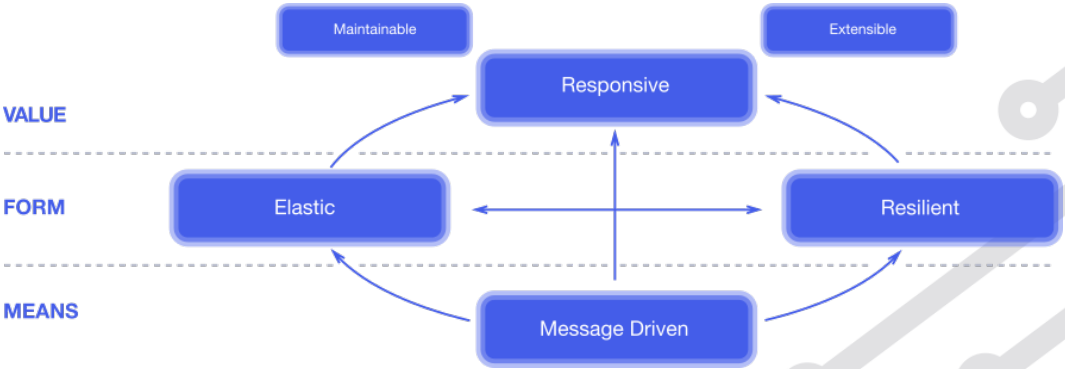
Desarrollo moderno con DevOps y Cloud Native

Víctor Orozco

23 de marzo de 2021

Nabenik

Aplicaciones reactivas



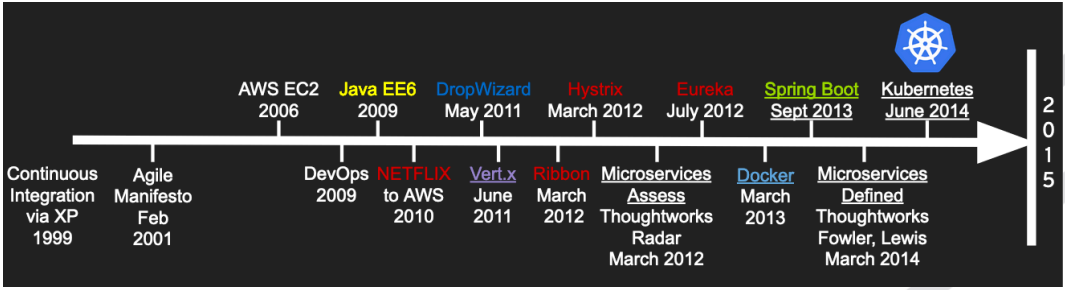
Cloud native

- Sistemas reactivos
- 12 factores cloud native
- Design patterns
- Domain Driven Design
- Microservices chassis e/ou service mesh
- Orquestación de contenedores

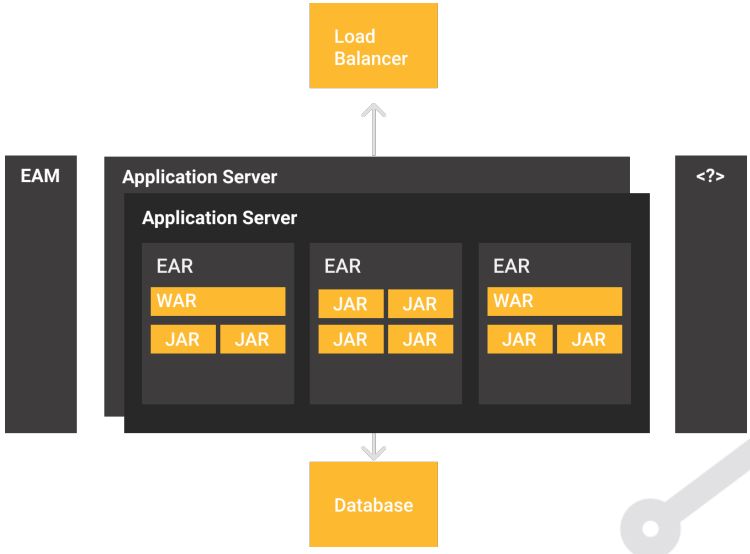
Cloud native

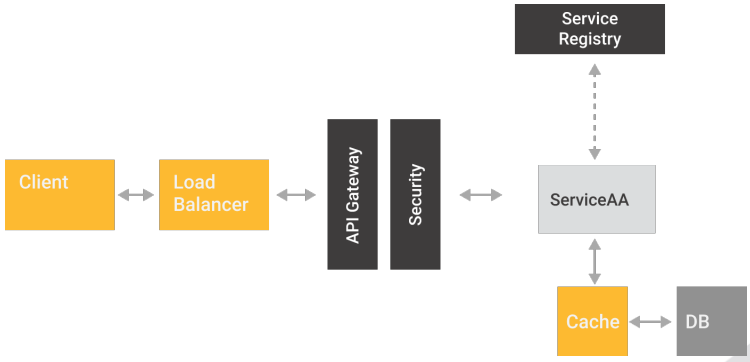
- (Nos gustaria tener) Sistemas reactivos
- (Es posible con la metodología de) 12 factores Cloud Native
- (Usamos soluciones probadas mediante) design patterns
- (Fragmentamos el sistema mediante) Domain Driven Design
- (Implementamos los servicios con) Microservices chassis y/o service mesh
- (Hacemos deployment) Mediante orquestación de containers

Historia

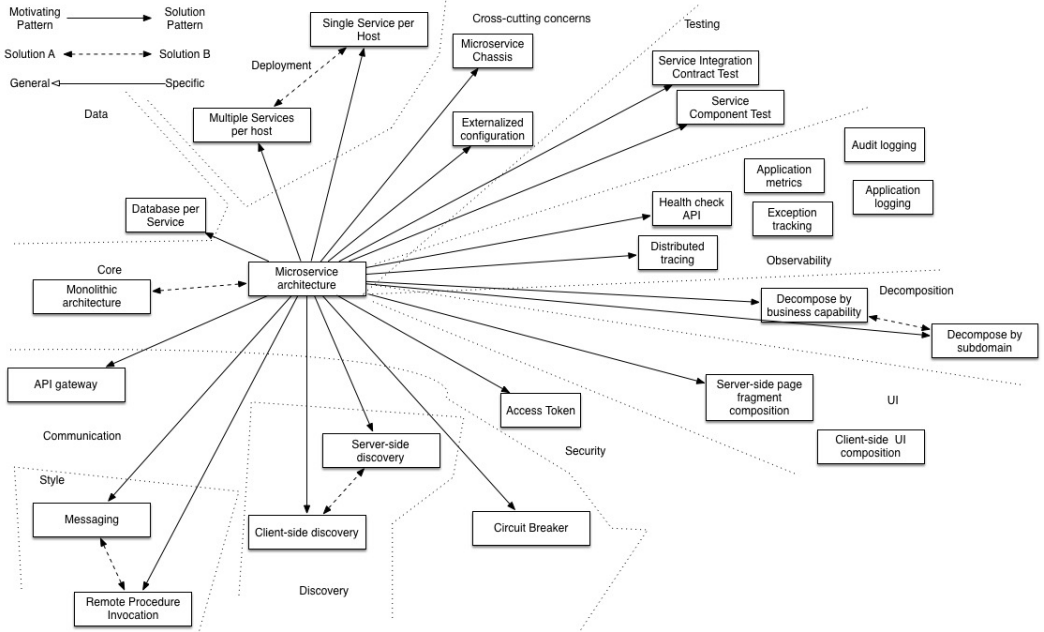


Créditos: Rafael Benevides

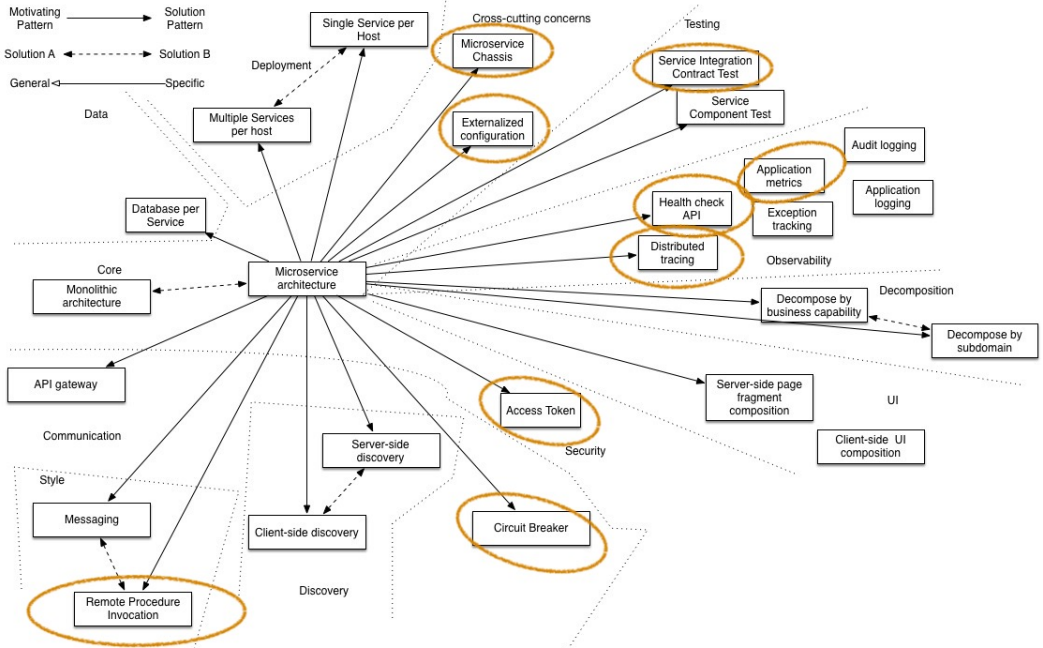




Patrones nuevos


Motivating Pattern
Solution Pattern
Solution A
Solution B
General
Specific
Data
Deployment
Single Service per Host
Multiple Services per host
Database per Service
Cross-cutting concerns
Microservice Chassis
Externalized configuration
Testing
Service Integration Contract Test
Service Component Test
Audit logging
Application metrics
Application logging
Health check API
Exception tracking
Distributed tracing
Observability
Core
Monolithic architecture
Microservice architecture
Decomposition
Decompose by business capability
Decompose by subdomain
API gateway
Server-side page fragment composition
UI
Client-side UI composition
Access Token
Security
Communication
Server-side discovery
Style
Messaging
Client-side discovery
Circuit Breaker
Discovery
Remote Procedure Invocation

T)


Motivating Pattern
Solution Pattern
Solution A
Solution B
General
Specific
Data
Deployment
Single Service per Host
Multiple Services per host
Database per Service
Cross-cutting concerns
Microservice Chassis
Externalized configuration
Testing
Service Integration Contract Test
Service Component Test
Audit logging
Application metrics
Application logging
Health check API
Exception tracking
Distributed tracing
Observability
Core
Monolithic architecture
Microservice architecture
Decomposition
Decompose by business capability
Decompose by subdomain
API gateway
Server-side page fragment composition
UI
Client-side UI composition
Access Token
Security
Communication
Server-side discovery
Style
Messaging
Client-side discovery
Circuit Breaker
Remote Procedure Invocation
Discovery

Microservice Chassis

Microservice Chassis

Chassis

Los frameworks Cloud Native son soluciones para problemas **cross-cuting concerns**.

Chassis Java

Spring Boot, MicroProfile, DropWizard, Akka

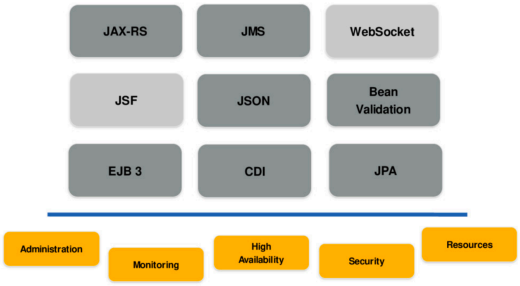
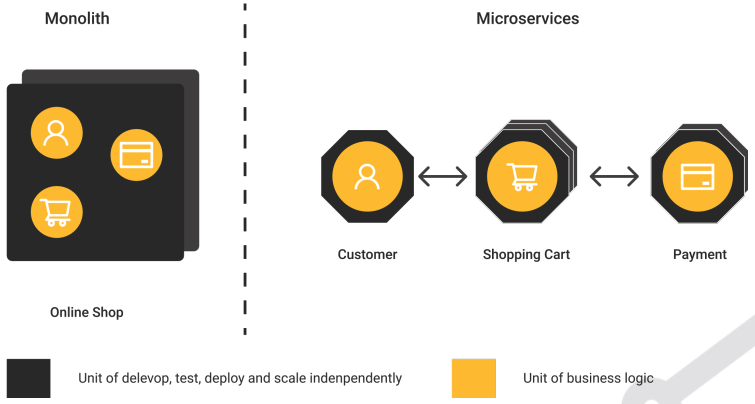
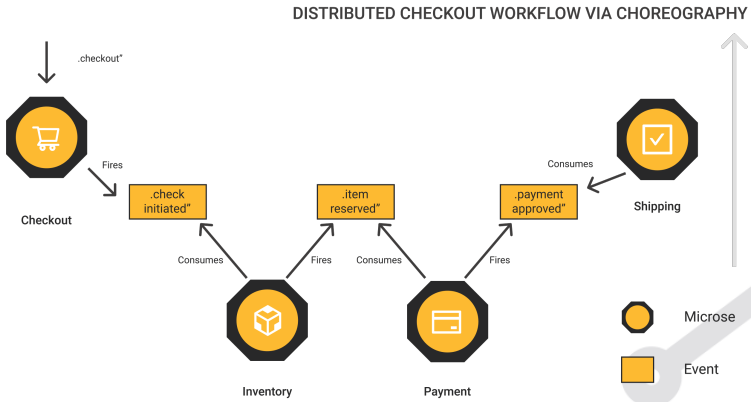


Figura 1: Créditos: Reza Rahman

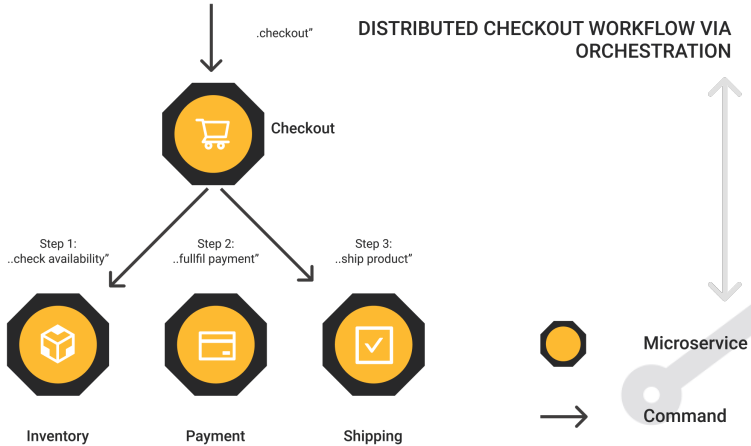
Microservice Chassis



Patterns complementarios - Event Sourcing, CQRS



Patterns complementarios - SAGA



Cross-cutting en el mundo real

- **Health checks & Metrics** - Recolectar métricas (Prometheus/Grafana) y establecer reglas de despliegue
- **Resilence & Fault Tolerance** - Service mesh -e.g. Likerd, Istio- y Chassis
- **Configuration** - Inyección de configuración en el ambiente
- **Authentication & Authorization** - API Gateway + Chassis
- **Standarized documentation** - OpenAPI + Swagger Server
- **Tracing** - Chassis + Zipkin
- **Remote Procedure & Messaging** - Chassis + K8S service discovery

12 factores Cloud Native

12 factores cloud native (Heroku)

Frameworks

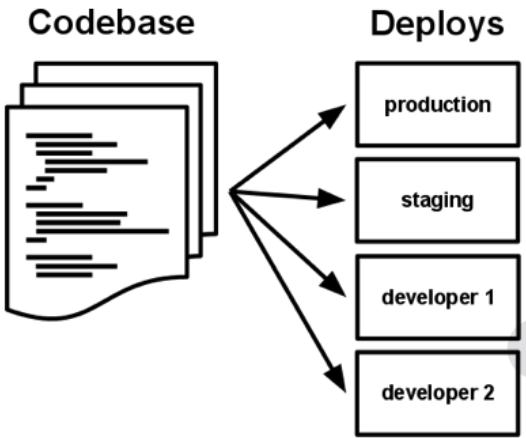
- Config
- Backing service
- Disposability

Cloud

- Codebase (Git-Flow)
- Dependencies (Maven)
- Build, Release, Run
- Processes
- Port binding
- Concurrency (Docker - k8s)
- Dev / Prod parity
- Logs
- Admin process

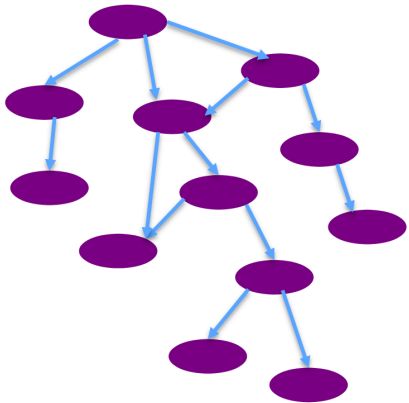
Codebase

- Una base de código con múltiples entornos de despliegue
- Un repositorio por aplicación / microservicio



Dependencias

- Una aplicación cloud native no ”depende”de algo en su entorno
- Dependencias isoladas y compilaciones repetibles

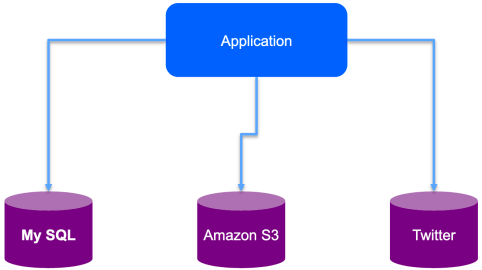


Configuración

- La configuración de una aplicación debe ser dinámica sin re-compilación/re-empaque
- Configuraciones inyectables

Backing services

- Acoplamiento debil. Siempre tratar backing services como componentes intercambiables y/o adjuntos

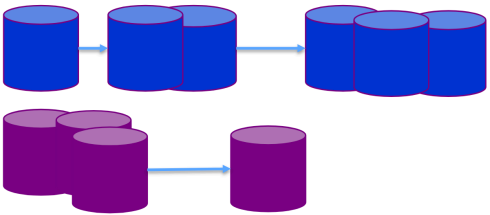


Build, release, run

- Separación de etapas de construcción, ejecución y lanzamiento
- CI/CD se hace obligatorio

Procesos

- Ejecutar la aplicación como uno o más procesos sin estado
- REST, Stateless, sesiones portables con JWT



Port binding

- Exponer los servicios con puertos dinámicos
- Kubernetes, Docker, etc.

Concurrencia

- Aplicaciones escalan de forma independiente replicándose
- Las nubes escalan mediante copias independientes, sin estado

Disposability

- Procesos arrancar rápido, mueren rápido, reinician rápido
- Procesos son tolerantes a fallas

Dev/Prod parity

- Entornos de desarrollo, certificación, producción lo más homogéneos posible

Logs

- Manipular logs de n copias de n servicios (streams de eventos)
- Permitir el análisis posterior

Admin process

- Manipular logs de n copias de n servicios (streams de eventos)
- Permitir el análisis posterior

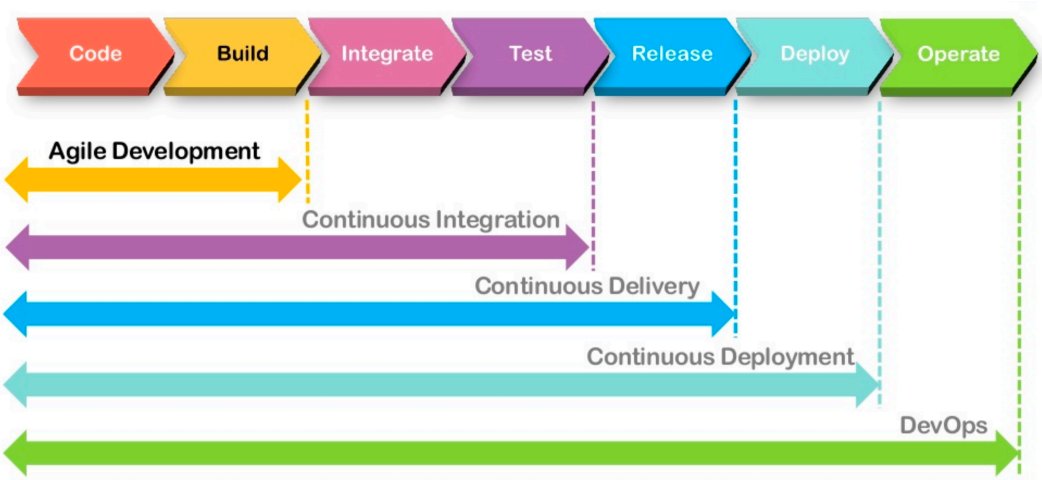
Estrategia

Estrategia

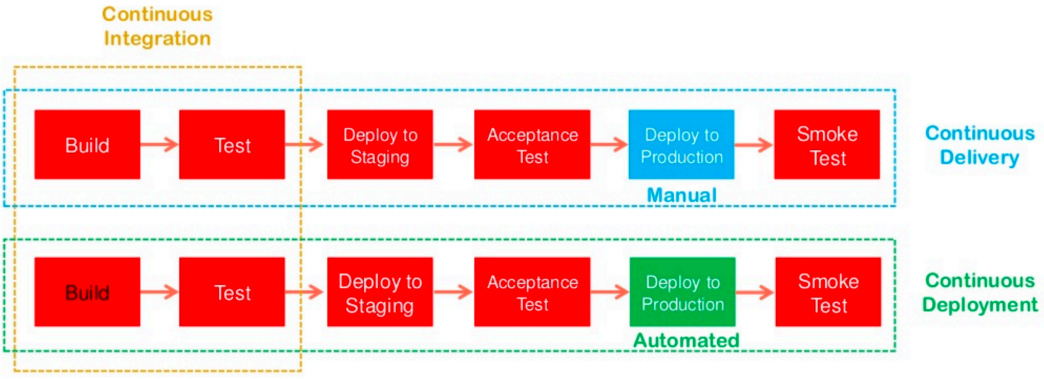
- Implementar DevOps sobre la base de código actual
- Evaluar opciones Cloud Native completas
- Fragmentar en Microservicios solo si es necesario

DevOps

Etapas



Delivery vs Deployment



Víctor Orozco

- vorozco@nabenik.com
- @tuxtor
- http://vorozco.com
- http://tuxtor.shekalug.org