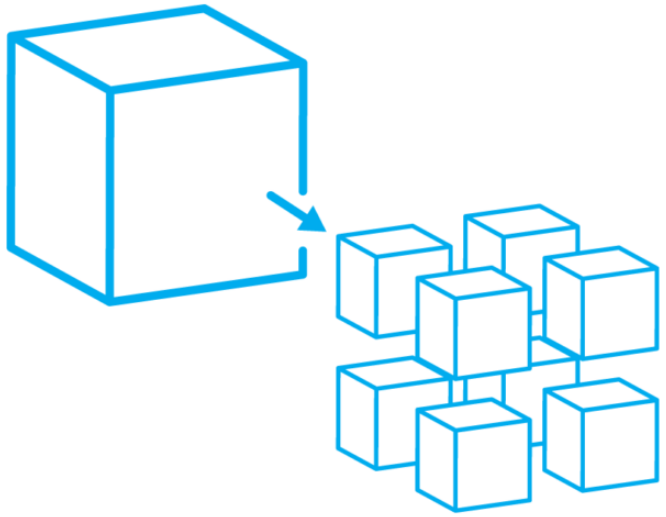


Microservicios – Arquitectura y Desarrollo

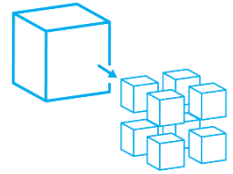
Nivel Avanzado

Por: Carlos Carreño

ccarrenovi@Gmail.com

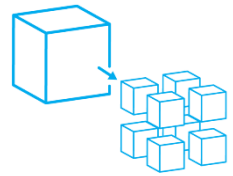
May, 2020

Modulo 2 Diseño de Microservicios



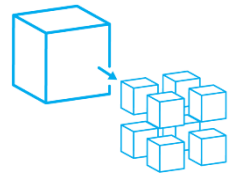
- Modelado de Microservicios
- Características de Microservicios
- Revisión de Patrones de Diseño de Microservicios
  - Direct Communication
  - API Gateway
  - Message Broker
  - Discovery (Registry Service)
  - Centralized Configuration
  - Retry
  - Circuit Breaker
  - Health Check API
- Los 12 Factores
- Modelos de Referencia
- Data y Microservicios
- Monitoreo y Alertas
- Deployment

Modelado de Microservicios



- El modelado del software, en general es una técnica para tratar con la complejidad inherente a estos sistemas. El uso de modelos ayuda al ingeniero de software a "visualizar" el sistema a construir. Además, los modelos de un nivel de abstracción mayor pueden utilizarse para la comunicación con el cliente.
- El modelar los Microservicios antes de construirlos aumenta la productividad
- En el modelado de los Microservicios utilizamos patrones de Microservicios que representan las buenas practicas aceptadas por la industria

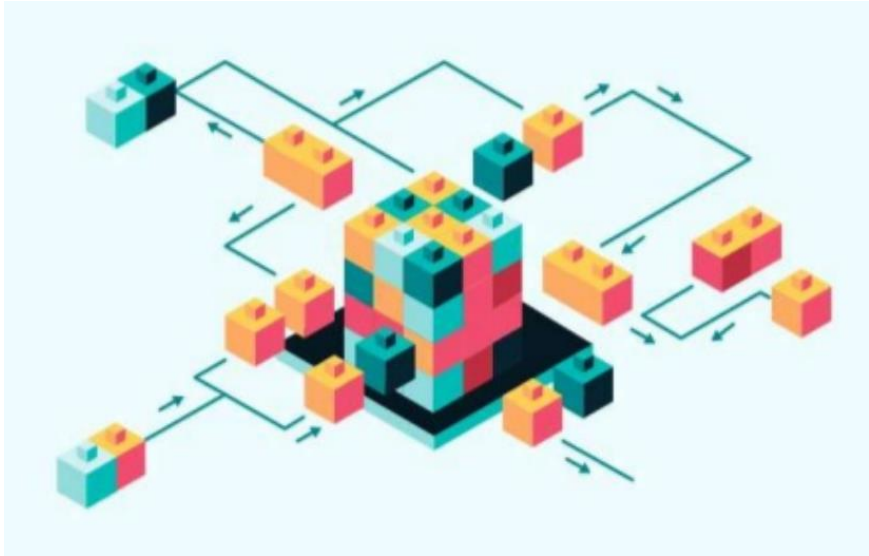
Características de los Microservicios



1) Modelado alrededor de un solo problema o dominio del negocio.
2) Implementa su propia lógica de negocios, almacenamiento persistente y colaboración externa.
3) Tiene un contrato publicado individualmente, también conocido como API
4) Capaz de ejecutarse de forma aislada
5) Independiente y desacoplado de otros servicios.
6) Fácilmente reemplazado o actualizado.
7) Escalado e implementado independiente de otros servicios

Los doce factores

- Los doce factores es una **metodología** que ayuda a construir aplicaciones de software como servicio. Prepara a los equipos del proyecto para construir un flujo para el desarrollo de aplicaciones en la nube que puedan **escalar de manera elástica**.



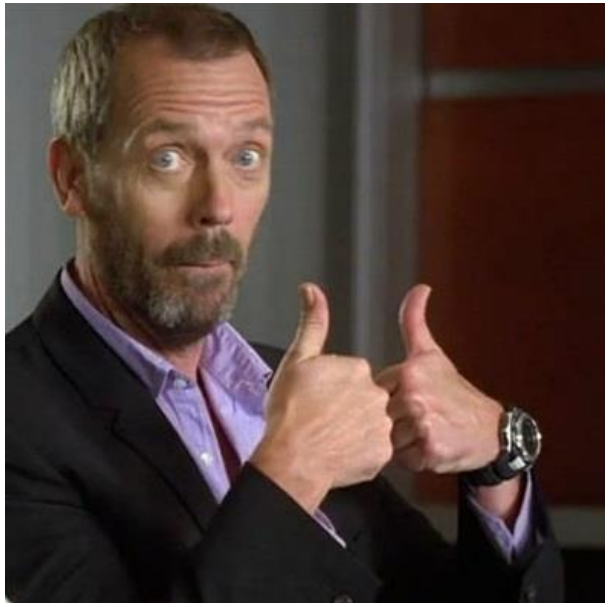
Los doce factores

- Principios de la disciplina de los doce factores

Ref: Manifiesto de los doce factores, https://12factor.net/es/

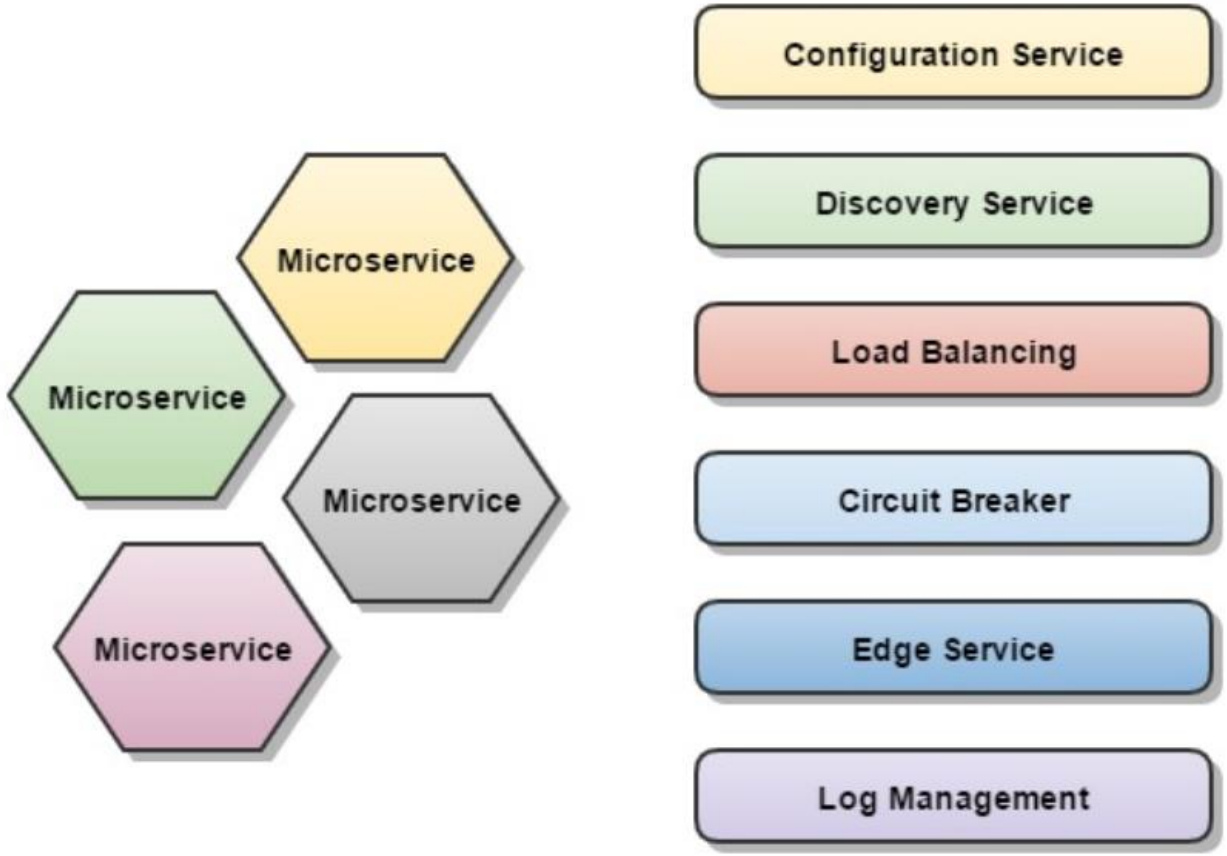
Implantación de Microservicios

- Para la implantación de una arquitectura de microservicios hemos tener en cuenta 3 aspectos principalmente:
  - Un modelo de referencia en el que definir las necesidades de una arquitectura de microservicios.
  - Un modelo de implementación en el que decidiremos y concretaremos la implementación de los componentes vistos en el modelo de referencia.
  - Un modelo de despliegue donde definir cómo se van a desplegar los distintos componentes de la arquitectura en los diferentes entornos.

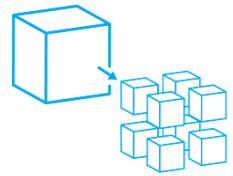


Modelo de Referencia

- Los siguientes serán los componentes que vamos a necesitar en una arquitectura de microservicios:

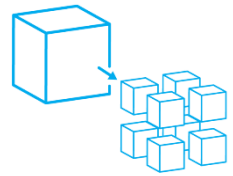


Modelo de Referencia



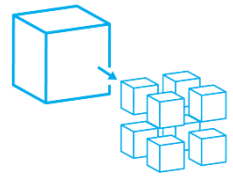
- **Servidor de configuración central**
  - Este componente se encargará de centralizar y proveer remotamente la configuración a cada microservicio. Esta configuración se mantiene convencionalmente en un repositorio Git, lo que nos permitirá gestionar su propio ciclo de vida y versionado.
- **Servicio de registro / descubrimiento**
  - Este servicio centralizado será el encargado de proveer los endpoints de los servicios para su consumo. Todo microservicio se registrará automáticamente en él en tiempo de bootstrap.
- **Balanceo de carga (Load balancer)**
  - Este patrón de implementación permite el balanceo entre distintas instancias de forma transparente a la hora de consumir un servicio.

...



- Tolerancia a fallos (Circuit breaker)
  - ❑Mediante este patrón conseguiremos que cuando se produzca un fallo, este no se propague en cascada por todo el pipe de llamadas, y poder gestionar el error de forma controlada a nivel local del servicio donde se produjo.
- Servidor perimetral / exposición de servicios (Edge server)
  - ❑Será un gateway en el que se expondrán los servicios a consumir.
- Centralización de logs
  - ❑Se hace necesario un mecanismo para centralizar la gestión de logs. Pues sería inviable la consulta de cada log individual de cada uno de los microservicios.

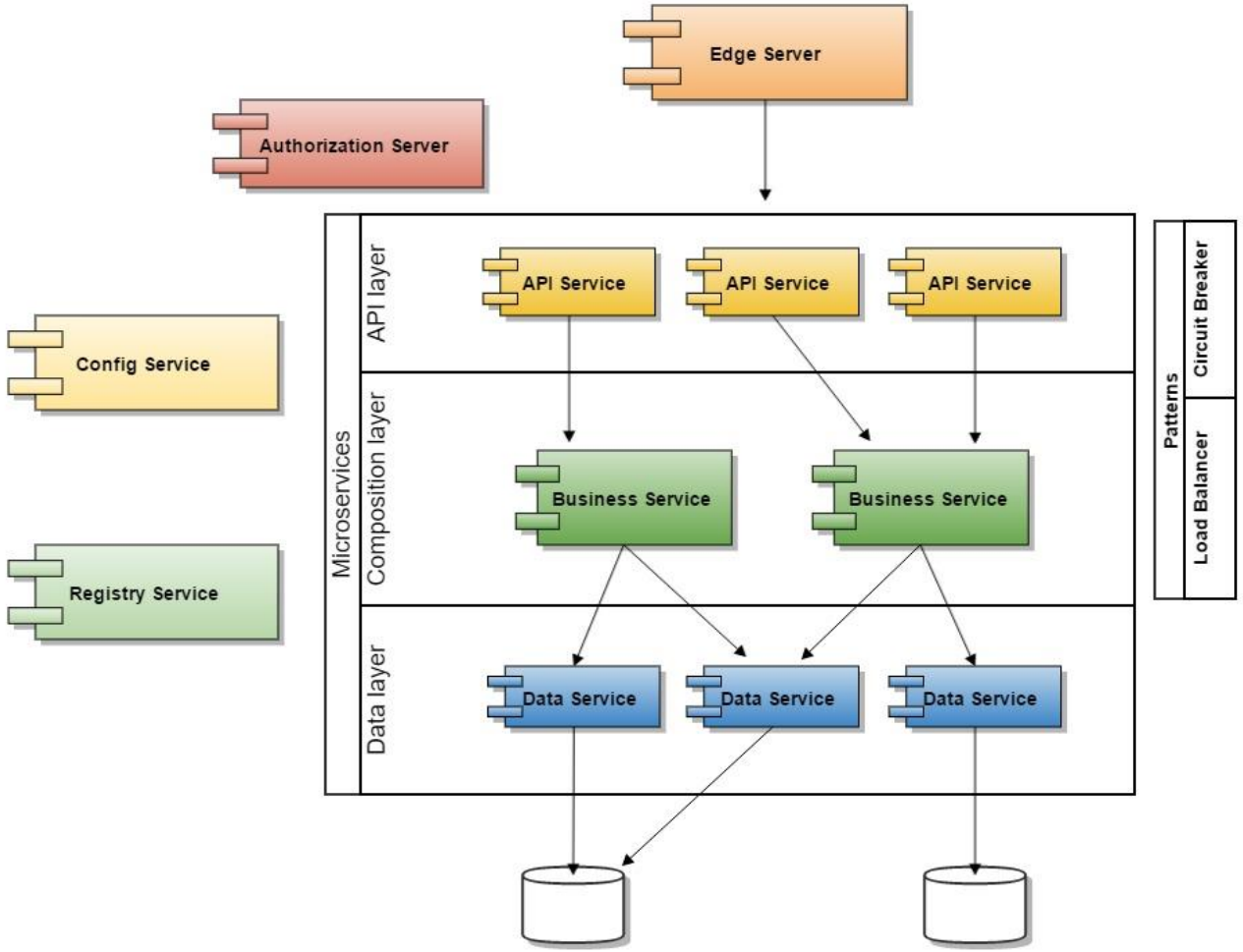
...



- Servidor de Autorización
  - Para implementar la capa de seguridad (recomendable en la capa de servicios API)
- Monitorización
  - Es interesante el poder disponer de mecanismos y algún dashboard para monitorizar aspectos de los nodos como, salud, carga de trabajo

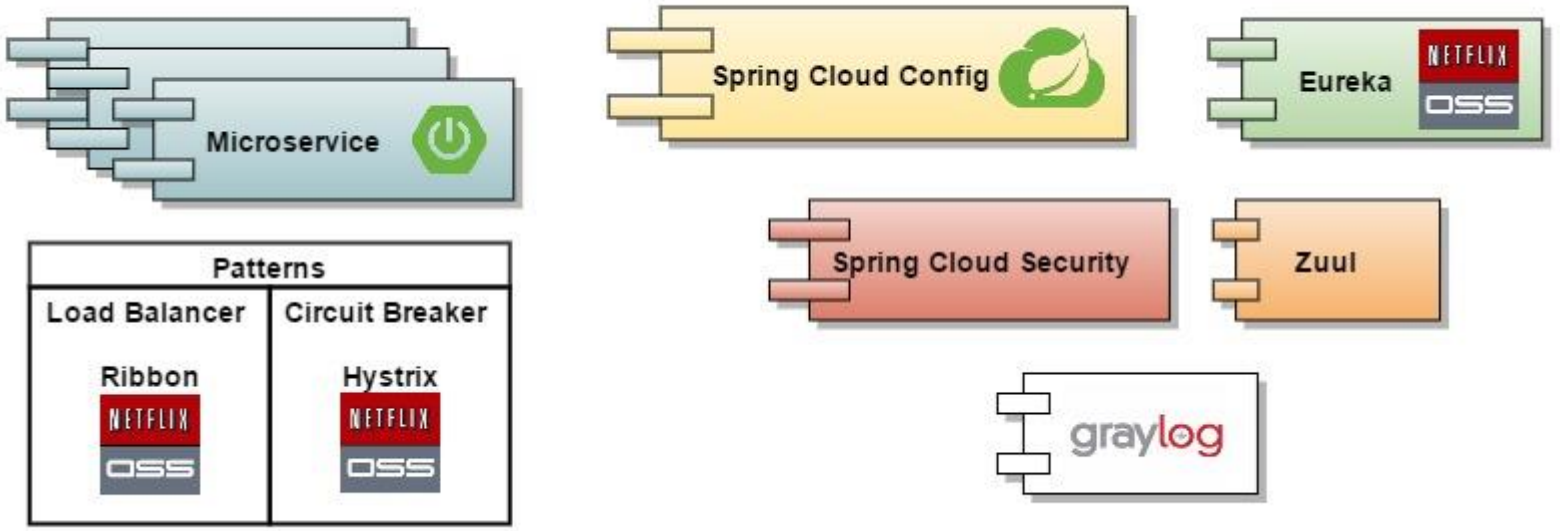
Capas de la Arquitectura de Microservicios

- El siguiente diagrama muestra la arquitectura de microservicios en capas.

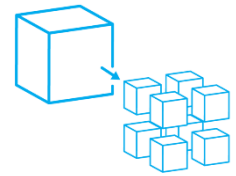


Modelo de Implementación

- Basándonos en el modelo de referencia, vamos a definir un modelo de implementación para cada uno de los componentes descritos. Para ello haremos uso del stack tecnológico de **Spring Cloud** y **Netflix OSS**



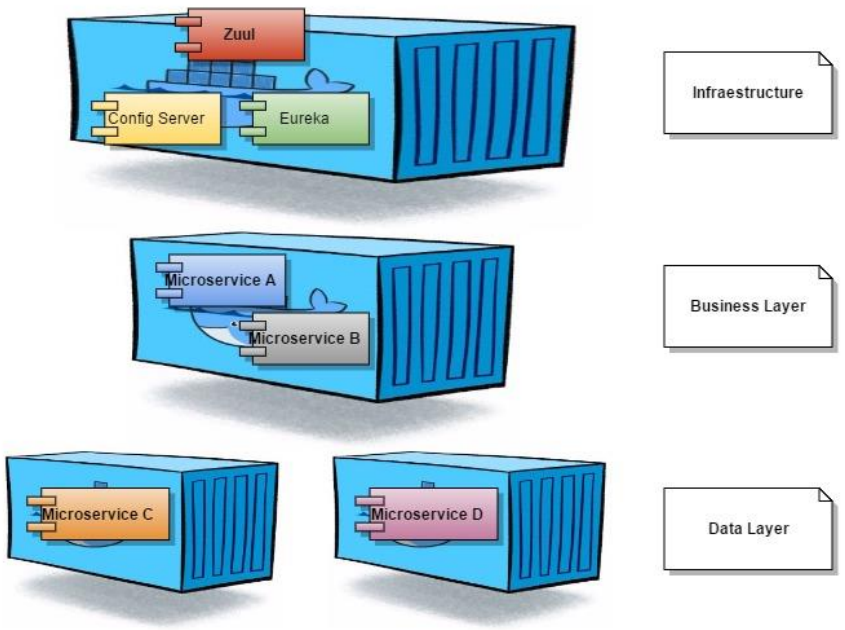
…



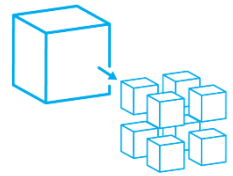
- Microservicios propiamente dichos: Serán aplicaciones Spring Boot con controladores Spring MVC. Utilizaremos Swagger para documentar y definir nuestro API.
- Config Server: microservicio basado en Spring Cloud Config. Utilizaremos Git como repositorio de configuración.
- Registry / Discovery Service: microservicio basado en **Eureka** de Netflix OSS.
- Load Balancer: utilizaremos **Ribbon** de Netflix OSS que ya viene integrado en REST-template de Spring.
- Circuit breaker: utilizaremos **Hystrix** de Netflix OSS.
- Gestión de Logs: utilizaremos **Graylog**
- Servidor perimetral: utilizaremos **Zuul** de Netflix OSS.
- Servidor de autorización: implementaremos el servicio con **Spring Cloud Security**.

Modelo de Despliegue

- Se refiere al modo en que vamos a organizar y gestionar los despliegues de los microservicios, así como a las tecnologías que podemos usar para tal fin.
- Existen convencionalmente dos tendencias en este sentido a la hora de encapsular microservicios:
  - ❑Máquinas virtuales.
  - ❑Contenedores.

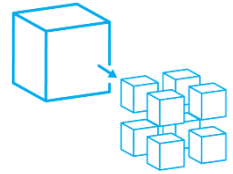


PaaS y Contenedores



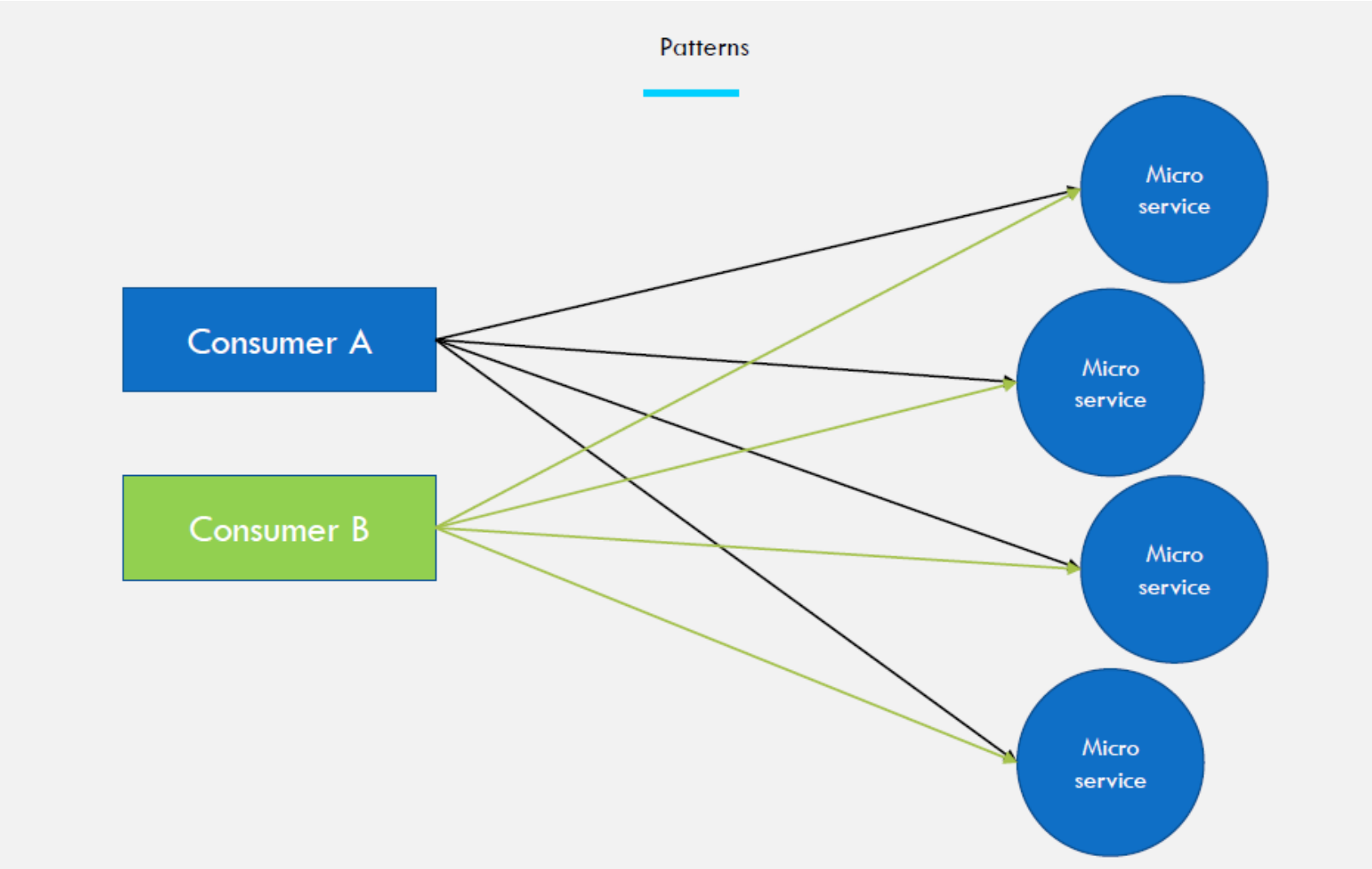
- Como tecnología de contenedores usaremos **Docker**, ya que esta tecnología es la que está teniendo mayor acogida y repercusión en entornos **cloud** y **PaaS**.
- Es conveniente pensar en la automatización y orquestación de los despliegues siguiendo el paradigma cloud. Devops (CI/CD)
- Las opciones son montar sobre una PaaS un cluster de Docker donde desplegar de forma *automágica* y transparente nuestros contenedores.
- Herramientas como **Kubernetes** y **OpenShift** aportan registry y config management a nivel de infraestructura, mientras que utilizaremos las opciones de Spring Cloud y Netflix OSS para implementar estos servicios.

Revisión de Patrones de Diseño de Microservicios

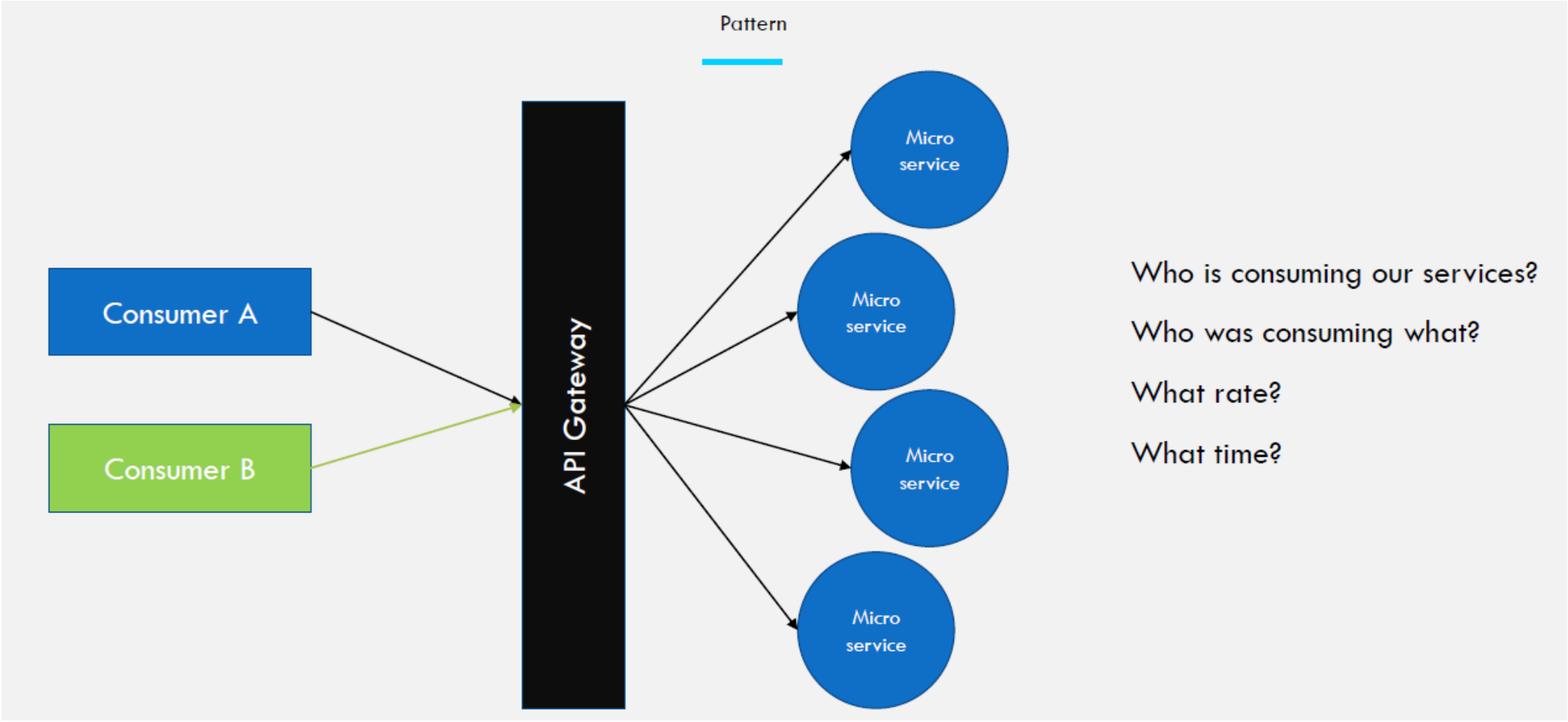


- Direct Communication
- API Gateway
- Message Broker
- Discovery (Registry Service)
- Centralized Configuration
- Retry
- Circuit Breaker
- Health Check API

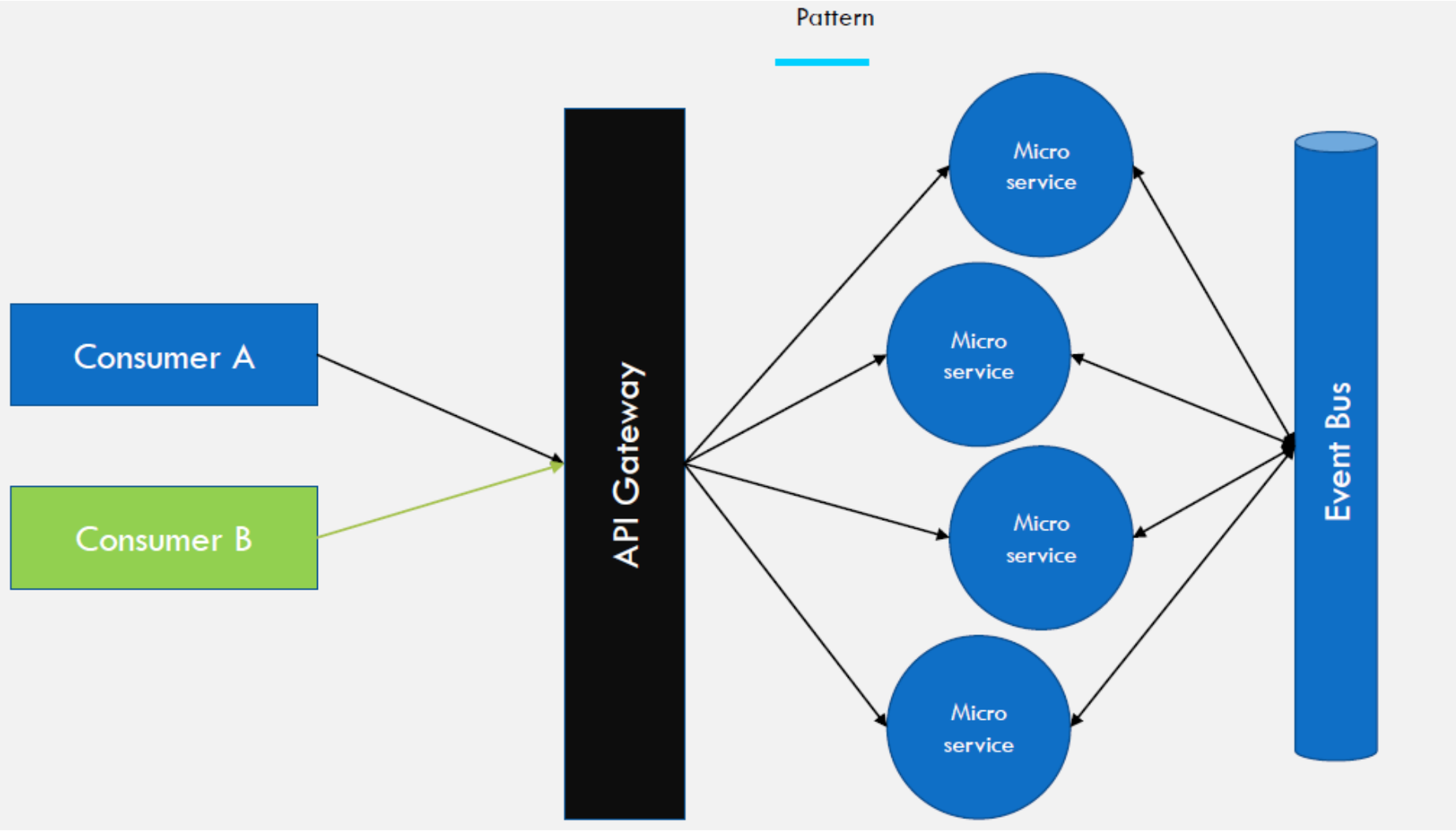
Direct Communication



API Gateway

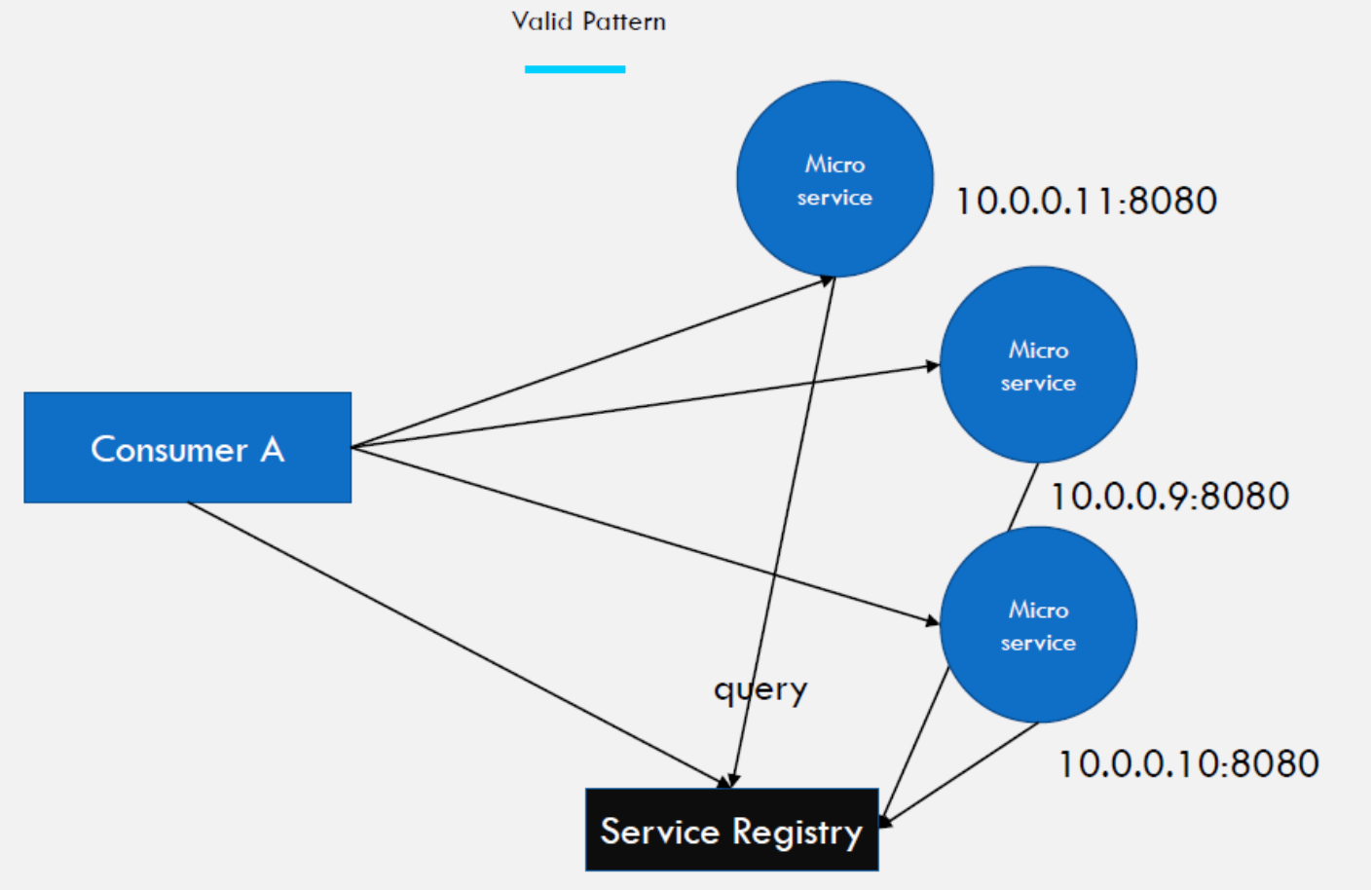

Pattern
Consumer A
Consumer B
API Gateway
Micro service
Micro service
Micro service
Micro service
Who is consuming our services?
Who was consuming what?
What rate?
What time?

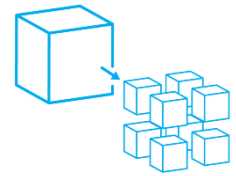
Message Broker



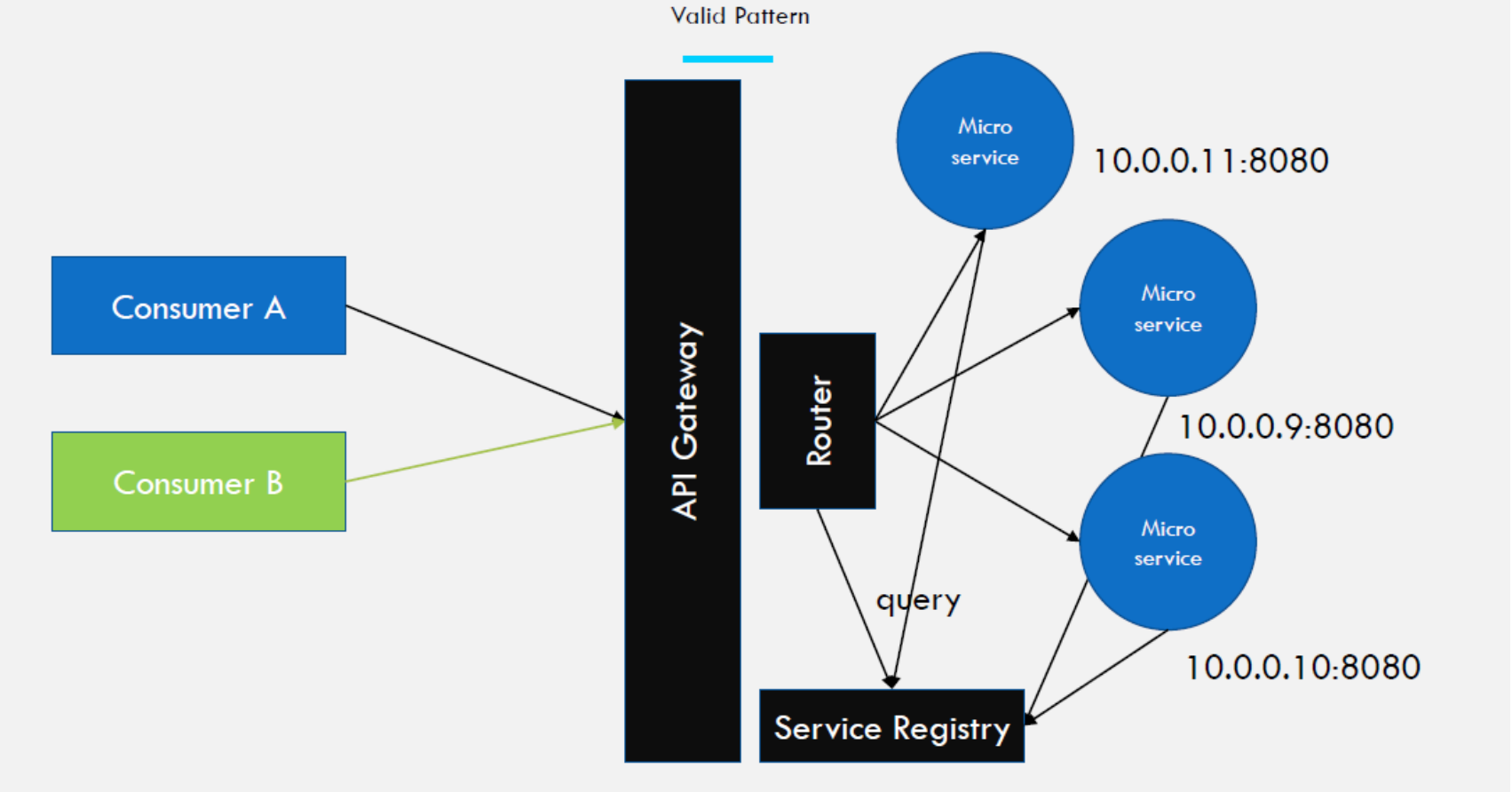
Discovery

- Client-Side Discovery

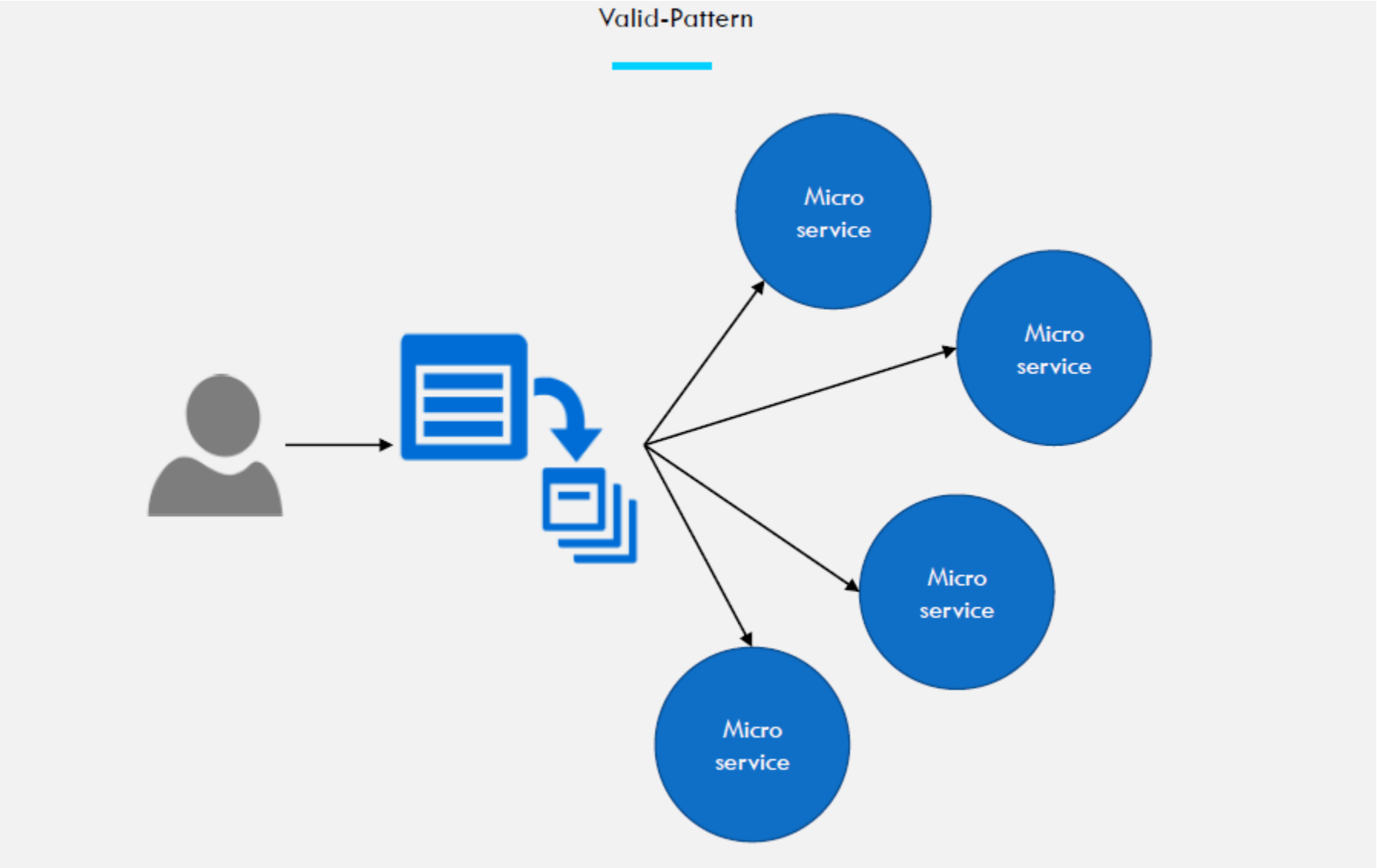




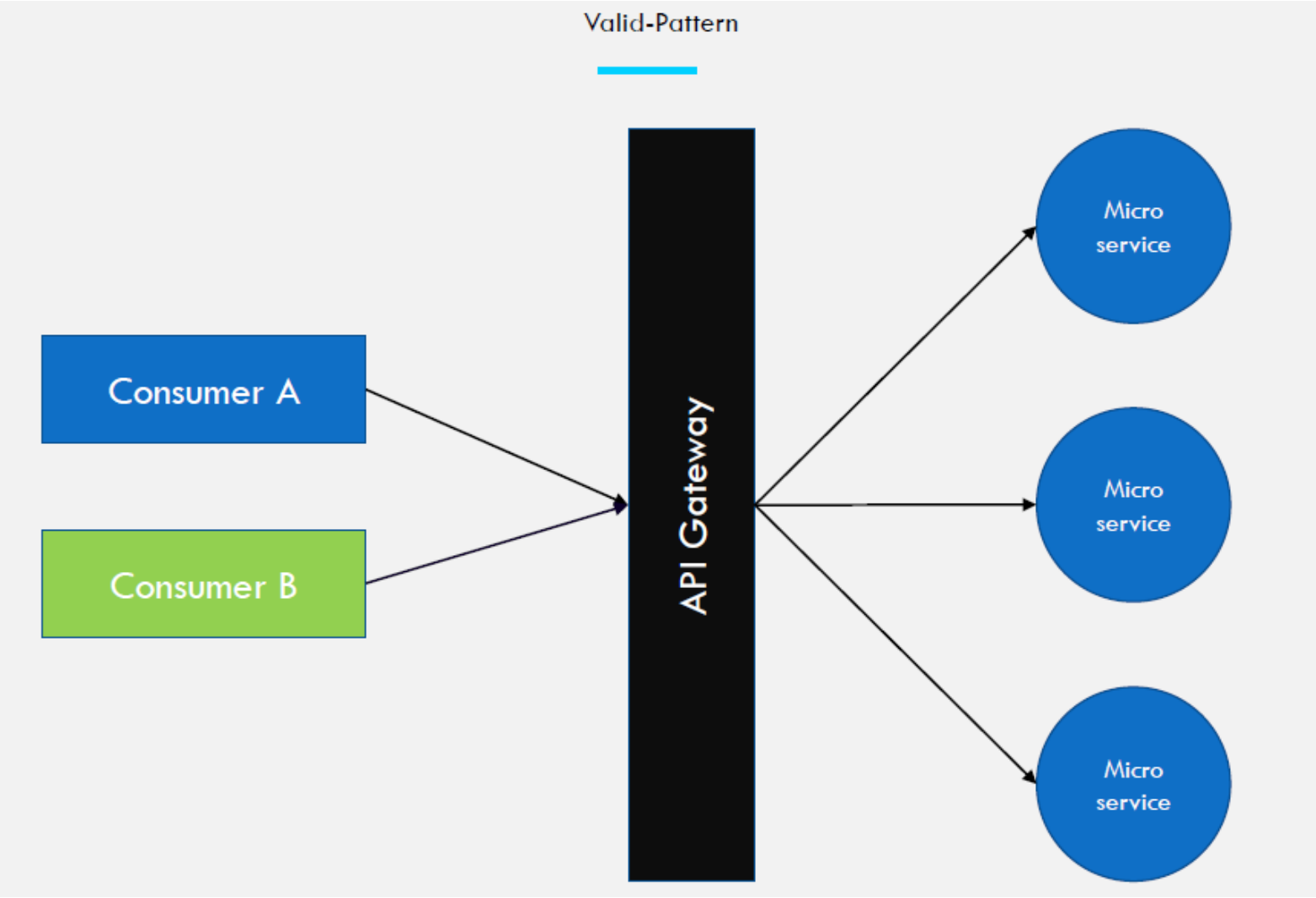
- Server-Side Discovery



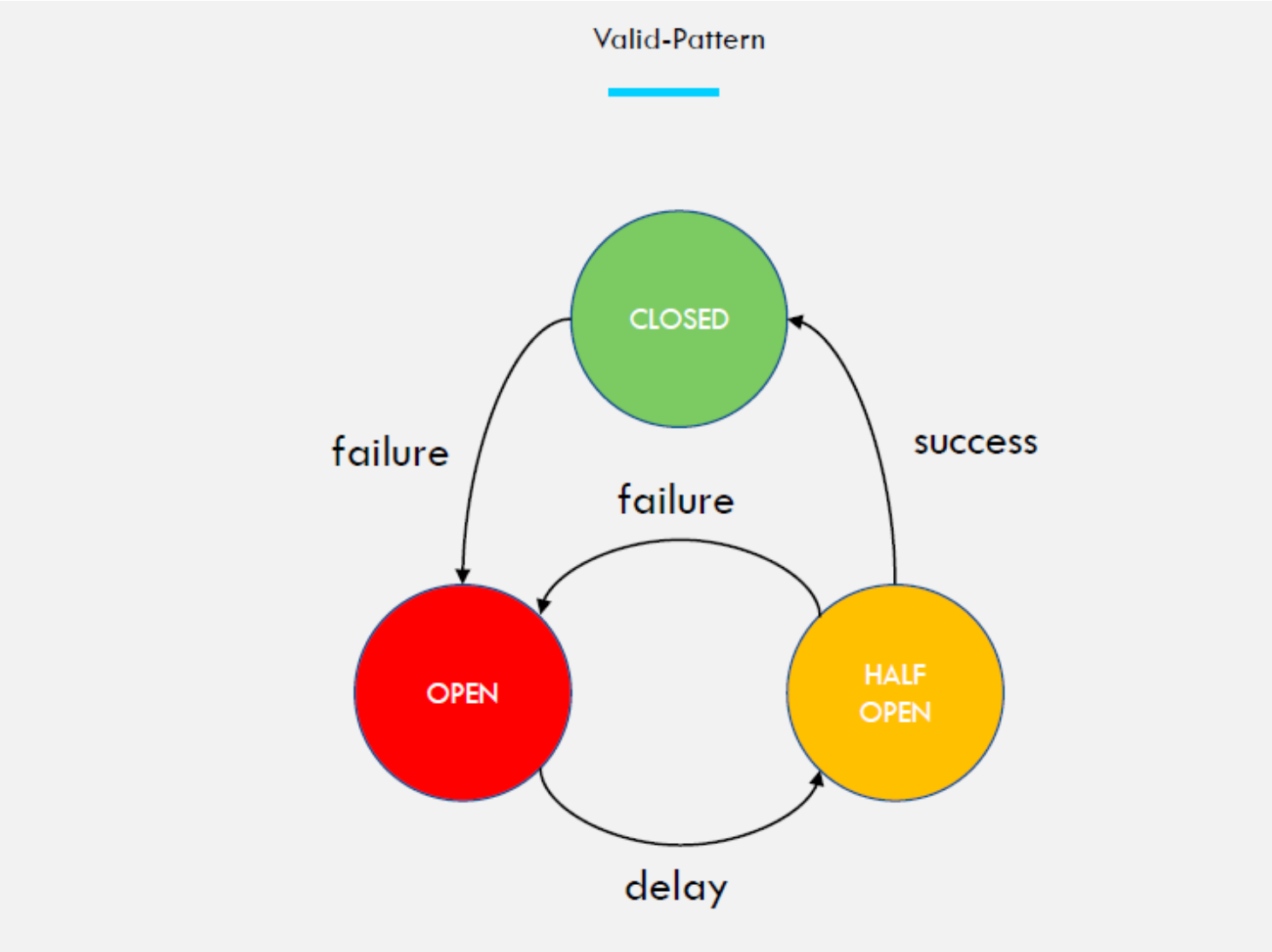
Centralized Configuration



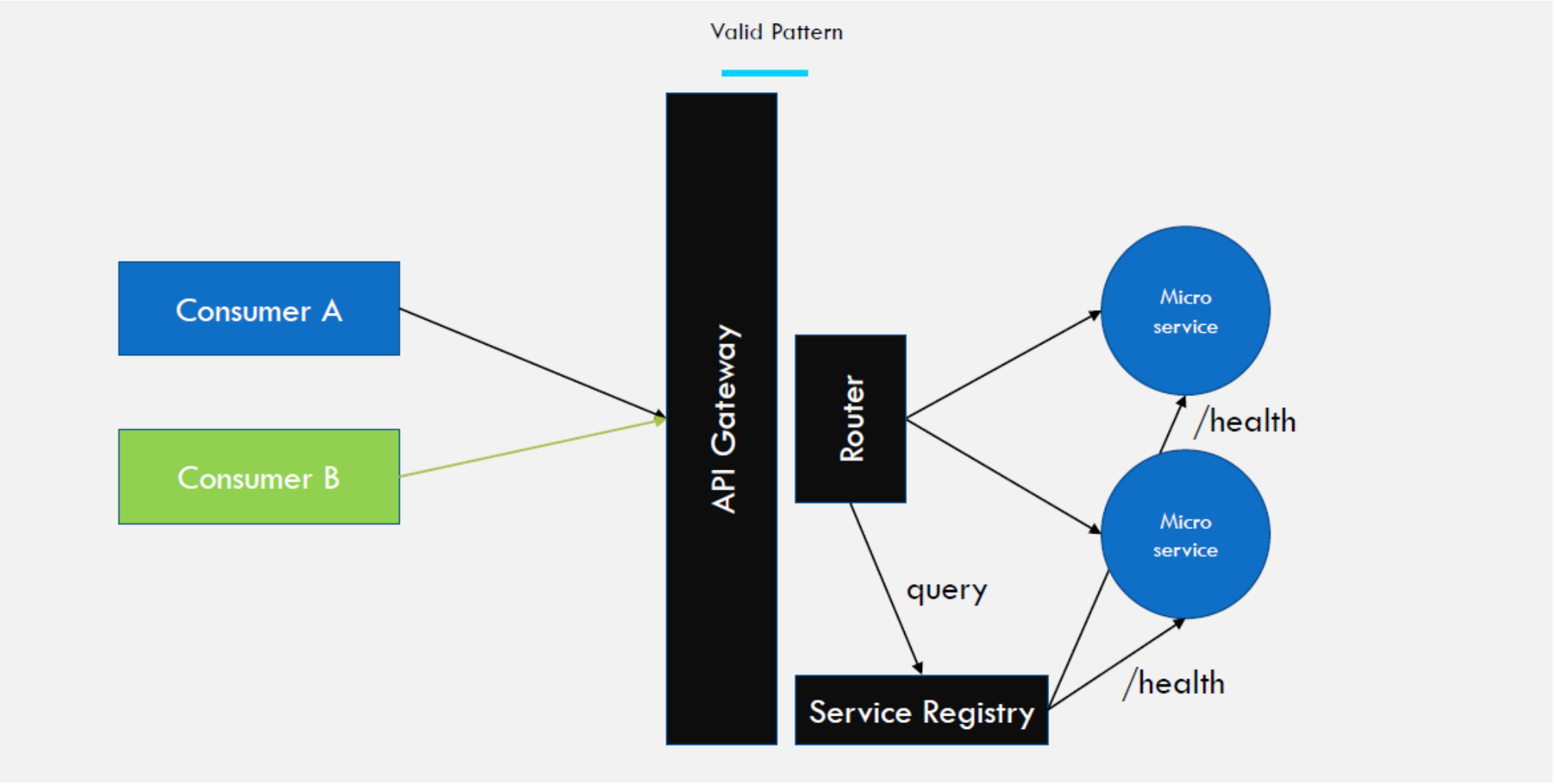
Retry


Valid-Pattern
Consumer A
Consumer B
API Gateway
Micro service
Micro service
Micro service

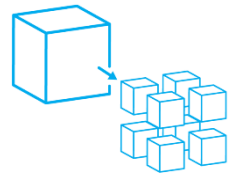
Resilencia: Circuit Breaker



Health Check API

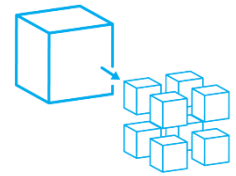


Netflix Zuul

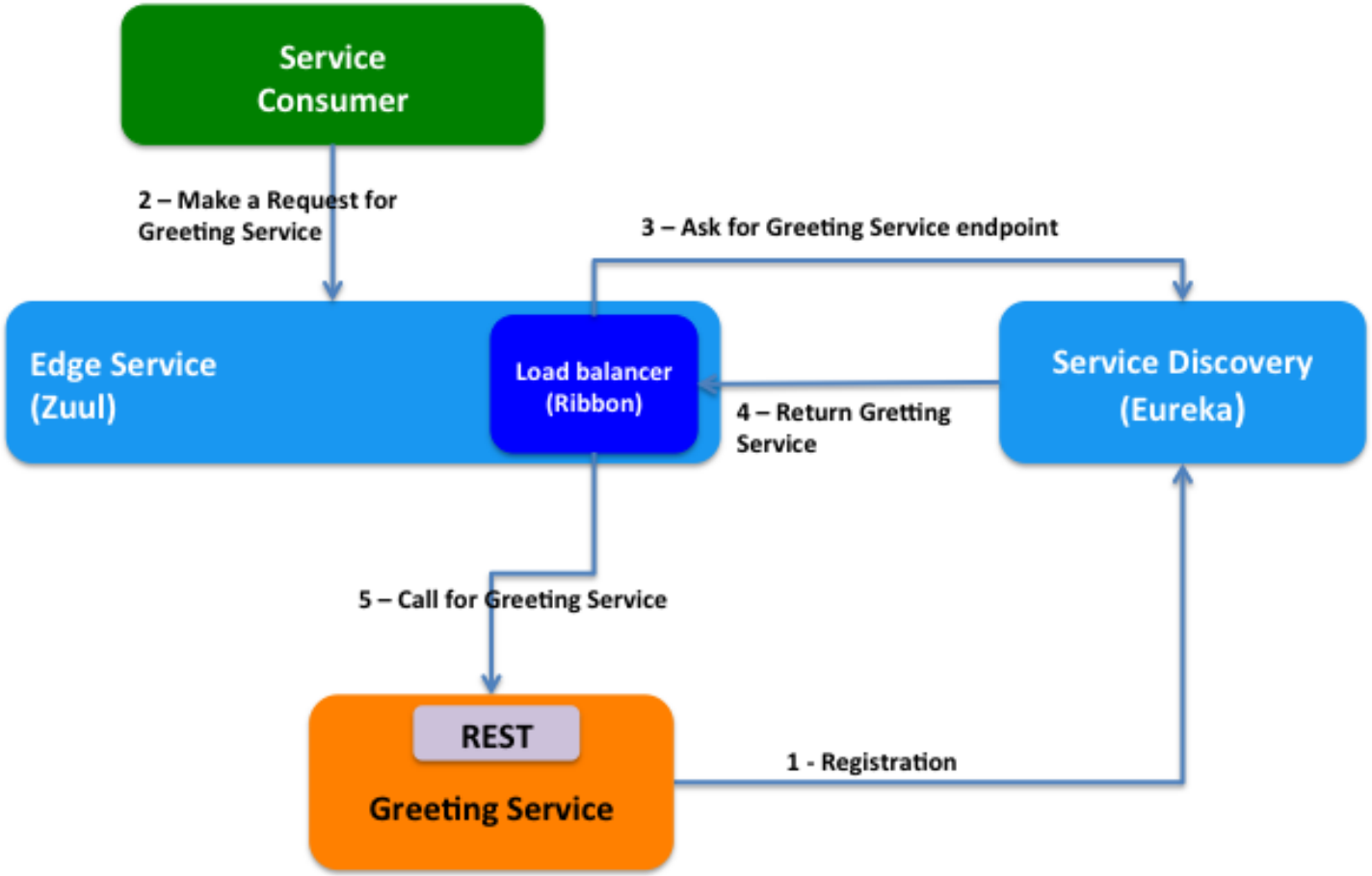


- Zuul, nuestra nueva herramienta, se puede definir como un proxy inverso o edge service que nos va a permitir tanto enrutar y filtrar nuestras peticiones de manera dinámica, como monitorizar y securizar las mismas.
- Este componente actúa como un punto de entrada a nuestros servicios, es decir, se encarga de solicitar una instancia de un microservicio concreto a Eureka y de su enrutamiento hacia el servicio que queramos consumir.

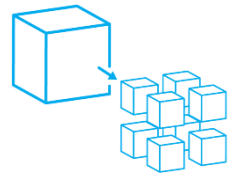
Zuul API Management



- Netflix Zuul es la dependencia en Spring Cloud, que nos permite implementar un API Management



Zuul API Management - Ventajas



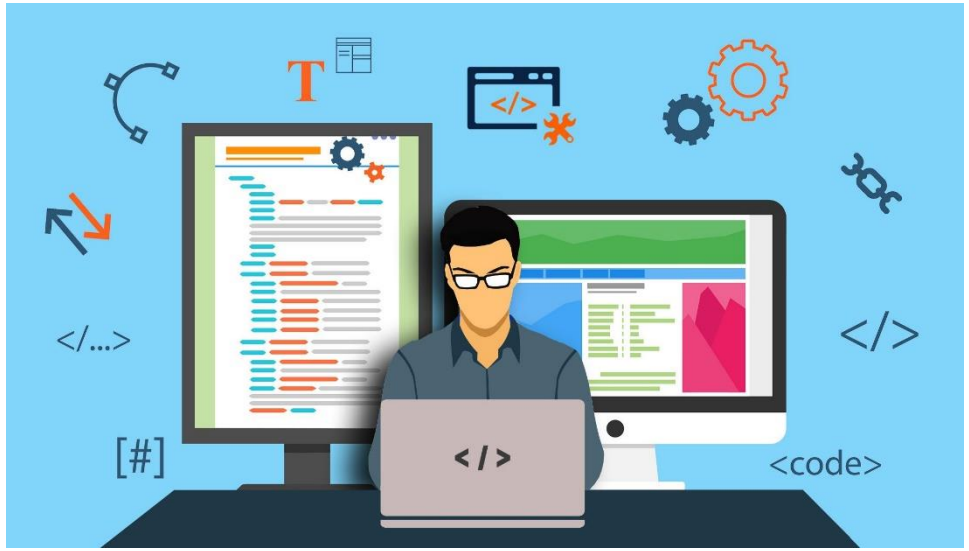
- Dispone de varios filtros enfocados a gestionar diferentes situaciones
- Transforma nuestro sistema en uno más ágil, capaz de reaccionar de manera más rápida y eficaz
- Puede encargarse de gestionar la autenticación de manera general al ser nuestro punto de entrada al ecosistema
- Es capaz de realizar el despliegue de filtros en caliente, de manera que podríamos realizar pruebas sobre nuevas funcionalidades sin parar la aplicación

Preguntas

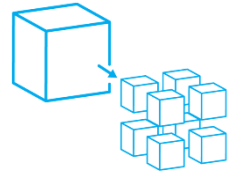
- Alguna pregunta?

Laboratorio

- Lab 03 Interacción entre Microservicios
- Lab 04 Cliente Rest con Netflix Feign
- Lab 05 Balanceo de Carga con Netflix Ribbon
- Lab 06 Balanceo de Carga con Ribbon(Visualizando el Puerto)



Referencias



- https://deku.github.io/microservice-architecture-es/
- https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern