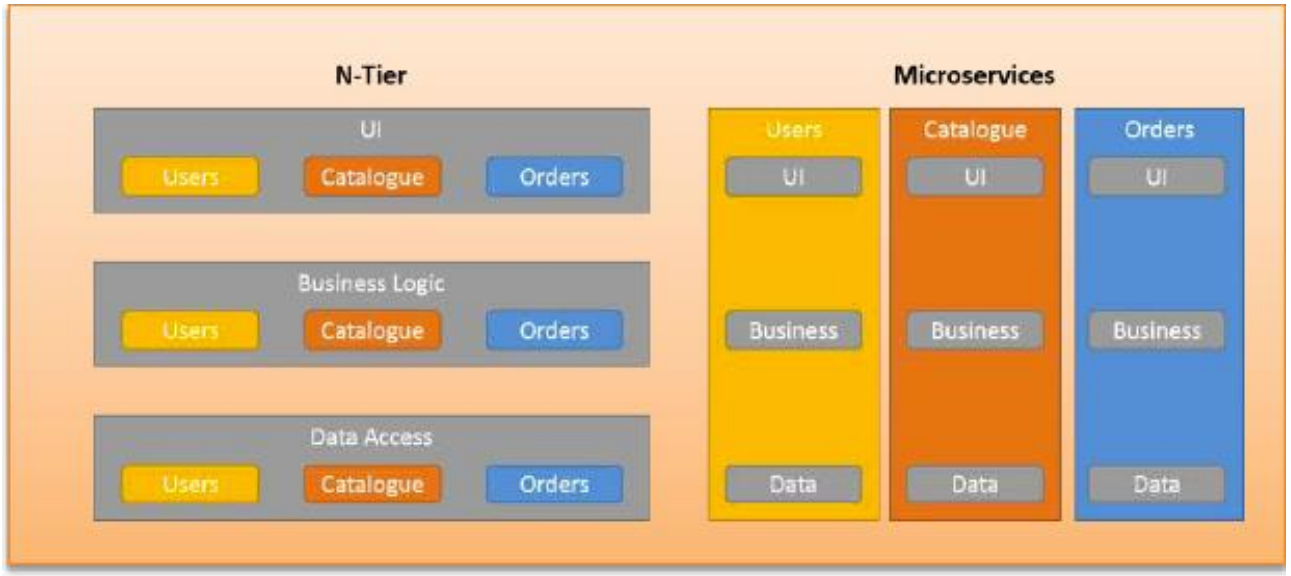
MÓDULO 1

Arquitectura de Microservicios

QUÉ SON MICROSERVICIOS

MicroServicio

- Es una arquitectura de aplicación.
- Su ámbito de diseño, implementación y gestión es la modularización de aplicaciones.
- No es un concepto de integración de aplicaciones.
- Para la integración los microservicios se exponen en protocolos ligeros (ej. API REST/JSON).



POR QUÉ USAR MICROSERVICIOS

- Los servicios en sí son muy simples de construir, pues se centran en hacer solamente una cosa bien, de forma que son fáciles de probar y se puede asegurar mayor calidad.

- Cada servicio podría construirse con las tecnologías y herramientas más adecuadas, permitiendo "Polyglot Programming" (las aplicaciones se deben escribir en una mezcla de lenguajes para explotar sus mejores características).

- Múltiples equipos pueden trabajar independientemente. Esto fomenta "continuous delivery" debido a que permite actualizaciones frecuentes mientras el resto del sistema se mantiene estable.

- Si un servicio deja de funcionar, solo afectará las partes que dependen directamente de él (si las hay). El resto operará normalmente.

MIROSERVICIOS /EVOLUCIÓN

Monolítica: Alto acoplamiento, cualquier cambio afecta a la totalidad de la aplicación, CI/CD una tarea casi imposible.

SOA: Menor acoplamiento, permitía desarrollar código en partes más pequeñas, pero todo debe estar comunicado y estrictamente desarrollado para que encaje con el resto del desarrollo.

Microservicios: Desacoplamiento total, permite desarrollar pequeños servicios de manera totalmente independiente (Agnósticos y Poliglotas). Su exposición de servicios es dada a través de la exposición de API´S.

CARACTERÍSTICAS COMUNES DE LOS MICROSERVICIOS

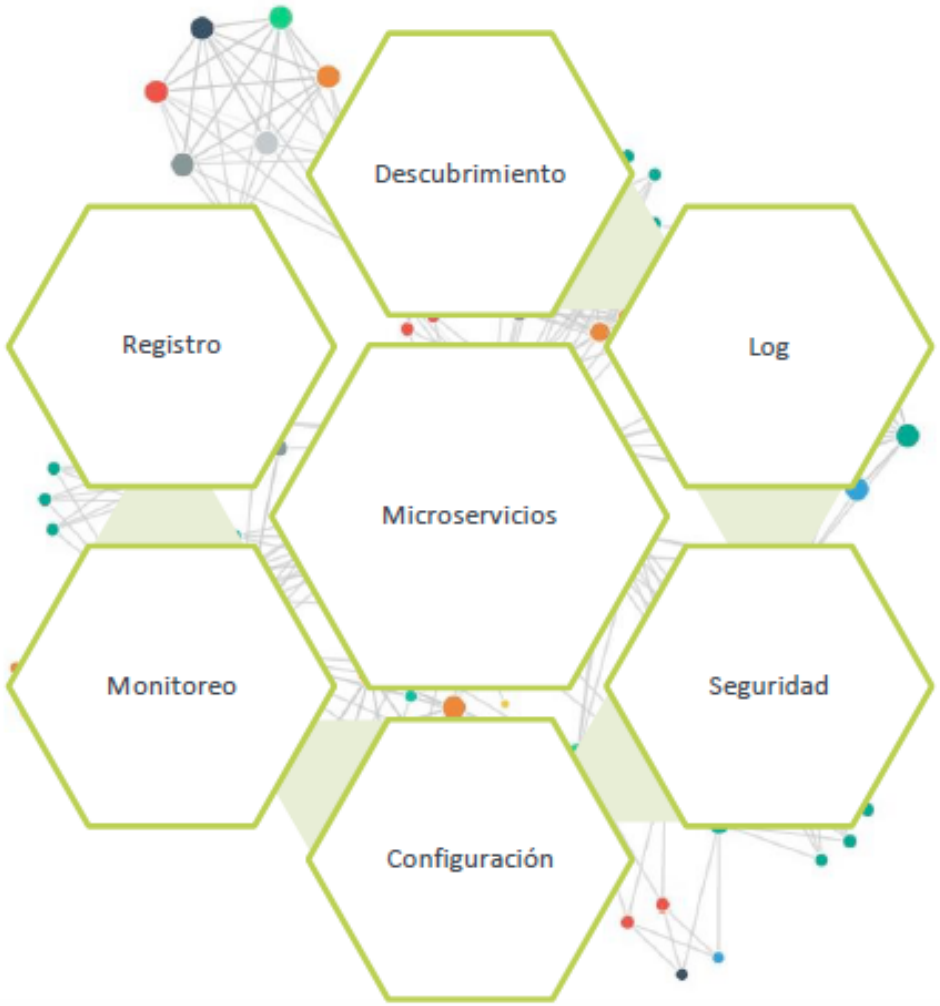
- Características de su software: Pueden ser descompuestos en diferentes partes independientes.
- Características de su organización: La manera en la que están organizados supone un contraste con el entorno monolítico.

CARACTERÍSTICAS COMUNES DE LOS MICROSERVICIOS

- Características de su arquitectura: Cada módulo es independiente ya que cada uno de ellos cuenta con su propia base de datos.

- Características de sus sistemas de aviso y actuación: Al estar varios servicios comunicados necesitamos contar con sistemas de aviso y actuación por si se registrara algún fallo de estos servicios.

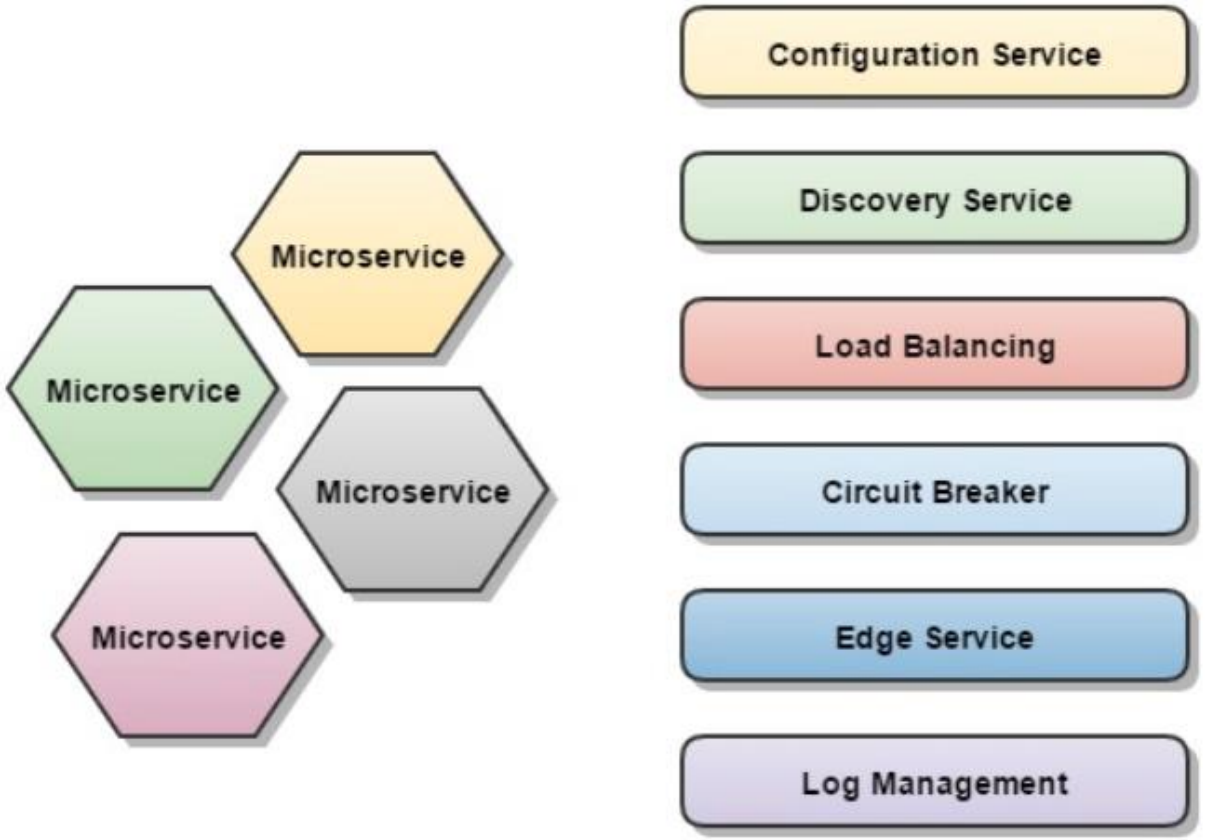
ARQUITECTURA Y COMPONENTES



Componentes Arquitectura de Microservicios

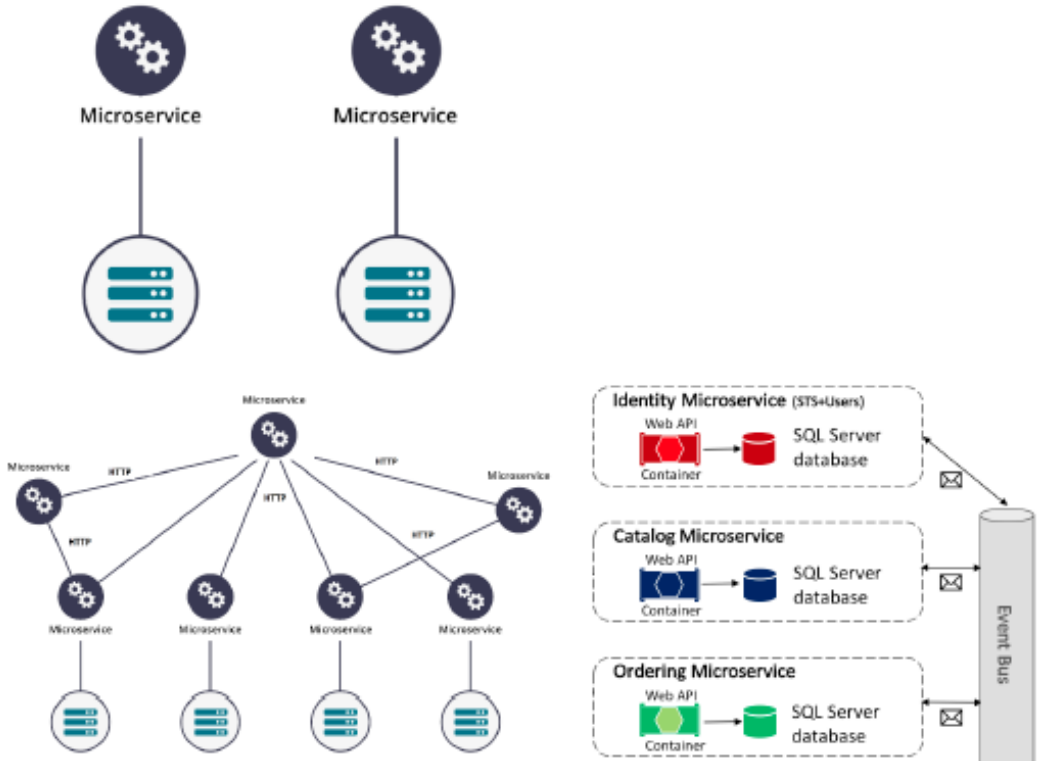
MODELO DE REFERENCIA

Estos son los componentes que vamos a necesitar en una arquitectura de microservicios:



ARQUITECTURA / ACCESO A DATOS Y COMUNICACIÓN

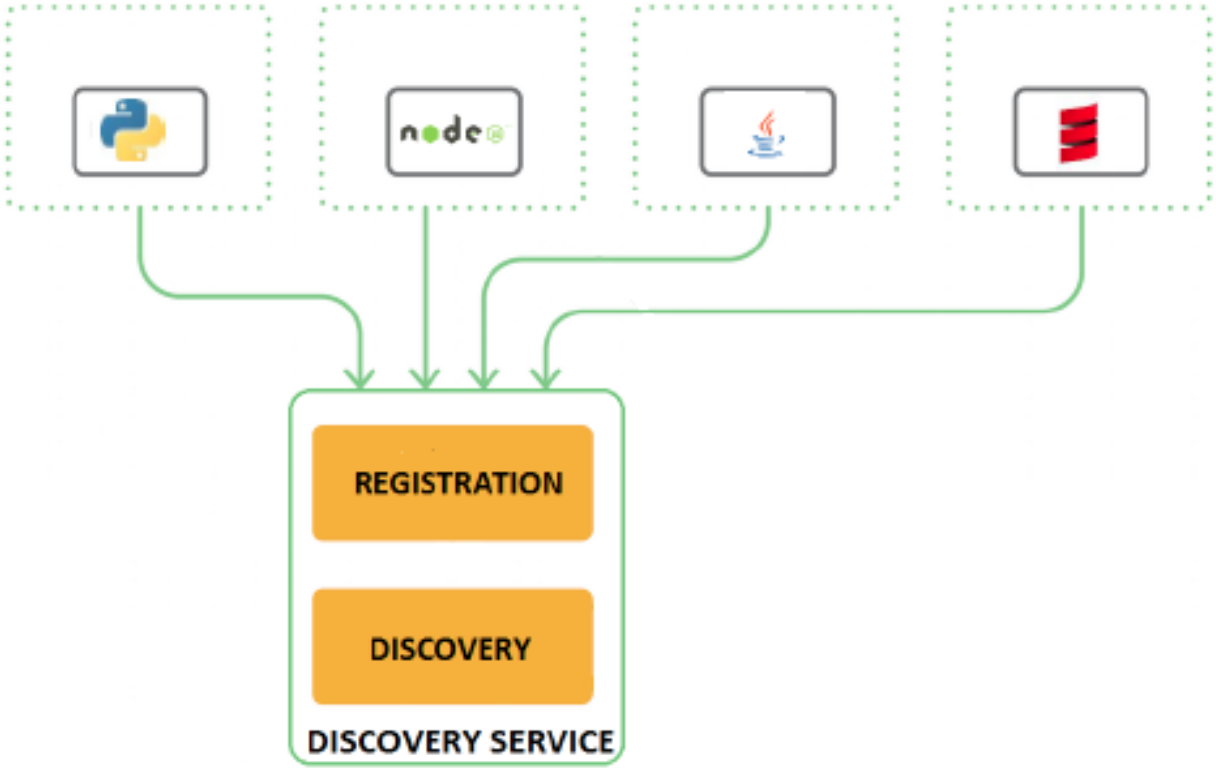
- De la misma manera que en una aplicación monolítica los microservicios pueden acceder a datos a través de capas de persistencia, drivers o configuración especifica para cada caso.
- Exponen funcionalidades a través de API's rest.
- Se comunican entre si con protocolos http o mensajería.



Comunicación y Acceso a Datos

COMPONENTES / DESCUBRIMIENTO Y ORQUESTACIÓN

- En la medida que nuestras aplicaciones crecen se hace necesario que despleguemos mayor número de microservicios o instancias de manera automática.
- Asignación de puerto e ip de manera delegada.
- Al ser asignado aleatoriamente, se hace necesario que descubramos y registremos los nuevos servicios creados automáticamente.

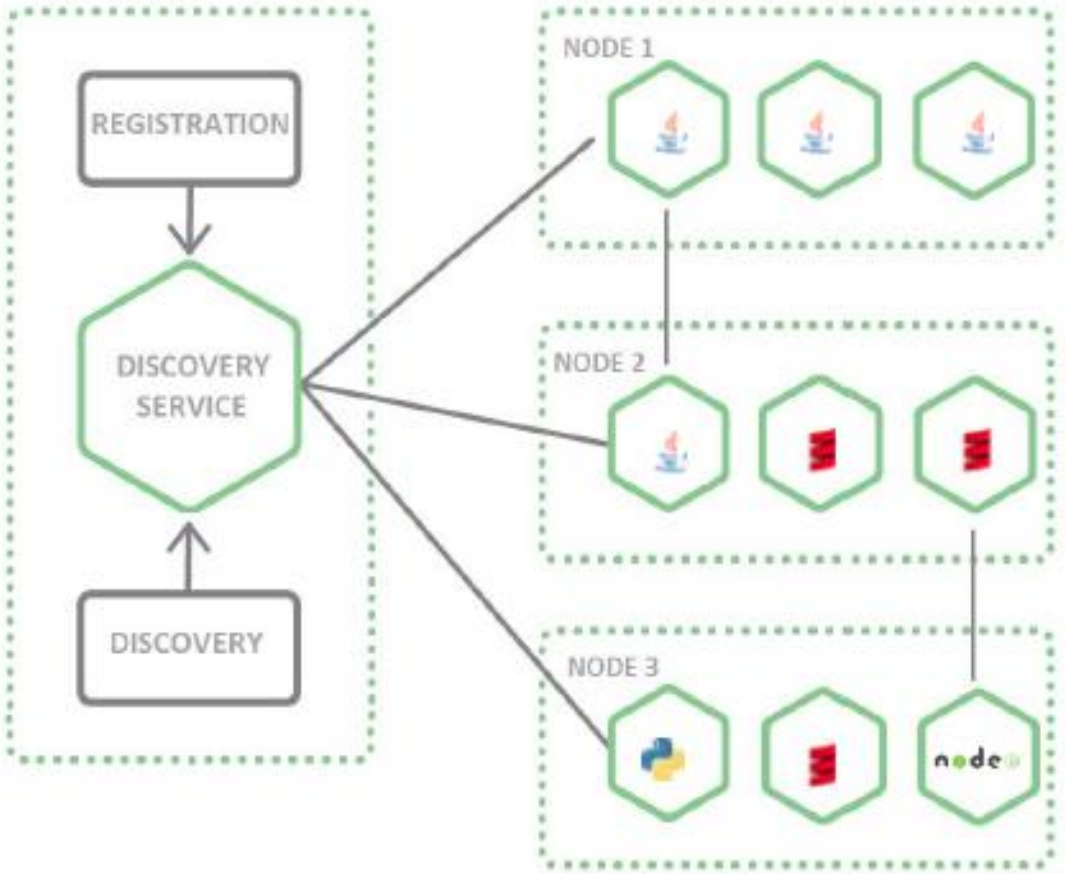


Servicio de Descubrimiento

COMPONENTES / DESCUBRIMIENTO Y ORQUESTACIÓN

Los procesos que componen el servicio de descubrimiento son:

- Registro
- Descubrimiento

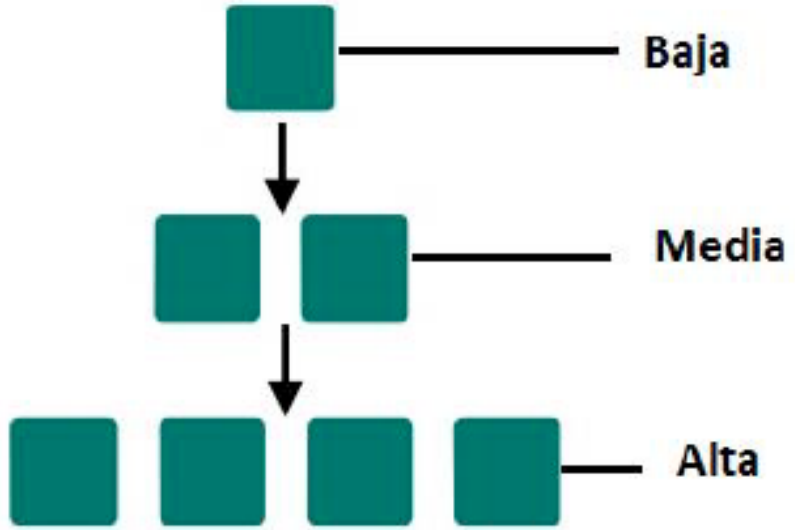


Servicio de Descubrimiento

COMPONENTES / DESCUBRIMIENTO Y ORQUESTACIÓN

Escalado en función de la demanda, como puede ser con base al consumo de memoria, cpu, peticiones, etc. los beneficios a resaltar son:

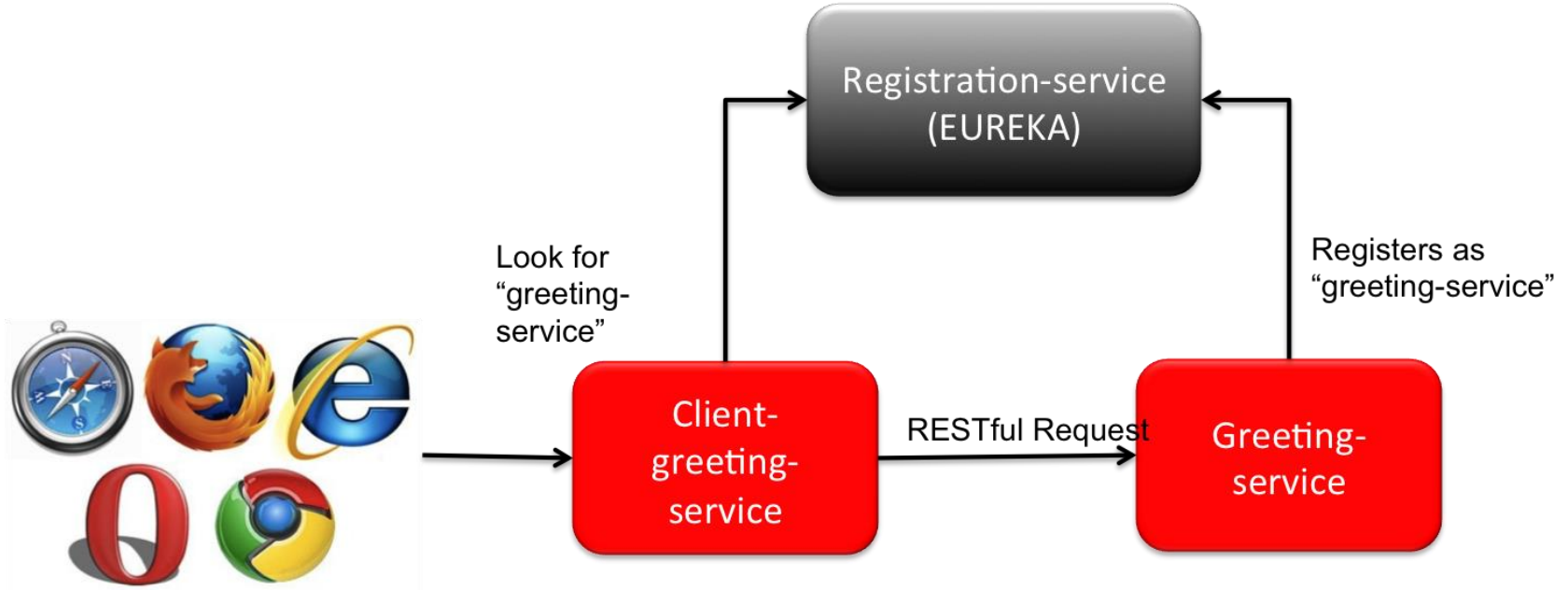
- Alta tolerancia a fallos: La recuperación se realiza de forma automática.
- Escalado horizontal con funcionamiento elástico.
- Despliegues basados en estrategias predefinidas.
- Abstracción de la capa de microservicios.
- Conocer el estado de nuestro ecosistema.
- Soporte Multi-región.



Escalado Elástico

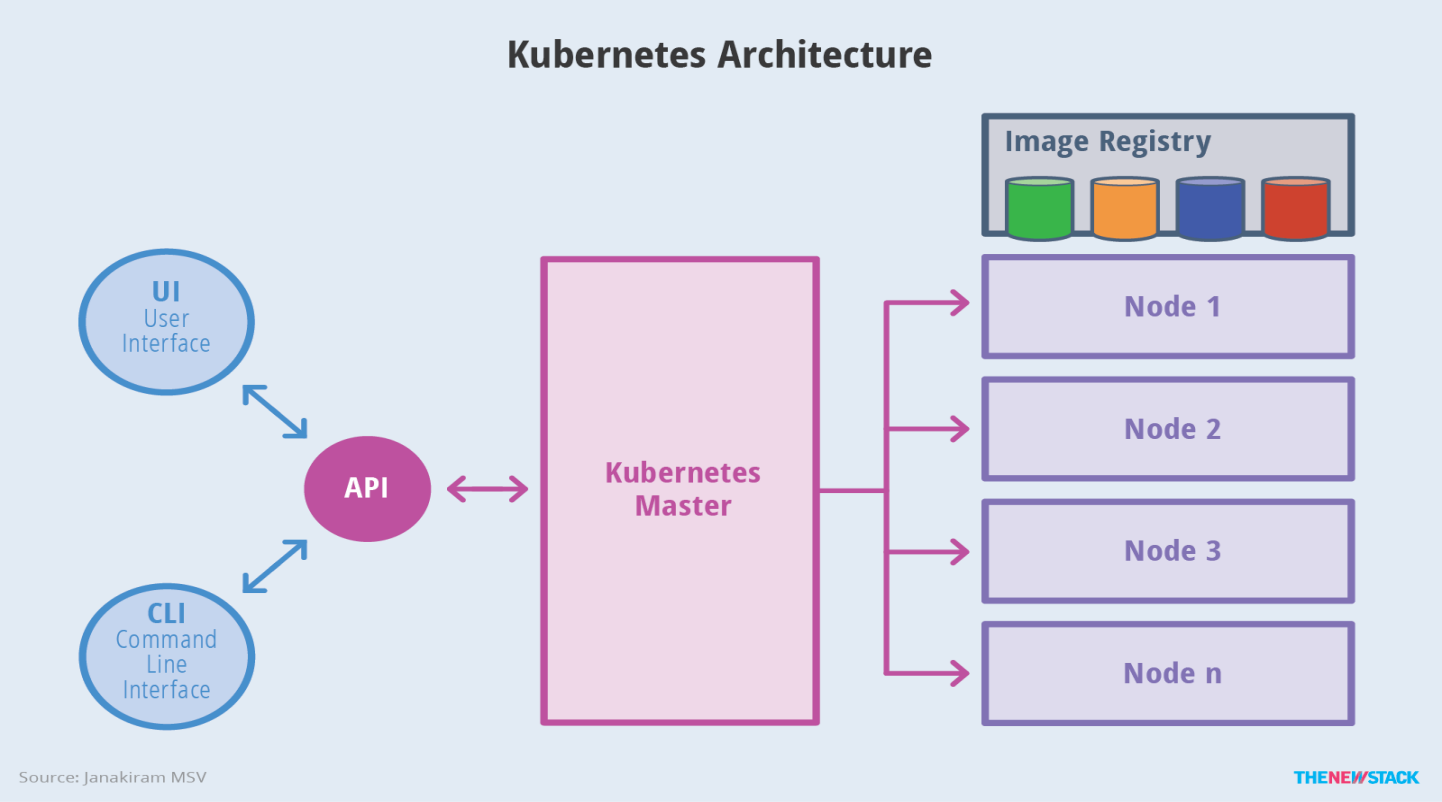
COMPONENTES / DESCUBRIMIENTO Y ORQUESTACIÓN

- **Herramientas disponibles para el servicio de Descubrimiento:**
  - Eureka (Spring Cloud Netflix)
  - Zookeper
  - Etcd
  - Consul



COMPONENTES / DESCUBRIMIENTO Y ORQUESTACIÓN

- **Herramientas para la orquestación:**
  - Ribbon + Eureka (Spring Cloud Netflix)
  - Kubernetes
  - Docker swarm



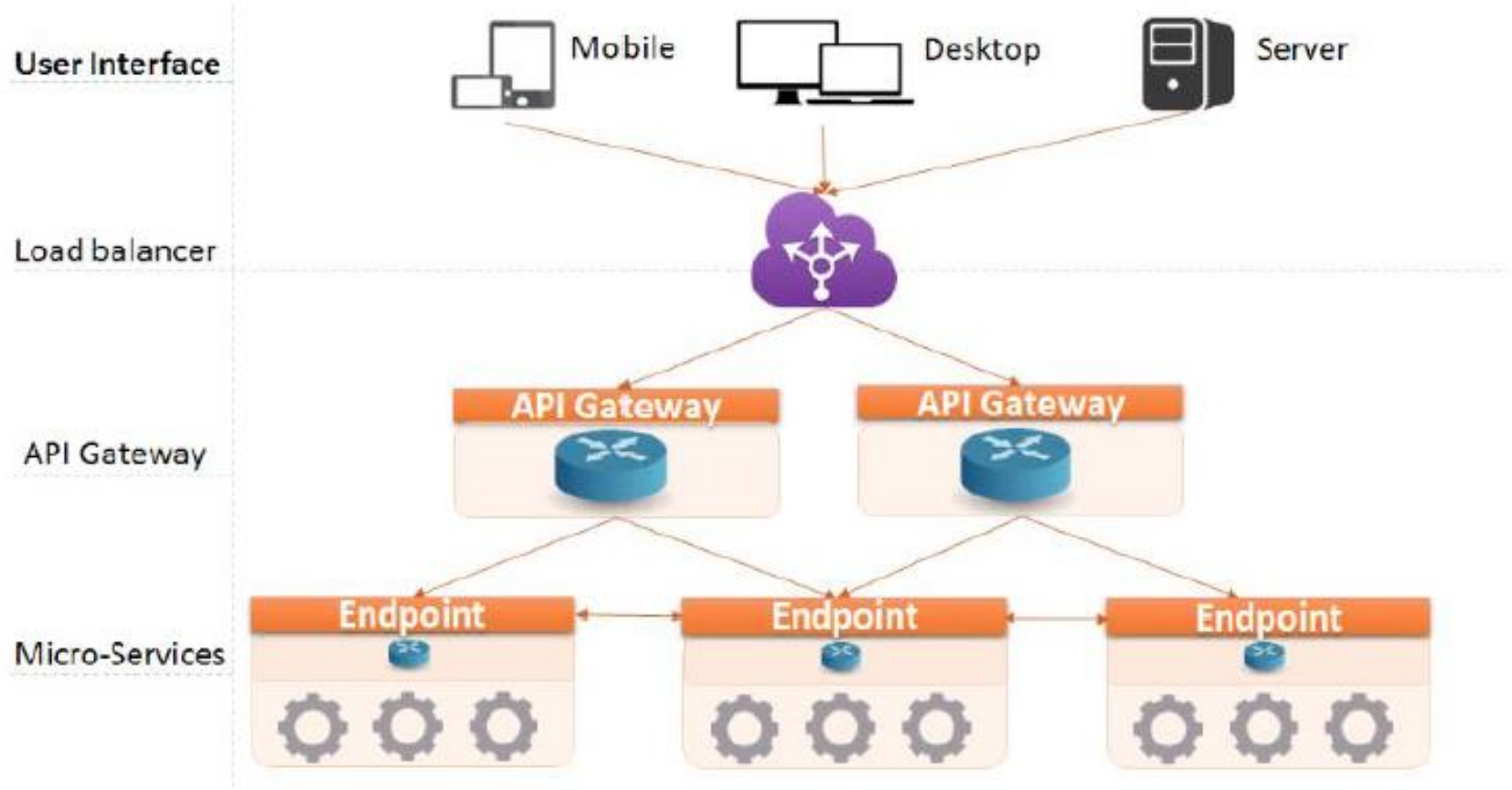
COMPONENTES / ENRUTAMIENTO

Es importante resaltar componentes de un **Api Gateway**:

- Converge las solicitudes a través de un único punto de entrada (Api Gateway).
- Mapeo global de peticiones (Url's).
- Abstracción de los microservicios.
- Filtros dinámicos, redireccionamiento con base en localización de las solicitudes.

- **Herramientas:**
  - Zuul (Spring Cloud Netflix)
  - Haproxy
  - Api Gateway (AWS)

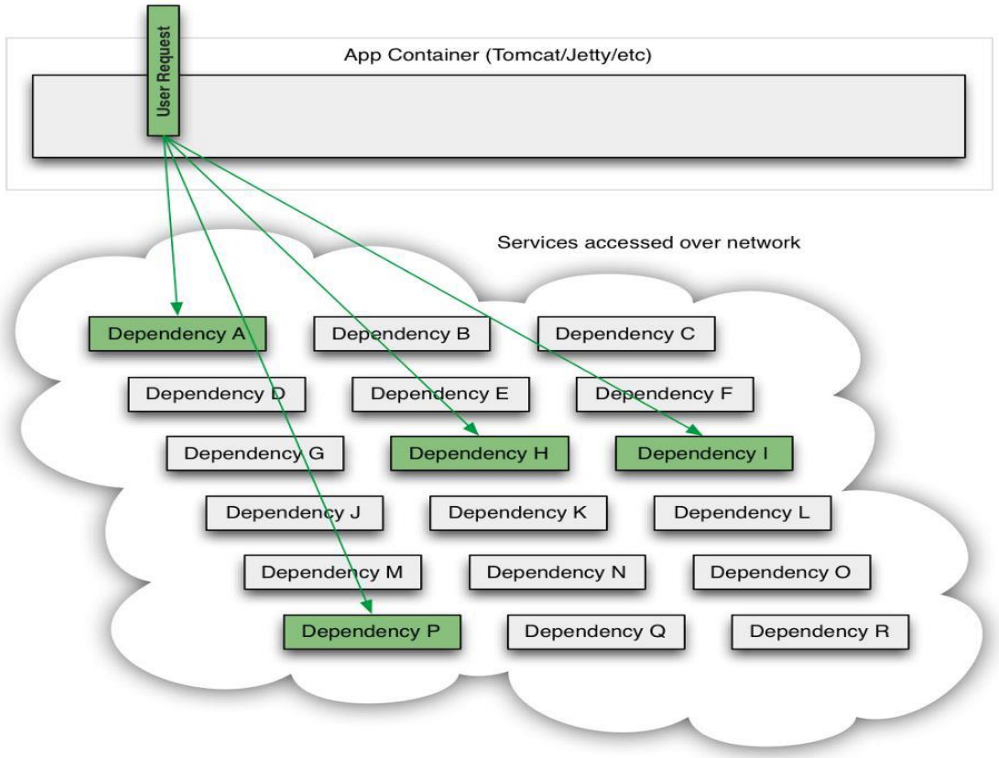
COMPONENTES / ENRUTAMIENTO

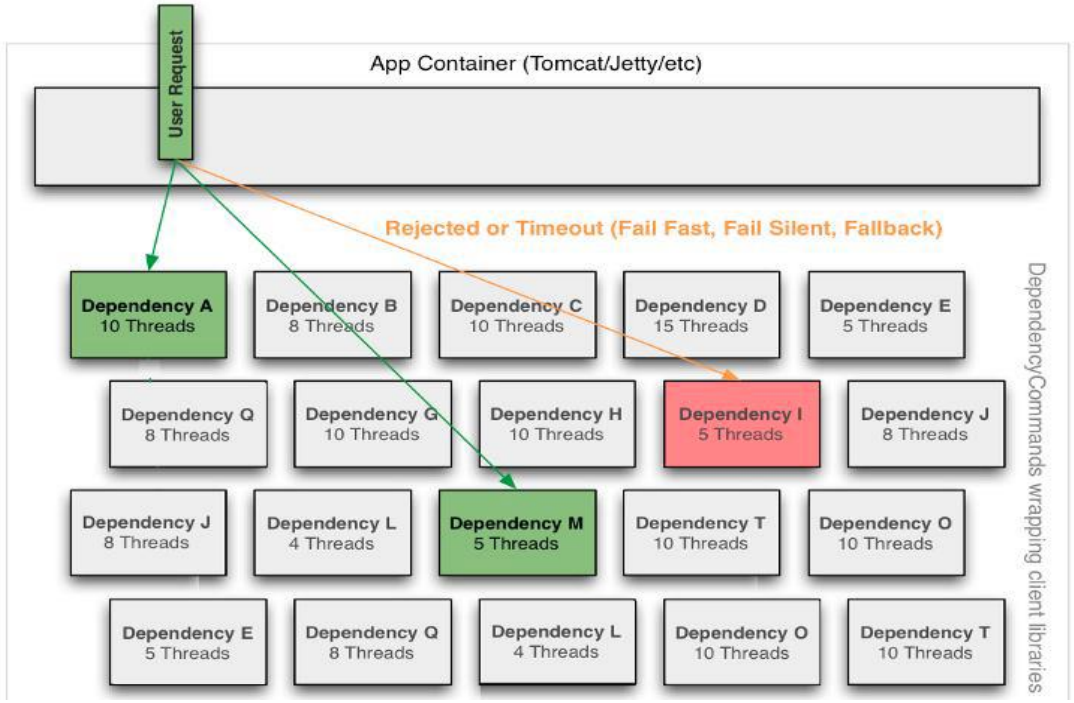


API Gateway

COMPONENTES / GESTIÓN DE ERRORES

En los sistemas distribuidos un fallo o demora en la respuesta puede causar una caída total de la aplicación. Por eso es necesario detectar las fallos y encapsular o contener la propagación.

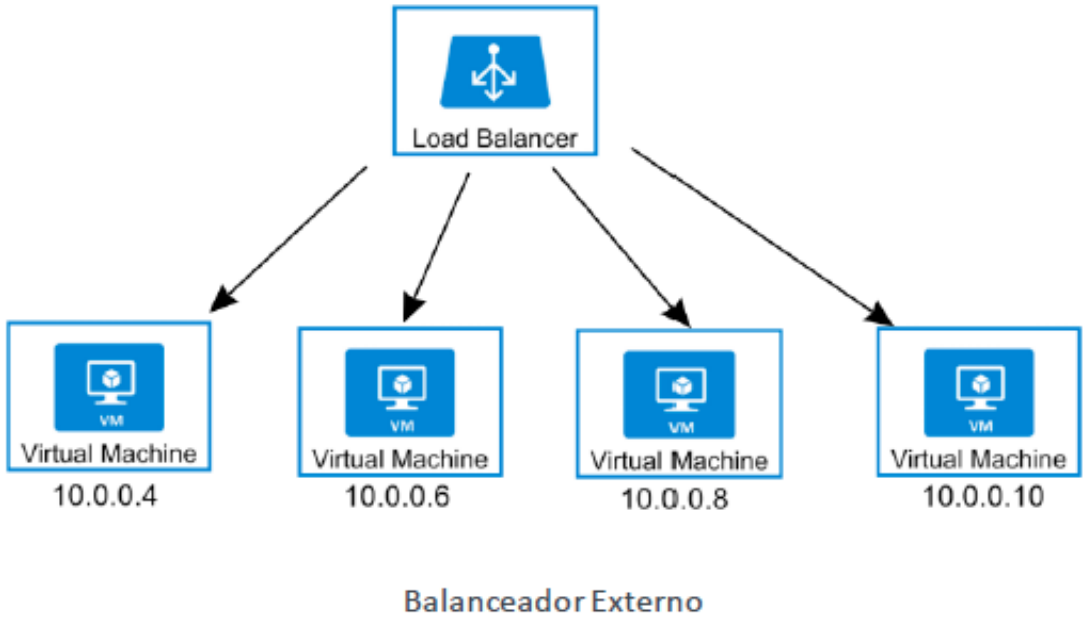




COMPONENTES / BALANCEADOR DE CARGA

- Es recomendado contar con un balanceador de carga externo que nos permita mejorar la fiabilidad de nuestra aplicación, incrementando además la disponibilidad y tolerancia a fallos.

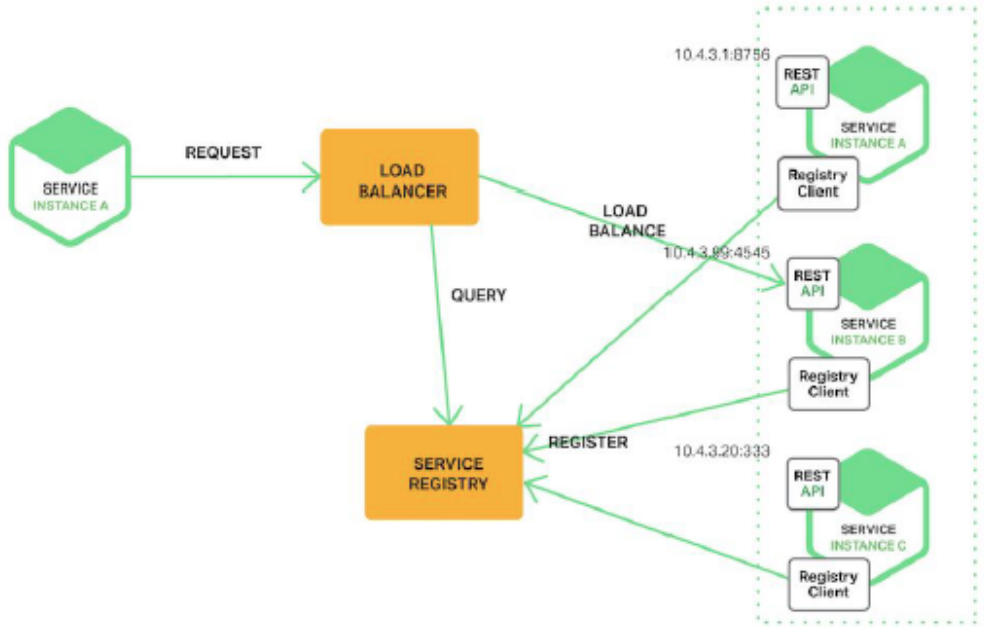
- Algunos ejemplos son:
  - AWS (ELB).
  - F5.



Balanceador en el Cliente

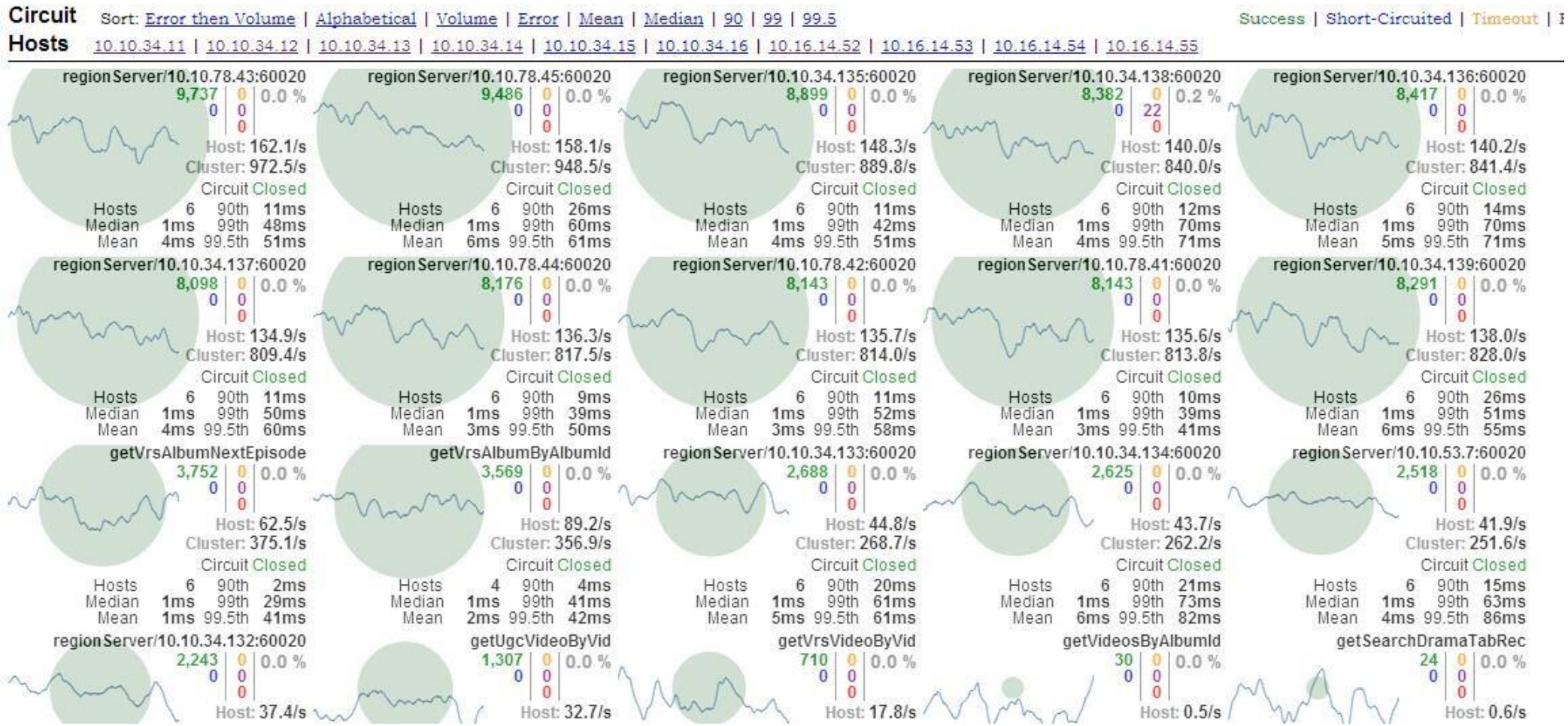
- Balancear la carga entre las distintas instancias de un microservicio. haciéndolo más efectivo a la hora de manejar las peticiones.
- Configuración de políticas de balanceo.
- Integración con el servicio de descubrimiento.

- **Ejemplos:**
  - Ribbon (Spring Cloud Netflix).



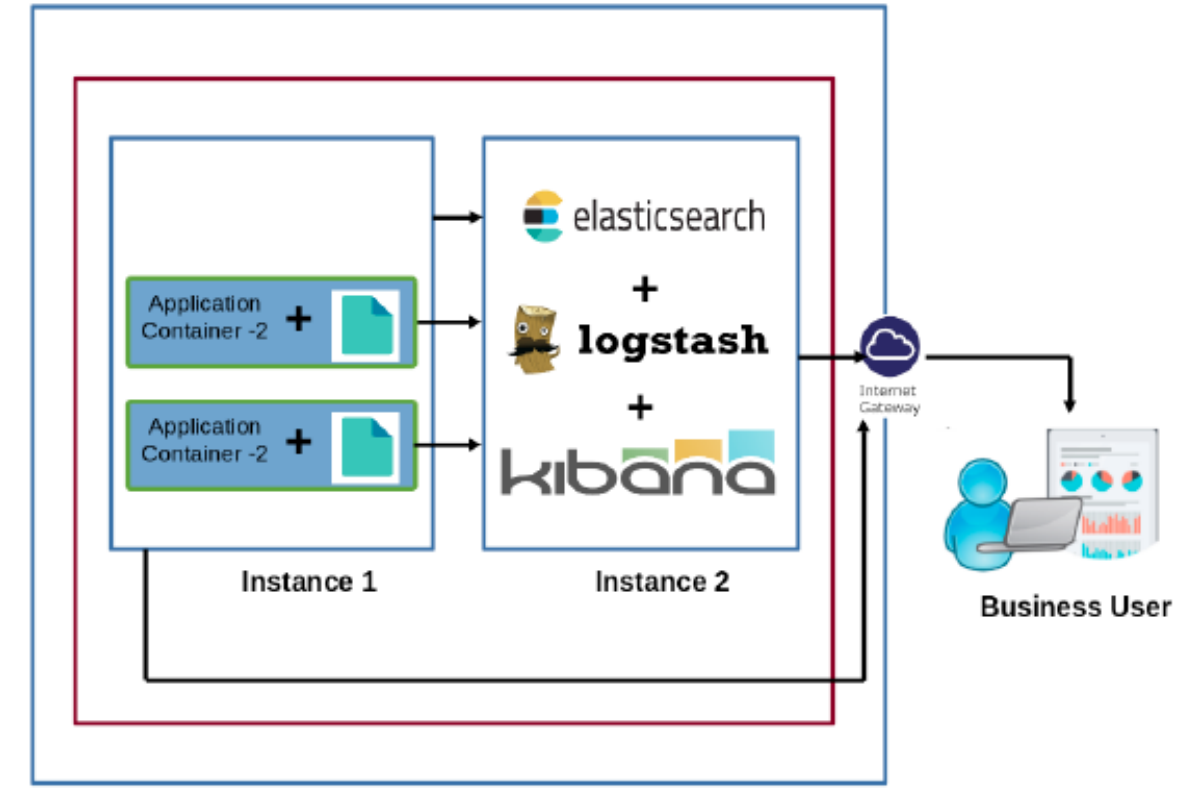
Balanceador en el Cliente

COMPONENTES / MONITORIZACIÓN Y LOGS



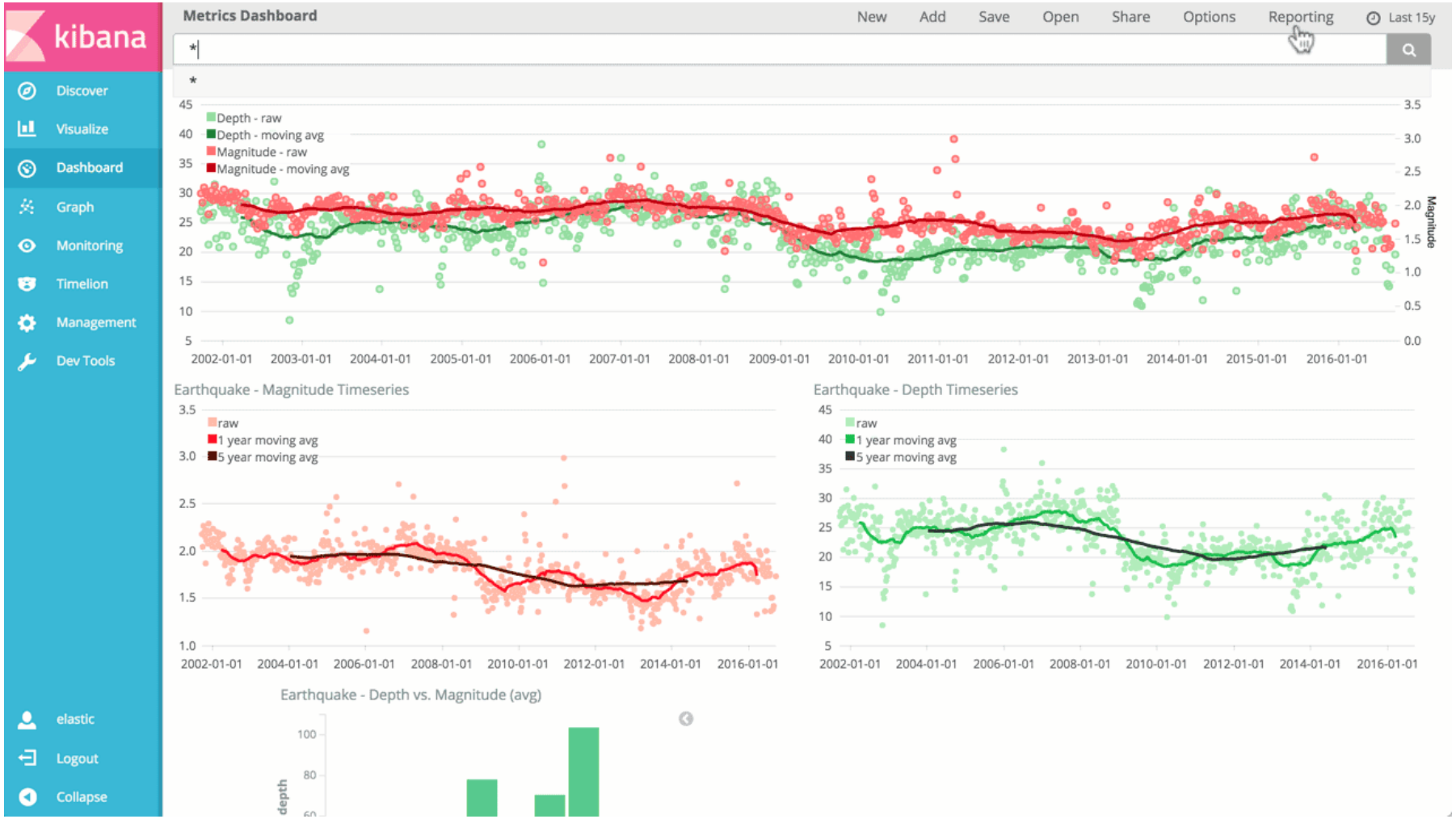
COMPONENTES / MONITORIZACIÓN Y LOGS

- La centralización y explotación de logs juega un papel importante de lo contrario seria inmanejable la administración de los mismos.
- **Ejemplo:**
  - ELK



ELK Explotación de Información

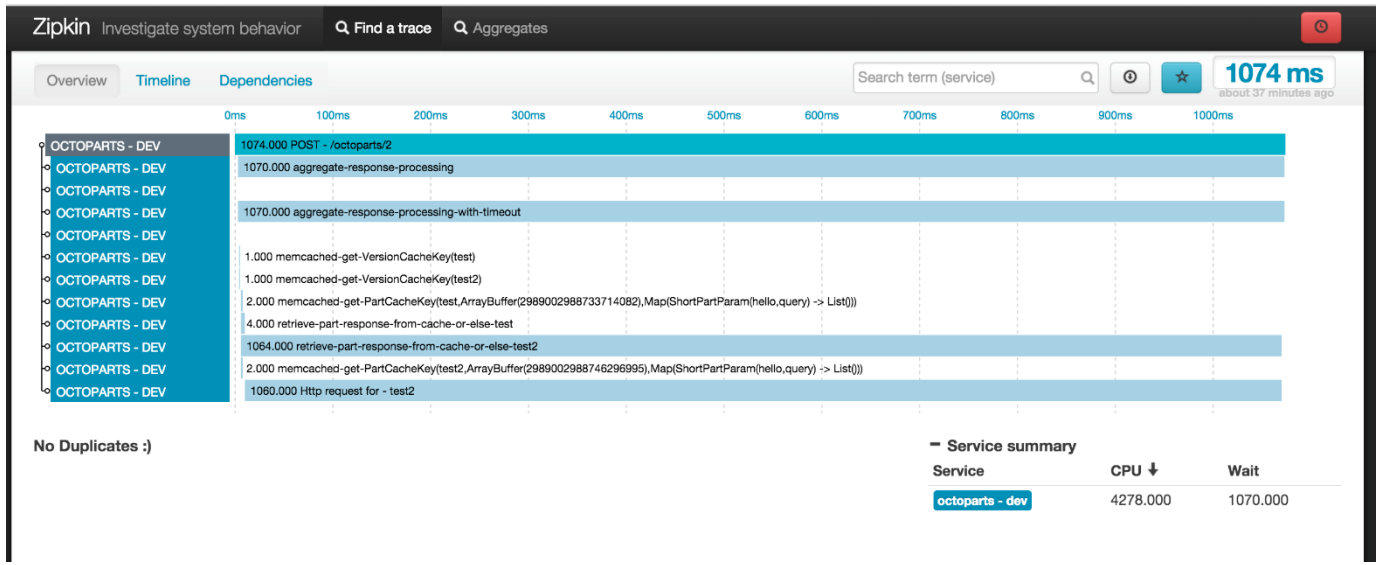
COMPONENTES / MONITORIZACIÓN Y LOGS



COMPONENTES / TRAZABILIDAD DE PETICIONES

- En un sistema distribuido debemos contar con la capacidad de registrar en nuestros logs las trazas a nivel de petición.

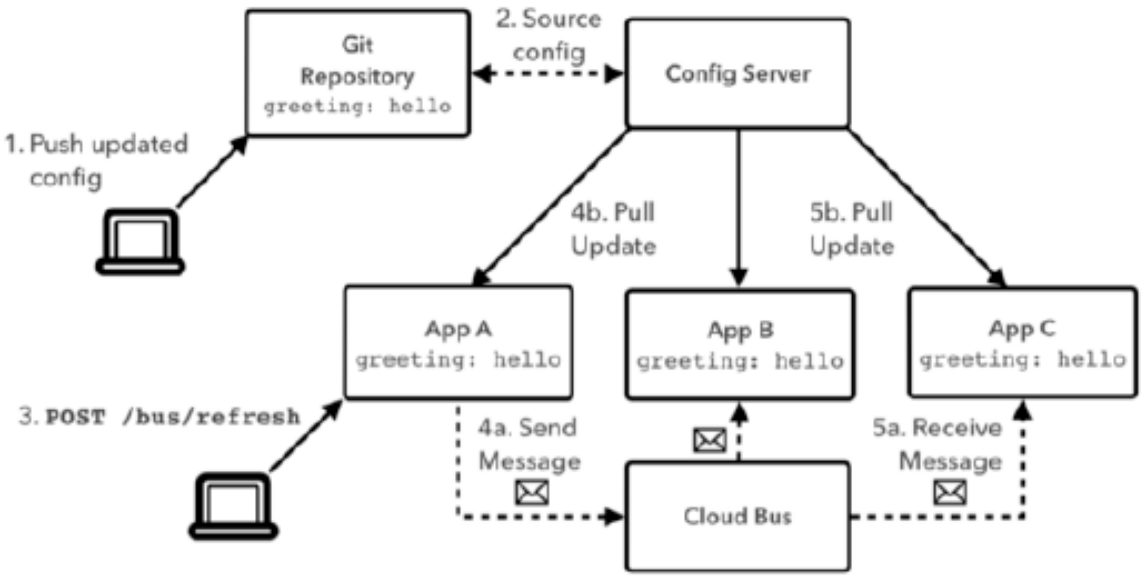
- Ejemplos:
  - Spring Sleuth + Zipkin



COMPONENTES / SERVICIOS DE CONFIGURACIÓN

- Configuración centralizada de la información.
- Algunos proveen la funcionalidad sincroniza los archivos de configuración con repositorios de git y captura de información en caliente por parte de los microservicios.

- **Ejemplo:**
  - Spring cloud config
  - Archadius
  - Consul
  - Zookeeper



Servicio de Configuración

CICLO DE VIDA CI/CD

- La creación de estos microservicios es apoyada por herramientas de iaas y paas que complementan el proceso de automatización.
- Dependiendo de la infraestructura y plataforma podrá variar el ciclo de CI/CD.
- Es aquí donde docker juega un papel importante ya que nos garantiza el despliegue de nuestros microservicios en cualquier plataforma.
- Utilizar herramientas como docker no es necesario para una arquitectura de microservicios, pero facilita mucho la labor.

CICLO DE VIDA CI/CD