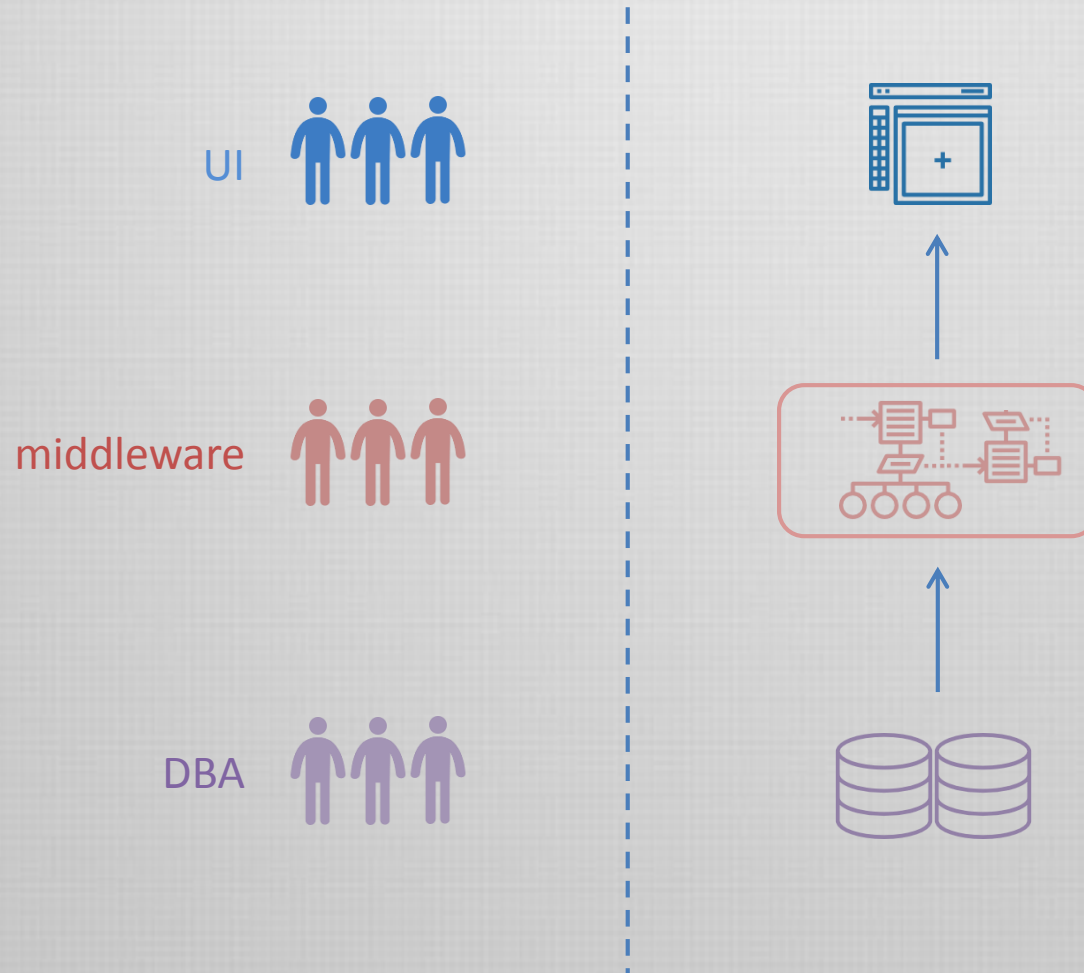


Ingeniería de Aplicaciones Web

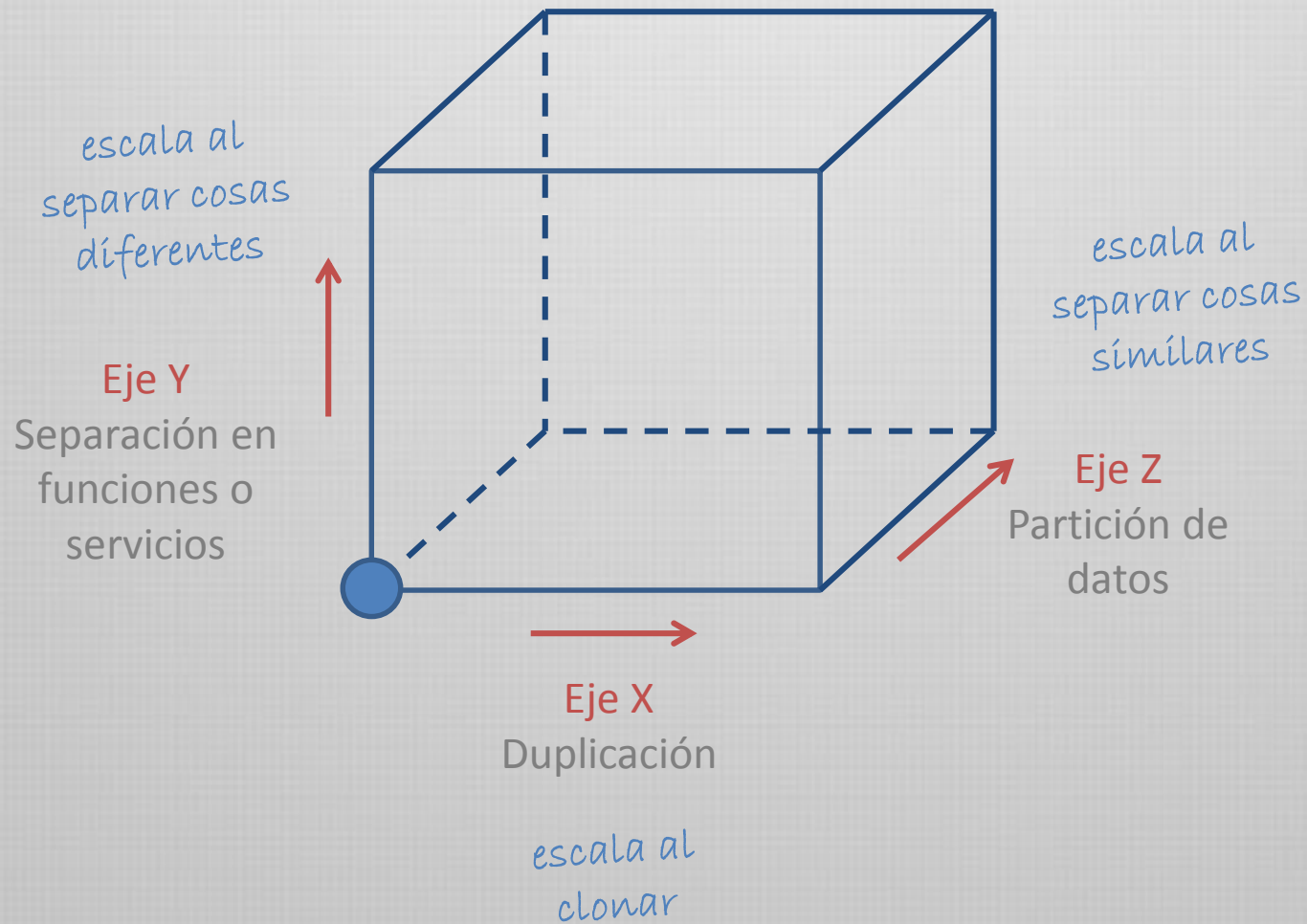
Diego C. Martínez

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Arquitecturas Monolíticas



Cubo de escalabilidad



Microservicios

Los **microservicios** son un **estilo arquitectónico** que organiza una aplicación como una **colección de servicios poco acoplados** que implementan lógica de negocios

“...the microservice architectural style is an approach to developing a **single application as a suite of small services**, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.”



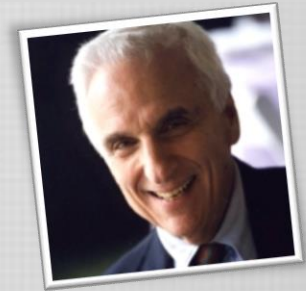
Martin Fowler



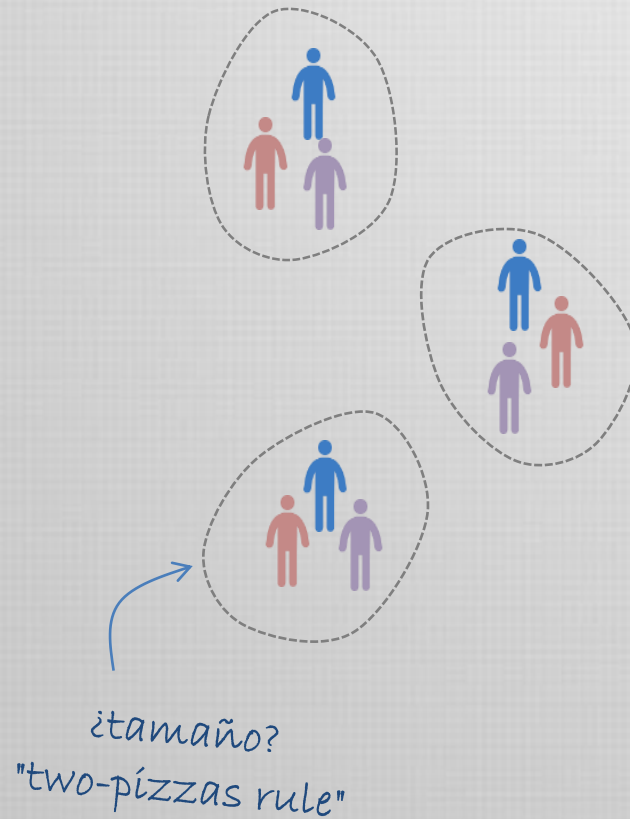
Adrian Cockcroft

“..fine grained SOA”

Microservicios

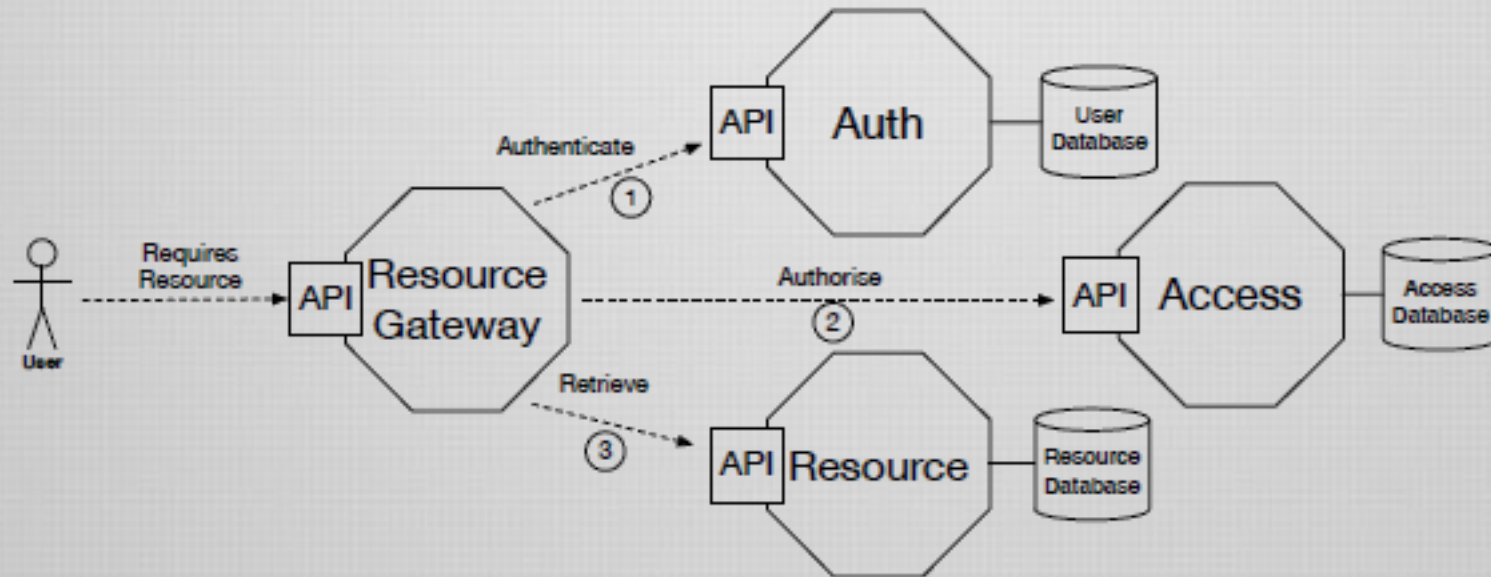


Melvyn Conway



Componentization via Services
Organized around Business Capabilities
Products not Projects
Smart endpoints and dumb pipes
Decentralized Governance
Decentralized Data Management
Infrastructure Automation
Design for failure
Evolutionary Design

Microservicios



Microservices: yesterday, today, and tomorrow. Nicola Dragoni et al

Comunicación entre servicios

Un aspecto clave en el diseño de una arquitectura de microservicios

Pueden existir varios **mecanismos de comunicación**


Requiere siempre la definición de una **API**



Application Programming Interface

*Un conjunto de funciones, procedimiento o servicios
que permite acceder a las funcionalidades de un
componente del sistema*

Para la comunicación técnica entre los servicios,
pueden usarse tecnologías **RPC** (como SOAP o JRMII) o **REST**



*La mas flexible
y preferida*

RPC

Remote Procedure Call



*técnica para realizar una llamada local y
que se ejecute un servicio remoto*

SOAP, Thrift, JRM, XML-RPC, JSON-RPC

*Genera fácilmente
stubs para el
cliente y el servidor*

*Algunos son de naturaleza
binaria (como JRM),
otros basados en XML
(como SOAP)*

*Algunos se basan en HTTP,
otros en protocolos
alternativos (como UDP)*

*Algunos mecanismos
dependen fuertemente de
tecnologías subyacentes,
como JRM
(cliente y servidor)*

*El marshalling de datos
tiene un costo adicional*

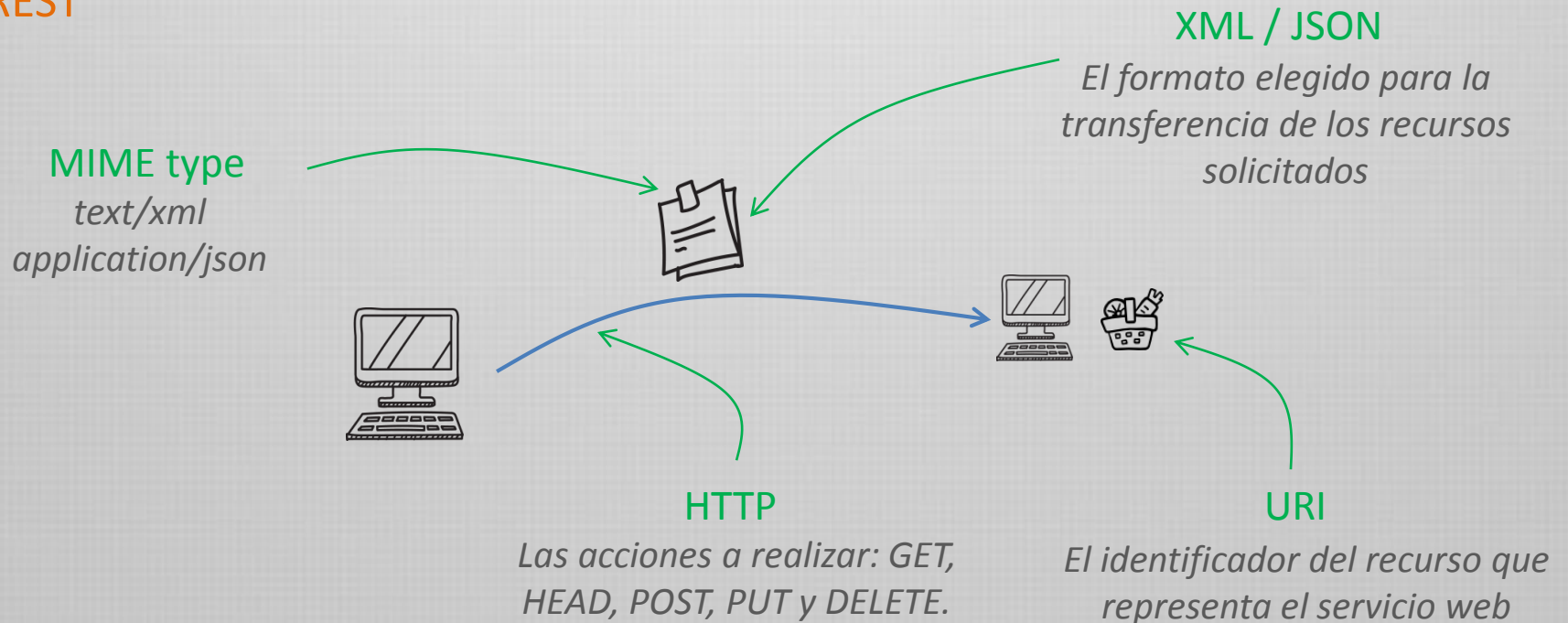
*A veces se oculta
demasiado que es una
llamada remota, no local*

REST

Representational State Transfer.

Es una abstracción de elementos de una arquitectura dentro de un sistema hipermedial distribuido.

REST

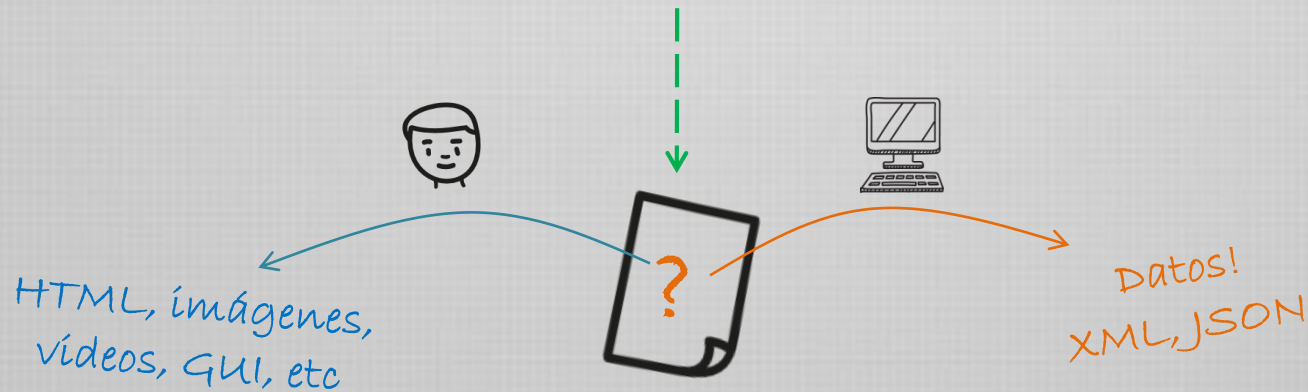


REST

REST ve el mundo como una **colección de recursos**. *abstracción de "datos"*

identificables por
medio de URI o URL

`http://unhost.com/ligafutbol/equipos/deportivoX`



REST – mensajes HTTP

REST asigna un rol especial a los mensajes HTTP, consistente con su uso web.

GET *Retrieve*

Recupera la representación de un recurso

HEAD *Retrieve*

Recupera metadatos de la representación de un recurso

POST *Create*

Estrictamente, crea un recurso.

Se usa también para actualizar un recurso o borrarlo

PUT *Update*

Actualiza un recurso.

Usualmente reemplazado por el método POST

DELETE *Delete*

Elimina un recurso.

Usualmente reemplazado por el método POST

<http://www.autopartes.com/partes>

```
<?xml version="1.0"?>
<p:partes xmlns:p="http://www...." xmlns:xlink="http://...">
  <p:autoparte id="ABC1" xlink:href="http://www.autopartes.com/partes/ABC1"/>
  <p:autoparte id="DEF2" xlink:href="http://www.autopartes.com/partes/DEF2"/>
  <p:autoparte id="GEW3" xlink:href="http://www.autopartes.com/partes/GEW3"/>
  <p:autoparte id="KLM1" xlink:href="http://www.autopartes.com/partes/KLM1"/>
</p:partes>
```

<http://www.autopartes.com/partes/DEF2>

```
<?xml version="1.0"?>
<p:partes xmlns:p="http://www...." xmlns:xlink="http://...">
  <p:autoparte>
    <p:id>DEF2</p:id>
    <p:desc>paragolpe delantero</p:desc>
    <p:auto>Fiat 600</p:auto>
    <p:stock>5</p:stock>
  </p:autoparte>
</p:partes>
```

hypermedia as the engine of application state
HATEOAS

Esquemas de comunicación

Como los servicios se ofrecen en procesos independientes existen dos modalidades de comunicación

Sincrónica

El llamador espera por la respuesta del servidor

Más simple de diseñar
Consistente con protocolos de red como HTTP



request/response

Asincrónica

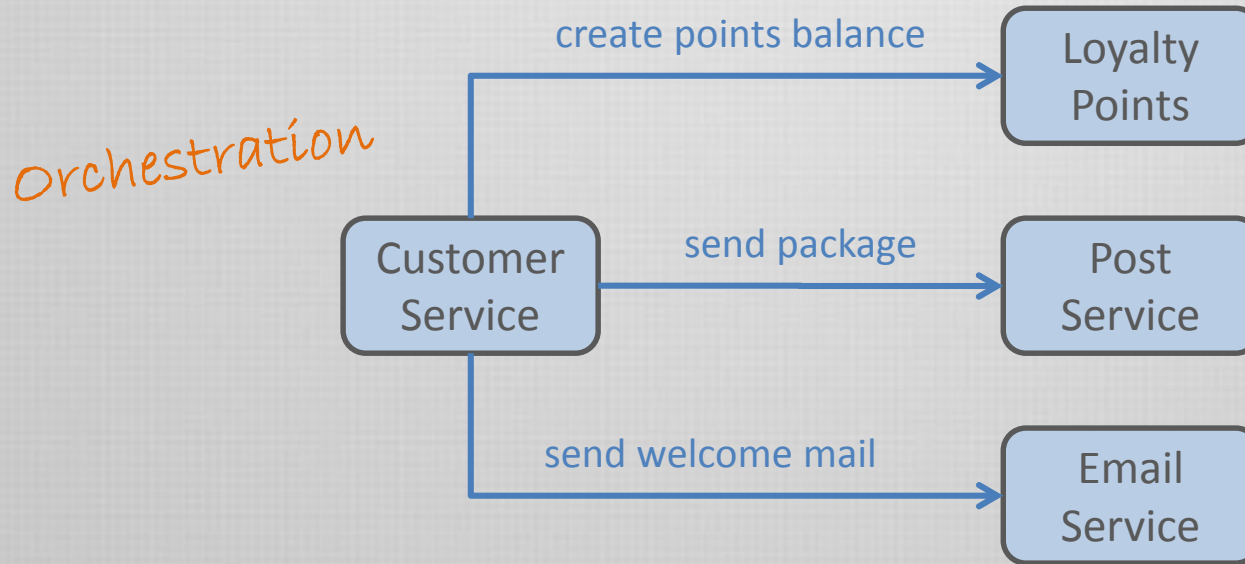
El llamador no espera por la respuesta del servidor

Útil para tareas prolongadas
Favorece la dinámica del proceso



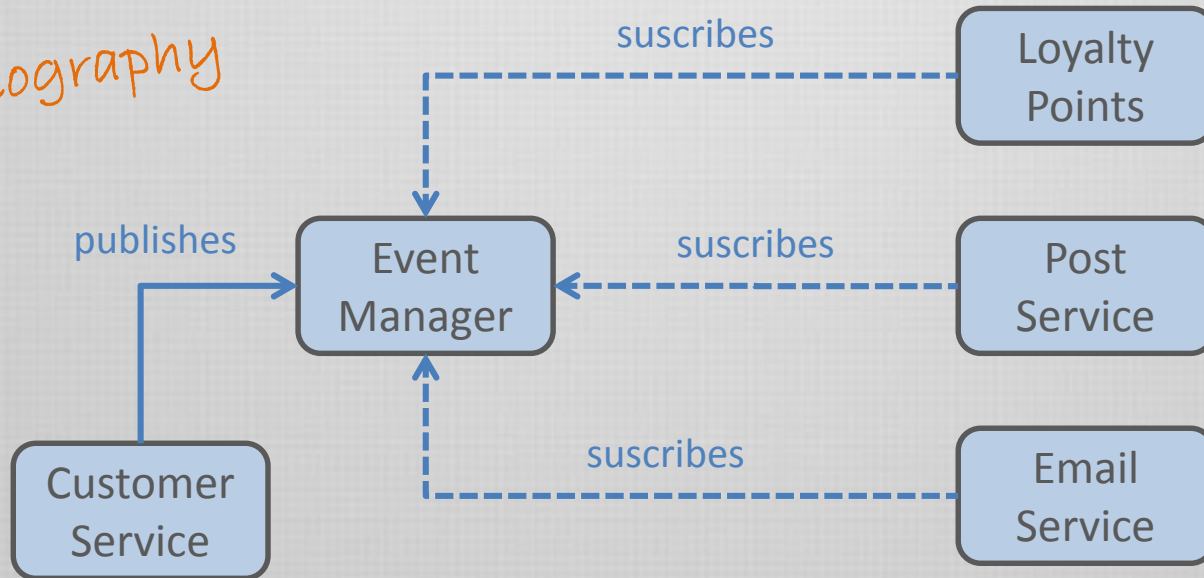
event-based

Orchestration vs Choreography



Orchestration vs Choreography

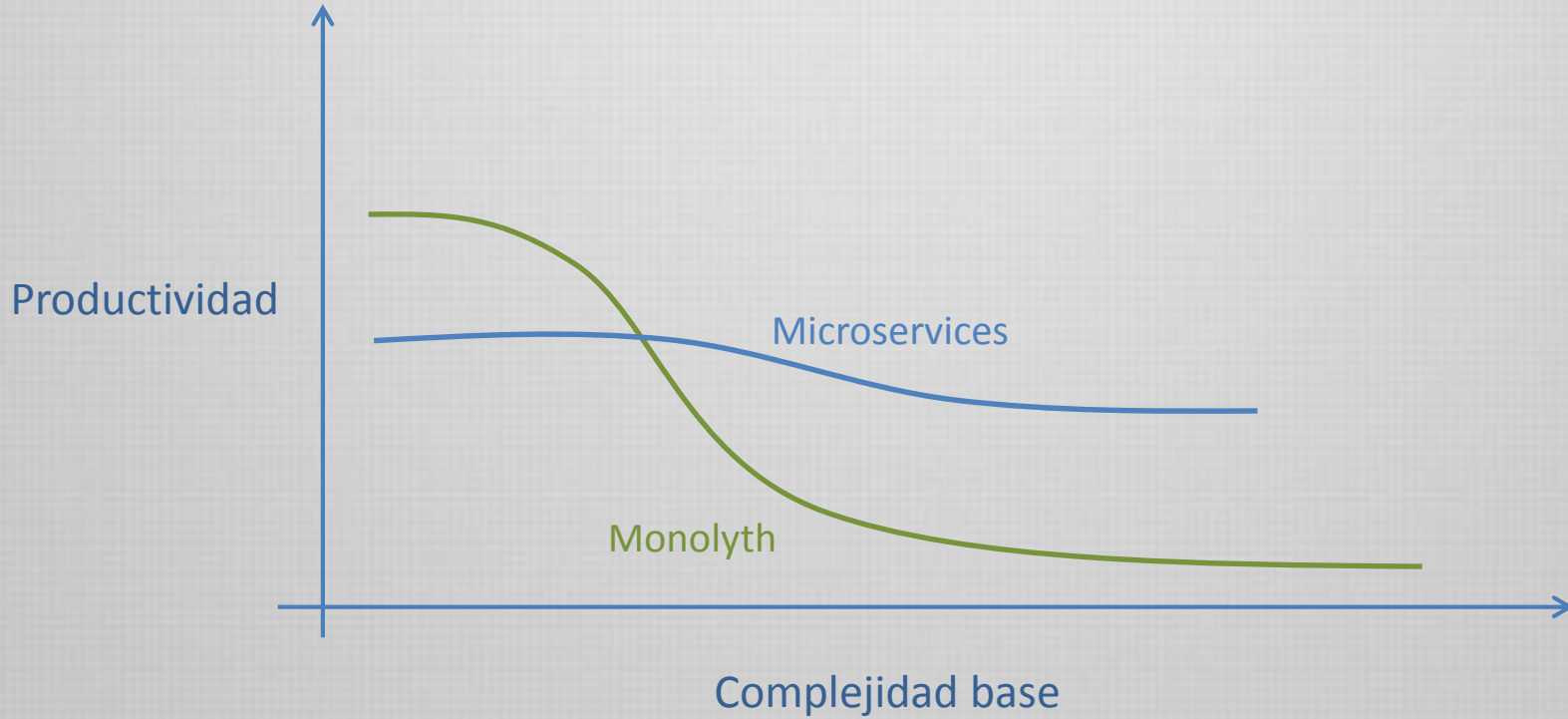
choreography



Ventajas - Monolythic vs Microservices

Monolyth	Microservices
Simplicidad	<i>Deployment</i> parcial
Consistencia	Mejor preservación modular <i>Continuidad</i> <i>Protección</i> <i>Comprensión</i>
Refactoring simple	Múltiples lenguajes y plataformas
IDEs poderosos	Escalabilidad!

Productividad vs Complejidad

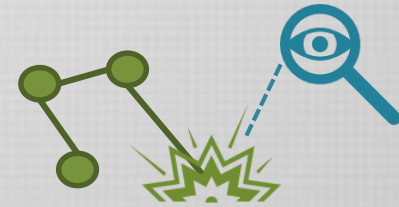


Prerequisitos

Competencias necesarias para poder encarar una arquitectura de microservicios



Provisionamiento rápido



Monitoreo Básico



Desarrollo rápido de
aplicaciones

Tener en cuenta además...

No comenzar directamente con microservicios

Procuran resolver problemas de escalabilidad

No aplicar microservicios sin DevOps

Es un sistema en movimiento constante

No administrar toda la infraestructura

La diversidad es compleja (múltiples DB, mensajería, cache, etc)

Utilizar servicios Cloud

Balancear la cantidad de microservicios

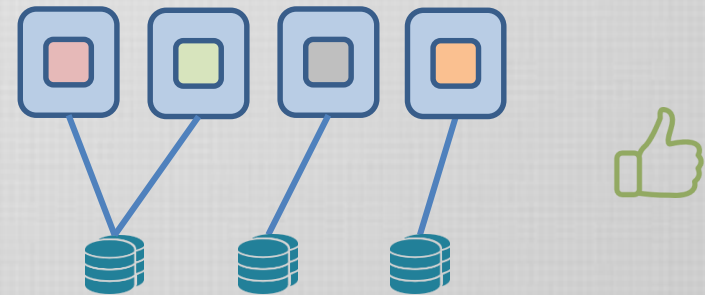
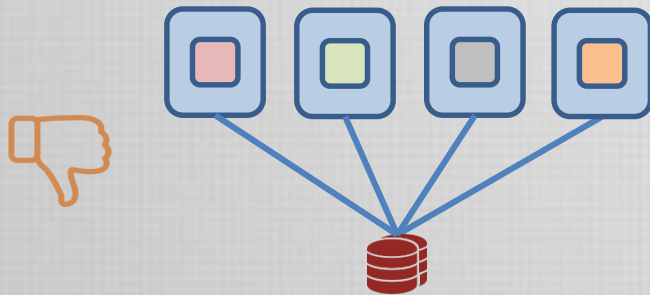
Muchos pequeños vs algunos medianos

Observar la latencia

El aumento de granularidad incrementa la latencia

Buenas prácticas para implementar microservicios

Crear *Data Stores* separados para cada microservicio



Puede requerir
Master Data Management

Buenas prácticas para implementar microservicios

Mantener el código al mismo nivel de madurez

Ante una **depuración** o **actualización**



Buenas prácticas para implementar microservicios

Hacer un *build* separado por cada microservicio

Cada microservicio debería poseer

código base separado

APIs bien definidas

Scripts de deployment separados

Artefactos desplegables separados

Control completo

(technical and management)

Buenas prácticas para implementar microservicios

Deploy en Containers

Simplifica el proceso de deployment

Docker es un estándar de facto



Buenas prácticas para implementar microservicios

Tratar los servidores como *stateless*



"servers like cattle, not pets"

Esto facilita el intercambio del servidor en un grupo

Microservicios \neq silver bullet

Complejidad de un sistema distribuido

Es necesario un mecanismo de comunicación entre servicios (mensajes, interfaces, sincronización, serialización)

Complejidad de desarrollo

No existe soporte integral de desarrollo (IDEs)

Degradación de rendimiento

Overhead de comunicación

Consistencia de los datos

La partición de los datos requiere atención especial en sincronización y consistencia

Dificultad para el testing e integración

Es difícil el diseño de casos de prueba integrales

Complejidad en la infraestructura

Crece la complejidad operativa y de gestión de toda la infraestructura



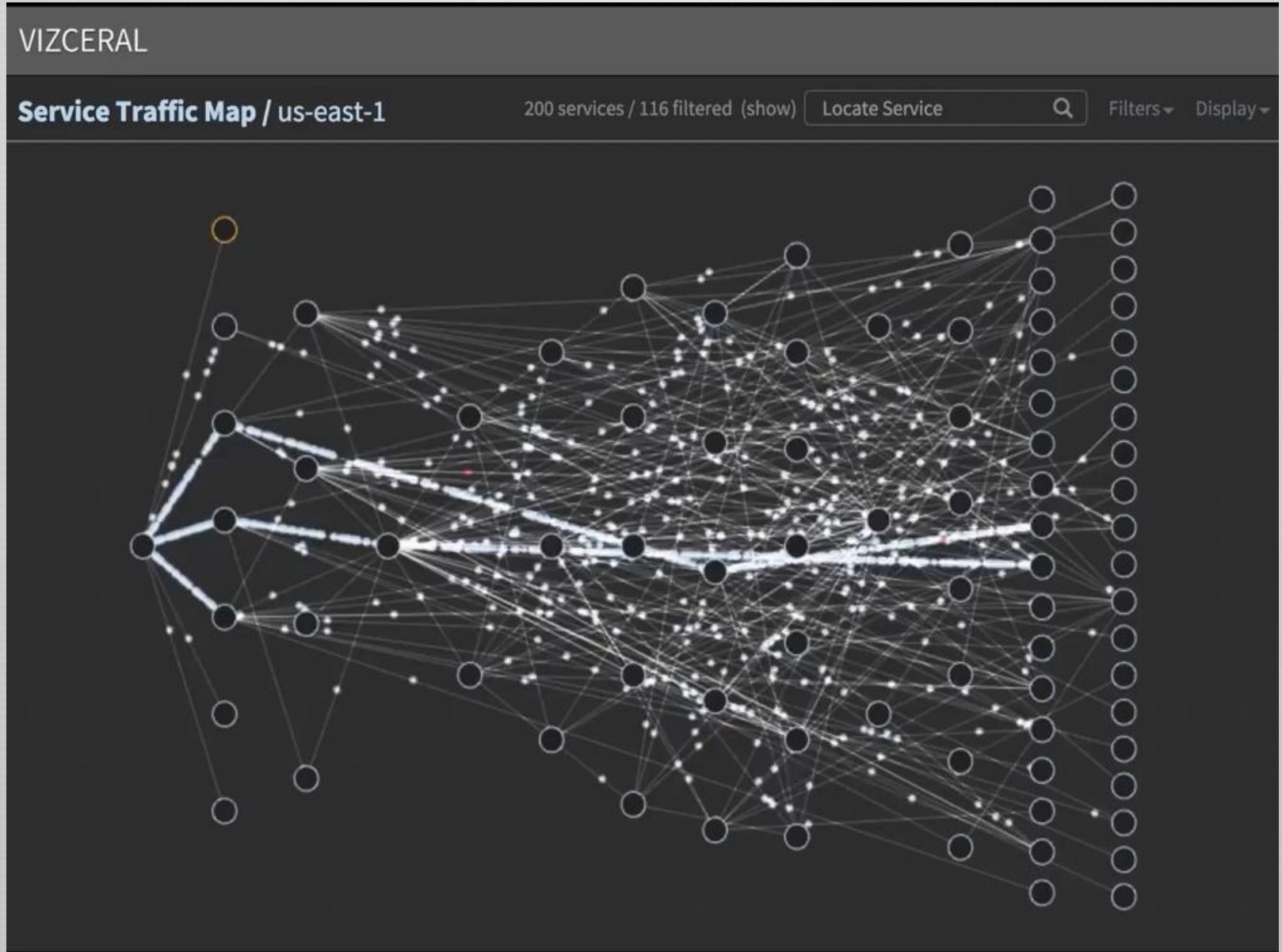
Fundada en 1997
Alquiler de DVDs por Internet

Suscripción mensual
Alquileres ilimitados
Sistema de recomendación de películas

En 2007 incorpora streaming de películas
En 2010, 20 millones de suscripciones

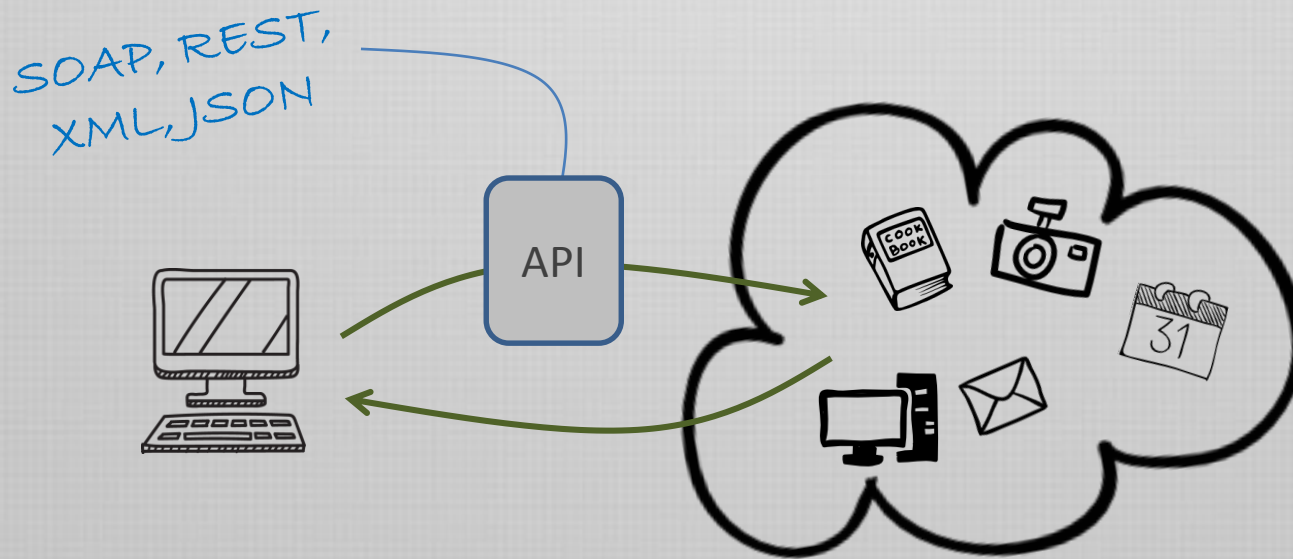
En 2013 comienzan a producir contenidos
House of Cards, Lilyhammer, Hemlock Grove,
Arrested Development S04, Orange is the New Black...

Netflix Microservices



Cloud Computing

Es la provisión de **servicios** via Internet. **Trending topic :)**



Cloud Computing - virtualizaciones



La virtualización de recursos es el núcleo de muchas arquitecturas *cloud*.

vista lógica, abstracta de recursos físicos

Ventajas de la virtualización

Uso mas eficiente de recursos. Existe un pool de recursos disponible para diferentes virtualizaciones en demanda.

Management. Las máquinas virtuales pueden crearse y configurarse automáticamente

Consolidación. Varios servidores físicos pequeños pueden consolidarse en uno solo de mayor capacidad. Menor consumo de energía.

Atención de emergencias. La portabilidad de las máquinas virtuales facilita la migración y las copias de respaldo.

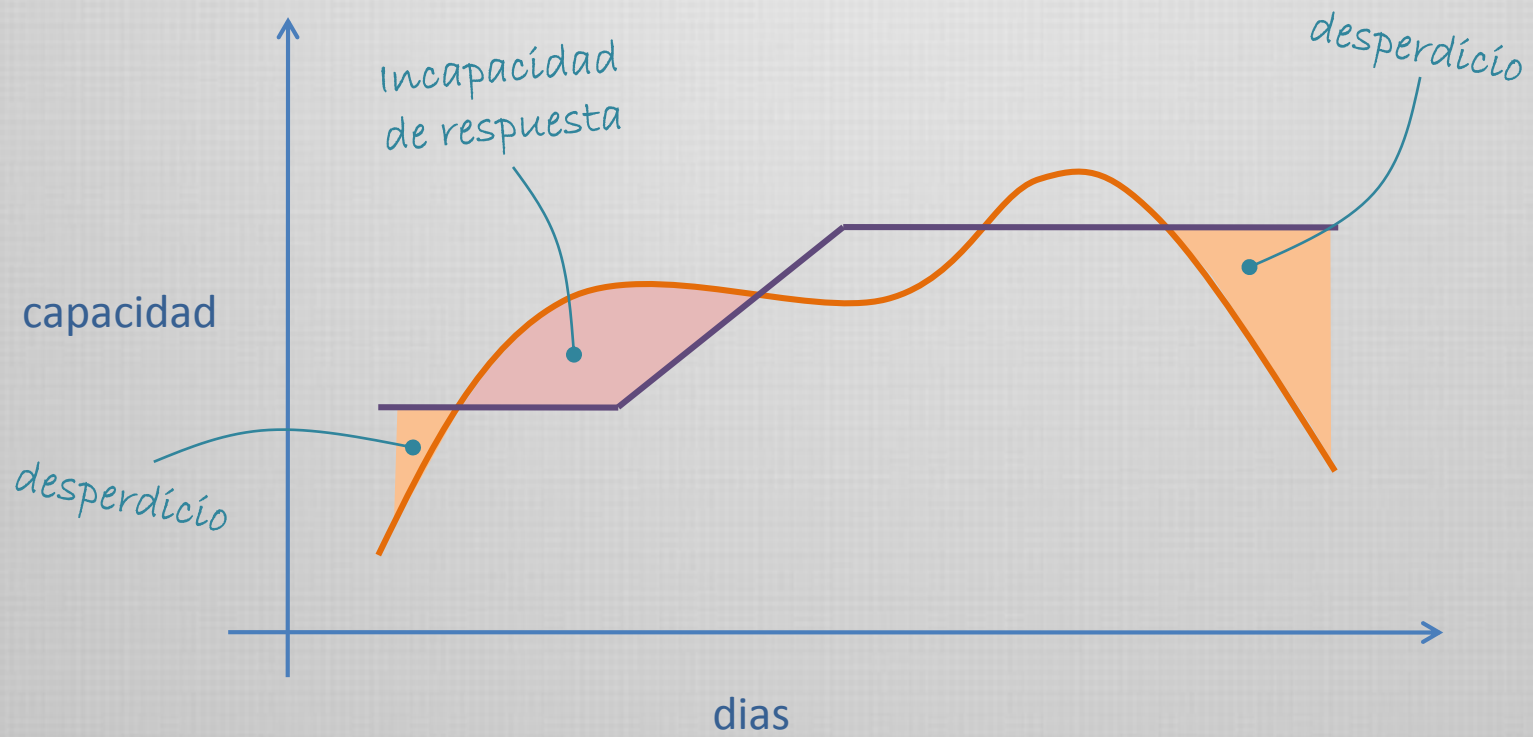
Cloud Computing

Características esenciales de cloud computing

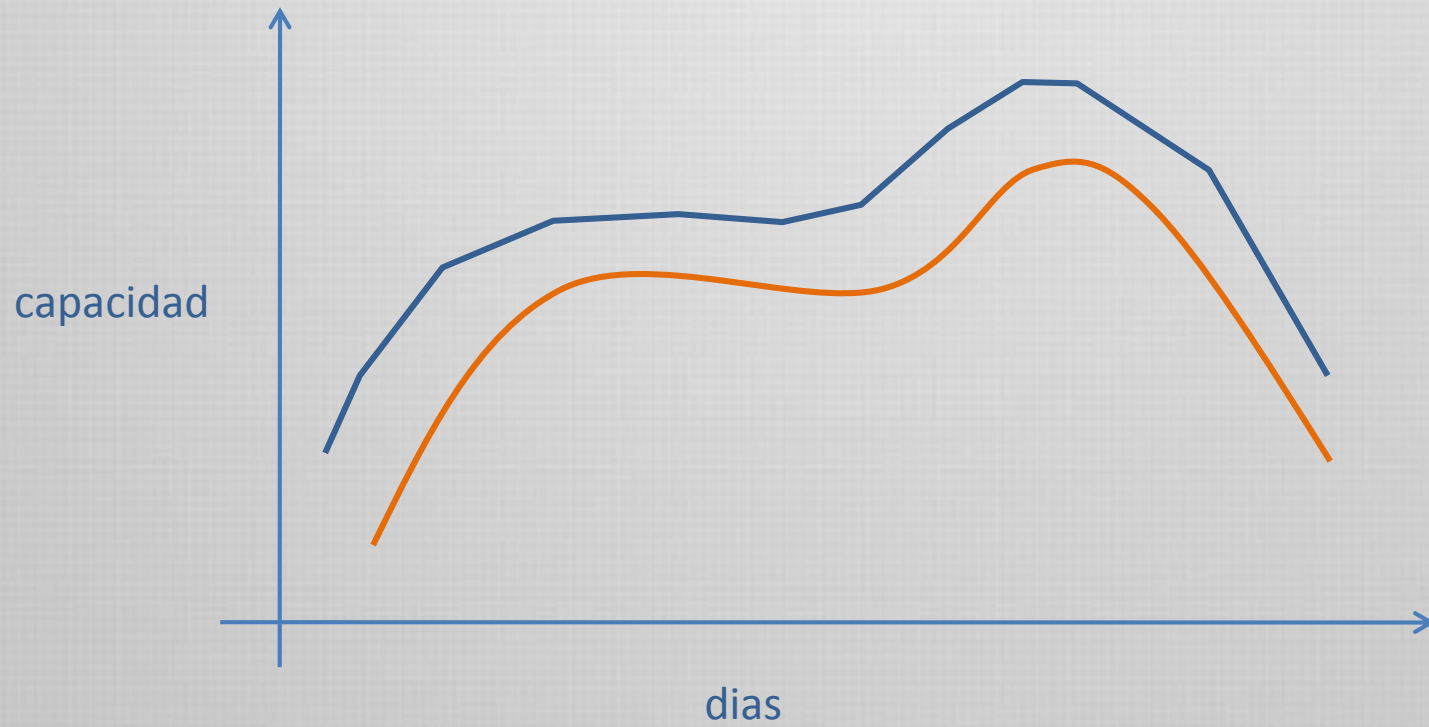
- Autoservicio en demanda** -----> Los servicios se proveen unilateralmente sin requerir interacción humana
- Acceso por red** -----> Los servicios están disponibles en la red en tiempo real a través de mecanismos estandarizados
- Resource pooling** -----> Atención paralelizada de los clientes, ajustados a la demanda actual de cada uno
- Elasticidad** -----> Los recursos se escalan apropiadamente. La percepción del cliente es la de recursos ilimitados
- Calidad** -----> Validación y tarifado basado en métricas cualitativas y cuantitativas en pos de la calidad del servicio.

National Institute of Standards and Technology (NIST)

Elasticidad



Elasticidad



Cloud Computing - providers



Infrastructure as a Service (IaaS)

Proveen a las organizaciones recursos computacionales como servidores, almacenamiento, soporte de red.

Amazon EC2, Dropbox, Google File System



Platform as a Service (PaaS)

Proveen una plataforma completa para el desarrollo de aplicaciones (basadas en la web).

Sistemas operativos, lenguajes, bases de datos. Tarifas variables

Google App Engine, Facebook, Microsoft Azure



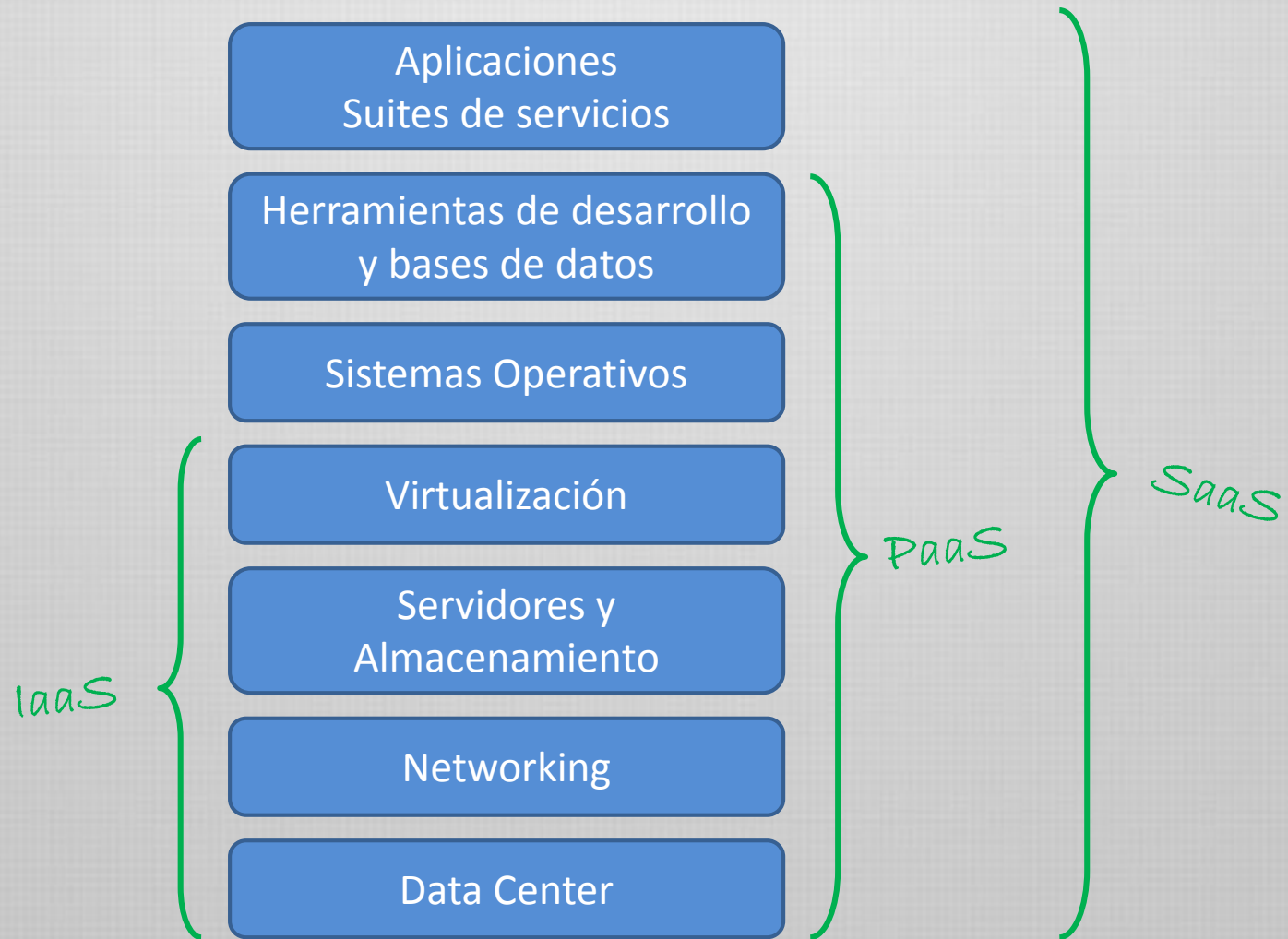
Software as a Service (SaaS)

Proveen aplicaciones por medio de servicios.

Mail, calendarios, redes sociales, administración de documentos.

Photoshop Express eCloud, Google Docs, Windows Live

Cloud Computing - providers



Cloud Computing

Algunas ofertas...

Amazon Simple Storage Service (S3).

Infraestructura de almacenamiento masivo.

Amazon SimpleDB

Base de datos en la nube, de acceso via servicios web.

Amazon Elastic Compute Cloud

Soluciones elásticas integrales. Servidores virtualizados

Google App Engine

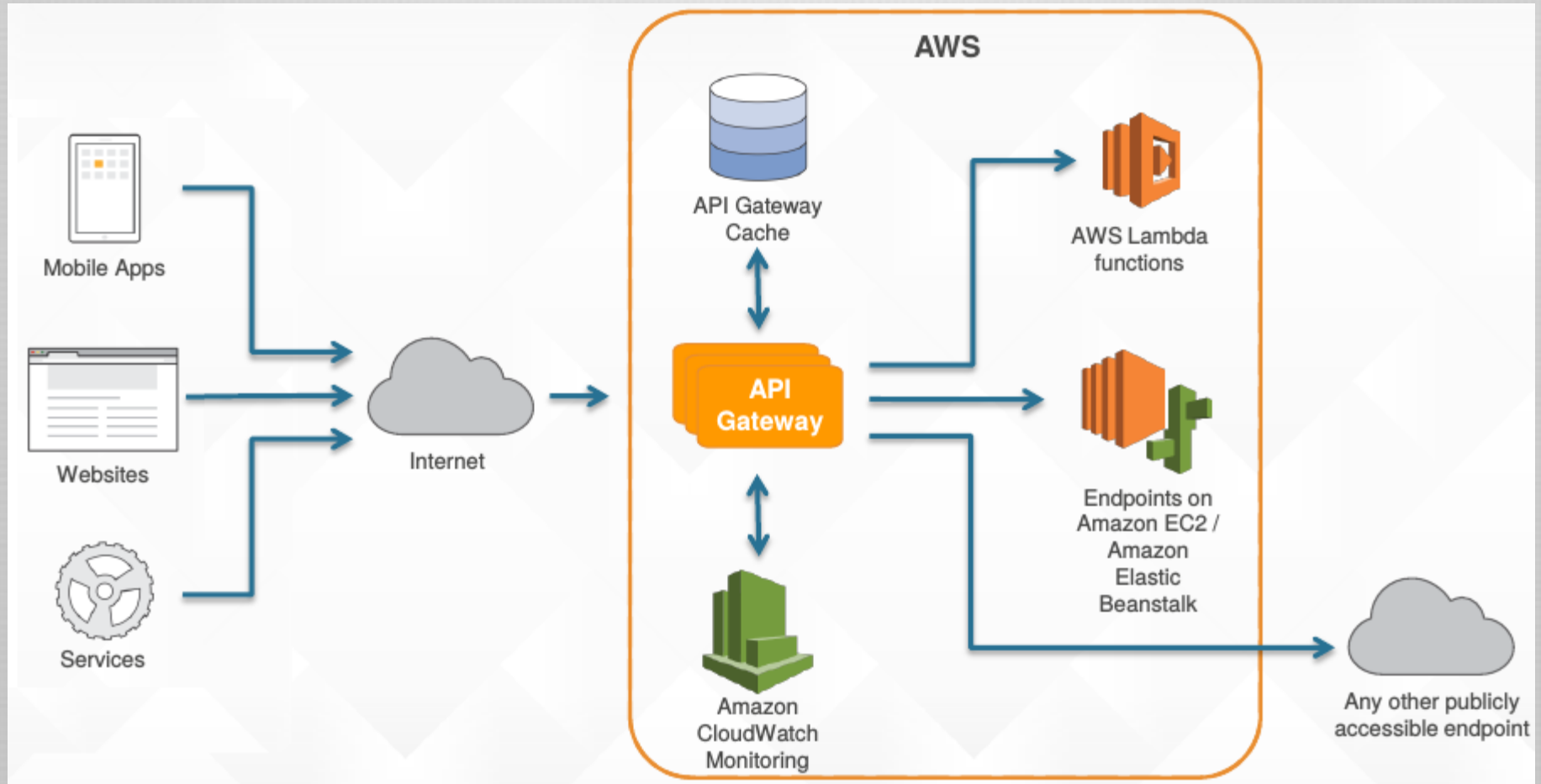
Servicio *Platform as a Service* de Google.

Infraestructura para el desarrollo de aplicaciones. Datastore, emails, users, etc.

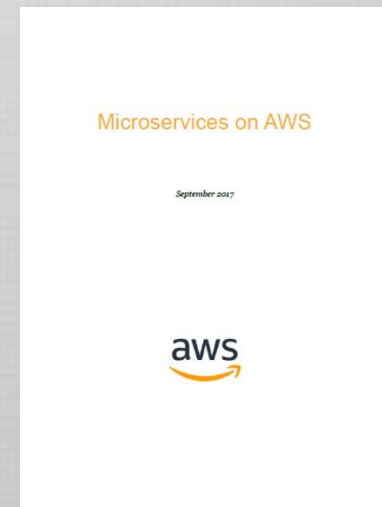
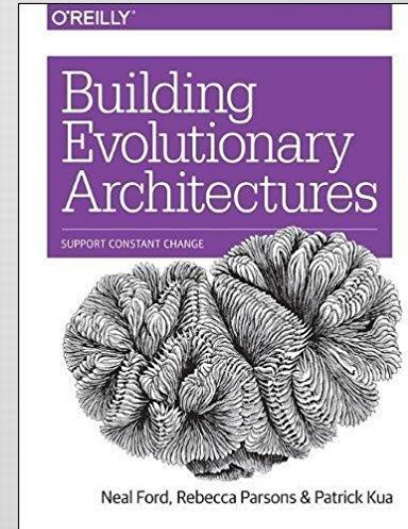
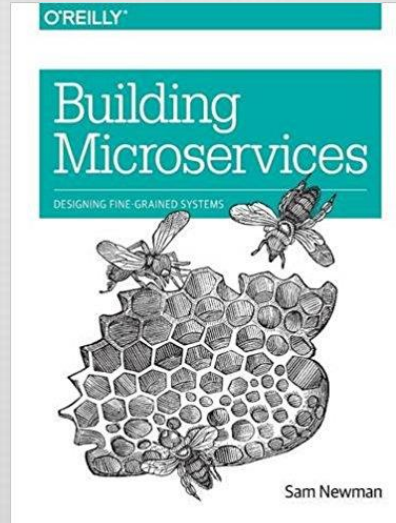
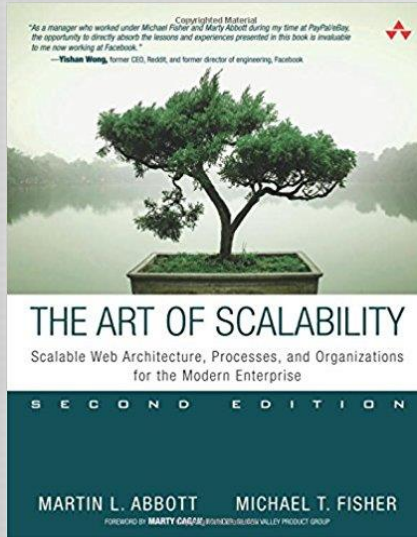
Windows Azure

Cloud Computing de Microsoft. Storage service, SQL service, desarrollo de aplicaciones.

AWS



Lecturas



1606.04036v1 [cs.SE] 13 Jun 2016

Lecturas

Martin Fowler website

<https://martinfowler.com/articles/microservices.html>

<https://martinfowler.com/articles/microservice-testing>

Chris Richardson's Microservices Blog

<http://microservices.io/>

Microservices • Martin Fowler

<https://youtu.be/wgdBVIX9ifA>

An Overview of Designing Microservices - March 2017 AWS Online Tech Talks

<https://youtu.be/ljs55IA8DIk>

Mastering Chaos - A Netflix Guide to Microservices

<https://youtu.be/CZ3wlurvHeM>