ESPECIALIDAD: JAVA NIVEL: AVANZADO MODALIDAD: ONLINE

Curso:

ARQUITECTURA DE MICROSERVICIOS I

(Spring Boot, Spring Cloud, Docker, Design, Patterns, Buildings, Deployments)

Jean Ramal
Instructor

Agenda - Sesión 02

Diseño de la Arquitectura de Microservicios

- Revisión general de los principales patrones de microservicios.
- Patrones a implementar: Chassis, Service Discovery, Load Balancer y Externalized Configuration.
- Spring Cloud (Netflix, Config Server).
- Creación del microservicio Cart y consumo de microservicio Product.

Patrones de la Arquitectura de Microservicios

Application architecture patterns

- Monolithic architecture
- Microservice architecture

Decomposition

- Decompose by business capability
- Decompose by subdomain
- Self-contained Service
- Service per team

Refactoring to microservices

- Strangler Application
- Anti-corruption layer

Data management

- Database per Service
- Shared database
- Saga
- API Composition
- CQRS
- Domain event
- Event sourcing

Transactional messaging

- Transactional outbox
- Transaction log tailing
- Polling publisher

Testing

- Consumer-driven contract test
- Consumer-side contract test
- Service component sest

Deployment patterns

- Multiple service instances per host
- Service instance per host
- Service instance per VM
- Service instance per Container
- Serverless deployment
- Service deployment platform

Cross cutting concerns

- Microservice chassis
- Externalized configuration
- Service Template

Communication Style

- Remote Procedure Invocation
- Messaging
- Domain-specific protocol
- Idempotent Consumer

External API

- API gateway
- Backend for front-end

Service discovery

- Client-side discovery
- Server-side discovery
- Service registry
- Self registration
- 3rd party registration

Reliability

- Circuit Breaker

Security

- Access Token

Observability

- Log aggregation
- Application metrics
- Audit logging
- Distributed tracing
- Exception tracking
- Health check API
- Log deployments and changes

UI patterns

- Server-side page fragment composition
- Client-side UI composition

Decomposition

- **Decompose by business capability:** Concepto del modelado de arquitectura empresarial y capacidad que hace la empresa para generar valor (Gestión de Pedidos, Gestión de Clientes).

- **Decompose by subdomain:** Correspondiente a los subdominios de DDD. Espacio problemático de la aplicación, el negocio como dominio (Catálogo de Producto, Gestión de Entrega).

Refactoring to microservices

- **Strangler Application:** Migre gradualmente un sistema heredado reemplazando gradualmente piezas específicas de funcionalidad con nuevas aplicaciones y servicios.

- **Anti-corruption layer:** Implemente una capa de fachada o de adaptador entre una aplicación moderna y un sistema heredado.

Data management

- **Database per Service:** Permite establecer claramente la propiedad de la información usando una base de datos, esquema, o tablas para uso exclusivo del servicio y solo ser accedido a través del servicio.

- **Saga:** Permite realizar transacciones distribuidas a través de una secuencia de transacciones locales y transacciones compensatorias en el caso de que falle la operación.

- **API Composition:** Permite definir una API componiendo los datos a través de la consulta de otros servicios.

- **CQRS:** Define una base de datos de consulta, que es una réplica de solo lectura. La aplicación mantiene la réplica actualizada al suscribirse a eventos de dominio publicados por el servicio que posee los datos, y otra base de datos para las operaciones transaccionales.

Data management

- **Domain Event:** Organiza la lógica empresarial de un servicio como una colección de agregados DDD que emiten eventos de dominio cuando se crean o actualizan.

- **Event sourcing:** Se encarga de capturar todos los cambios que se pueden producir en nuestra aplicación como una secuencia de eventos. El estado actual de la aplicación deriva del procesamiento del histórico de todos los eventos.

Deployment

- **Single Service per Host:** Despliegue de un servicio en su propio host.

- **Service-per-container:** Despliegue de un servicio en su propio host utilizando contenedores.

Cross cutting concerns

- **Microservice Chassis:** El desarrollo de una arquitectura de microservicios será implementada por un framework o conjunto de framework.

- **Externalized configuration:** Externalizar o centralizar toda la configuración de la aplicación, incluidas las credenciales de la base de datos y la ubicación de la red.

External API

- **API gateway:** Punto de entrada único para todas las APIs. La puerta de enlace puede implementar seguridad, filtros y otros temas transversales.

- **Backend for front-end:** Punto de entrada único para las APIs para cada tipo de cliente.

Service discovery

- **Client-side discovery:** Despliegue de un servicio en su propio host.
- **Server-side discovery:** Servidor que permite registrar, descubrir y balancear los servicios.
- **Service registry:** Despliegue de un servicio en su propio host utilizando contenedores.

Reliability

- **Circuit breaker:** Punto de entrada único para todas las APIs. La puerta de enlace puede implementar seguridad, filtros y otros temas transversales.
- **Bulkhead:** Aísle los elementos de una aplicación en grupos para que, si uno falla, los demás sigan funcionando.
- **Retry:** Al conectarse a un servicio o recurso de red volver a intentar de manera transparente una operación que falló anteriormente.

Security

- **Access Token:** Permite identificar al usuario de forma segura transfiriendo un token en cada petición.
- **Federated Identity:** Delegue la autenticación a un proveedor de identidad externo..

Observability

- **Distributed tracing:** Permite trazar las solicitudes a los microservicios para poder identificar de forma centralizada todo lo relacionado al mismo.
- **Health check API:** Un servicio tiene un punto final de API de verificación de estado (por ejemplo, HTTP /health) que devuelve el estado del servicio.

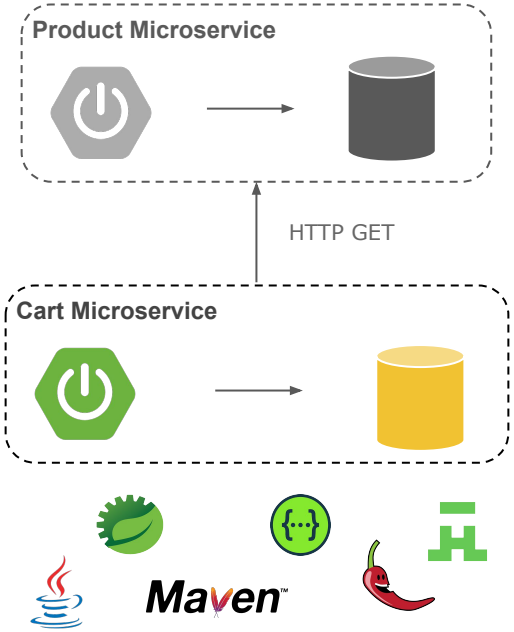
Observability

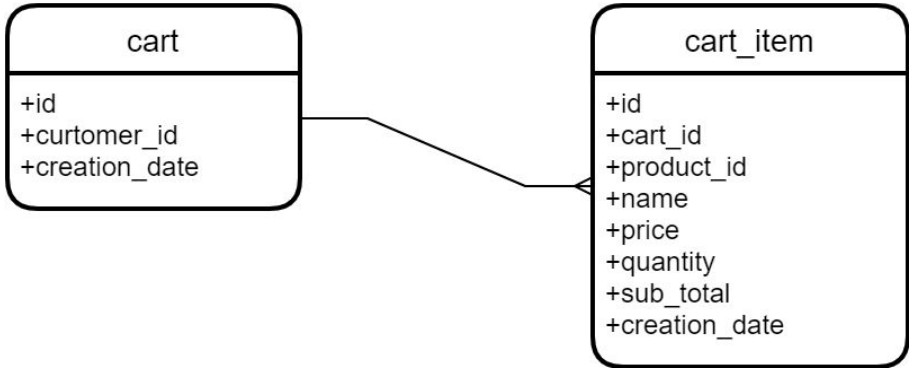
- **Log aggregation:** Utilizar un servicio de registro centralizado que agregue registros de cada instancia de servicio. Los usuarios pueden buscar y analizar los registros. Pueden configurar alertas que se activan cuando aparecen determinados mensajes en los registros.

- **Application metrics:** Instrumentar un servicio para recopilar estadísticas sobre operaciones individuales. Métricas agregadas en el servicio de métricas centralizadas, que proporciona informes y alertas.

- **Exception tracking:** Informa todas las excepciones a un servicio de seguimiento de excepciones centralizado que agrega y rastrea las excepciones y notifica a los desarrolladores.

Microservicios de Carrito





Microservice chassis pattern

- El desarrollo de una arquitectura de microservicios será implementada por un framework o conjunto de librerías.
- Implementación con **Spring Boot y Spring Cloud**, MicroProfile (Quarkus, Helidon), Micronaut.

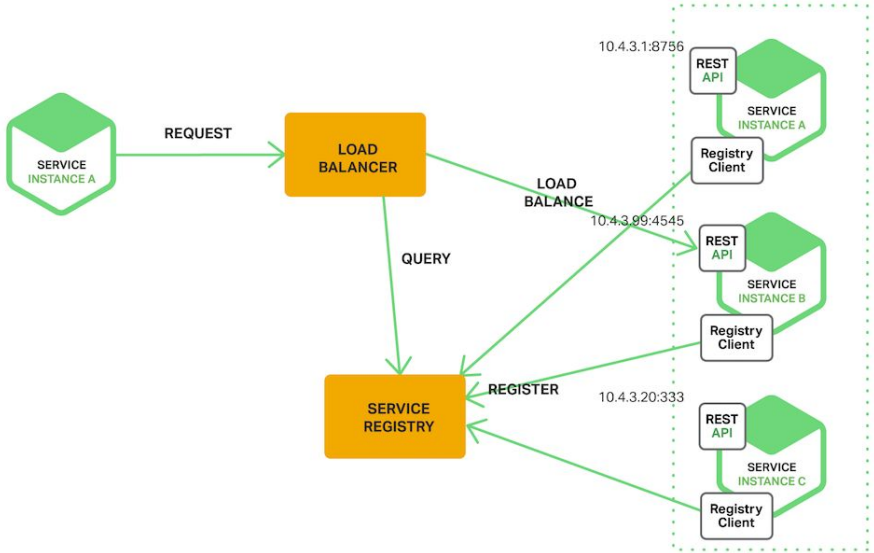
Spring Cloud

Spring Cloud proporciona herramientas para construir rápidamente algunos patrones comunes en sistemas distribuidos.

- Distributed/versioned configuration
- Service registration and discovery
- Routing
- Service-to-service calls
- Load balancing
- Circuit Breakers
- Global locks
- Leadership election and cluster state
- Distributed messaging

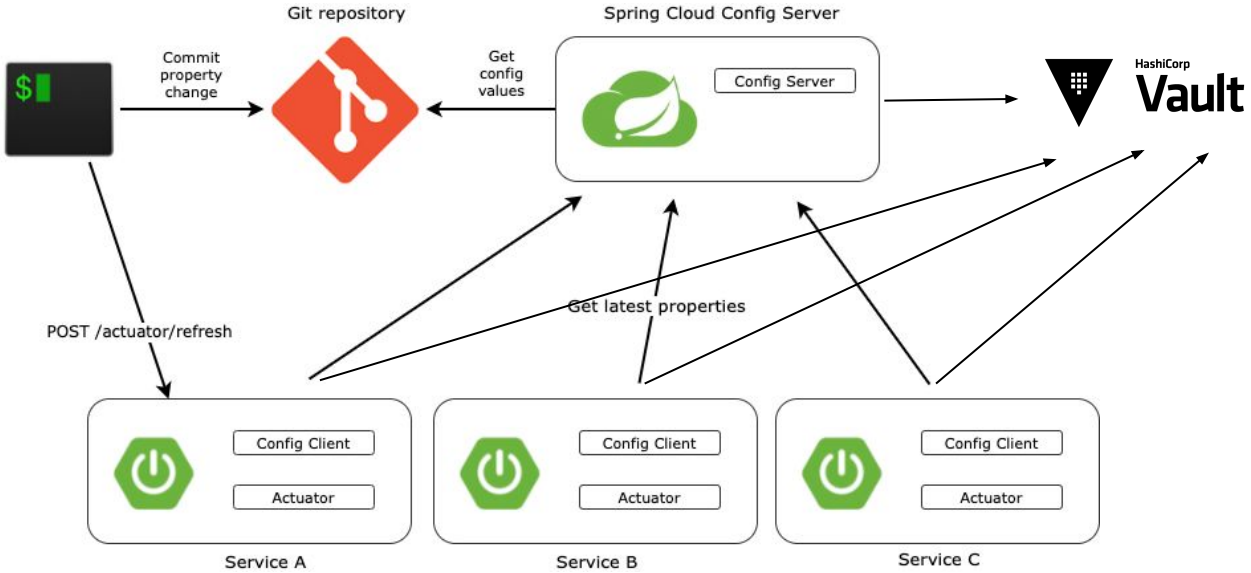
Server side discovery pattern

- Servidor que lleva el control de los servicios. Permite registrar, descubrir y balancear los servicios.
- Implementación en Spring con **Spring Cloud Netflix**



Externalized configuration pattern

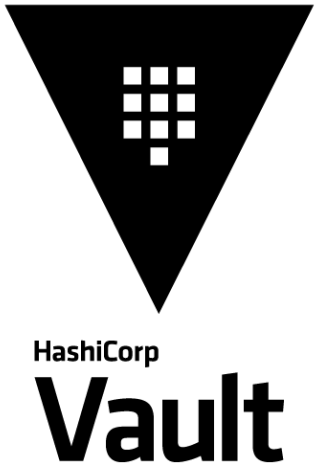
- Externalizar y centralizar la configuración.
- Implementación en Spring con **Spring Cloud Config**.



HashiCorp Vault

HashiCorp Vault es un sistema de gestión de secretos y encriptación basados en la identidad. Se puede acceder a los secretos u otros datos sensibles a través de Vault UI, CLI o su API Http.

- Flujo: autenticacion y autorizacion, validación y Acceso
- Almacenamiento de secreto seguro
- Secretos dinámicos
- Encriptación de datos
- Alta y Renovación
- Revocación

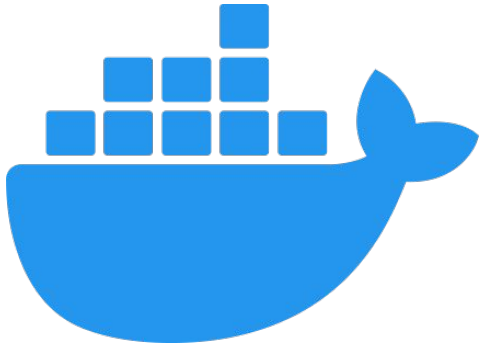


Docker

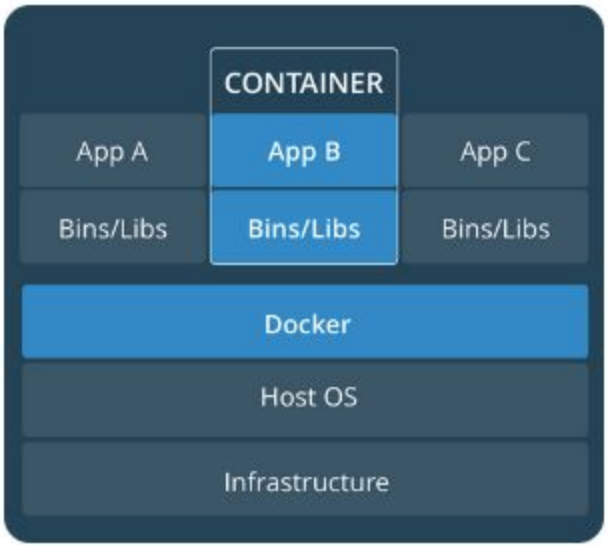
Es una plataforma que permite crear, probar e implementar aplicaciones rápidamente.

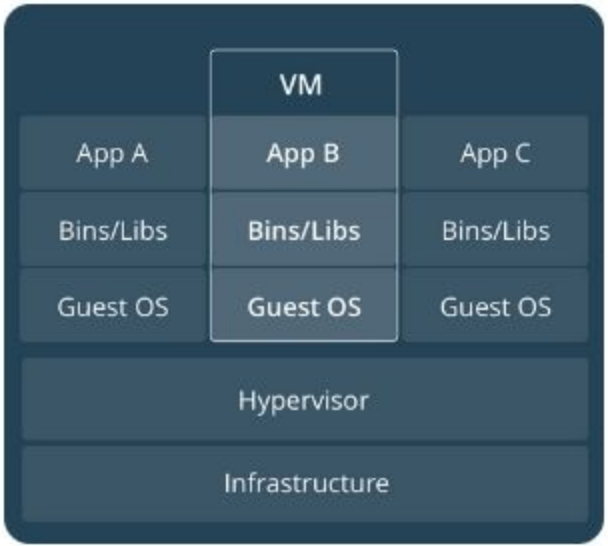
Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución.

Con Docker, puede implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que su código se ejecutará.



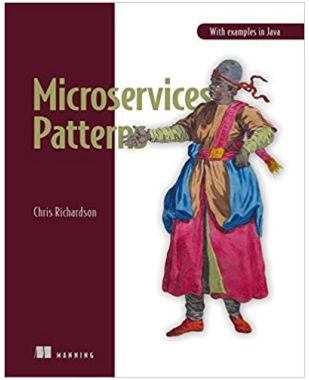
Docker


CONTAINER
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Docker
Host OS
Infrastructure


VM
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Infrastructure

Referencias

https://docs.microsoft.com/es-es/azure/architecture/guide/technology-choices/microservices-assessment
https://docs.microsoft.com/es-es/azure/architecture/microservices/design/
https://microservices.io/patterns/index.html
https://docs.microsoft.com/es-es/azure/architecture/patterns/
https://docs.microsoft.com/es-es/azure/architecture/microservices/design/patterns
https://docs.microsoft.com/es-es/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice
https://docs.microsoft.com/es-es/azure/architecture/microservices/model/domain-analysis
https://docs.microsoft.com/es-es/azure/architecture/microservices/model/microservice-boundaries
https://docs.microsoft.com/es-mx/dotnet/architecture/microservices/architect-microservice-container-applications/microservices-architecture
https://docs.microsoft.com/es-es/azure/architecture/microservices/migrate-monolith
https://spring.io/projects/spring-cloud
https://spring.io/projects/spring-cloud-netflix
https://spring.io/projects/spring-cloud-config



¡GRACIAS!

Correo: jean.ramal@codearti.com

Cel: +51994641153

Link: https://www.linkedin.com/in/jeanramal/