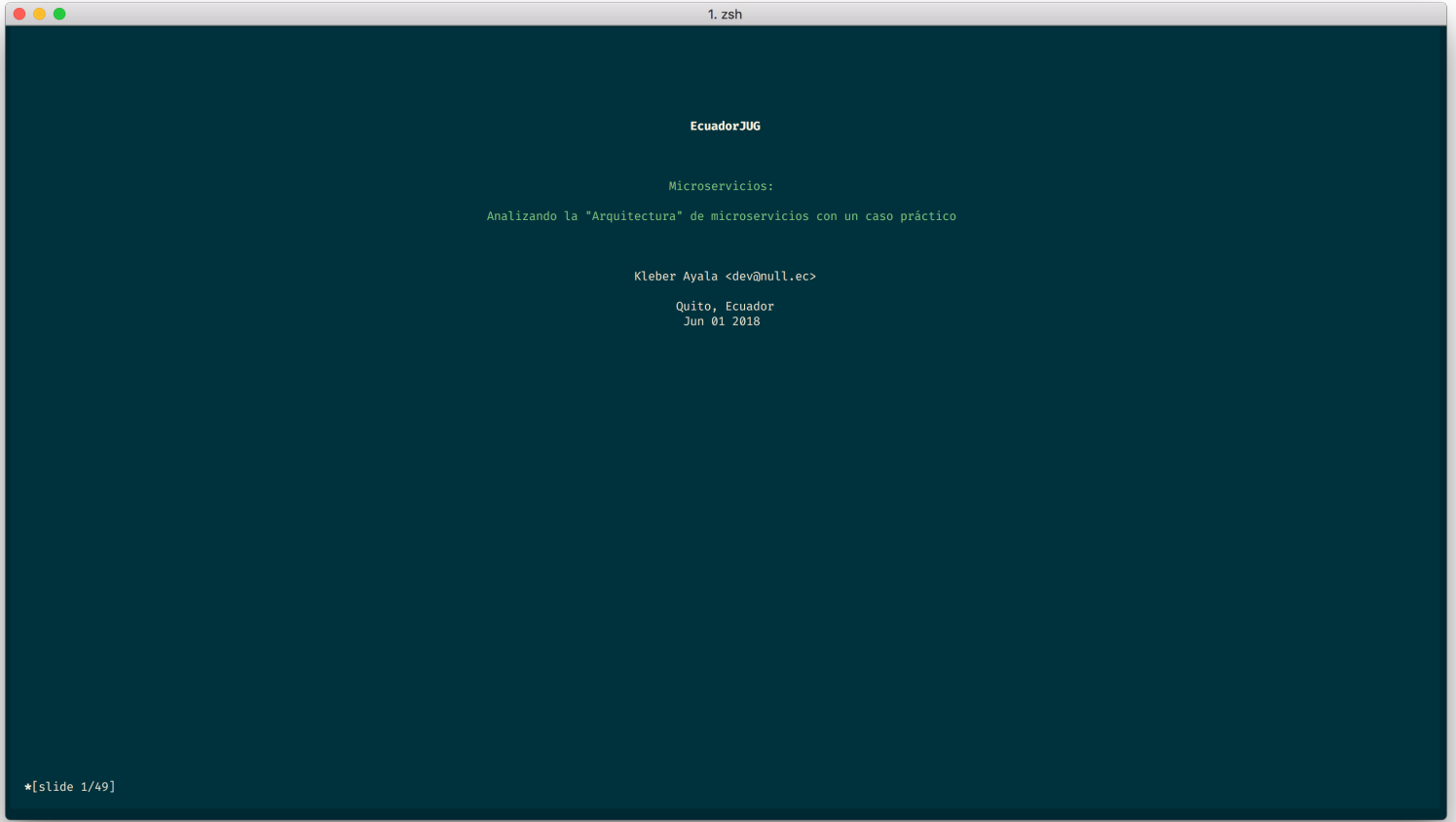
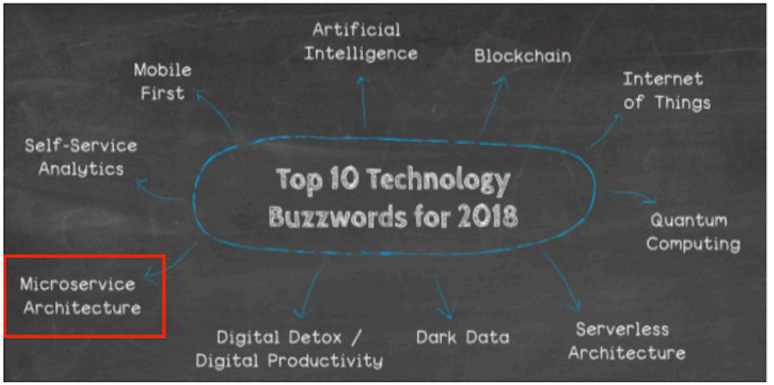

1. zsh
EcuadorJUG
Microservicios:
Analizando la "Arquitectura" de microservicios con un caso práctico
Kleber Ayala <dev@null.ec>
Quito, Ecuador
Jun 01 2018
*[slide 1/49]

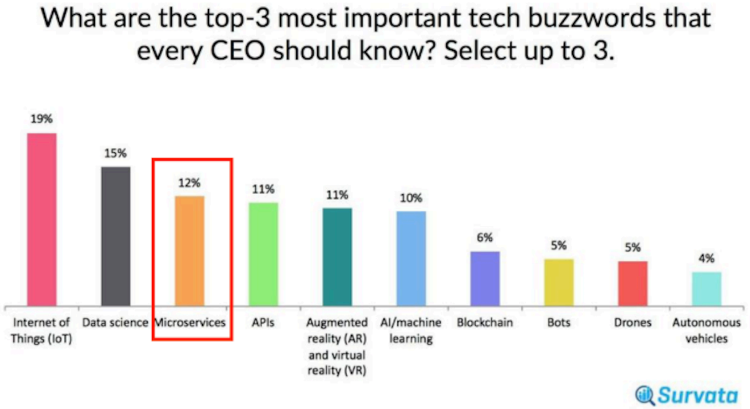
Buzzword Oriented Architecture

* block chain,
* virtual reality,
* serverless,
* self-driving cars,
* artificial intelligence,
* containers,
* **microservices,**
* …

Buzzword Oriented Architecture in a nutshell

* block chain = Una base de datos fancy
* virtual reality = teléfonos en cascos
* serverless = servidores en la nube de Amazon
* self-driving cars = control de crucero extremadamente caro
* artificial intelligence = miles de IF statements
* containers = dos líneas de configuración por línea de código
* **microservices = reduciendo la complejidad aumentando la complejidad**
* …


Artificial Intelligence
Blockchain
Mobile First
Internet of Things
Self-Service Analytics
Top 10 Technology Buzzwords for 2018
Quantum Computing
Microservice Architecture
Digital Detox / Digital Productivity
Dark Data
Serverless Architecture


What are the top-3 most important tech buzzwords that every CEO should know? Select up to 3.
19%
15%
12%
11%
11%
10%
6%
5%
5%
4%
Internet of Things (IoT)
Data science
Microservices
APIs
Augmented reality (AR) and virtual reality (VR)
AI/machine learning
Blockchain
Bots
Drones
Autonomous vehicles
Survata


Quantum Computing
Microservices
Dark Data
Technology Buzzwords in 2018
Digital Detox
Actionable Analytics

Luis Falcones
May 30, 07:55

Que temas estas interesado en tratar en el JUG ?

Microservicios

Xavier Pazmiño
May 30, 07:55

Que temas estas interesado en tratar en el JUG ?

docker en microservicioa

Paul Yaguachi
May 15, 21:44

Que temas estas interesado en tratar en el JUG ?

arquitectura de microservicios

danielkard
May 8, 09:22

Que temas estas interesado en tratar en el JUG ?

Practicidad de microservicios

Galo Latorre +1
May 4, 12:27

Que temas estas interesado en tratar en el JUG ?

Microservicios

giovanny cholca
Apr 28, 21:58

Que temas estas interesado en tratar en el JUG ?

microservicios

Que temas estas interesado en tratar en el JUG ?

microservicios con Java,java de, docker, swarm, kubernetes. Arquitectura concéntrica multiplataforma para apps móviles y web.

1. zsh

¿Qué es Buzzword Oriented Architecture?

```
.-------------------------------------------------------------------------------------------------------------------------------------------------------------.
| BOA es un estilo de arquitectura que se apoya en la adopción de tecnologías de moda,                                                                        |
| escuchando palabras de la industria que están de moda, sin mucha información y promesas                                                                     |
| de la panacea que solucionará todos tus problemas, incluso los que aun no los tienes.                                                                       |
`-------------------------------------------------------------------------------------------------------------------------------------------------------------'
```

*[slide 5/49]

Buzzword Oriented Architecture no es Nuevo

* De programación procedural ⟶ programación Orientación Objetos,
* De Orientación a Objetos ⟶ Component Based Development.
* De Object-oriented analysis and design (OOA/OOD), Model-driven architecture (MDA) ⟶ Domain-Driven Design.
* De Component Based Development ⟶ SOA
* De SOA ⟶ Microservicios

It's time to architect a solution for buzzword mania

SIMON TUCK
OTTAWA
PUBLISHED FEBRUARY 8, 2001
UPDATED APRIL 9, 2018

If you can't use the future tense without babbling on about "going forward," or you think the "global leader" you work for makes "end-to-end solutions" based on "a scalable, robust, object-oriented architecture that fully leverages XML" and not products, you may be fluent in techno-speak.

The technology industry, like many others, has created its own jargon. Those working for a "New Economy," "startup" or "pre-IPO" company -- many of which are "repositioning" their "client-focused" operations towards a "path to profits" -- may feel "empowered" by all the insider babble. But it's fog to many.

1. zsh

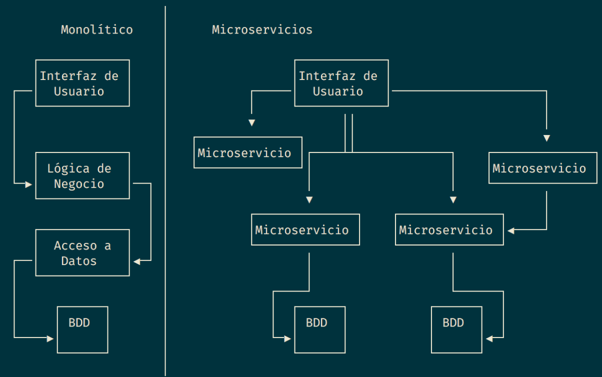
Microservicios

Monolítico	Microservicios	
Plataforma de transmisión de Video	Carga	Streaming
	Codecs	Descargas
	Recomendaciones	Suscripciones

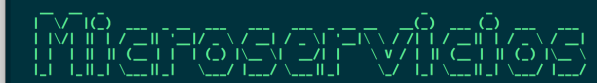
* El primero es una unidad grande única: un Monolítico.

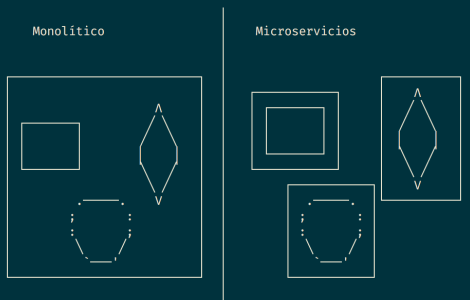
* El segundo es un conjunto de servicios pequeños y específicos. Cada servicio provee una funcionalidad concreta.

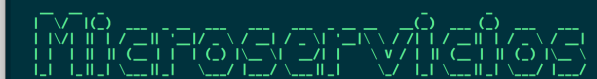
Microservicios



*[slide 8/49]

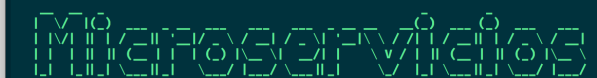



Monolítico
Microservicios



Ventajas potenciales:

* Desarrollo Independiente
* Despliegue independiente
* Escalabilidad independiente
* Reusabilidad

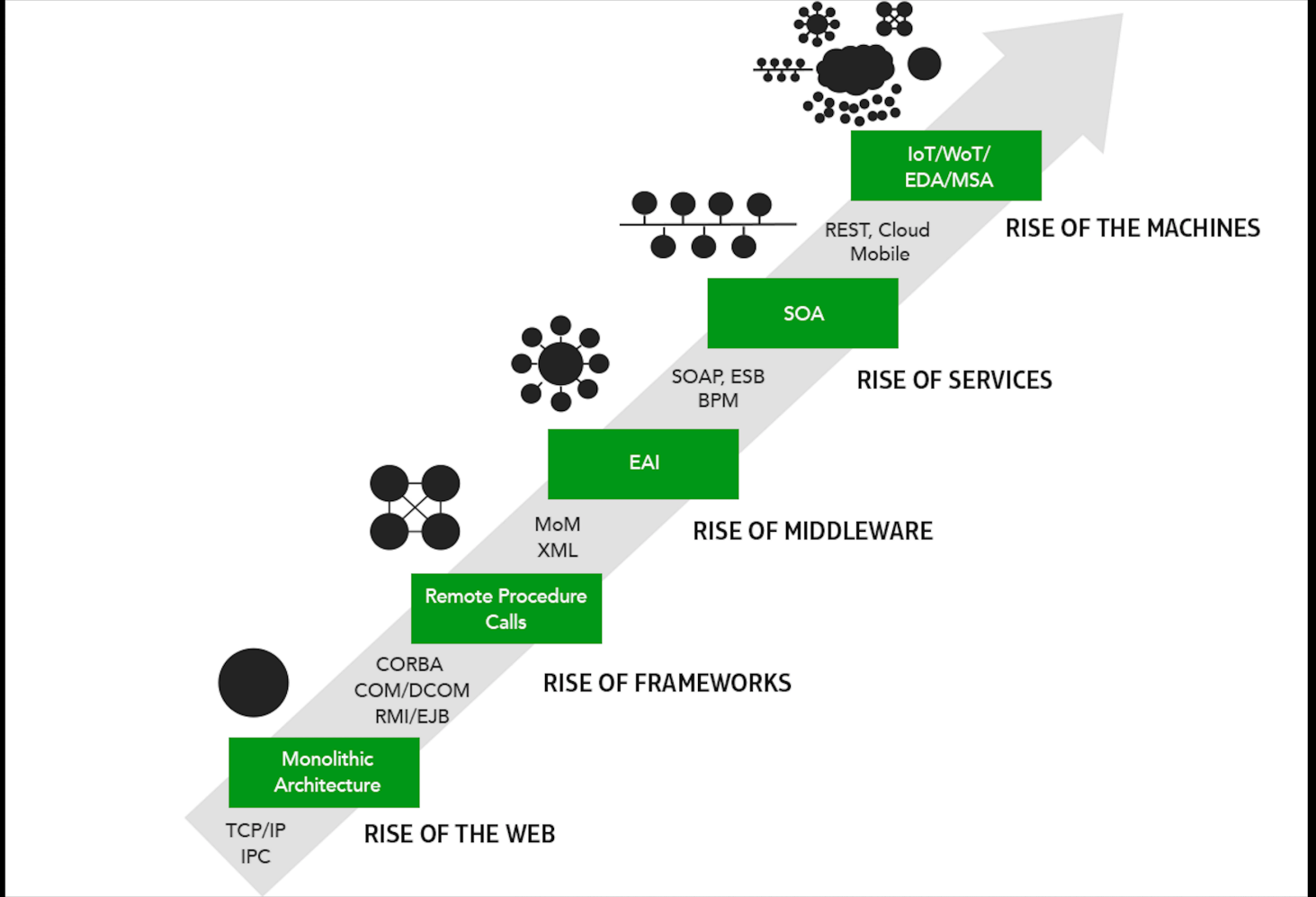


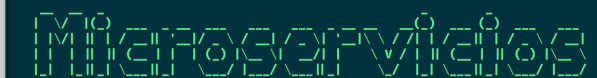
Si esto es tan genial, ¿por qué no se ha hecho antes?

Los problemas no son nuevos:

* Integración
* Escalabilidad
* Reusabilidad
* Mantenimiento
* Time to Market

¡Re Inventemos la rueda!


IoT/WoT/
EDA/MSA
REST, Cloud
Mobile
RISE OF THE MACHINES
SOA
SOAP, ESB
BPM
RISE OF SERVICES
EAI
MoM
XML
RISE OF MIDDLEWARE
Remote Procedure
Calls
CORBA
COM/DCOM
RMI/EJB
RISE OF FRAMEWORKS
Monolithic
Architecture
TCP/IP
IPC
RISE OF THE WEB



Otras iniciativas a lo largo del tiempo

* Orientación a Objetos (SOLID, GRASP)

* Desarrollo basado en componentes (Douglas McIlroy, Mass Produced Software Components, 1968).

* Domain-driven design (Evans, 2003).

* SOA, (Gartner, 1996)

* OSGI, 1999

The key goals of CBD are as follows:

- Save time and money when building large and complex systems: Developing complex software systems with the help of off-the-shelf components helps reduce software development time substantially. Function points or similar techniques can be used to verify the affordability of the existing method.
- Enhance the software quality: The component quality is the key factor behind the enhancement of software quality.
- Detect defects within the systems: The CBD strategy supports fault detection by testing the components; however, finding the source of defects is challenging in CBD.

Some advantages of CBD include:

Some advantages of CBD include:

- Minimized delivery:
  - Search in component catalogs
  - Recycling of pre-fabricated components
- Improved efficiency:
  - Developers concentrate on application development
- Improved quality:
  - Component developers can permit additional time to ensure quality
- Minimized expenditures

The specific routines of CBD are:

- Component development
- Component publishing
- Component lookup as well as retrieval
- Component analysis
- Component assembly

Related Terms

Component
Component Object Model (COM)
Aspect-Oriented Software Development

ciples OO

Videos Images News Maps More

12,100,000 results (0.54 seconds)

.L.I.D. STANDS FOR:

S — Single responsibility principle.
O — Open closed principle.
L — Liskov substitution principle.
I — Interface segregation principle
D — Dependency Inversion prin

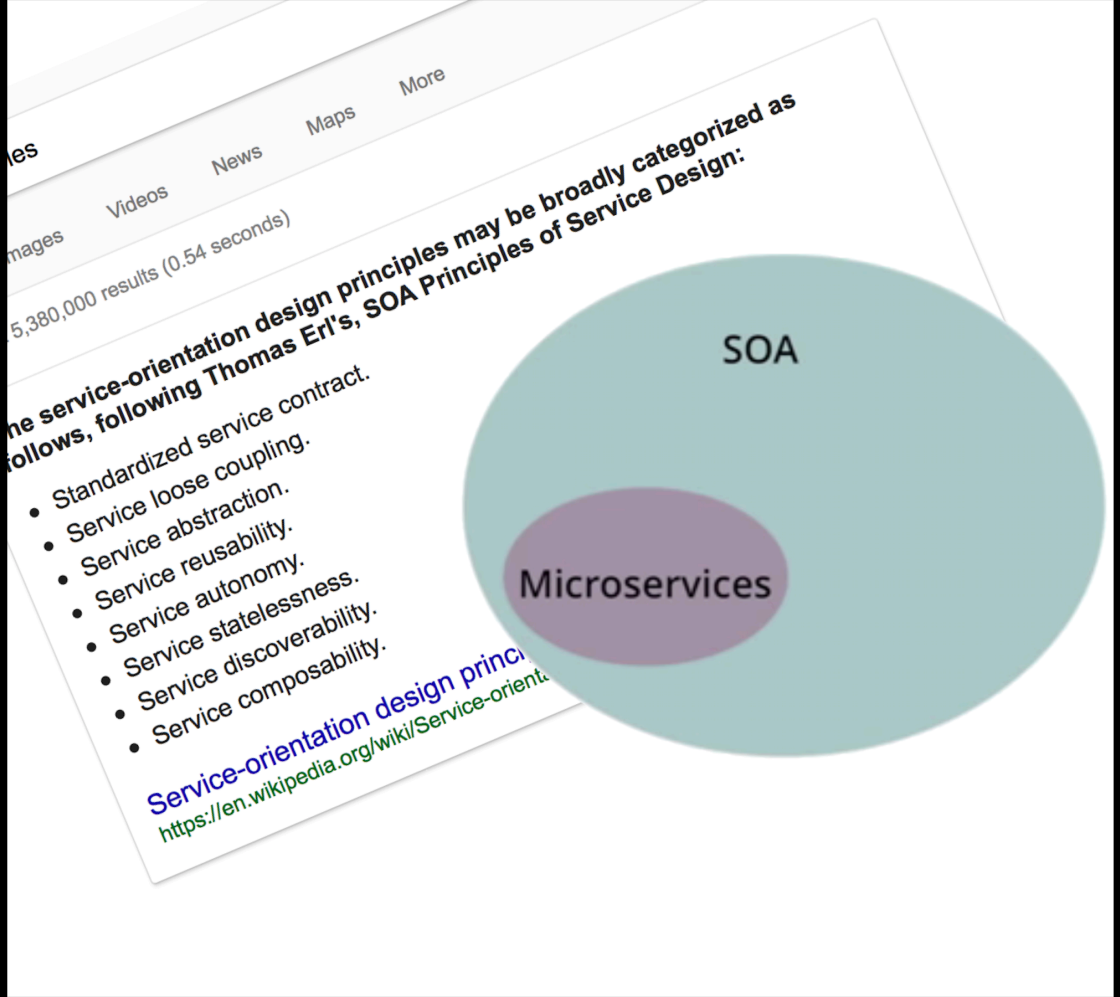
S.O.L.I.D The first 5 princir
https://medium.com/.../s-o-l-i-d-t'

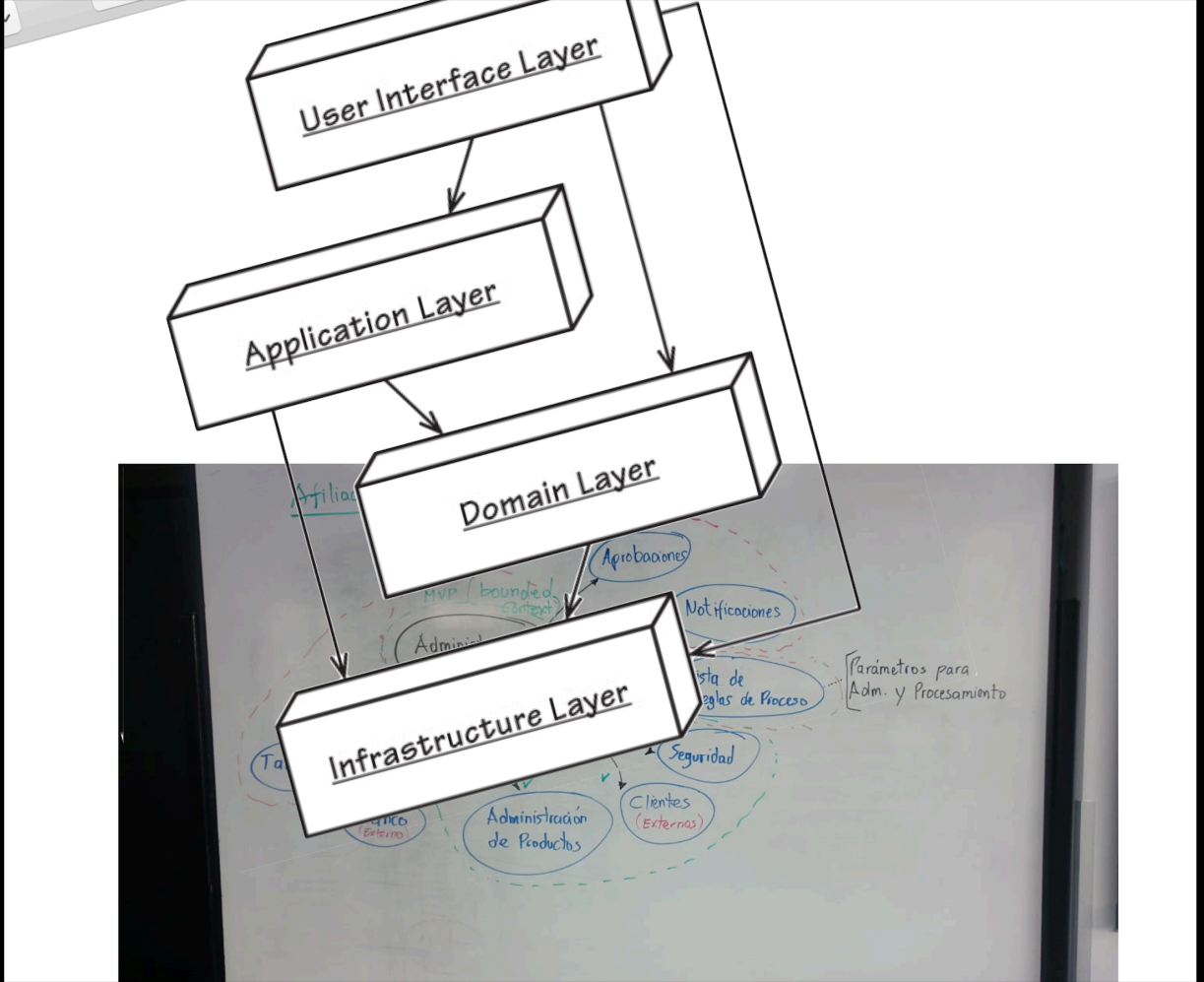
General responsibility assignment software patterns (or
objects in object-oriented design.

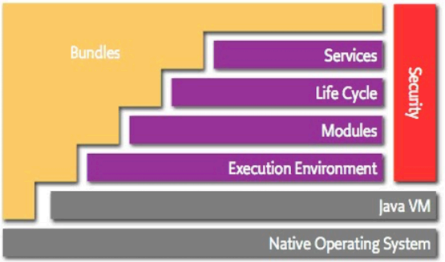
The different patterns and principles used in GRASP are contro
variations, and pure fabrication. All these patterns answer some
These techniques have not been invented to create new ways of w
object-oriented design.

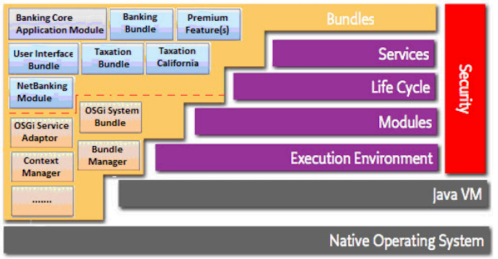
Computer scientist Craig Larman states that "the critical design tool fo
other technology."[1] Thus, GRASP are really a mental toolset, a learning

Contents [hide]


nes
Videos
News
Maps
More
mages
5,380,000 results (0.54 seconds)
he service-orientation design principles may be broadly categorized as
follows, following Thomas Erl's, SOA Principles of Service Design:
Standardized service contract.
Service loose coupling.
Service abstraction.
Service reusability.
Service autonomy.
Service statelessness.
Service discoverability.
Service composability.
Service-orientation design princ
https://en.wikipedia.org/wiki/Service-orient
SOA
Microservices


User Interface Layer
Application Layer
Domain Layer
Infrastructure Layer
Aprobaciones
Notificaciones
Parámetros para Adm. y Procesamiento
Seguridad
Clientes (Externos)
Administración de Productos

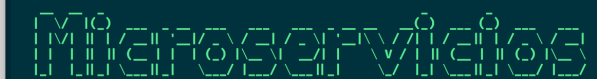




OSGi

The OSGi Alliance, formerly known as the Open Services Gateway initiative, is an open standards organization founded in March 1999 that originally specified and continues to maintain the OSGi standard. Wikipedia

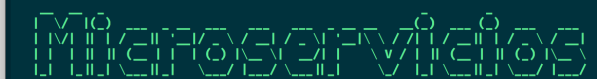
Founded: 2000



¿Pero que ha cambiado ahora?

* Popularidad de la tecnología de contenedores (Docker en particular).

* Tecnología de orquestación (como Kubernetes, Mesos, Consul, etc ...).



¿Que retos y problemas tenemos con los microservicios?

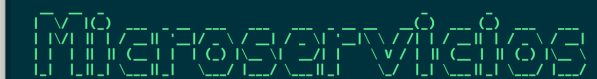
* Mayor complejidad para los desarrolladores

* Mayor complejidad para los operaciones

* Mayor complejidad para devops

* Requiere un nivel de expertcia y conocimiento muy alto

* Los sistemas del mundo real están muy mal delimitados



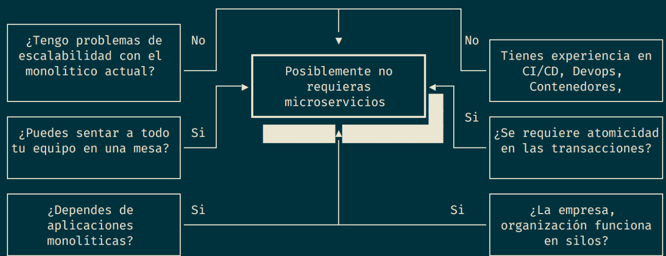
¿Que retos y problemas tenemos con los microservicios?

* Se obvian con frecuencia las complejidades de estado
* Las complejidades de la comunicación se ignoran a menudo
* Versionar se vuelve complicado
* Transacciones distribuidas
* Los microservicios pueden llegar a ser monolitos disfrazados

1. zsh

Microservicios

¿Que retos y problemas tenemos con los microservicios?

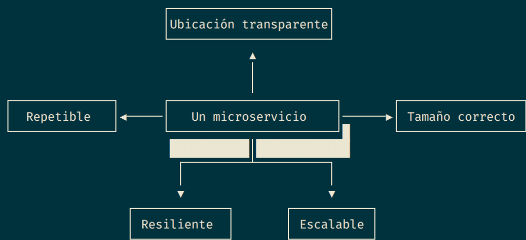


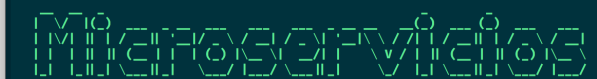
*[slide 16/49]

1. zsh

Microservicios

¿Que retos y problemas tenemos con los microservicios?





Otras consideraciones con microservicios

* Desarrollo en general
* Enrutamiento y Descubrimiento
* Resiliencia del cliente
* Seguridad
* Logging y tracing
* Compilación y despliegue

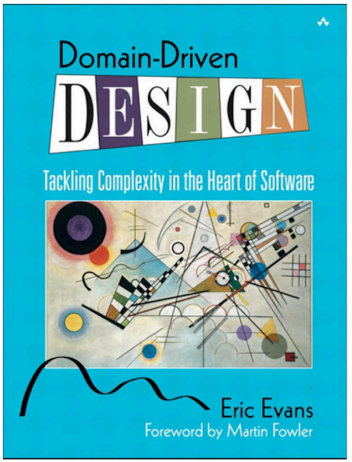
1. zsh

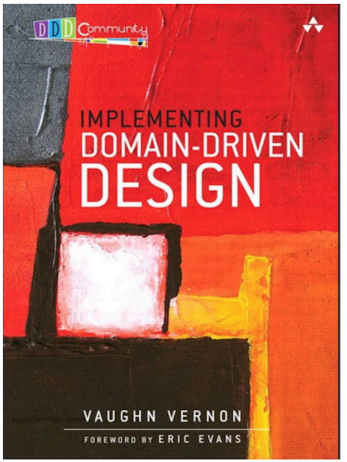
Microservicios

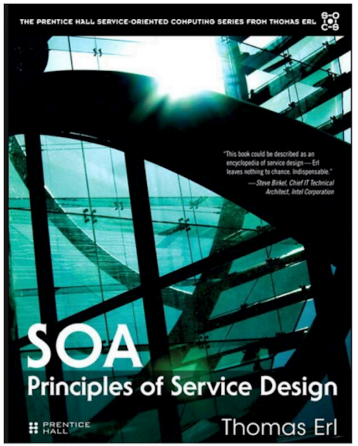
Desarrollo en general

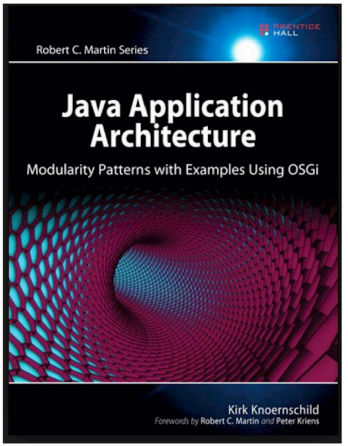
```
┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐
│ ┌───────────────┐ │
  │Granularidad   │
│ │del servicio   │ │
  └───────┬───────┘
│         ▼         │
  ┌───────────────┐
│ │Protocolos de  │ │
  │Comunicación   │
│ └───────┬───────┘ │
          ▼
│ ┌───────────────┐ │
  │Diseño de las  │
│ │ interfaces    │ │
  └───────┬───────┘
│         ▼         │
  ┌───────────────┐
│ │  Gestión de   │ │
  │configuración  │
│ └───────┬───────┘ │
          ▼
│ ┌───────────────┐ │
  │Procesamiento  │
│ │ de eventos    │ │
  └───────────────┘
└ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘
```


AN INTRODUCTION TO
COMPONENT-BASED
SOFTWARE DEVELOPMENT
KUNG-KIU LAU
SIMONE DI COLA
Series on Component-Based Software Development – Vol. 3


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler


DDD Community
IMPLEMENTING
DOMAIN-DRIVEN
DESIGN
VAUGHN VERNON
FOREWORD BY ERIC EVANS

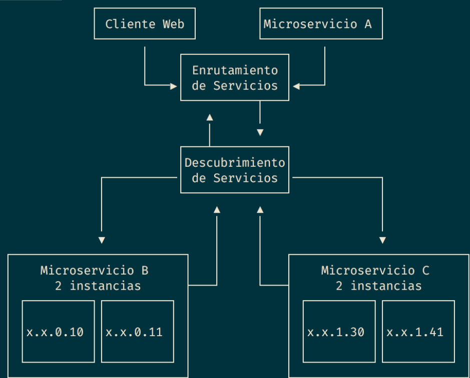

THE PRENTICE HALL SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL
"This book could be described as an encyclopedia of service design — Erl leaves nothing to chance. Indispensable."
—Steve Birket, Chief IT Technical Architect, Intel Corporation
SOA
Principles of Service Design
PRENTICE HALL
Thomas Erl

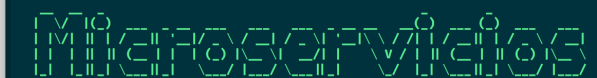

Robert C. Martin Series
PRENTICE HALL
Java Application Architecture
Modularity Patterns with Examples Using OSGi
Kirk Knoernschild
Forewords by Robert C. Martin and Peter Kriens

1. zsh

Microservicios

Enrutamiento y Descubrimiento





Enrutamiento y Descubrimiento

* Consul (https://www.consul.io/) Service Discovery and Configuration Made Easy

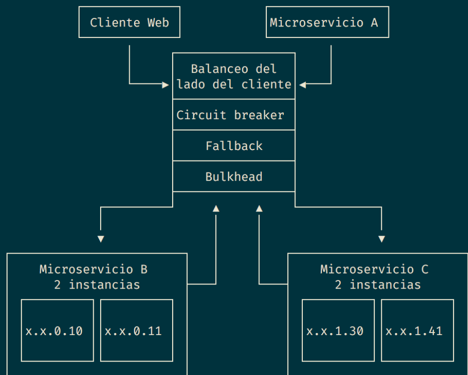
* Eureka (https://github.com/Netflix/eureka) REST based service used for locating services
                                               for the purpose of load balancing and failover of middle-tier servers.

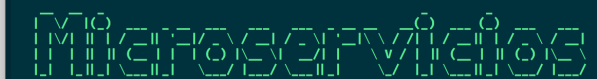
* Zookeeper (https://stackshare.io/zookeeper) Because coordinating distributed systems is a Zoo

1. zsh

Microservicios

Resiliencia del cliente

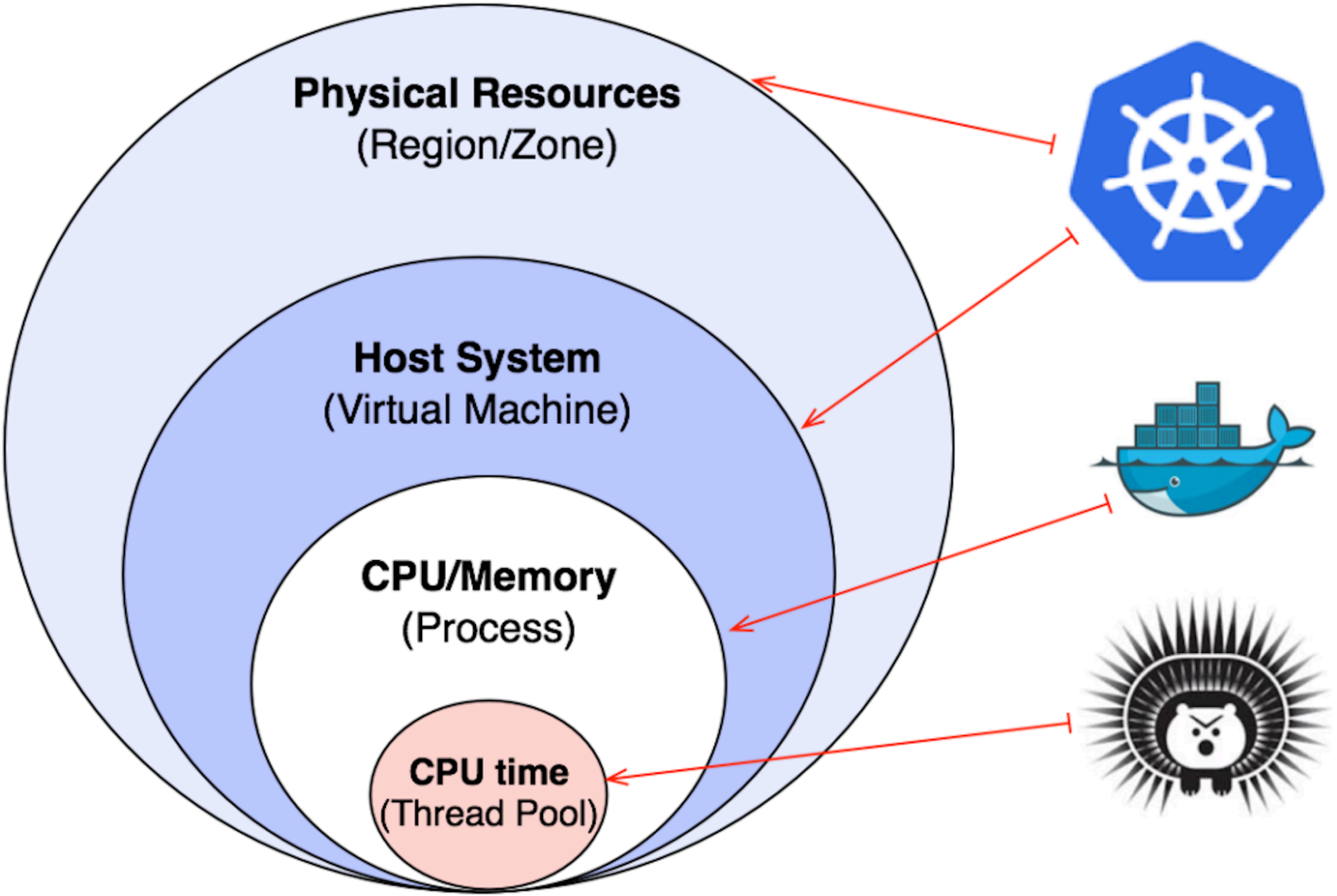


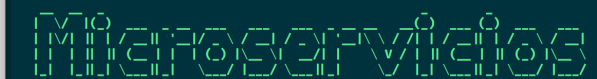


Resiliencia del cliente

* Resilience4j (https://github.com/resilience4j/resilience4j) Fault tolerance library designed for Java8 and functional programming

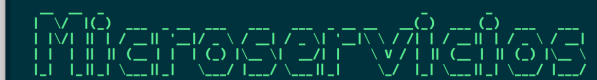
* Hystrix (https://github.com/Netflix/Hystrix) Latency and Fault Tolerance for Distributed Systems


Physical Resources
(Region/Zone)
Host System
(Virtual Machine)
CPU/Memory
(Process)
CPU time
(Thread Pool)



Seguridad

* Autenticación
* Control de acceso y autorización
* Asegurar APIs y aplicaciones
* No confiar en datos externos, validar / codificar / sanitizar
* Serialización / Deserialización segura
* Compartir "Secrets" de forma segura



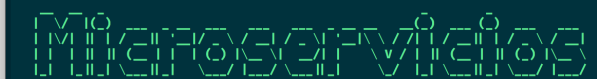
Seguridad

Malas Ideas:

* Credenciales en el código fuente

* Credenciales en Dockerfiles

* Usar variables de entorno para pasar credenciales



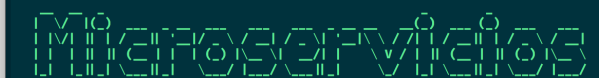
Seguridad

Opciones:

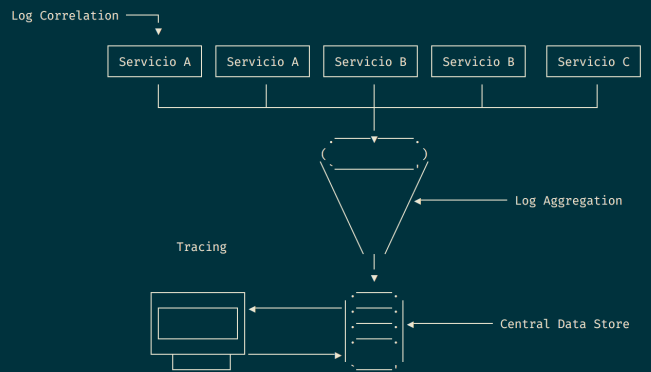
* Kubernetes → native secrets management API
  http://kubernetes.io/docs/user-guide/secrets

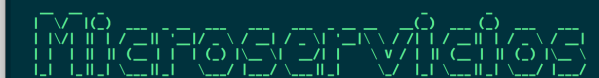
* Docker/Swarm
  Secret Management https://github.com/docker/docker/pull/27794

* DC/OS - Secrets API
  https://docs.mesosphere.com/1.8/administration/secrets/secrets-api/



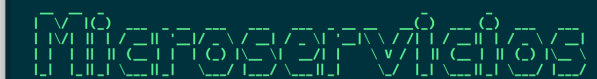
Logging y tracing





Logging y tracing

* Cloud Sleuth (https://cloud.spring.io/spring-cloud-sleuth/) Spring Cloud Sleuth implements a distributed tracing solution for Spring Cloud
* Zipkin (https://zipkin.io) Distributed tracing system
* ELK Stack (https://www.elastic.co/elk-stack) Elasticsearch, Logstash, and Kibana



Compilación y despliegue

* Pipelines
* Infraestructura como código
* Servidores inmutables

1. zsh

Microservicios

Compilación y despliegue

Compilación	Pruebas unitarias/integración	Empaquetamiento/despliegue de artefactos	Creación de Imagenes	Commit repo a Registry

Desarrollo

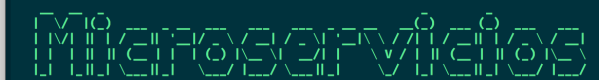
- prueba de plataforma
- Despliegue imagen / nuevo servidor desplegado

Pruebas

- prueba de plataforma
- Despliegue imagen / nuevo servidor desplegado

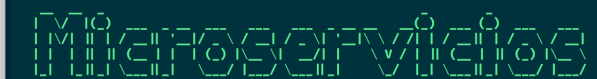
Producción

- prueba de plataforma
- Despliegue imagen / nuevo servidor desplegado



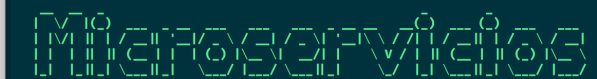
Frameworks de Java populares

* Spring Cloud (http://projects.spring.io/spring-cloud/)
* Dropwizard (https://www.dropwizard.io/1.3.2/docs/)
* WildFly Swarm (http://wildfly-swarm.io)
* Istio (https://istio.io)
* Karaf (https://karaf.apache.org)
* Netflix OSS (https://netflix.github.io)
* Docker (https://www.docker.com)
* Kubernetes (https://kubernetes.io)



Frameworks y herramientas Sugeridas

* Netflix Eureka: para descubrimiento de servicios
* Consul: para guardar configuraciones, parámetros
* Zuul: para ruteo inteligente y programable
* Netflix Ribbon: para balanceo de carga del lado del cliente
* Zipkin: para trazabilidad
* Turbine: para agregación de métricas
* Netflix Feign: para implementar Clientes REST Declarativos
* Hystrix: circuit breaker
* RabbitMQ: broker de mensajería



"Un caso práctico"

Antecedentes:

* Imaginemos un banco grande que ya tiene sus sistemas monolíticos funcionando mal o bien.

* Los días picos, como quincenas, fin de mes y feriados siempre tienen problemas especialmente en la banca en línea.

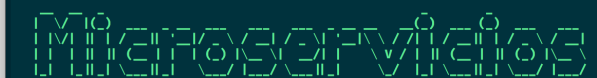
* Según las estadísticas, las funcionalidades de consulta de autenticación, posición consolidada, transferencias son las que más peticiones tienen en horas pico.

* La banca en línea es un war que tiene todas las funcionalidades embebidas, más de 12 años código espagueti.

* El banco funciona en silos, las áreas de desarrollo, release y operaciones no trabajan en equipo, utilizan un proceso manual y en cascada para la puesta en producción.

* El core bancario es un Monolítico escrito en Cobol que funciona en un MainFrame

* El banco también escucho de los beneficios de usar microservicios y quiere subirse al tren de la modernidad.



"Un caso práctico"

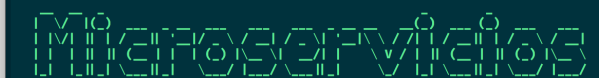
¿Qué problemas vemos?

* No tienen experiencia en CI/CD, DevOps, Contenedores

* Las transferencias requieren atomicidad

* Tiene dependencias de aplicaciones monolíticas

.--.
| ¿Es prudente hacer microservicios en este escenario? |
`--'



"Un caso práctico"

¿Qué podemos hacer?

Empezar con pasos pequeños:

* La aplicación no usa un building tool ⟶ Maven o Gradle

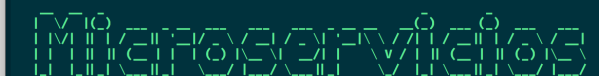
* No versionamos los artefactos ⟶ maven release plugin, Semantic Versioning https://semver.org

* No realizamos despliegue de los artefactos compilados ⟶ Nexus, Artifactory

* No utilizamos herramientas de Integración y Despliegue continuo: Jenkins

* No realizamos pruebas unitarias ⟶ Empecemos con realizar pruebas a los issues nuevos.

* No realizamos funcionales automáticas ⟶ Selenium, Cucumber.



"Un caso práctico"

¿Qué podemos hacer?

Siguientes pasos:

* Crear una imagen de Docker del monolítico.

* Empezar a utilizar en ambientes de desarrollo y pruebas la imagen

* Desplegar la imagen de Docker a un Docker Registry.

* Si todo va bien, podemos empezar a realizar en producción una mezcla de instancias de servidores y contenedores.

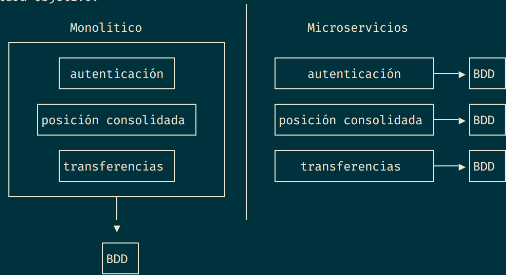
* Si todo va bien, orquestemos con kubernetes

1. zsh

Microservicios

"Un caso práctico"

Arquitectura Objetivo:



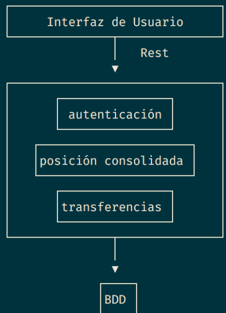
*[slide 37/49]

1. zsh

Microservicios

"Un caso práctico"

Separamos la interfaz de usuario, debería verse exactamente igual a la original:

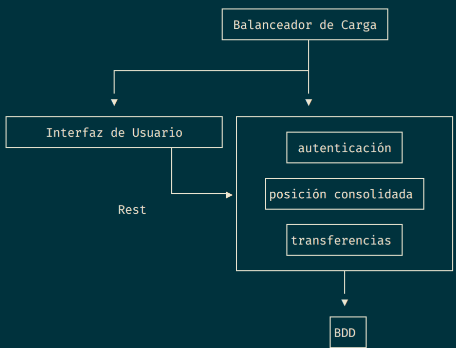


Es el mismo Monolítico
anterior sin modificar nada

1. zsh

Microservicios

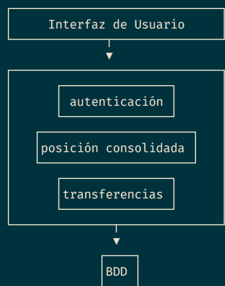
"Un caso práctico"
Primer paso a producción:



Es el mismo Monolítico
anterior sin modificar nada

Microservicios

"Un caso práctico"
Segundo paso a producción, quitamos la vieja Interfaz de Usuario:



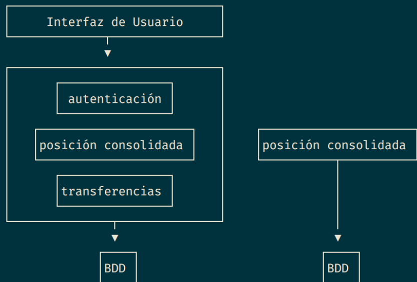
*[slide 40/49]

1. zsh

Microservicios

"Un caso práctico"

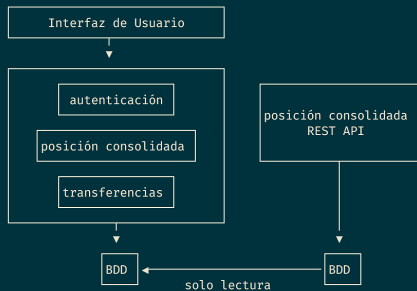
Creamos un nuevo servicio independiente de posición consolidada

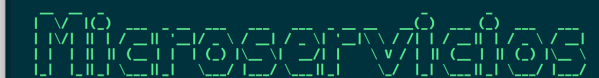


1. zsh

Microservicios

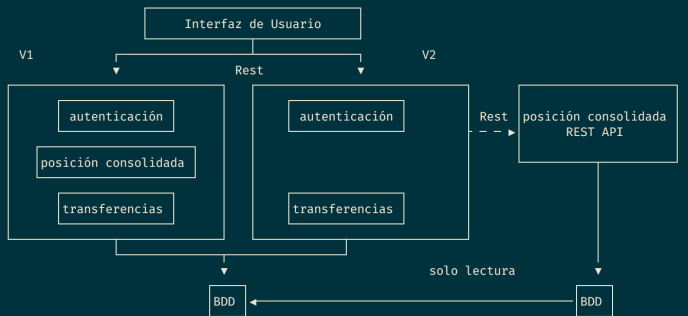
"Un caso práctico"
Creamos un nuevo servicio independiente de posición consolidada





"Un caso práctico"

Tercer paso a producción, servicio independiente de posición consolidada (dark launch)

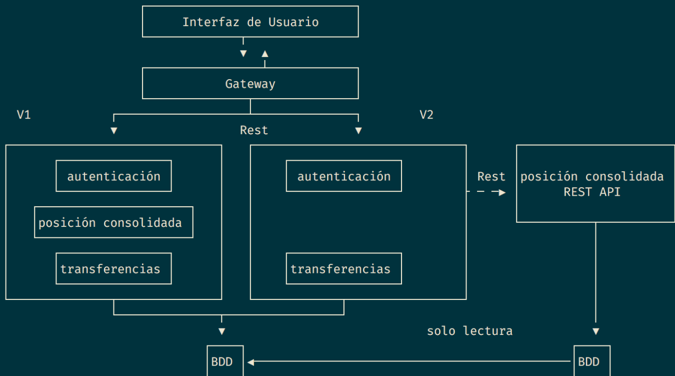


1. zsh

Microservicios

"Un caso práctico"

Podemos usar un Gateway para un mejor control sobre las peticiones y seguimos validando

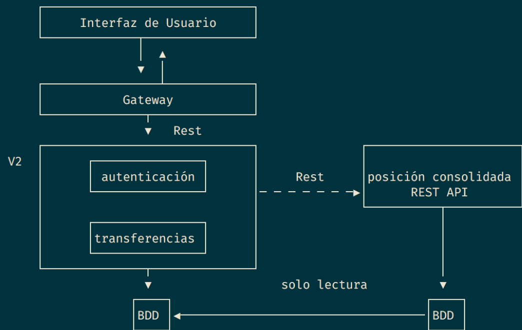


1. zsh

Microservicios

"Un caso práctico"

Si todo va bien, damos de baja a la versión 1 del backend



*[slide 45/49]

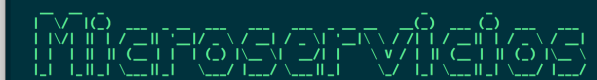
1. zsh

Microservicios

"Un caso práctico"

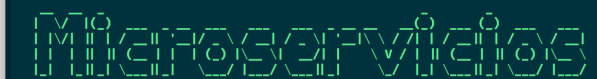
Si todo vsale bien, buscamos una forma de desconectar las bases de datos

```
                 ┌──────────────────────────┐
                 │   Interfaz de Usuario    │
                 └──────────────────────────┘
                         │  ▲
                         ▼  │
                 ┌──────────────────────────┐
                 │         Gateway          │
                 └──────────────────────────┘
                           │
                           ▼    Rest
                 ┌──────────────────────────┐           ┌───────────────────┐
          V2     │    ┌──────────────┐      │   Rest    │posición consolidada│
                 │    │autenticación │      │           │     REST API      │
                 │    └──────────────┘      ├──────────▶└───────────────────┘
                 │    ┌──────────────┐      │                     │
                 │    │transferencias│      │                     │
                 │    └──────────────┘      │                     │
                 └──────────────────────────┘                     │
                           │                                      │
                           ▼                                      ▼
                        ┌─────┐                                ┌─────┐
                        │ BDD │                                │ BDD │
                        └─────┘                                └─────┘
```



"Un caso práctico"

Si todo va bien, continuamos con el próximo servicio :)



"Un caso práctico Conclusiones"

* Hacer microservicios no es nada fácil y tampoco es barato

* La “arquitectura” de microservicios tiene que ver más con los procesos técnicos de empaquetado y operaciones que con el diseño intrínseco del sistema.

* Los límites apropiados de los componentes continúan siendo uno de los retos más importantes.

* La experiencia en DevOps CI/CD es primordial

* Mantengamos nuestras arquitecturas simples, identifiquemos limites apropiados de los componentes.

* Es mejor empezar con un monolítico bien diseñado, que pueda desacoplar funcionalidades, que tener un espagueti de servicios difícil de controlar

1. zsh

Microservicios

"Un caso práctico Conclusiones"

Preguntas?