zuul is a gateway service that provides dynamic routing, monitoring, resiliency, security, and more.

- *Netflix Zuul Project Page*

Using Spring Cloud & Netflix Zuul

pom.xml

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Camden.SR2</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Using Spring Cloud & Netflix Zuul

pom.xml

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-zuul</artifactId>
</dependency>
```

Using Spring Cloud & Netflix Zuul Reverse Proxy

Application.java

```java
@SpringBootApplication
@EnableZuulProxy
public class Application {

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

Using Spring Cloud & Netflix Zuul with Service Discovery

application.properties

```
spring.application.name=gateway-service
eureka.client.defaultZone=http://localhost:8761/eureka
```

OR

application.yml

```
spring:
  application:
    name: gateway-service
eureka:
  client:
    defaultZone: http://localhost:8761/eureka
```

Using Spring Cloud & Netflix Zuul Without Service Discovery

application.properties

```
spring.application.name=gateway-service
ribbon.eureka.enabled=false
```

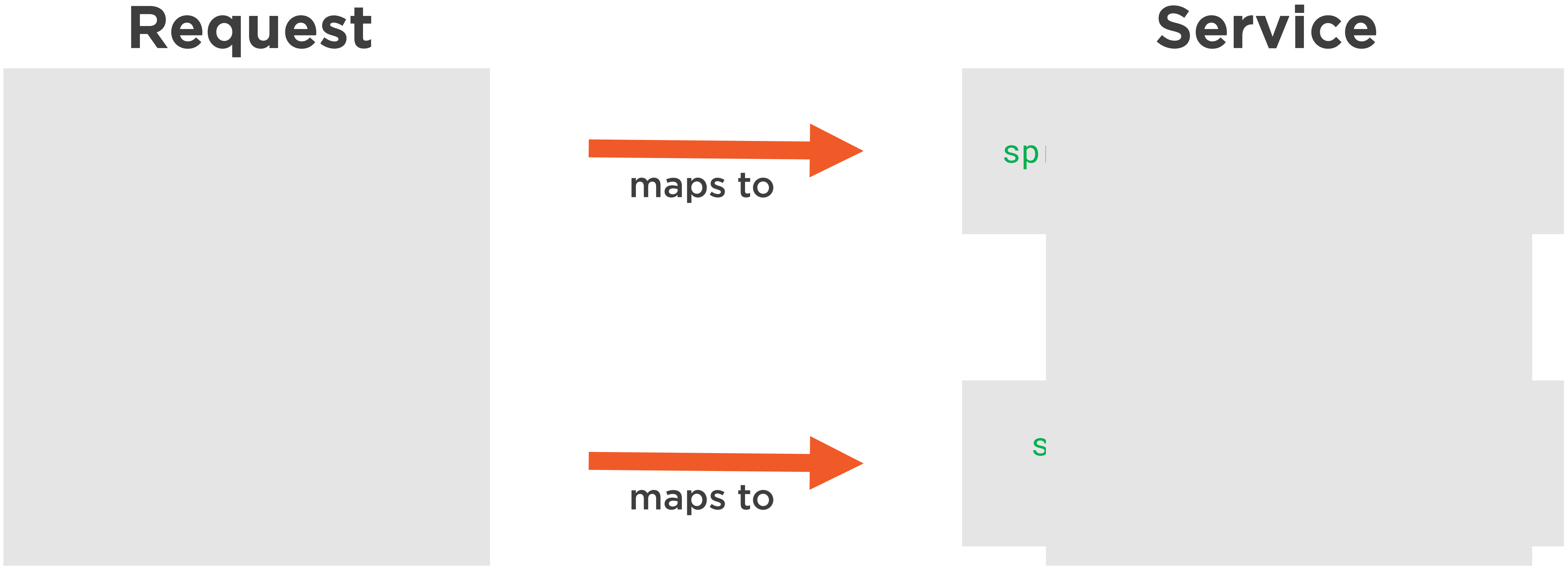
OR

application.yml

```
spring:
  application:
    name: gateway-service
ribbon:
  eureka:
    enabled: false
```

Configuring Routes with Spring Cloud & Netflix Zuul

Default Route to Service Resolution with Service Discovery



** **prefix is stripped by default. Use** `zuul.stripPrefix=false` **to disable***

*** **All services are added by default. Use** `zuul.ignoredServices=<pattern>` **to ignore services***

Netflix Zuul with Service Discovery: Precise Routing

application.properties

```
spring.application.name=gateway-service
zuul.routes.<route_name>.path=/somepath/**
zuul.routes.<route_name>.serviceId=some_service_id
zuul.ignored-services=some_service_id
```

OR

application.yml

```
spring:
  application:
    name: gateway-service
zuul:
  routes:
    <route_name>:
      path: /somepath/**
      serviceId: some_service_id
  ignored-services: some_service_id
```

Netflix Zuul Without Service Discovery: Precise Routing

application.properties

```
spring.application.name=gateway-service
zuul.routes.<route_name>.path=/somepath/**
zuul.routes.<route_name>.url=http://some_url_address/
```

OR

application.yml

```
spring:
  application:
    name: gateway-service
zuul:
  routes:
    <route_name>:
      path: /somepath/**
      url: http://some_url_address/
```

Creating Filters with Spring Cloud & Netflix Zuul

Filters allow you to intercept and control requests and responses.

Filter Types

`pre`	Before the request
`post`	After the request
`route`	Direct the request
`error`	Handle request errors

Creating a Filter: Extend & Implement ZuulFilter

MyFilter.java

```
public class MyFilter extends ZuulFilter {

    // implement methods
    ...

}
```

```java
@Override
public Object run() {

}


@Override
public boolean shouldFilter() {

}


@Override
public String filterType() {

}

@Override
public int filterOrder() {

}
```

- Filter logic goes here. Current implementation ignores return
- Whether or not the `run()` method should execute
- The type of filter: `pre`, `route`, `post`, `error`
- The order of execution with respect to other filters of the same type

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

// Get the servlet request
HttpServletRequest req = ctx.getRequest();

// Get the servlet response
HttpServletResponse res = ctx.getResponse();

// Set a variable
ctx.set("foobar", "PRE_FILTER_EXECUTED");

// Get a variable
String foobar = (String) ctx.get("foobar");
```

Sharing Between Filters: RequestContext

Holds request, response, state, and data information

Only available for the duration of the request

Creating a Filter: Define an @Bean Which Returns the Filter

MyConfig.java

```
@Configuration
public class MyConfig {

    @Bean
    public ZuulFilter myFilter() {
        return new MyFilter();
    }
}
```

Summary

The need for intelligent routing

Gateway service

Netflix Zuul

- `@EnableZuulProxy`
- Configuring routes
- Writing filters