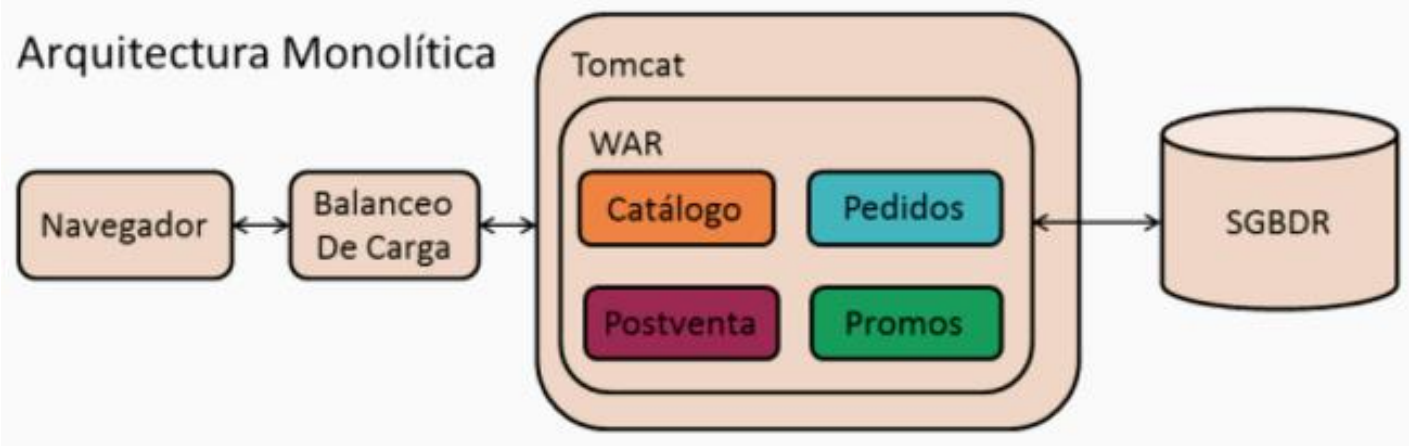
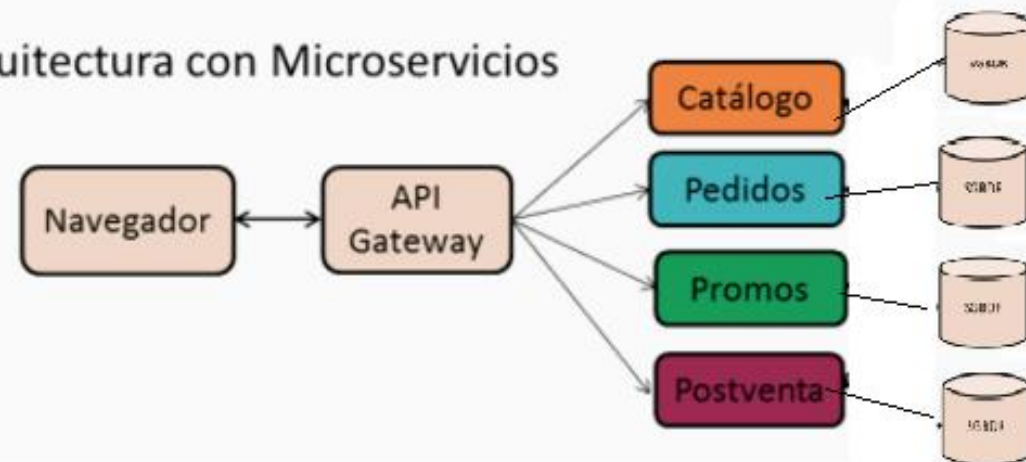


MICROSERVICIOS

Arquitectura Monolítica



Arquitectura con Microservicios

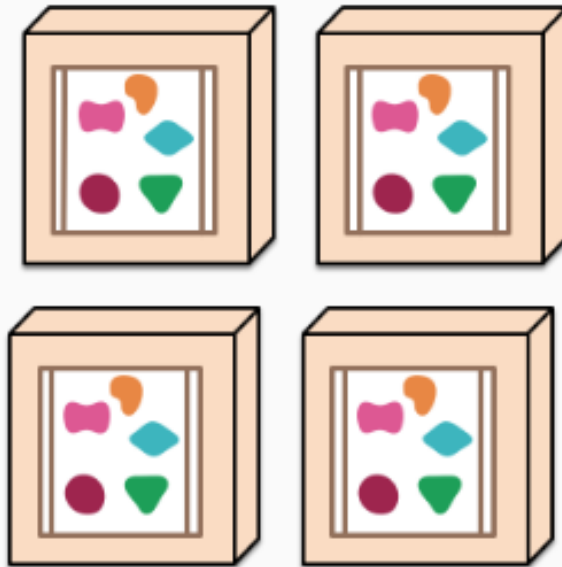


Una arquitectura de microservicios pone cada elemento de funcionalidad en un servicio separado y escala redistribuyendo estos servicios a través de los servidores, replicándolos según se vaya necesitando.

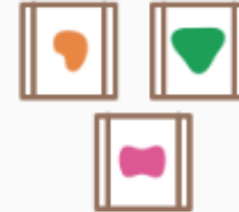
A monolithic application puts all its functionality into a single process...



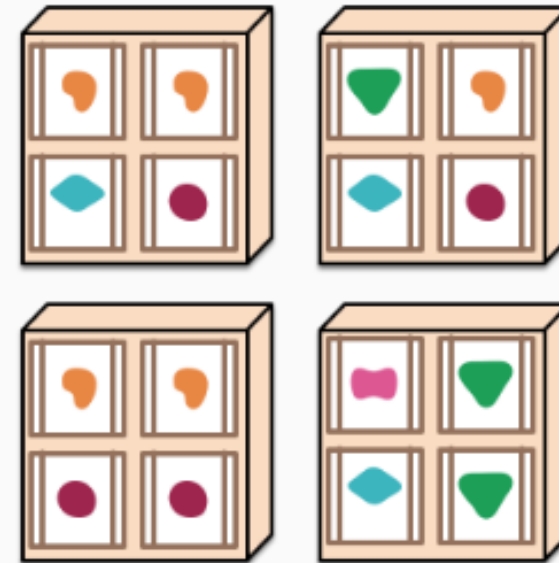
... and scales by replicating the monolith on multiple servers



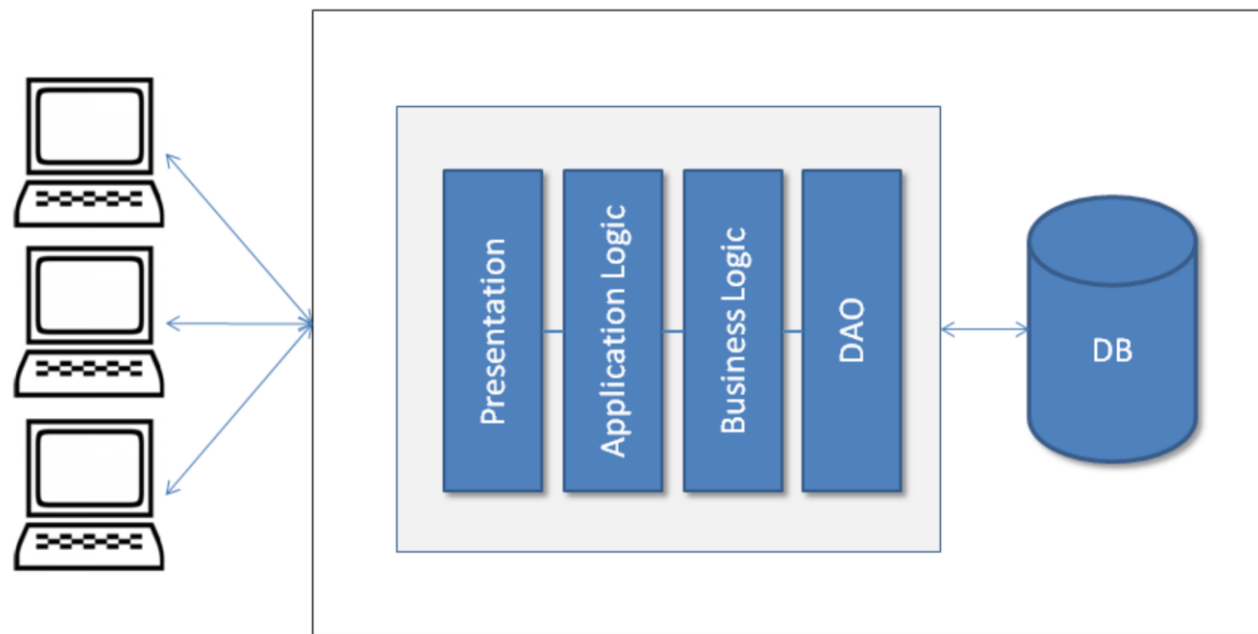
A microservices architecture puts each element of functionality into a separate service...



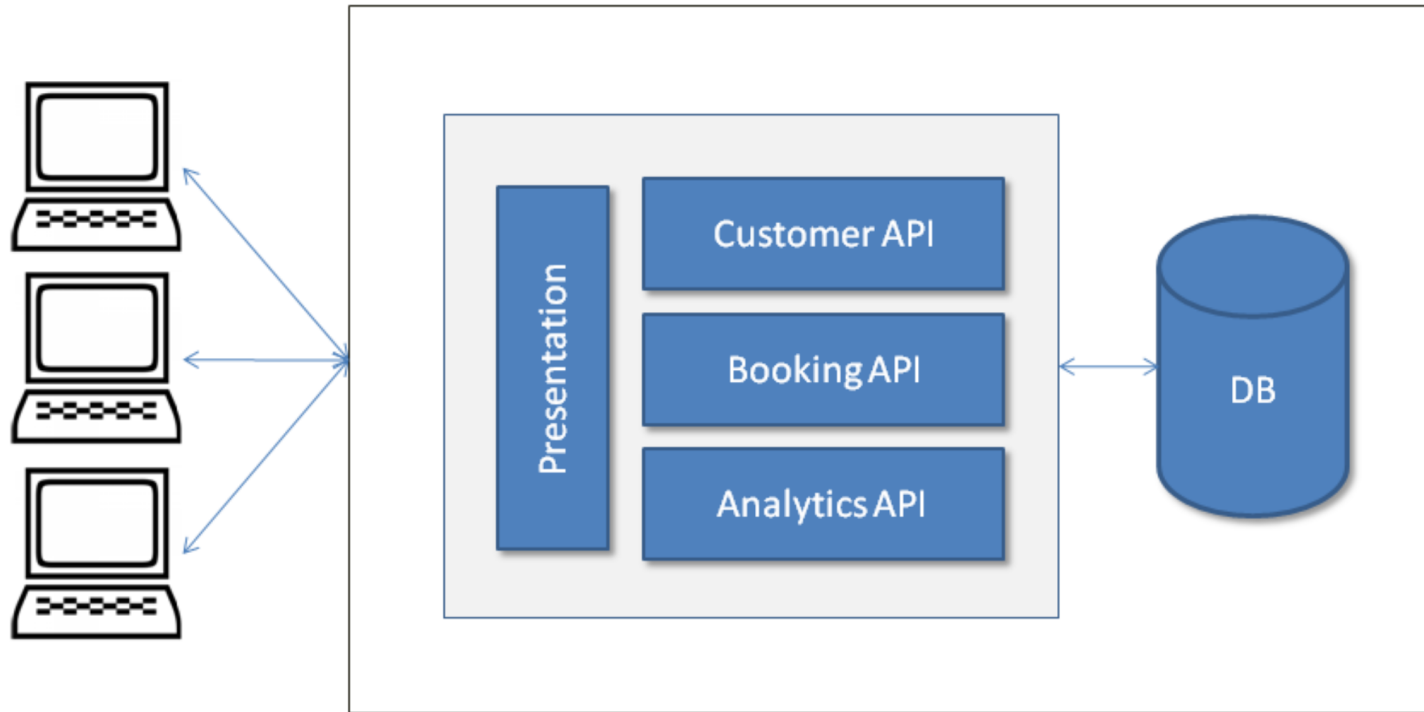
... and scales by distributing these services across servers, replicating as needed.



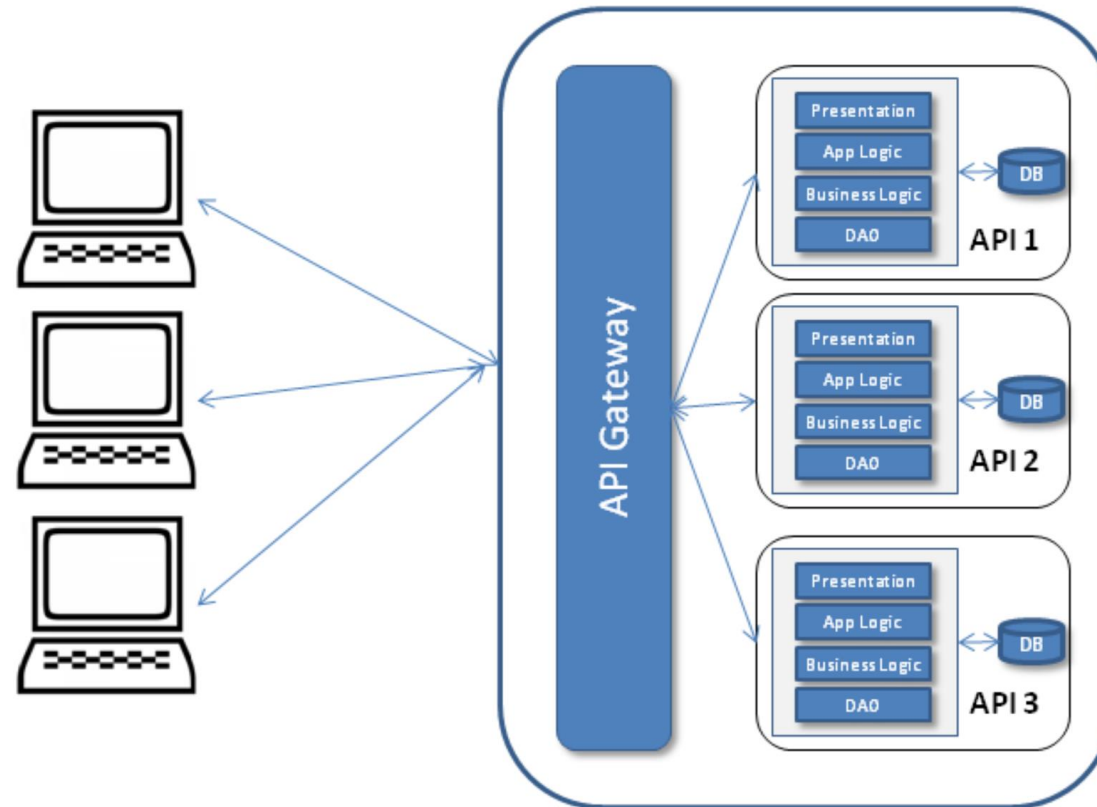
Arquitectura Monolítica



Arquitectura Monolítica con Servicios

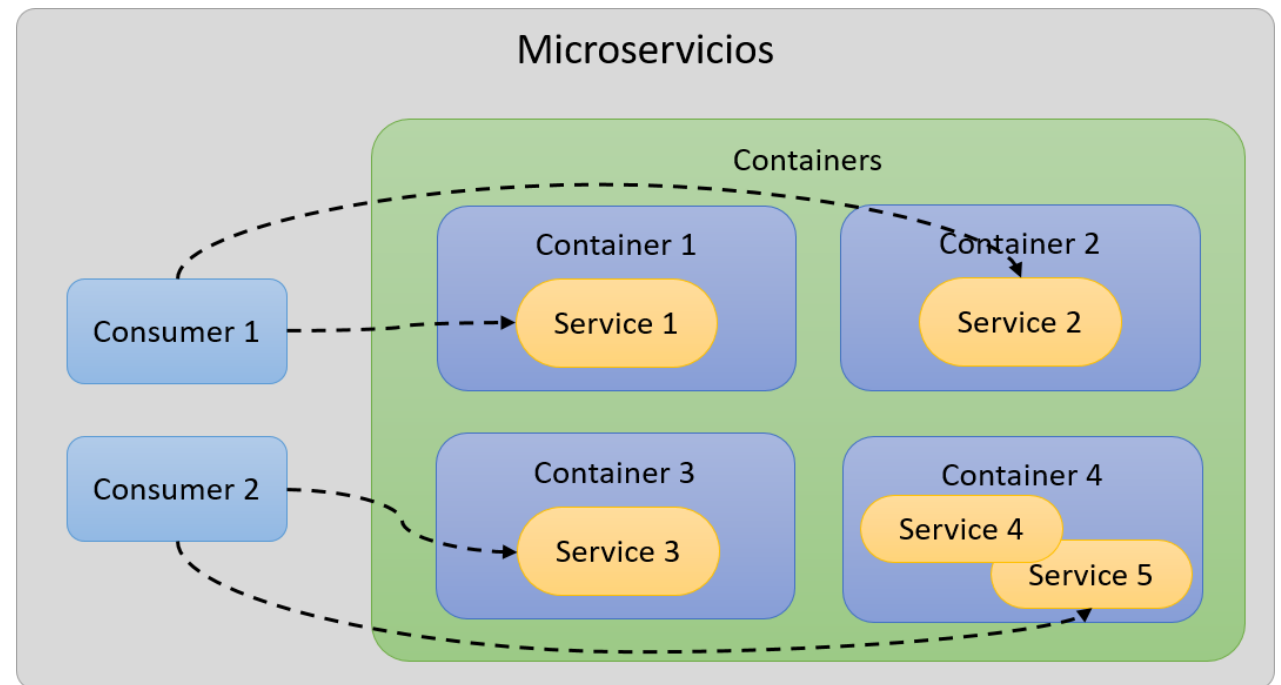


Microservicios



Microservicios

- Varios procesos o servicios
- Varios servidores, containers
- Lo habitual es que cada instancia de servicio sea un proceso.
- Por lo tanto, los servicios deben interactuar con HTTP, AMQP o un protocolo binario como TCP, en función de la naturaleza de cada servicio.



Microservicios

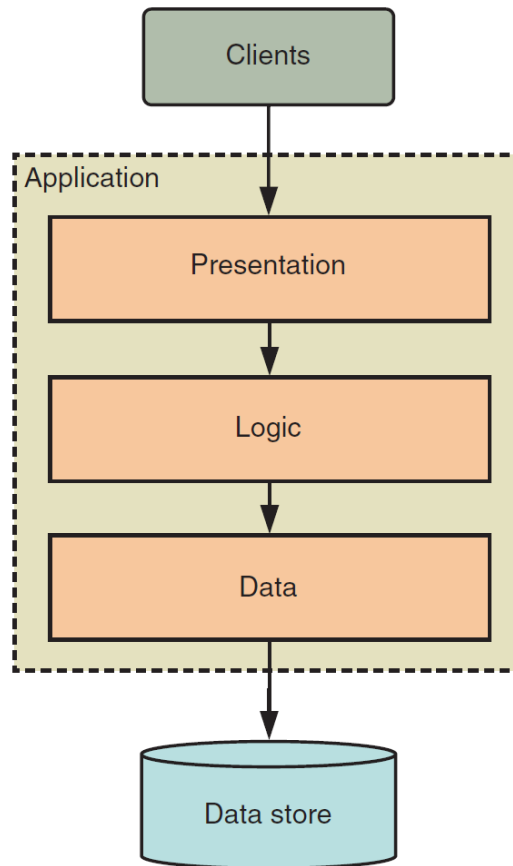
- *Martin Fowler y James Lewis* los microservicios se definen como un estilo arquitectural, una forma de desarrollar una aplicación, basada en un conjunto de pequeños servicios, cada uno de ellos ejecutándose de forma autónoma y comunicándose entre si mediante mecanismos livianos, generalmente a través de peticiones **REST** sobre HTTP a través de sus **Apis**.

Microservices

- Tony Pujals:
- *En mi modelo mental, pienso en procesos livianos autocontenidos (como en contenedores) que se comunican a través de HTTP, creados y desplegados con un esfuerzo y proceso relativamente pequeño, proporcionando APIs enfocadas de manera limitada a sus consumidores.*

Principios de Microservicios

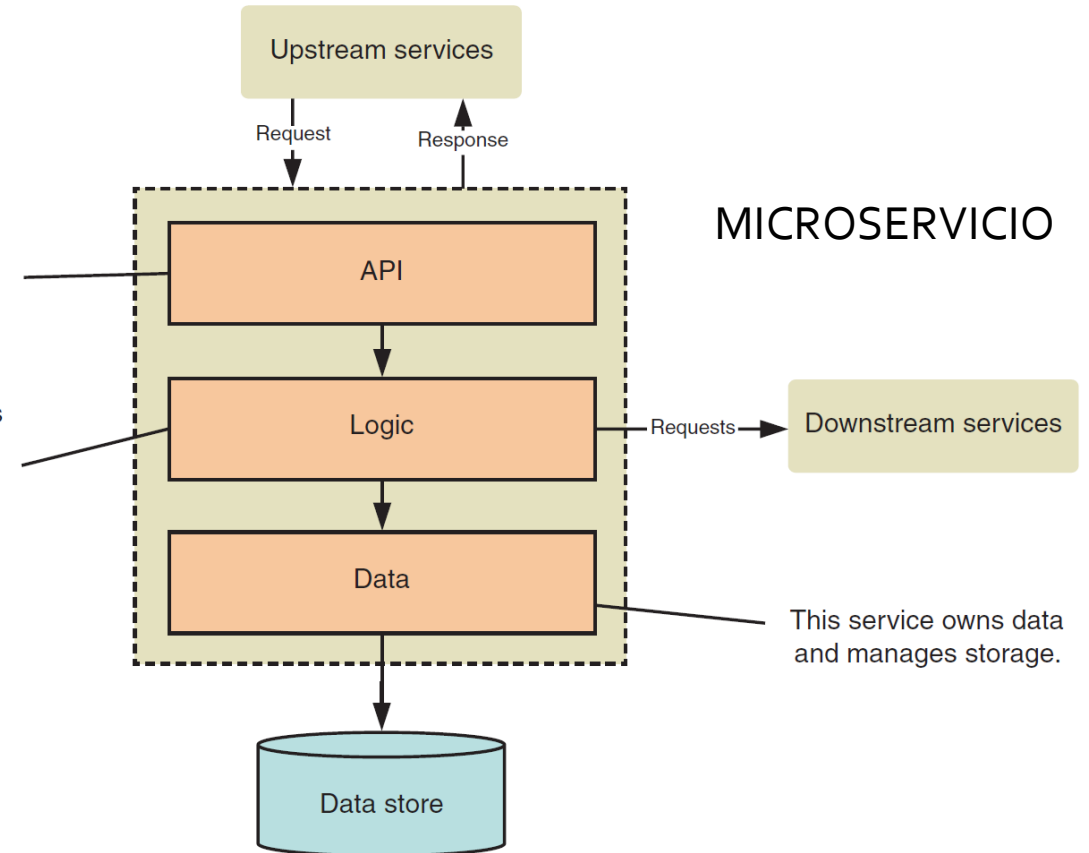
MONOLITICO



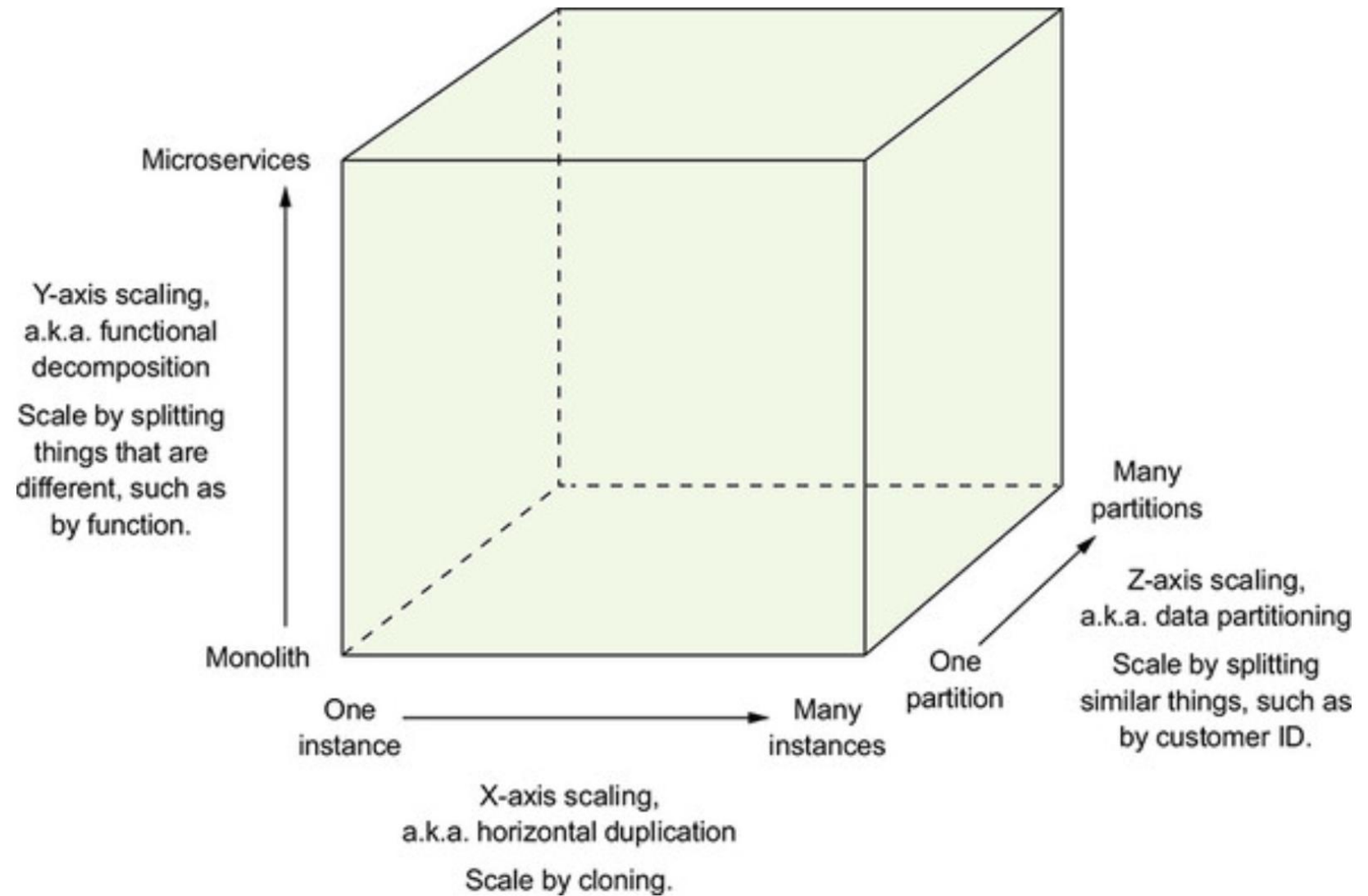
The presentation layer exposes API resources for other services.

The logic layer performs work, potentially interacting with other services.

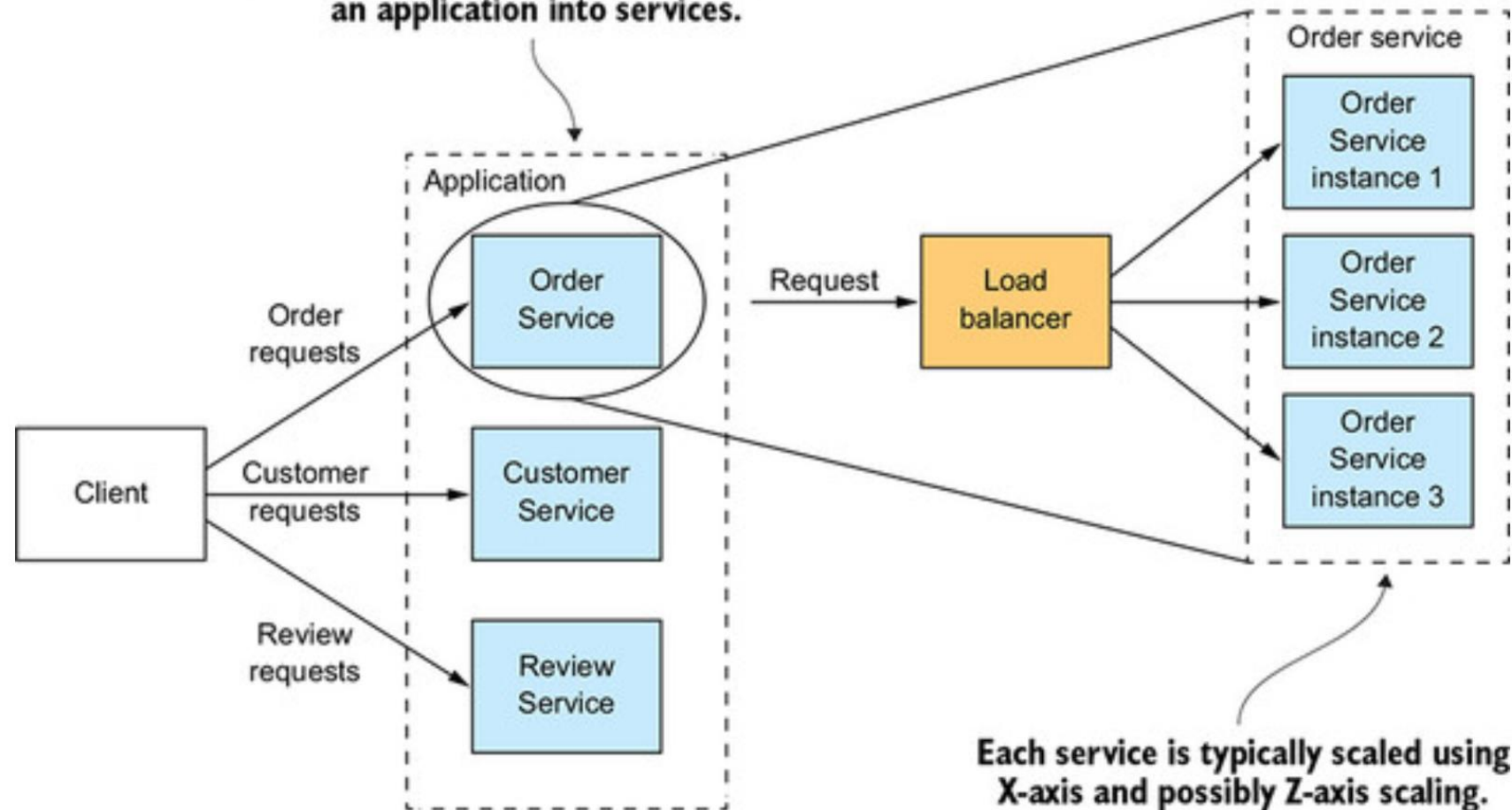
MICROSERVICIO

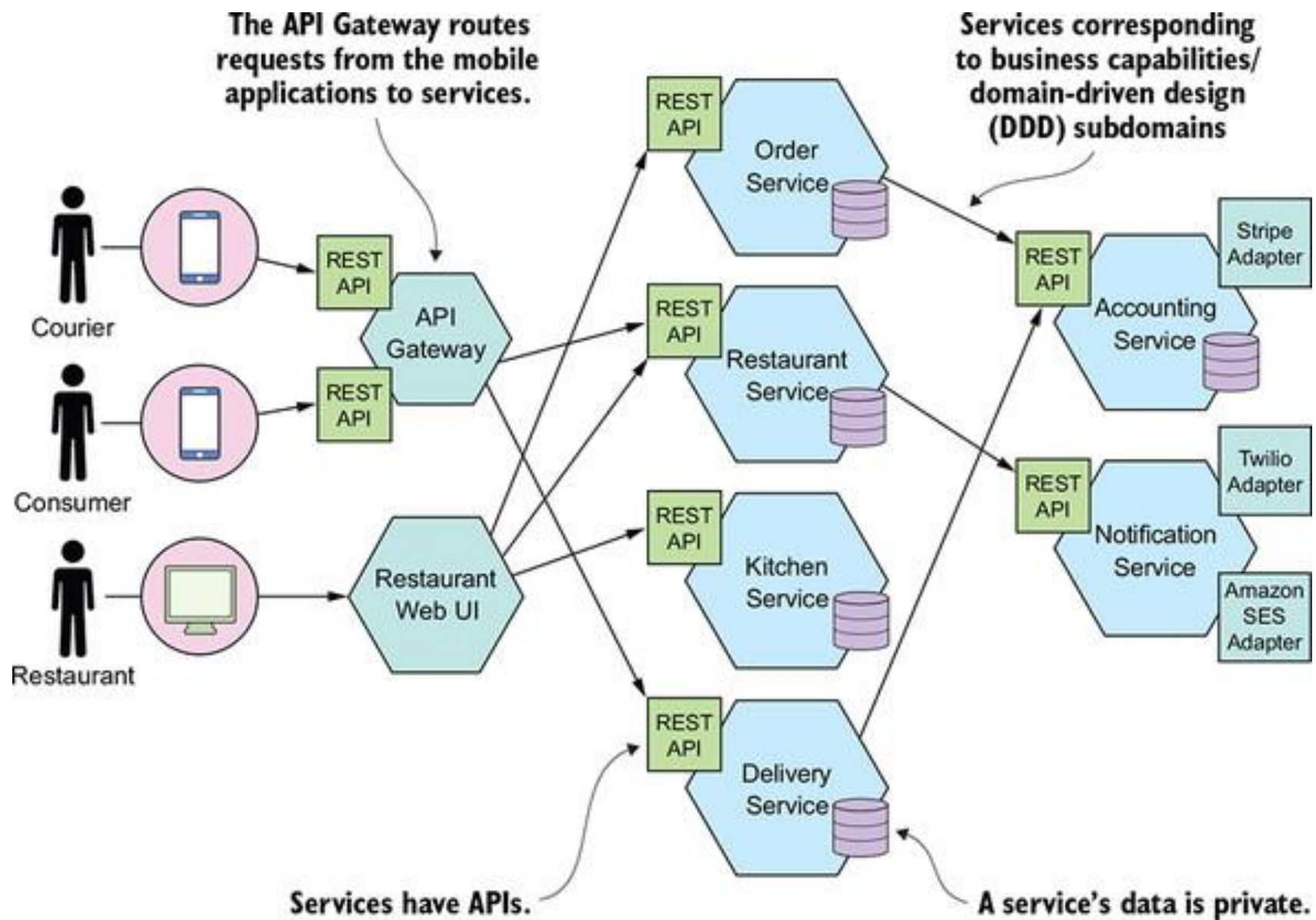


Scale Cube



Y-axis scaling functionality decomposes an application into services.

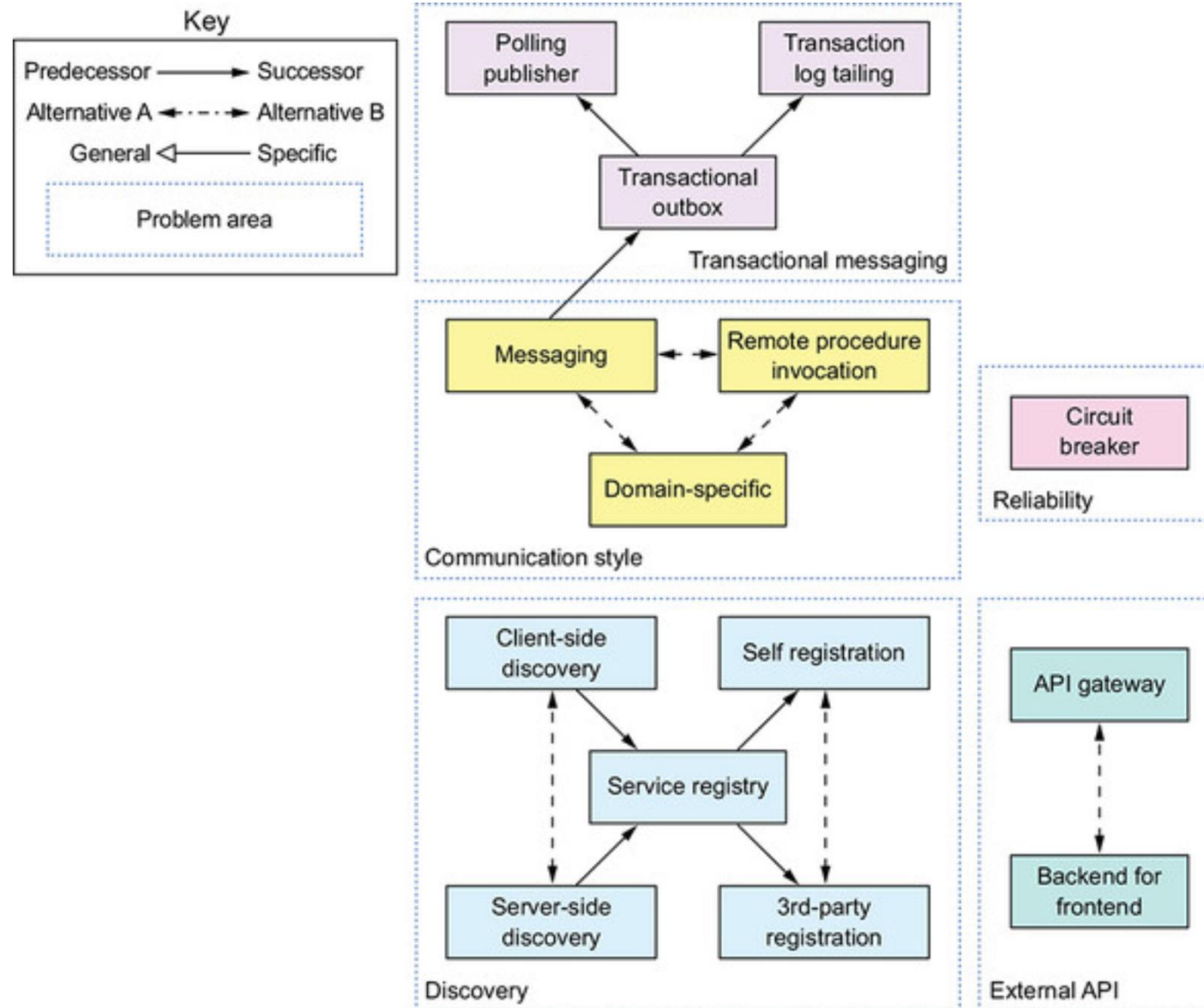




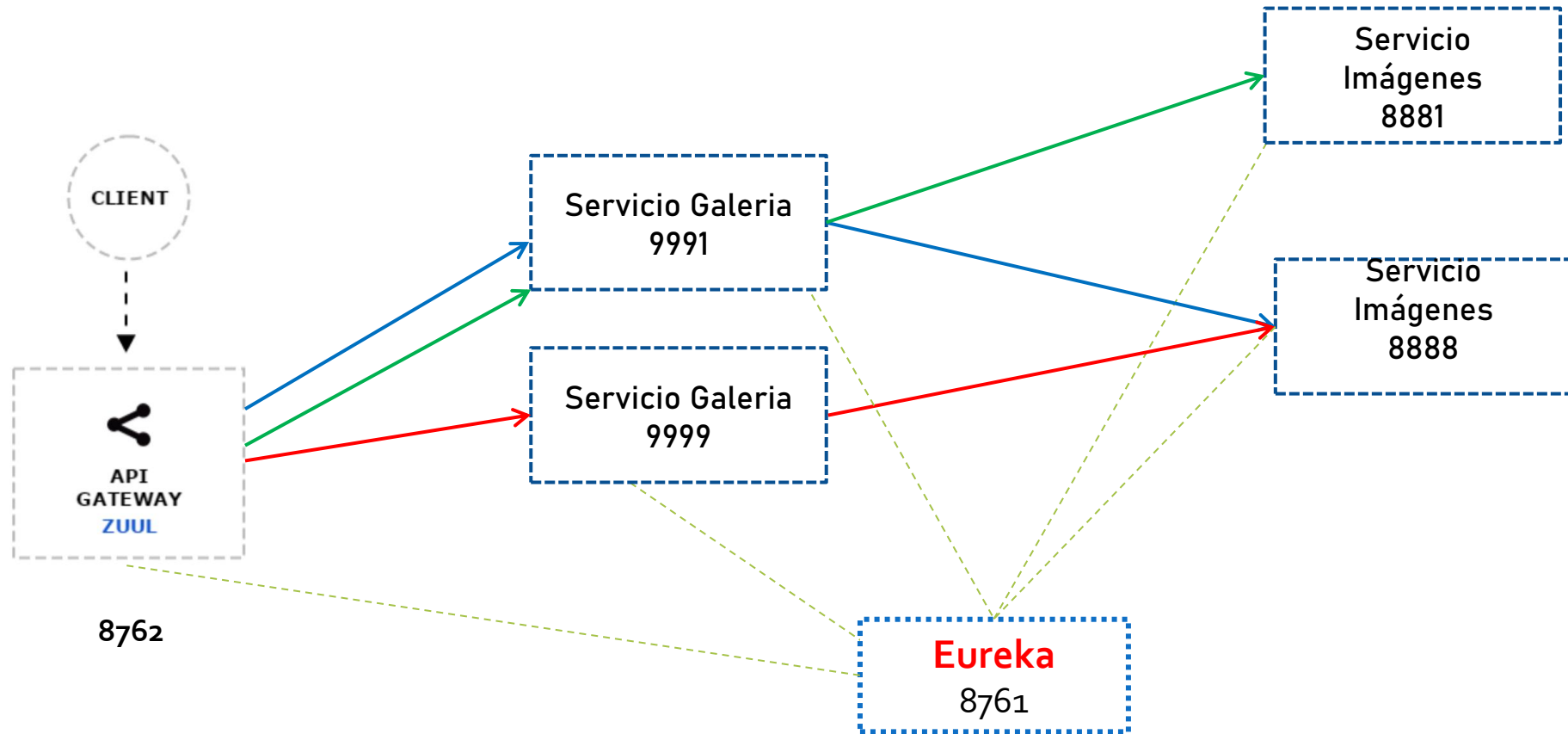
Beneficios de la arquitectura de microservicios

- Permite la entrega e implementación continuas de aplicaciones grandes y complejas.
- Los servicios son pequeños y fáciles de mantener.
- Los servicios se pueden implementar de forma independiente.
- Los servicios son escalables de forma independiente.
- La arquitectura de microservicio permite que los equipos sean autónomos.
- Permite una fácil experimentación y adopción de nuevas tecnologías.
- Tiene mejor aislamiento de fallas.

Figura 1.12. Los cinco grupos de patrones de comunicación



IMPLEMENTACIÓN DE MICROSERVICIOS



EUREKA

Eureka

- El servidor **Netflix Eureka** se utiliza para el descubrimiento y registro de servicios.
- Patrón de diseño: *Service discovery and registration*.
- No solo le permite registrarse y descubrir servicios, sino que también proporciona equilibrio de carga utilizando Ribbon o **Eureka Client**.

Eureka Client

- Netflix Ribbon se utiliza para equilibrar la carga. Se integra con los servicios Zuul y Eureka para proporcionar balanceo de carga para llamadas internas y externas.

Funcionamiento

- Cada microservicio, durante su arranque, se comunicará con el servidor Eureka para notificar que está disponible, dónde está situado, sus metadatos...
- De esta forma Eureka mantendrá en su registro la información de todos los microservicios del ecosistema.
- El nuevo microservicio continuará notificando a Eureka su estado cada 30 segundos, lo que denominan '**heartbeats**'.
- Si después de **tres** periodos Eureka no recibe notificación de dicho microservicio lo eliminará del registro. De la misma forma una vez vueltos a recibir **tres** notificaciones considerará el servicio disponible de nuevo.

Que ofrece?

- **Abstracción de la localización física de los microservicios:** cualquier microservicio que sea un cliente Eureka solo necesita conocer el identificador del microservicio al que desea invocar y Eureka resolverá su localización, puerto...
- **Conocer el estado de nuestro ecosistema de microservicios:** Eureka proporciona un dashboard que permite ver los microservicios existentes actualmente en el registro.
- Se puede configurar como cluster incrementando notablemente su tolerancia a fallos.
- Eureka proporciona soporte a multiregión, pudiendo definir diferentes agrupaciones de microservicios.
- Eureka fue inicialmente desarrollado por Netflix, quien utiliza AWS como IaaS, así Eureka fue diseñado para integrarse fácilmente con los servicios de Amazon.
- Eureka se integra con Asgard para la realización de despliegues haciendo más sencillo el cambio a nuevas o viejas releases.

Configurar Servidor Eureka

- En un archivo `application.yml` configure:

```
spring:
  application:
    name: eureka-server
server:
  port: 8761
eureka:
  client:
    registerWithEureka: false
    fetchRegistry: false
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
  instance:
    preferIpAddress: false
    hostname: localhost
    instanceId: ${eureka.instance.hostname}:${server.port}
```

En la Clase Principal de Spring Boot

```
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
```

```
@SpringBootApplication
```

```
@EnableEurekaServer
```

```
public class EurekaserverApplication {
```

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

```
}
```

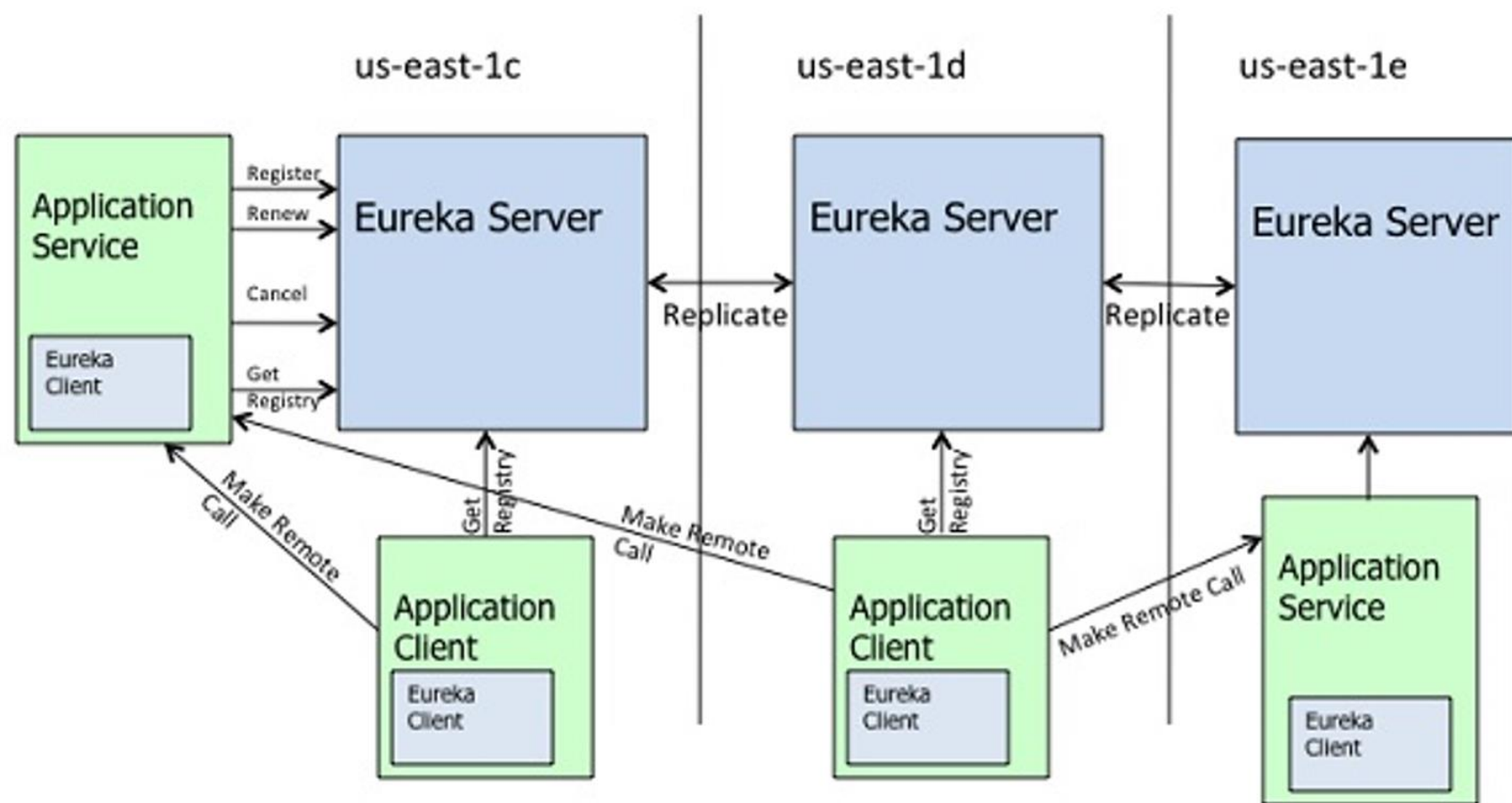

Las dependencias a considerar:

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
  </dependency>
</dependencies>
```

Version de Spring Boot Sugerida

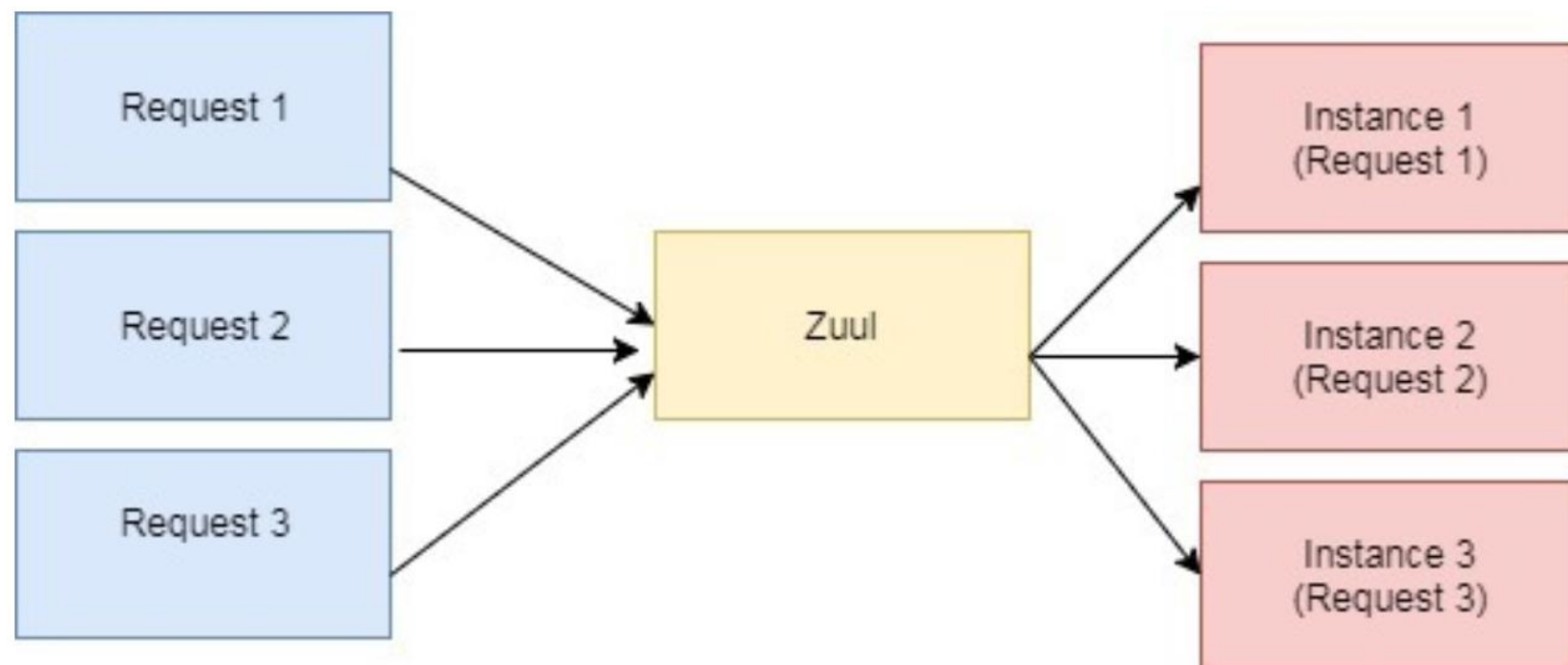
```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.1.5.RELEASE</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```



GATEWAY ZUUL

Zuul

- Servidor que funciona como **API Gateway/Edge Service** siendo el punto de entrada al ecosistema de microservicios.
- Servidor puerta de enlace o servidores **proxy**. Estos están configurados para enrutar solicitudes a diferentes microservicios o aplicaciones frontend.
- Zuul, entre muchas otras cosas, obtiene las ubicaciones de servicio de Eureka
- **Cuando Zuul recibe una solicitud, recoge una de las ubicaciones físicas disponibles y las envía a la instancia de servicio real.** Todo el proceso de almacenamiento en caché de la ubicación de las instancias de servicio y el reenvío de la solicitud a la ubicación real se proporciona de forma inmediata sin necesidad de configuraciones adicionales.
- Aquí, podemos ver cómo Zuul está encapsulando tres instancias diferentes del mismo servicio:



Dependencias en pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>
</dependencies>
```

application.properties

server.port = 8762

spring.application.name = zuul-server

eureka.client.service-url.default-zone = http://localhost:8761/eureka/

Un prefijo que puede agregarse al comienzo de todas las solicitudes.

zuul.prefix = / api

Desactiva el acceso a los servicios utilizando el nombre del servicio (es decir, servicio de galería).

Solo se debe acceder a ellos a través de la ruta definida a continuación.

zuul.ignored-services = *

Mapa de rutas a servicios

zuul.routes.gallery-service.path = /gallery/**

zuul.routes.gallery-service.service-id = gallery-service

#podemos comentarlo si no queremos que la capa de images sea inaccesible

zuul.routes.image-service.path = /image/**

zuul.routes.image-service.service-id = image-service

Debe existir
en Eureka

Debe existir
en Eureka

Clase Principal de Spring Boot

```
package com.curso.zuul;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

@SpringBootApplication
@EnableEurekaClient           // Actúa como un cliente eureka
@EnableZuulProxy             // Habilitar Zuul
public class ZuulApplication {

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

}
```

SERVICIO REST DE IMÁGENES

application.properties

`server.port=8888`

`spring.application.name=image-service`

`eureka.client.registerWithEureka=true`

`eureka.client.fetchRegistry=true`

`eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/`

Dependencias a considerar

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>true</scope>
  </dependency>
</dependencies>
```

Version de Spring Boot Sugerida

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.1.5.RELEASE</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```


Servicio de Imágenes

```
public class Image {  
    private int id;  
    private String name;  
    private String url;  
  
    public Image(int id, String name, String url) {  
        this.id = id;  
        this.name = name;  
        this.url = url;  
    }  
  
    public int getId() {  
        return id;  
    }  
    public void setId(int id) {  
        this.id = id;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public String getUrl() {  
        return url;  
    }  
    public void setUrl(String url) {  
        this.url = url;  
    }  
}
```

```
import com.demo.eurekaclient.entidades.Image;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.core.env.Environment;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.Arrays;  
import java.util.List;  
  
@RestController  
@RequestMapping("/")  
public class ServiciosDemo {  
    @Autowired  
    private Environment env;  
  
    @GetMapping("/images")  
    public List<Image> getImages() {  
        List<Image> images = Arrays.asList(  
            new Image( id: 1, name: "Th Witcher", url: "https:  
            new Image( id: 2, name: "Tolkien", url: "https://w  
            new Image( id: 3, name: "Playmobil", url: "https:/  
        );  
        return images;  
    }  
}
```

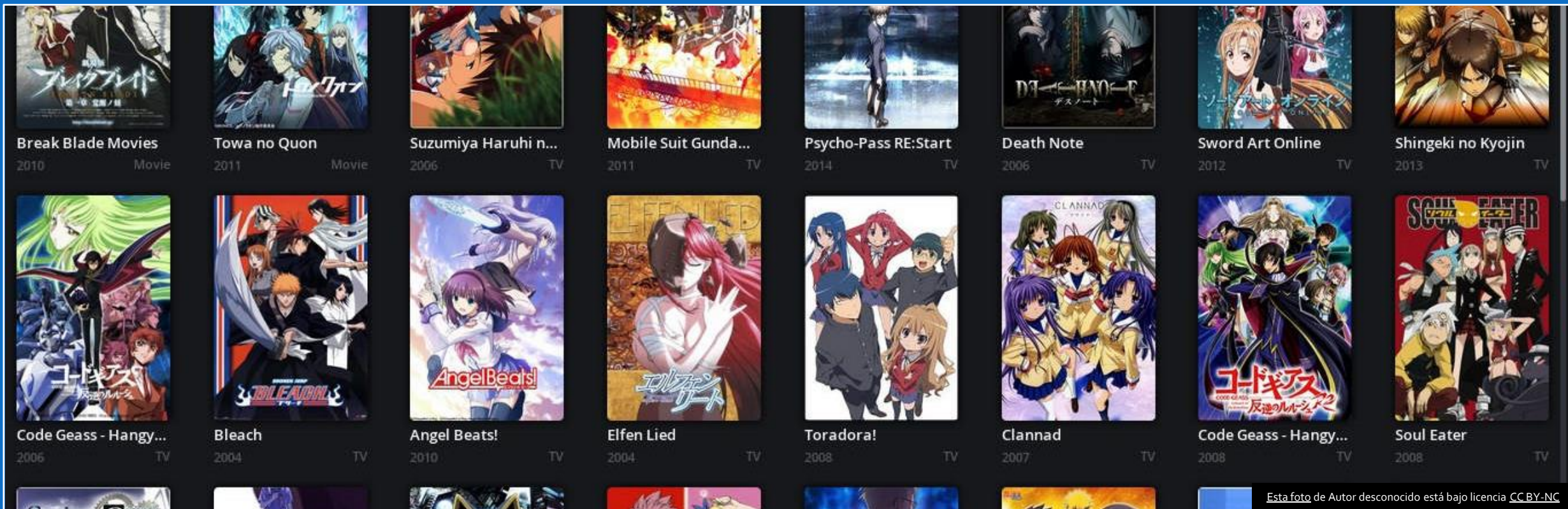
Clase principal de Spring Boot

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
@EnableEurekaClient
public class EurekaclientApplication {

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

}
```



Esta foto de Autor desconocido está bajo licencia [CC BY-NC](#)

SERVICIO REST DE GALERIA DE IMÁGENES

Usa al servicio de imagenes

application.properties

`server.port=9999`

`spring.application.name=gallery-service`

`eureka.client.registerWithEureka=true`

`eureka.client.fetchRegistry=true`

`eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka`

```

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
@EnableEurekaClient
public class FrontApplication {

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

}

@Configuration
class RestTemplateConfig {

    // Crea un bean para que restTemplate pueda llamar a los servicios REST
    @Bean
    @LoadBalanced // Load balance entre servicios corriendo en diferentes puertos.
    public RestTemplate restTemplate() { return new RestTemplate(); }

}

```

```
import java.util.List;

public class Galeria {
    private int id;
    private List<Object> images;

    public Galeria() {
    }

    public Galeria(int galleryId) {
        this.id = galleryId;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public List<Object> getImages() {
        return images;
    }

    public void setImages(List<Object> images) {
        this.images = images;
    }
}
```

```
import com.curso.front.entidades.Galeria;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.env.Environment;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
@RequestMapping("/")
public class ServiciosDemo {
    @Autowired
    private RestTemplate restTemplate;

    @Autowired
    private Environment env;

    @GetMapping("/")
    public String home() {
        // Esto es útil para la depuración
        // Al tener múltiples instancias de servicio de galería ejecutándose en diferentes puertos.
        // Nosotros balanceamos la carga entre ellos y mostramos qué instancia recibió la solicitud.
        return "Hola de Galeria Service corriendo en el puerto: " + env.getProperty("local.server.port");
    }
}
```

```

@GetMapping("/{id}")
public Galeria getGallery(@PathVariable final int id) {
    // crea gallery object
    Galeria galeria = new Galeria();
    galeria.setId(id);

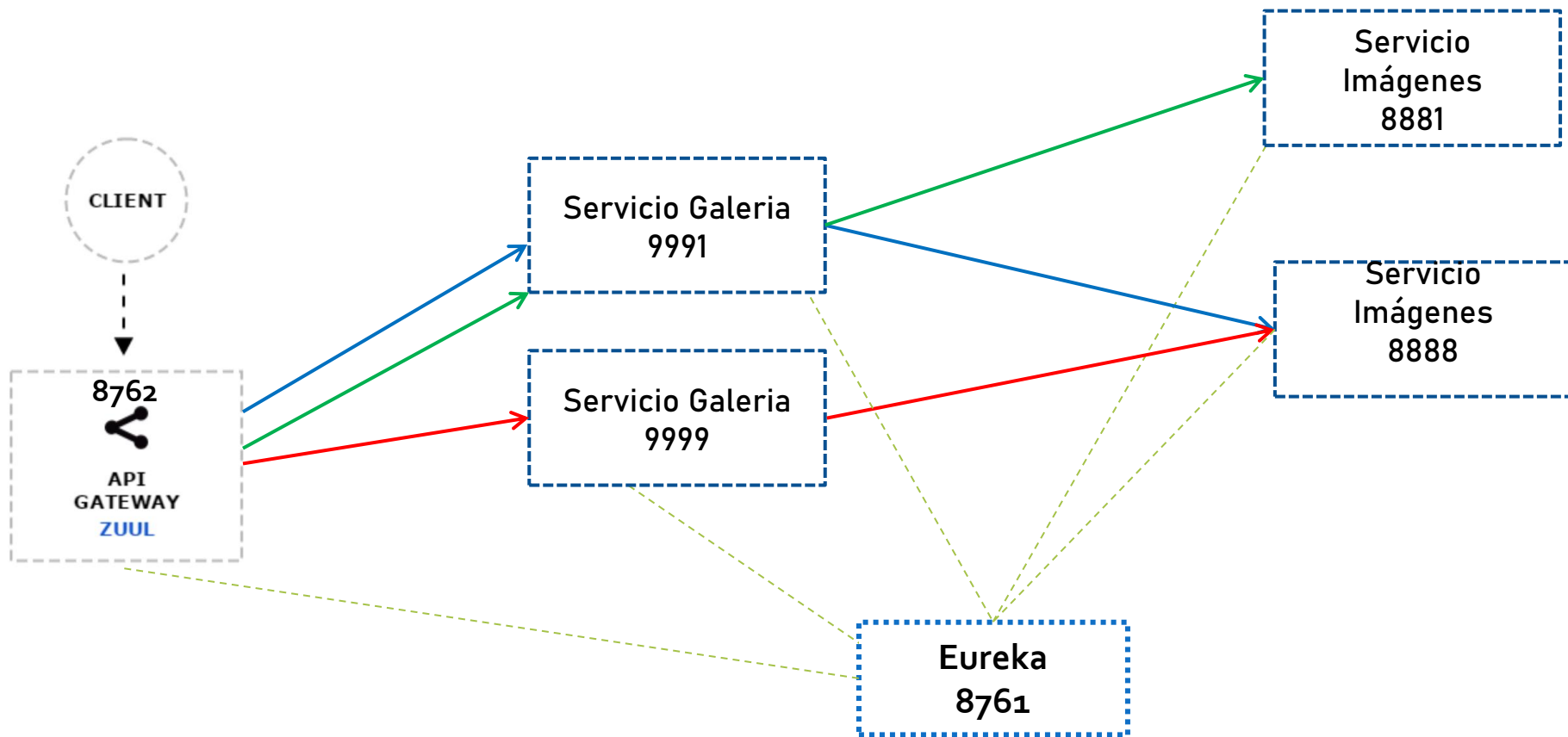
    // obtiene una lista de imágenes
    List<Object> imagenes = restTemplate.getForObject( url: "http://image-service/images/", List.class);
    /*Al llamar a cualquier servicio desde el navegador, no podemos llamarlo por su nombre como lo hacemos aqui
    desde el servicio de getGallery: esto se usa internamente entre los servicios. Entonces puedes usar Zuul
    para los navegadores
    */
    galeria.setImages(imagenes);

    return galeria;
}

// ----- Admin Area -----
// Para usuarios de rol de administrador cuando se usa Spring Security
@GetMapping("/admin")
public String homeAdmin() {
    return "Este es administrador de area de Galeria service corriendo en el puerto: " + env.getProperty("local.
}
}

```

PRUEBAS



Pasos

- 1) Levantar el Servidor Eureka
- 2) Levantar el Servicio de Imágenes
instancia1 Puerto:8888, instancia2: Puerto:8881

Para levantar el servicio en otro Puerto ir a la carpeta donde está el jar y ejecutar:
java -jar eurekaclient-o.o.1-SNAPSHOT.jar --server.port=8888 u 8881

- 3) 3ro. Levantar el Servicio de Galleria de Imágenes
instancia 1 Puerto:9999, instancia2 Puerto:9991


Para levantar el servicio en otro Puerto ir a la carpeta donde está el jar y ejecutar:
java -jar front-o.o.1-SNAPSHOT.jar --server.port=9999 ó 9991

(1) WhatsApp

Eureka

+

localhost:8761

HOMELAST 1000 SINCE STARTUP

System Status

Environment	test
Data center	default

Current time	2019-09-18T13:11:30 -0500
Uptime	00:02
Lease expiration enabled	false
Renews threshold	6
Renews (last min)	3

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
GALLERY-SERVICE	n/a (1)	(1)	UP (1) - host.docker.internal:gallery-service:9999
IMAGE-SERVICE	n/a (1)	(1)	UP (1) - host.docker.internal:image-service:8888
ZUUL-SERVER	n/a (1)	(1)	UP (1) - host.docker.internal:zuul-server:8762

General Info

Name	Value
total-avail-memory	545mb
environment	test
num-of-cpus	8

Pruebas – usando Gateway Zuul

❑ Probar el Servicio de Galleria de Imágenes:

- localhost:8762/gallery

- Saldrá:

“Hola de Galeria Service corriendo en el puerto: 9999”

- De nuevo localhost:8762/gallery

“Hola de Galeria Service corriendo en el puerto: 9991”

Se confirma balanceo!

Pruebas

Ahora el servicio gallery:

<http://localhost:9999/1> se llama directamente en la red interna.

Con Zuulu: <http://localhost:8762/gallery/1> que a su vez el servicio llama gallery-service de Eureka balanceado. Sale:

```
{ "id": 1, "images": [ { "id": 1, "name": "The  
Witcher", "url": "https://www.imdb.com/gallery/rg1859820288/mediaviewer/rm109682433"}, { "id": 2, "name": "Tolkien", "url": "https://w  
ww.imdb.com/gallery/rg1859820288/mediaviewer/rm268985856"}, { "id": 3, "name": "Playmobil", "url": "https://www.imdb.com/gallery/  
rg1859820288/mediaviewer/rm2625857024"} ] }
```

Considerar que el servicio de Galeria de Imágenes llama al servicio del Imágenes también Balanceado, ver este último caso el log del servicio de imágenes puede salir:

Service Image:8888 ó Service Image:8881 , pruebe varias veces y verá.