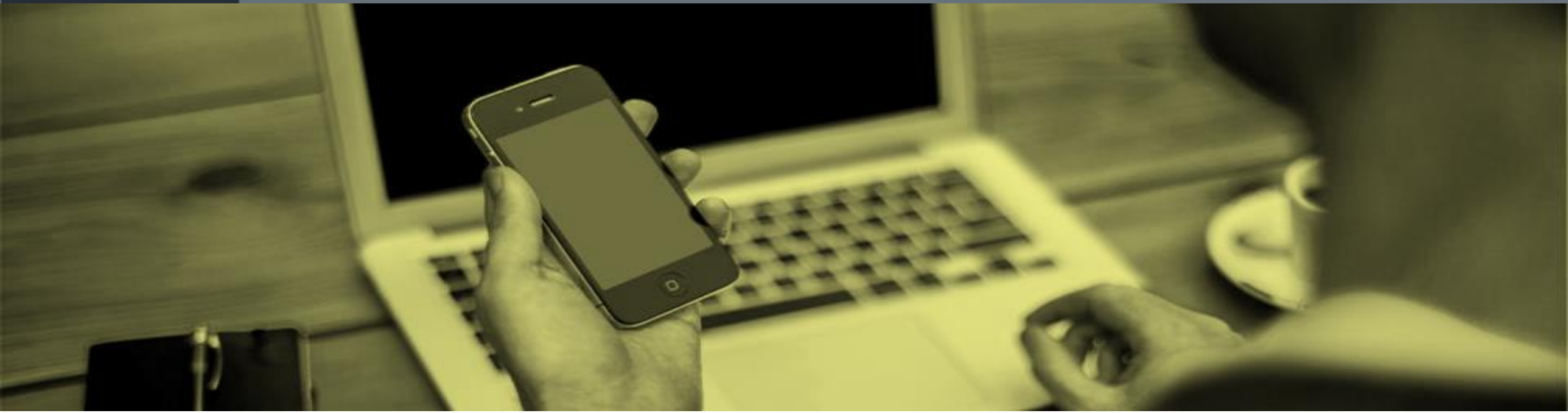
Orquestación de Microservicios

Introducción a arquitecturas de desarrollo modernas basadas en sistemas distribuidos.

TODO 1

Expositor



Nombres: Jaider Anillo Garcia

Perfil: Jaider es Ingeniero de Sistemas de la Fundación Universitaria Tecnológico Comfenalco, con un Diplomado en Gerencia de Proyectos Informáticos y un Magister en Gerencia de Proyectos de la Universidad Viña del Mar.

TODO 1

Agenda

01 Microservicios

- Definición
- Evolución
- Porque Usarlos?
- Patrones

02 Arquitectura

- Componentes de arquitectura
- Acceso a datos y comunicación

03 Componentes

- Componentes necesarios para una correcta implementación de aplicaciones basadas en sistemas distribuidos

04 Ciclo de Vida

- Procesos CI/CD para microservicios

05 Recursos

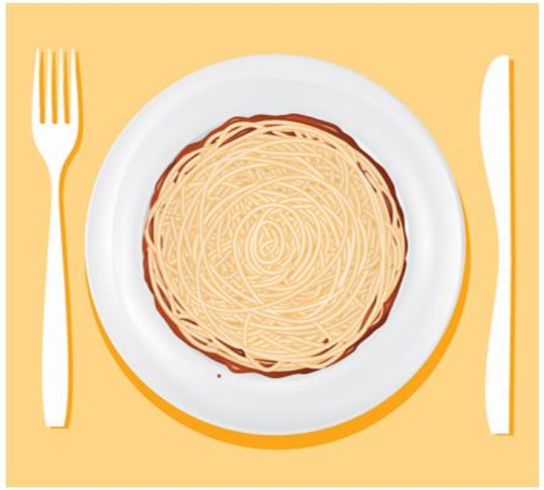
- Recursos que complementan y facilitan el desarrollo de aplicaciones bajo sistemas distribuidos

Microservicios / Definición

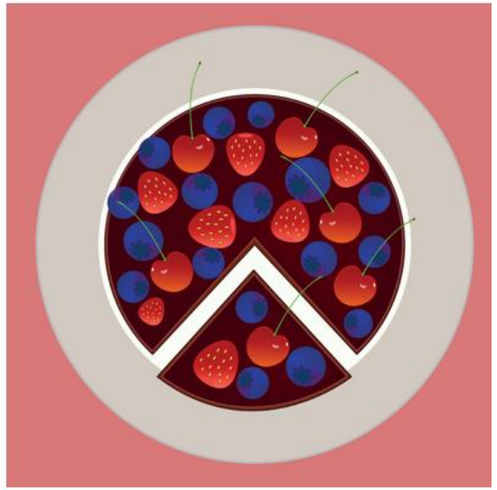
- **Sistemas Distribuidos:** La computación distribuida o informática en malla (grid) es un modelo para resolver problemas de computación masiva utilizando un gran número de ordenadores organizados en clústeres incrustados en una infraestructura de telecomunicaciones distribuida.

- **Microservicios:** La Arquitectura de microservicios es una aproximación para el desarrollo software que consiste en construir una aplicación como un conjunto de pequeños servicios, los cuales se ejecutan en su propio proceso y se comunican con mecanismos ligeros (normalmente una API de recursos HTTP). Cada servicio se encarga de implementar una funcionalidad completa del negocio.

Microservicios / Evolución



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

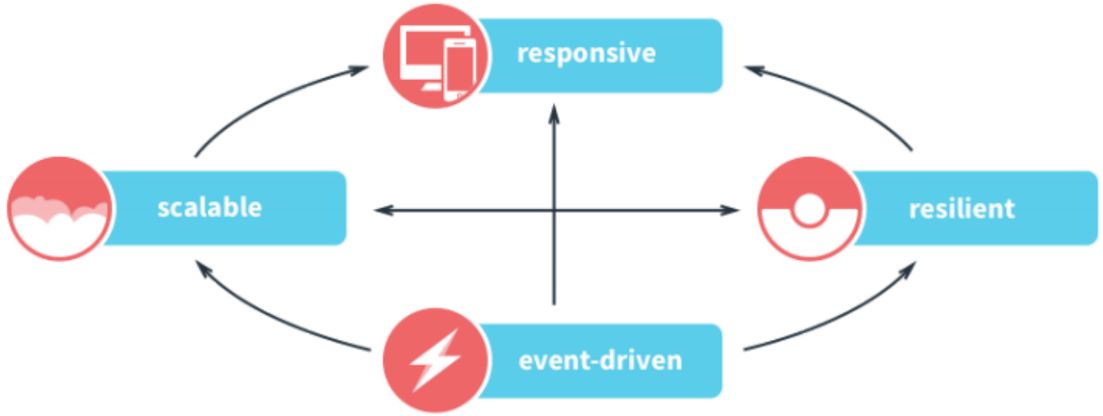


SOA: Menor acoplamiento, permitía desarrollar código en partes más pequeñas, pero todo debe estar comunicado y estrictamente desarrollado para que encaje con el resto del desarrollo.

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.

Microservicios / Porque usarlos?

- Ciclo de vida CI/CD más rápido.
- Reducir tiempo de salida al mercado.
- Protocolos de red para exponer servicios y comunicarse.
- Integración con otros sistemas
- Servicios de alta disponibilidad y rendimiento.
- Centralizar funcionalidad.
- Mejor control de fallos y testeabilidad.



Reactive Manifesto

Microservicios / Patrones

- Existen muchos patrones relacionados con la arquitectura de microservicios, mas sin embargo la correcta selección e implementación de los mismos puede conllevar al éxito o fracaso.

- Empresas lideres a nivel mundial en implementación de microservicios tales como: Amazon, Netflix e Ebay, han liberado herramientas que facilitan el uso de esta arquitectura.

- Algunos de los patrones serán revisados dentro del desarrollo de esta sesión, mas sin embargo veamos algunos de los mas importantes a la hora de desarrollar microservicios.

Microservicios / Patrones

- **DDD (Domain Driven Design)**

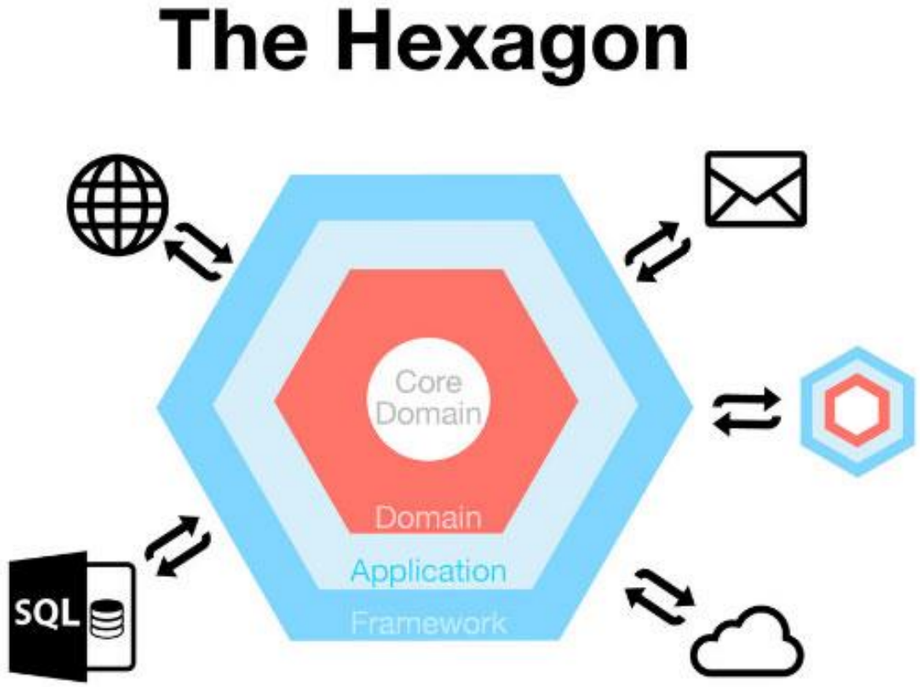
  Domain driven design es un enfoque para el desarrollo de software definido por Eric Evans en su libro *"Domain-driven design: Tackling Complexity in the Heart of Software"*, que se centra en un modelo rico, expresivo y en constante evolución para resolver problemas del dominio de una forma semántica.

  Aspectos:
  Lenguaje Ubicuo

  Capas Conceptuales:
  Interface de Usuario
  Aplicación
  Dominio
  Infraestructura

Arquitectura Hexagonal

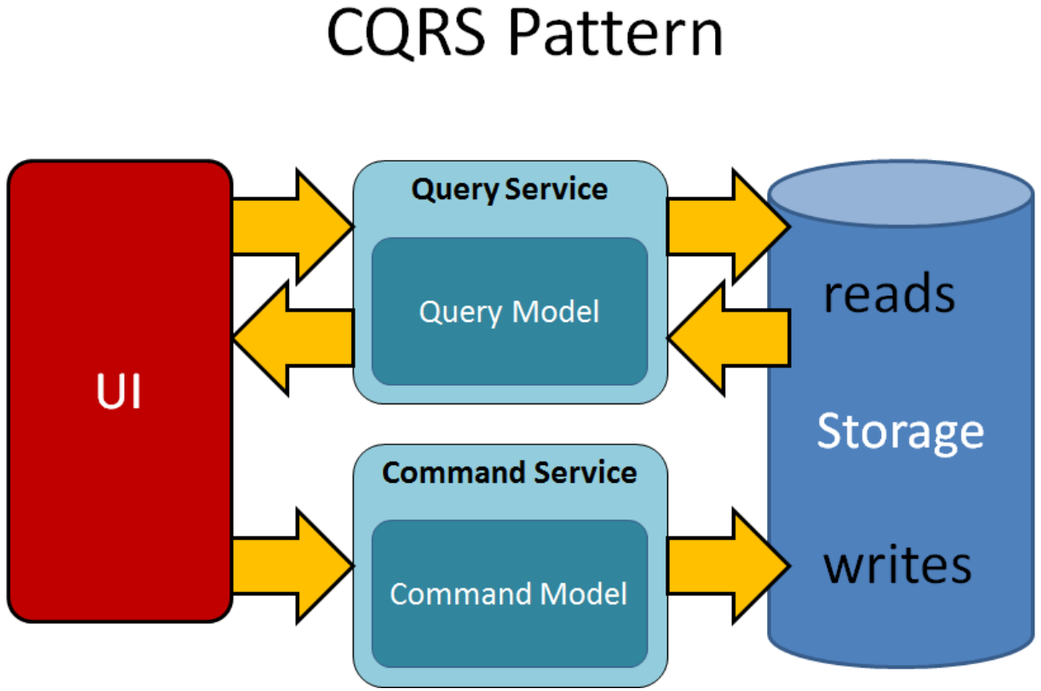
- Llamada también ports and adapters.
- Objetivo: El principal objetivo de la arquitectura hexagonal es separar nuestra aplicación en distintas capas que tienen su propia responsabilidad.



Arquitectura Hexagonal

CQRS (Command Query Responsibility Segregation)

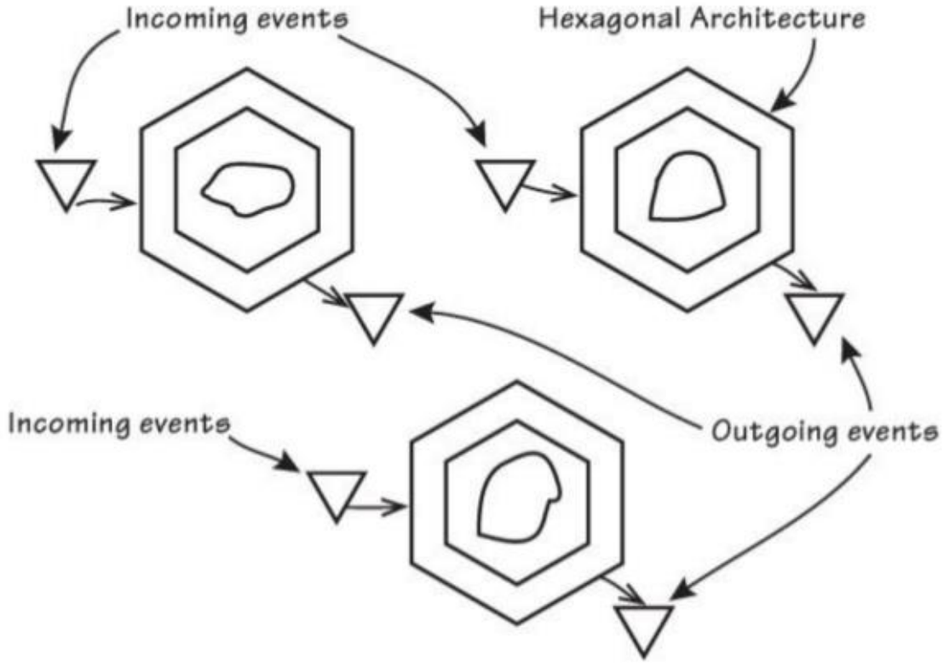
- En esta arquitectura contamos contamos con Comando y Consultas, con el fin de desagregar funcionalidades.
- Permite disponer de bases de datos independientes adecuadas para las necesidades.
- Permite escalar por separado las funcionalidades.



Detalle Patrón CQRS

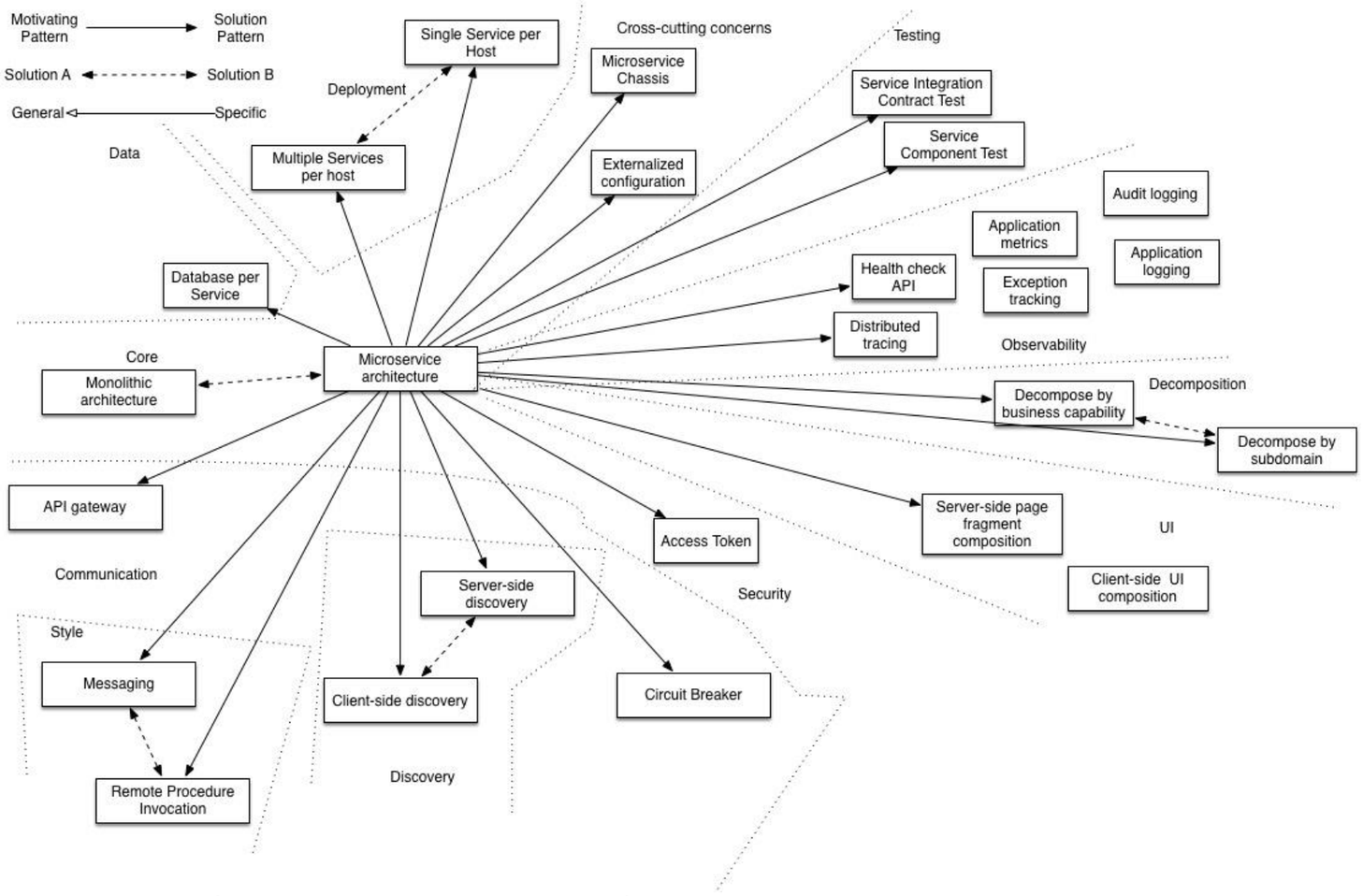
Event Driven Architecture

- La arquitectura enfocada a eventos mediante el envío y recepción de eventos nos permite desacoplar responsabilidades.
- Los servicios internos o externos que reciban los eventos darán tratamiento a ellos y emitirán nuevos eventos de ser necesario.



Evnt Driven con Arquitectura Hexagonal

Microservicios / Patrones

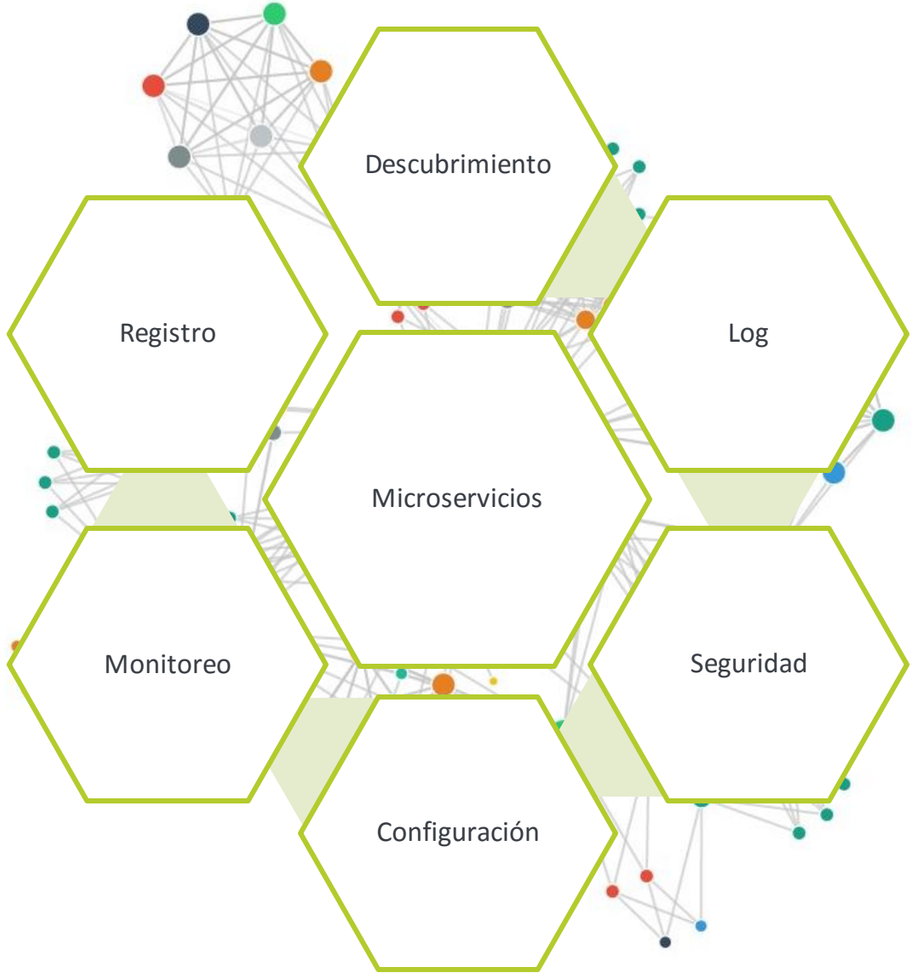


Resumen de Patrones para Arquitecturas de Microservicios

Imagen: Chris Richardson

Arquitectura / Componentes

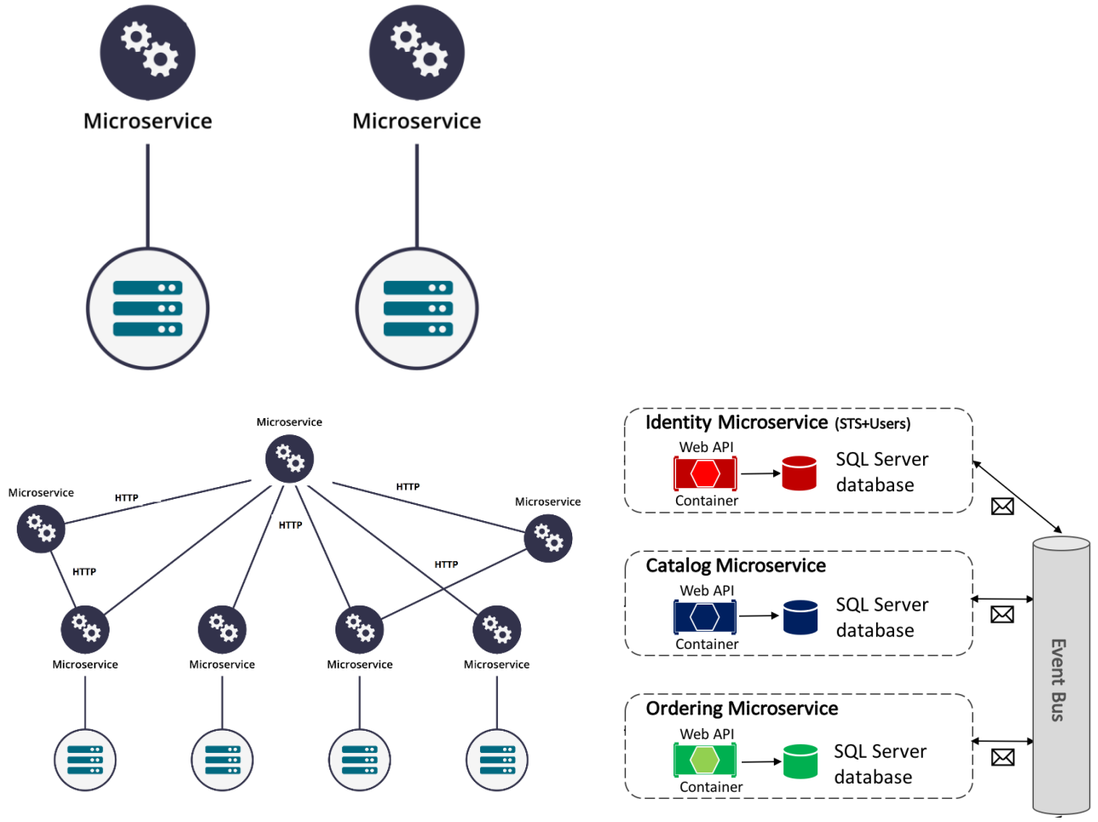
- Los microservicios son mas pequeños, desacoplados y perfeccionados para realizar pequeñas tareas de manera eficiente.
- Aparte de la funcionalidad principal del microservicio, necesitaremos componentes específicos para hacer una correcta implementación de la arquitectura.
- Al lado derecho podemos observar los retos de arquitectura planteados para el desarrollo de microservicios.



Componentes Arquitectura de Microservicios

Arquitectura /Acceso a Datos y Comunicación

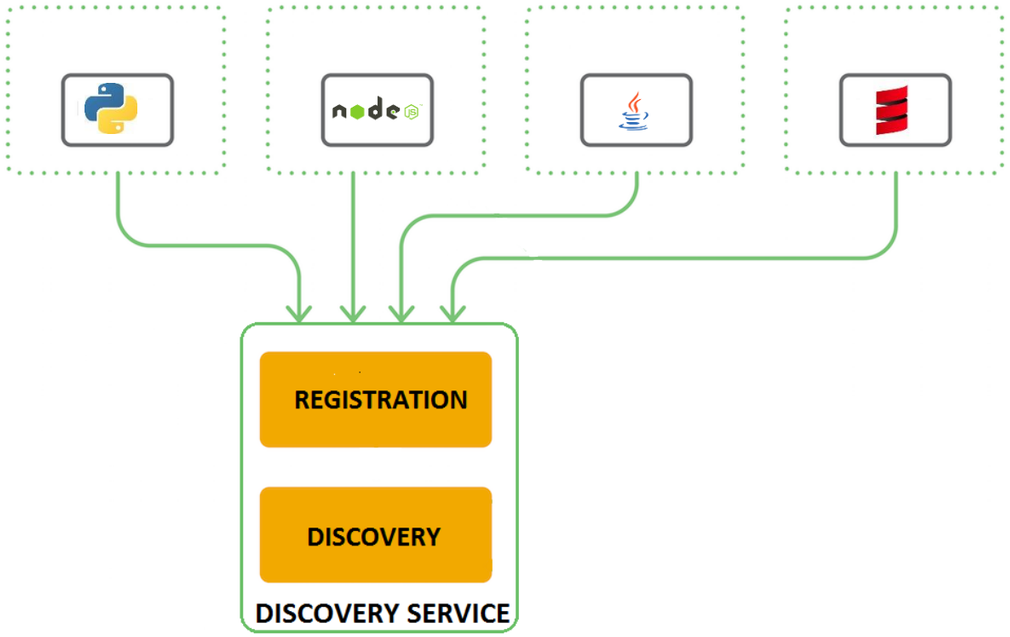
- De la misma manera que en una aplicación monolítica los microservicios pueden acceder a datos a través de capas de persistencia, drivers o configuración especifica para cada caso.
- Exponen funcionalidades a través de API’s rest.
- Se comunican entre si con protocolos http o mensajería.



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.

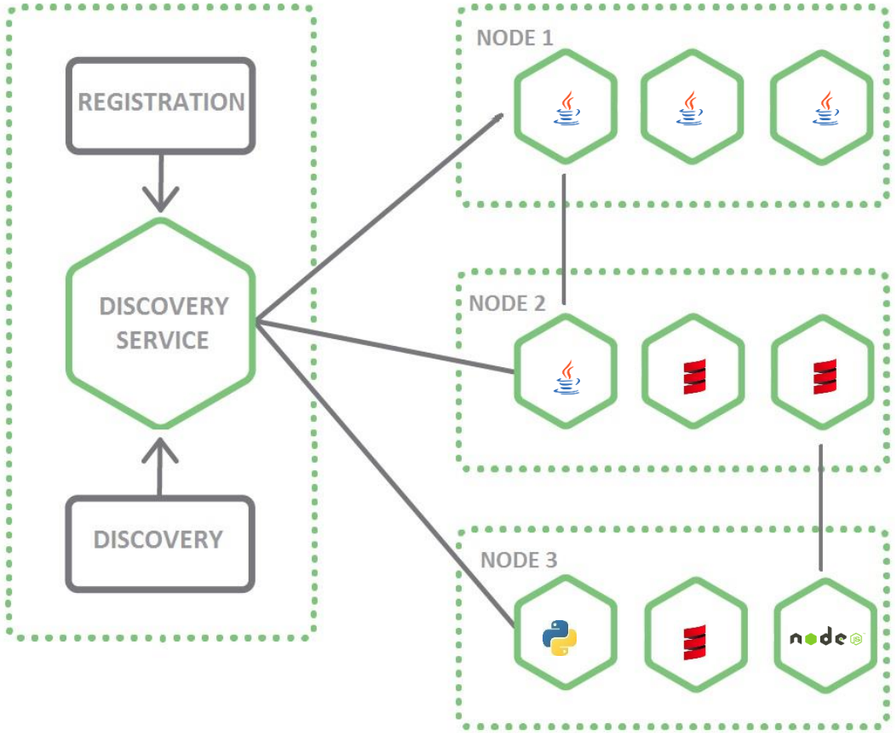


Servicio de Descubrimiento

Componentes /Descubrimiento y Orquestación

Los procesos que componen el servicio de descubrimiento son:

- **Registro:** Consiste en el registro de cada nueva instancia, dejando consignado sus datos de ubicación, como nodo, puerto e ip (Entre otros).
- **Descubrimiento:** Es el proceso que permite tener acceso a la información del registro.

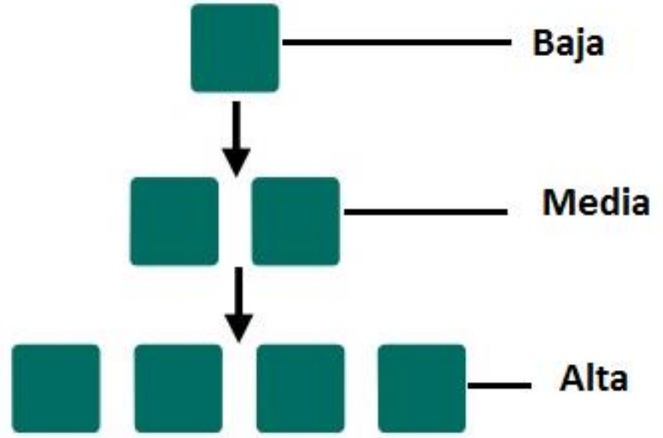


Servicio de Descubrimiento

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

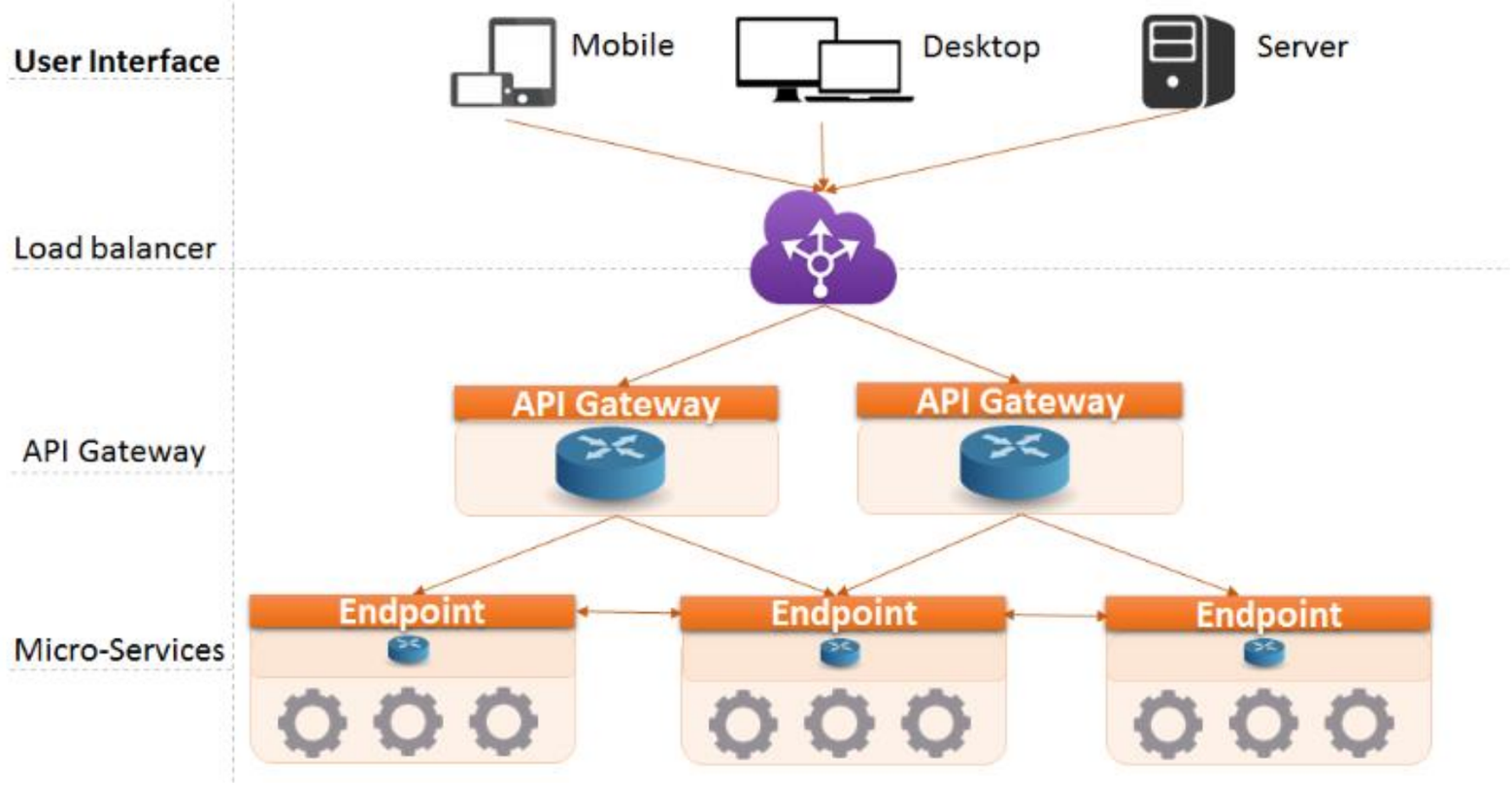
Componentes / Enrutamiento

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)

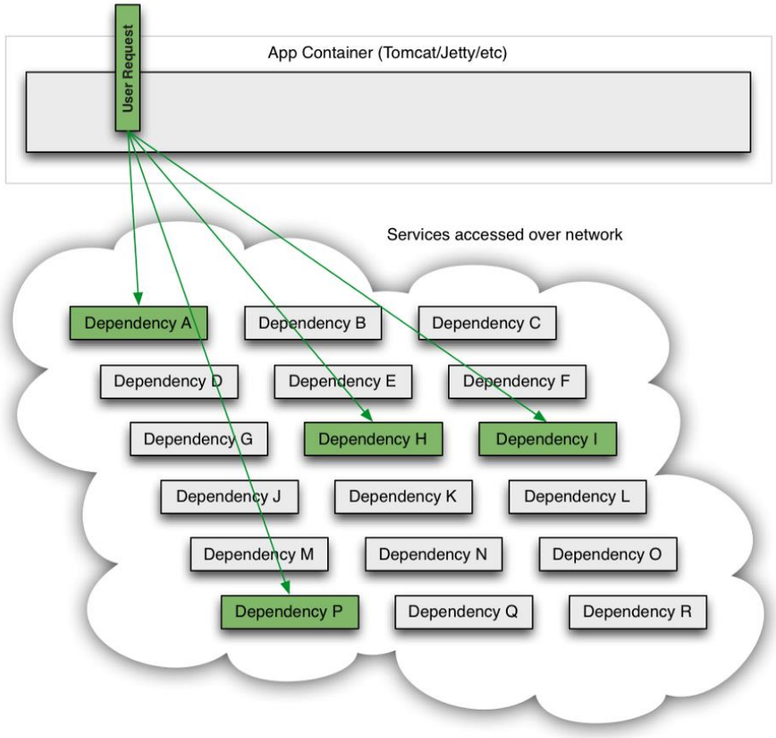
Componentes / Enrutamiento



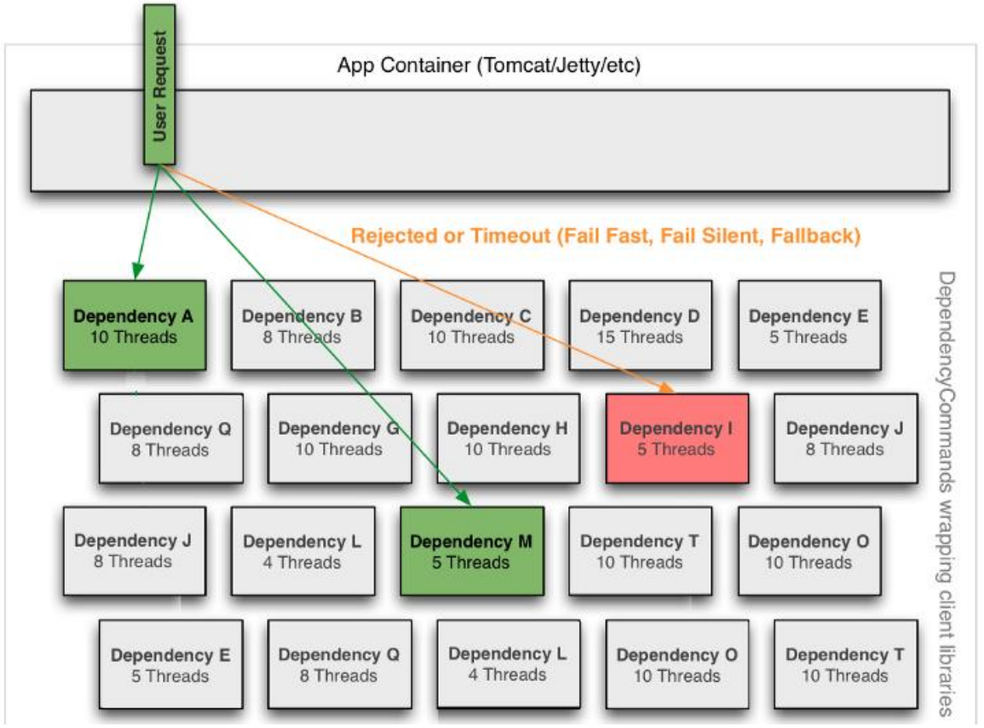
API Gateway

Componentes / Gestión de Errores

En los sistemas distribuidos una falla o demora en la respuesta puede causar una caída total de la aplicación. Por ende es necesario detectar las fallas y encapsular o contener la propagación.

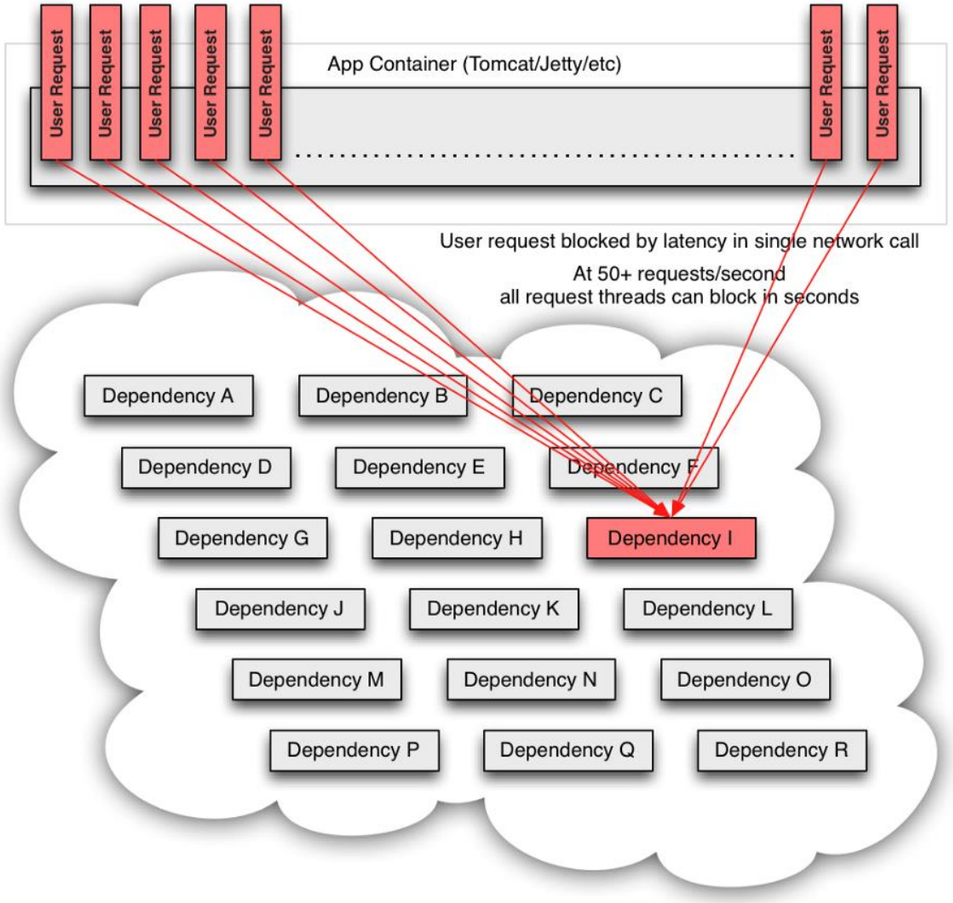


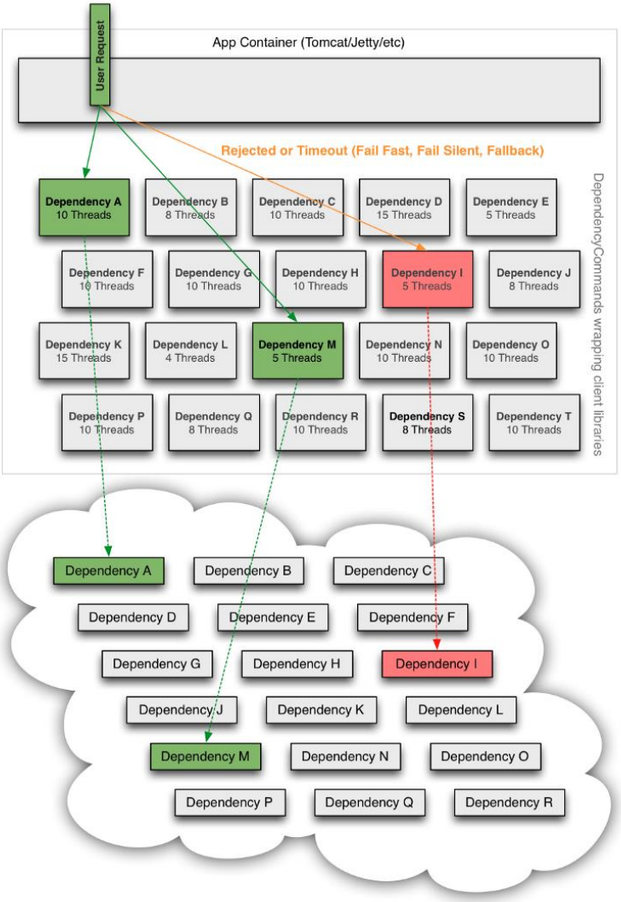
Funcionamiento Normal



Falla en Solicitud

Componentes / Gestión de Errores

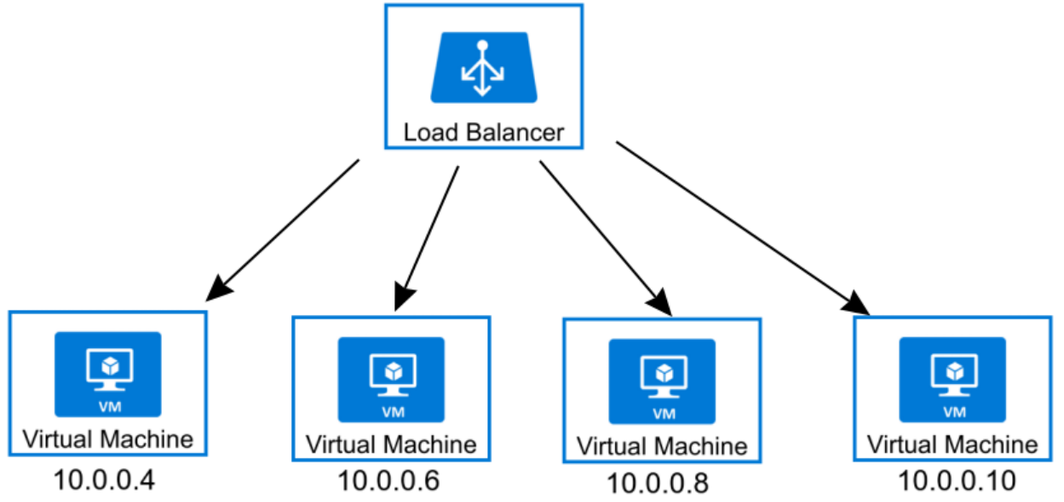




Componentes / Balanceador de Carga

- 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.

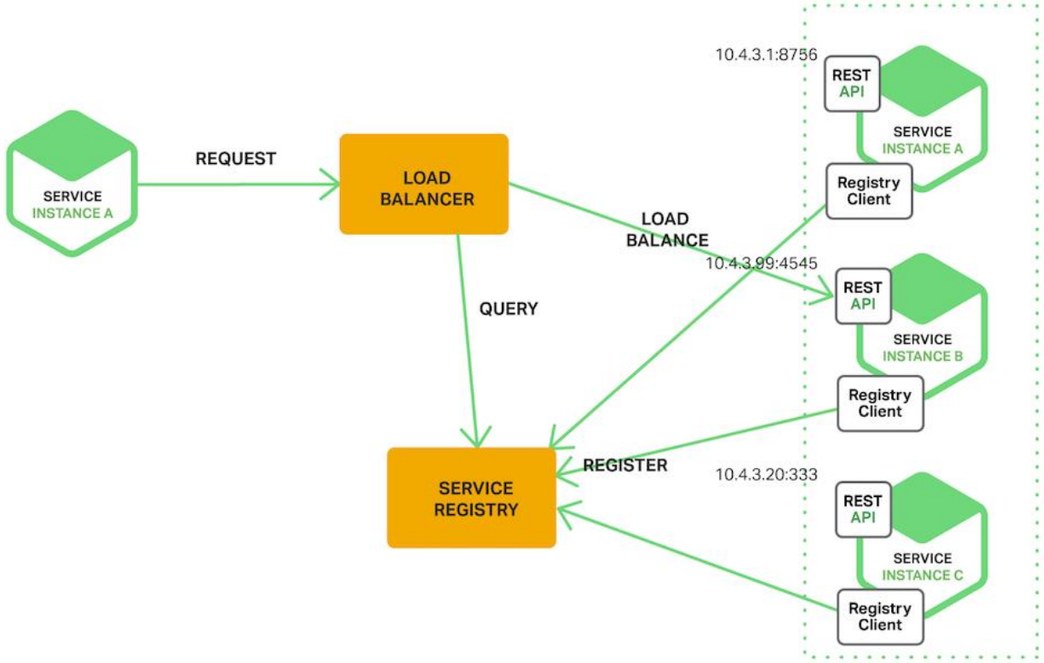


Balanceador Externo

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).

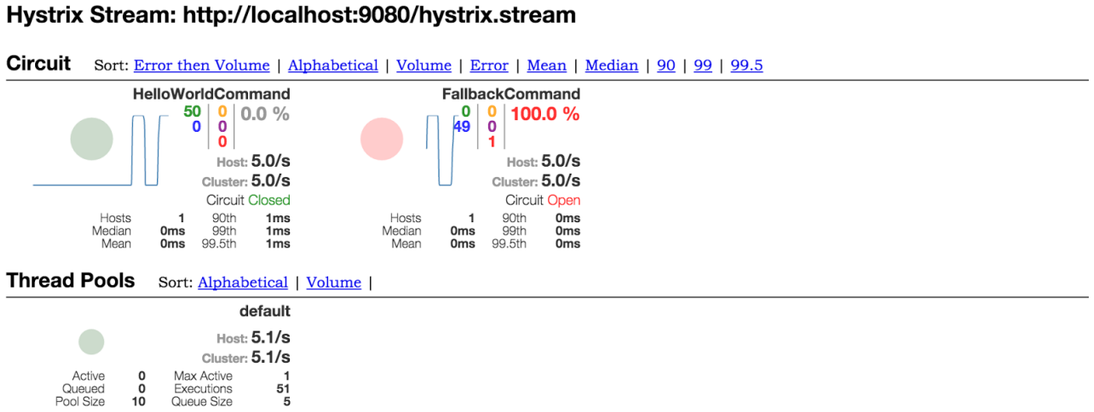


Balanceador en el Cliente

Componentes / Monitorización

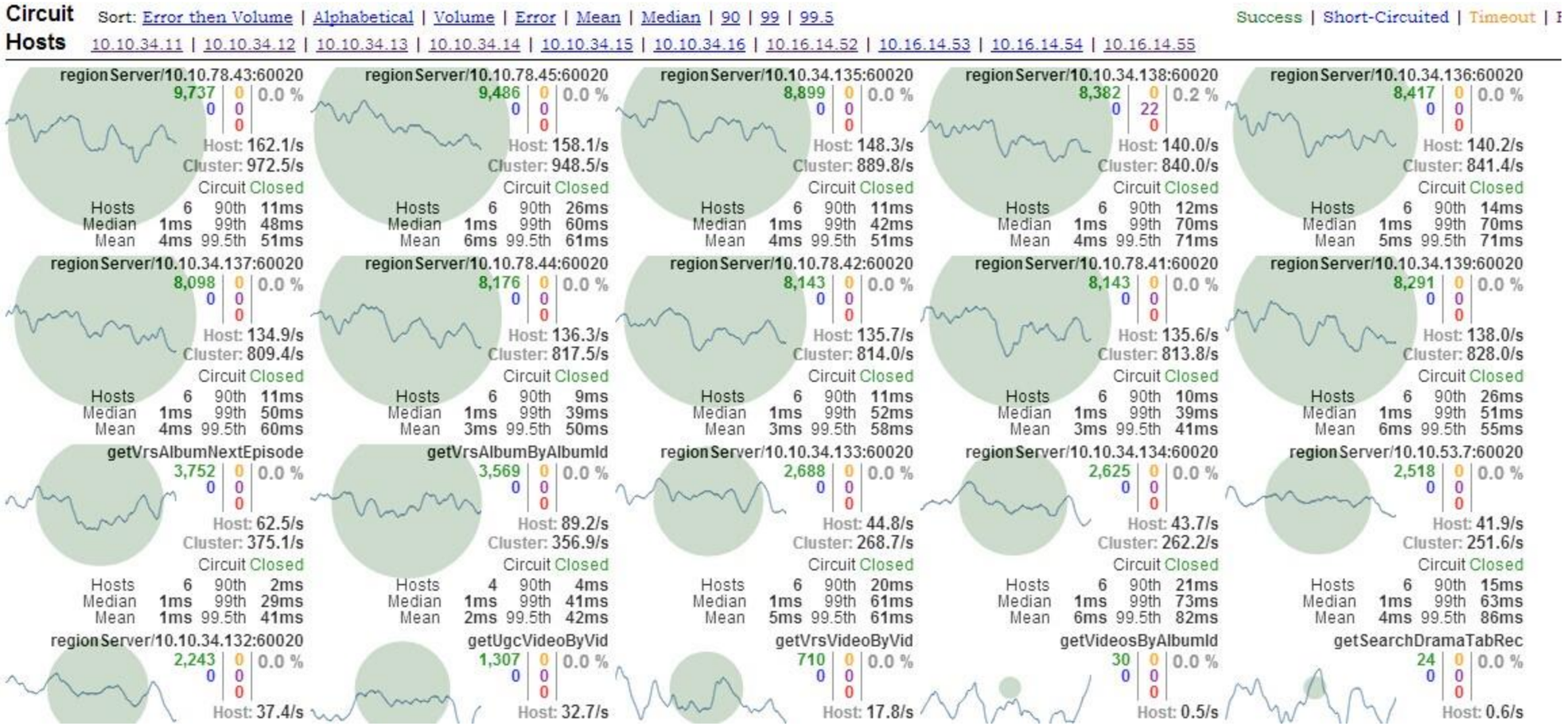
- Un aspecto importante a resaltar en la arquitectura a microservicios es la monitorización, contar con un panel unificado con información en tiempo real de cada microservicio.

- Ejemplos:
  - Hystrix + turbine + Spring boot admin



Hystrix Dashboard

Componentes / Monitorización y Logs

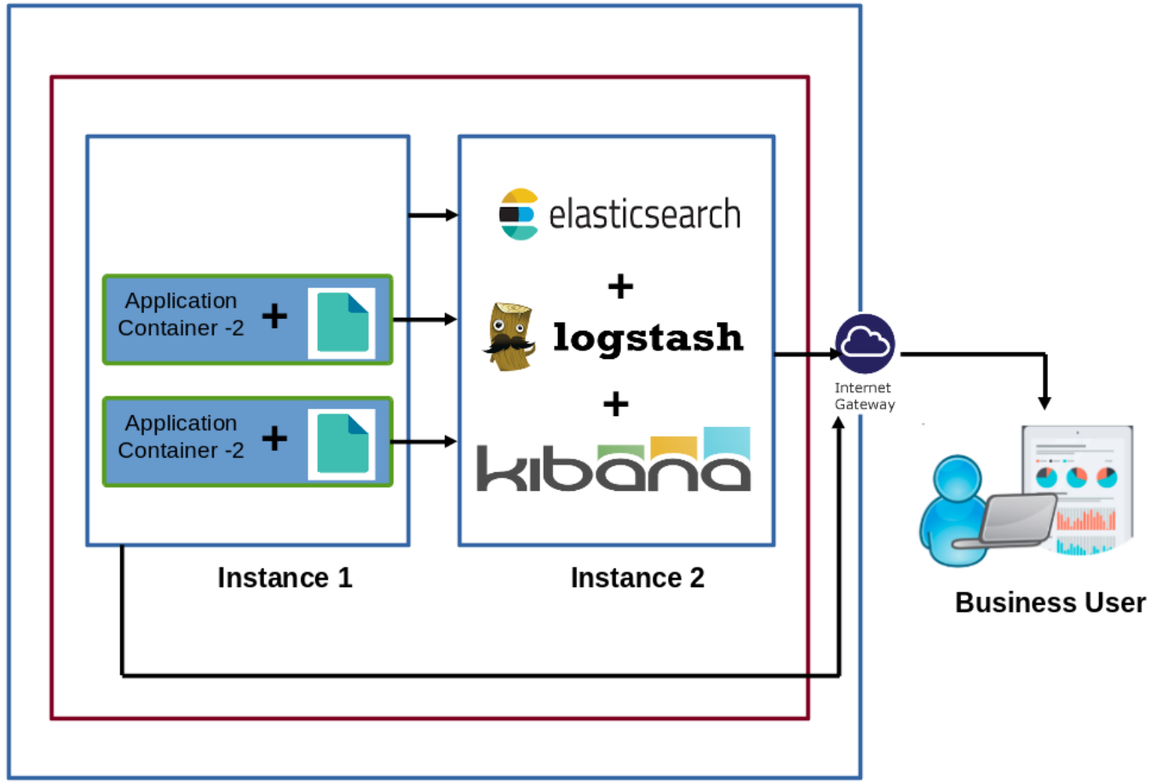


Hystrix Dashboard

Componentes / Monitorización y Logs

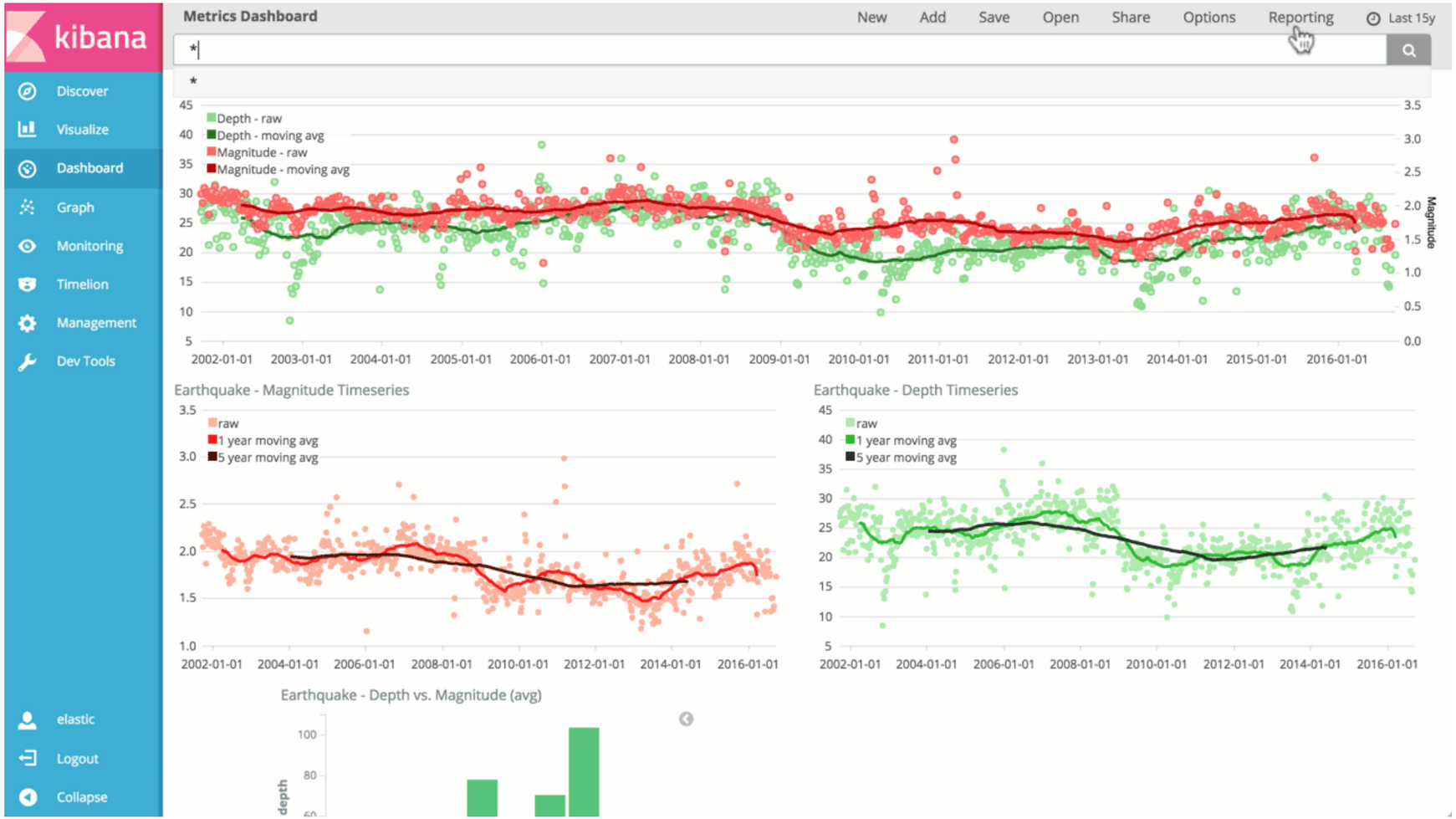
- La centralización y explotación de logs juega un papel importante de lo contrario seria inmanejable la administración de los mismos.

- **Ejemplo:**
  - ELK



ELK Explotación de Información

Componentes / Monitorización y Logs

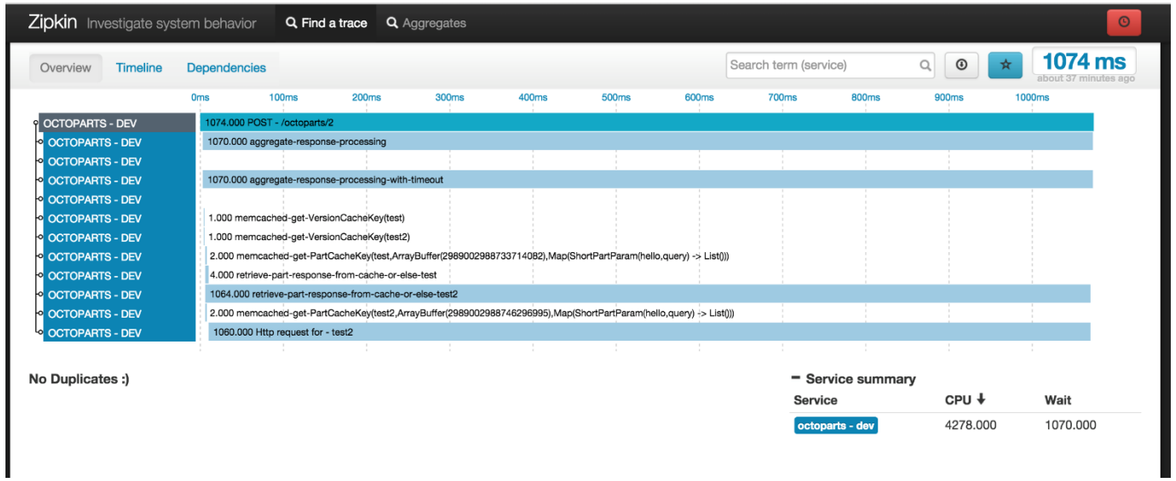


Kibana

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

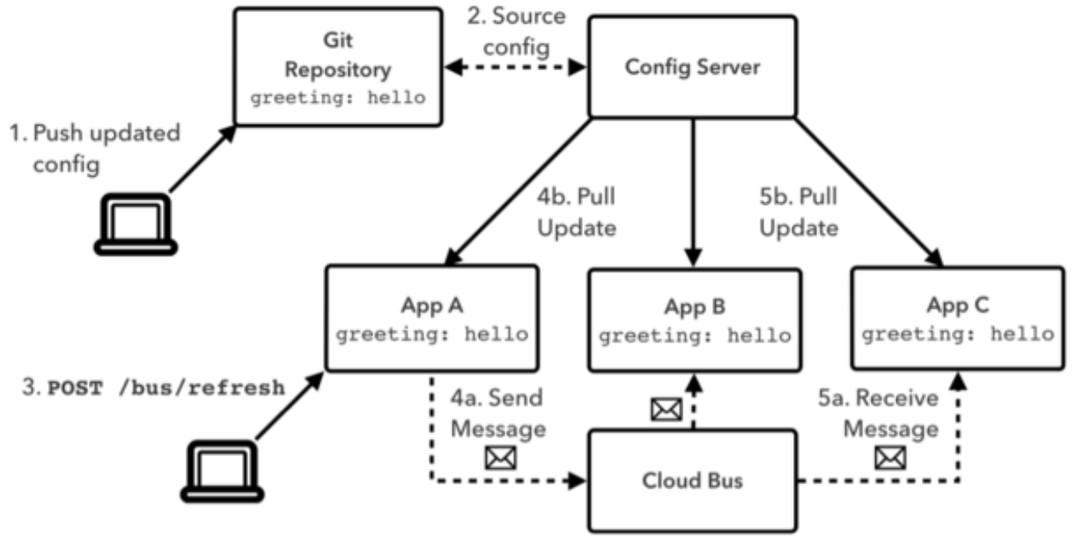


Análisis de Solicitudes con Zipkin

Componentes / Servicios de Configuración

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

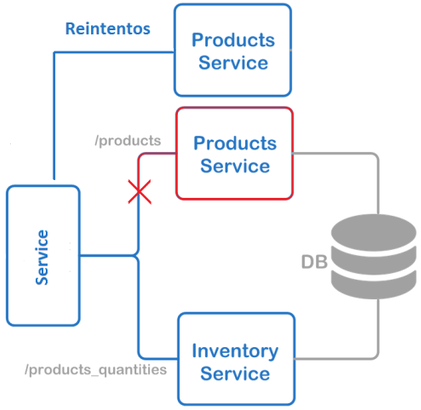
- **Ejemplo:**
  - Spring cloud config
  - Archadius
  - Consul
  - Zookeeper



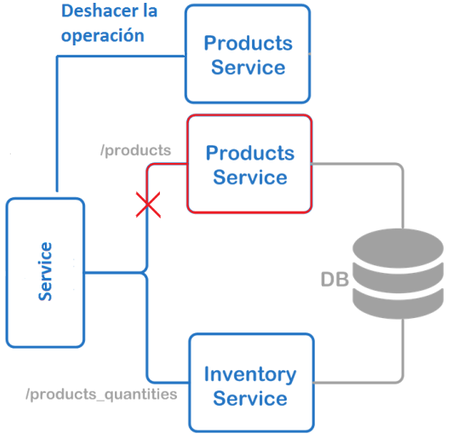
Servicio de Configuración

Componentes / Transaccionalidad

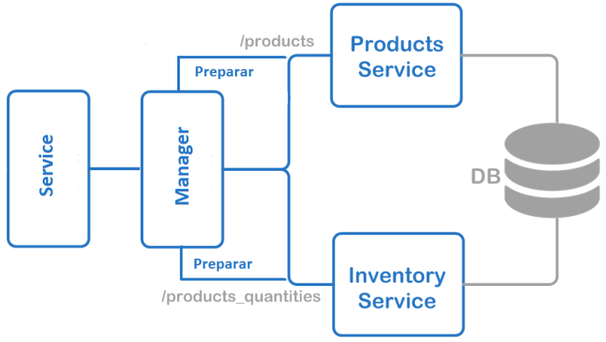
- **Política de reintentos:** usada en caídas temporales del servicio. Reintentar las operaciones pude solventar el problema.
- **Deshacer operaciones en su totalidad:** Es importante conservar un estado previo para conservar la consistencia de datos al momento de deshacer la operación.
- **Ejecución distribuida:** Puede utilizarse un servicio centralizado para la operación que garantice la correcta finalización de la misma. Commit de dos fases.



Reintentos



Deshacer

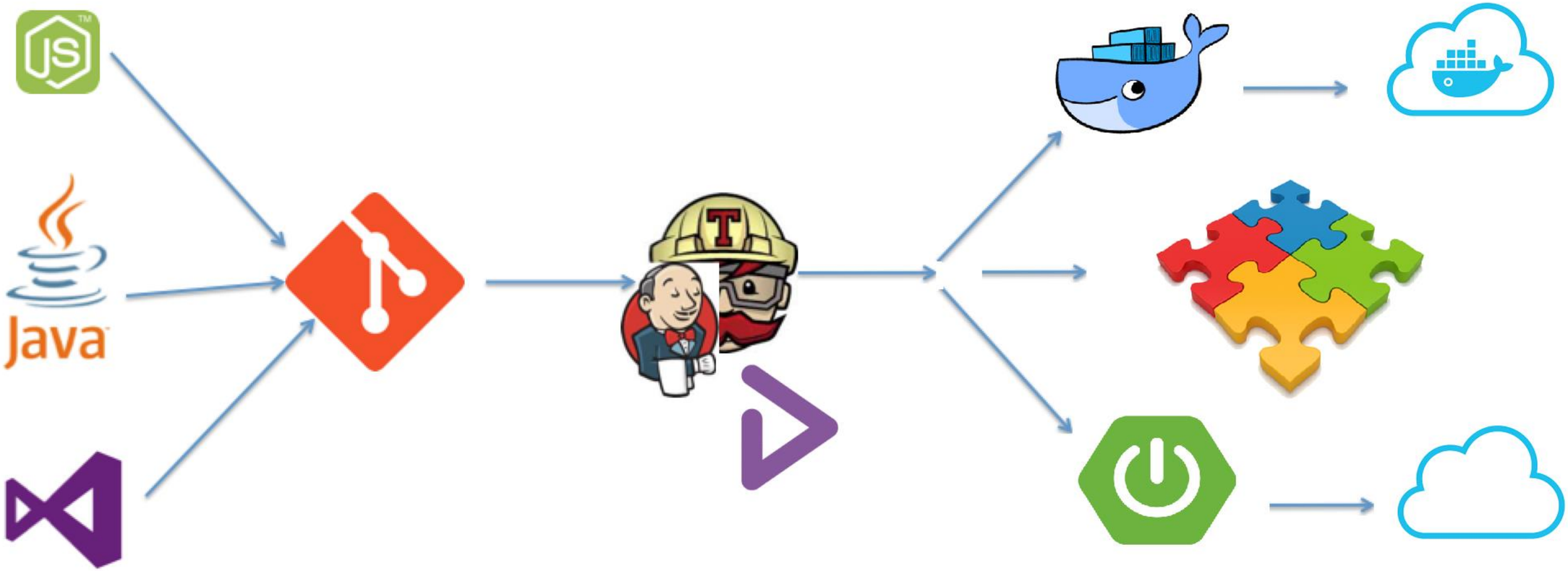


Distribuida Coordinada

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



Repositorio CI/CD Contenedor o No. Registry o Cloud

Recursos / Paas (Platform as a Service)

- **Nos ofrece la abstracción de la capa de plataforma donde se encuentran las herramientas para la operación.**
  - Monitorización y consolas administrativas
  - Logs
  - Despliegues, CI/DC
  - Registro de aplicaciones
  - Replicación
  - Estadísticas
  - Enrutamiento
  - Balanceadores
- **Plataformas**
  - Aws
  - Cloudfoundry
  - Google cloud
  - OpenShift
  - Azure
  - Digital ocean (+/-)

Recursos / Iaas (Infrastructure as a Service)

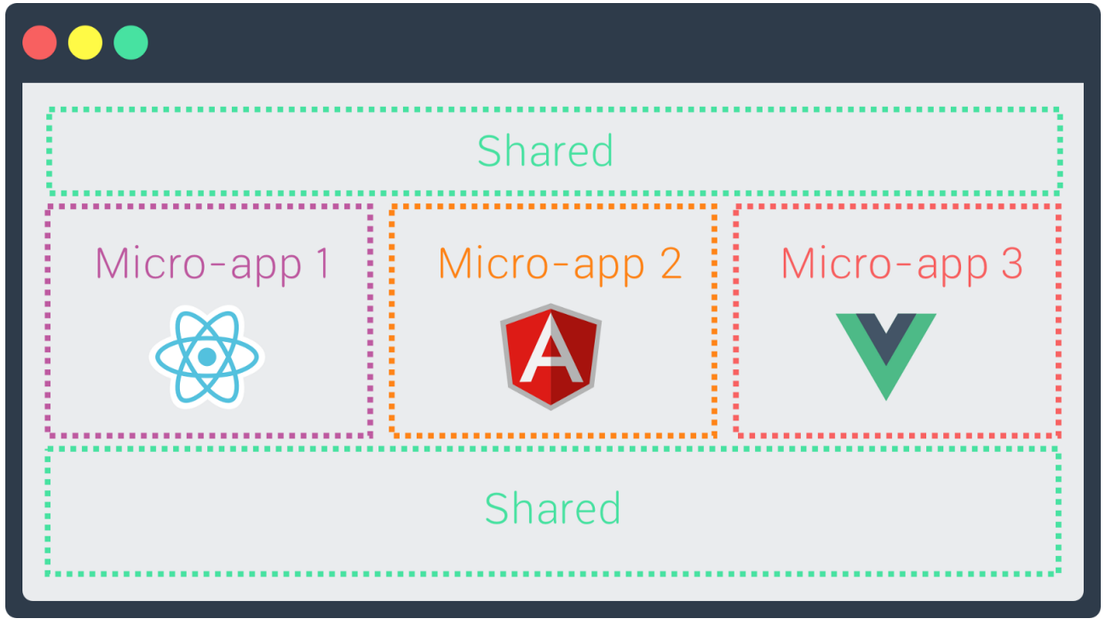
- **Nos ofrece la abstracción de la capa física de infraestructura.**
  - Servicios de red
  - Seguridad
  - Almacenamiento
  - Máquinas virtuales
- **Ejemplos:**
  - AWS
  - Digital ocean
  - Google
  - OpenStack (CPD Propio)
  - DC/OS Apache mesos (CPD Propio)

Microservicios / FrontEnd

Es posible ?

- La respuesta a su duda es SI.
- Segregar funcionalidades para creación de microservicios.
- Diversidad de Frameworks.
- Comunicación entre microservicios.

Microservicios FrontEnd



Casos de Uso

- Aplicaciones con alto nivel de concurrencia.
- Tiendas online
- Streaming
- Real time
- Analítica de datos
- Seguridad

Referencias

- **Book: Microservices Best Practices for Java**
  - Author: Michael Hofmann – Erin Schnabel – Katherine Stanley
  - IBM
- **Book: Designing Reactive Systems**
  - Author: Hugh Mackee
  - O'REILLY
- **Book: Developing Reactive Microservices**
  - Author: Markus Eisele
  - O'REILLY
- **Book: Reactive Microsystem**
  - Author: Jonas Bonér
  - O'REILLY
- **Book: Reactive Microservices Architecture**
  - Author: Jonas Bonér
  - O'REILLY
- **Book: Reactive Microservices Architecture**
  - Author: Chris Richardson / Floyd Smith
  - NGINX
- https://es.wikipedia.org/wiki/Arquitectura_de_microservicios
- http://www.reactivemanifesto.org/es
- https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/2h-2014-global-internet-phenomena-report.pdf
- https://www.sandvine.com/downloads/general/global-internet-phenomena/2016/global-internet-phenomena-report-latin-america-and-north-america.pdf
- https://es.slideshare.net/InfoQ/scalable-microservices-at-netflix-challenges-and-tools-of-the-trade

Gracias

JAIDER ANILLO GARCIA
DESARROLLADOR SENIOR
janillo@todo1.com