SpringBoot Microservices

Contenidos

1. Fundamentos de Arquitectura y Microservicios

Comprender los fundamentos de la arquitectura de software y cómo los microservicios se integran en ella.

2. Desarrollo de Microservicios con Spring Boot

Aprender a desarrollar microservicios utilizando Spring Boot 3 y aplicar buenas prácticas en su construcción.

3. Construcción y Testing de Microservicios

Desarrollar y probar un microservicio con acceso a datos, y despliegue en Docker.

4. Gestión y Despliegue de Contenedores

Profundizar en la gestión de contenedores y en su despliegue en diferentes entornos.

Contenidos

5. Gobierno y Documentación de Microservicios

Implementar estrategias de gobierno, documentación y monitorización de microservicios.

6. Kubernetes

Implementar aplicaciones en Kubernetes

7. DevSecOps

Comprender los conceptos y prácticas de automatización y control en el ciclo de vida de desarrollo.

8. Observabilidad en Microservicios

Integrar la observabilidad sobre arquitecturas orientadas a microservicios

Contenidos

9. **Arquitecturas EDA**

Comprender los conceptos de las arquitecturas orientadas a eventos.

Sesión 1: Fundamentos de Arquitectura y Microservicios

Objetivo: Comprender los fundamentos de la arquitectura de software y cómo los microservicios se integran en ella.

Sesión 1: Fundamentos de Arquitectura y Microservicios

Patrones de Arquitectura

Descripción: Los patrones de arquitectura proporcionan soluciones reutilizables a problemas comunes en el diseño de software. En el contexto de los microservicios, algunos patrones relevantes incluyen:

- Microkernel: Útil para sistemas donde la extensibilidad es clave. La funcionalidad básica está en el núcleo, y los módulos pueden conectarse y desconectarse dinámicamente.
- SOA (Service-Oriented Architecture): Arquitectura que comparte principios con los microservicios, enfocándose en la reusabilidad de servicios.
- Event-Driven Architecture: Arquitectura en la que los componentes se comunican mediante eventos, lo que es útil en sistemas desacoplados y escalables.

Referencias:

- Patterns of Enterprise Application Architecture (Martin Fowler)
- Microservices Patterns (Chris Richardson)

Código Ejemplo:

- Ejemplo de arquitectura basada en eventos con Kafka: GitHub - event-driven-kafka

Sesión 1: Fundamentos de Arquitectura y Microservicios

Patrones de Diseño

Descripción: Los patrones de diseño son soluciones reutilizables a problemas recurrentes en el diseño de software. En el desarrollo de microservicios, los patrones más relevantes incluyen:

- Circuit Breaker: Previene el fallo en cascada en caso de que un microservicio falle.
- API Gateway: Centraliza las solicitudes hacia diferentes microservicios.
- Saga: Manejo de transacciones distribuidas sin la necesidad de un coordinador central.

Referencias:

- Design Patterns: Elements of Reusable Object-Oriented Software (Erich Gamma, et al.)
- Designing Data-Intensive Applications (Martin Kleppmann)

Código Ejemplo:

- Implementación de patrón Circuit Breaker con Resilience4j: GitHub - resilience4j

Sesión 1: Fundamentos de Arquitectura y Microservicios

Atributos de Calidad

Descripción: Los microservicios impactan directamente atributos de calidad como:

- Escalabilidad: Facilidad para escalar componentes individualmente según la demanda.
- Disponibilidad: Resiliencia ante fallos gracias a la arquitectura distribuida.
- Mantenibilidad: Modularidad que facilita actualizaciones y despliegues continuos.

Referencias:

- Building Microservices: Designing Fine-Grained Systems (Sam Newman)
- The Twelve-Factor App

Código Ejemplo:

- Ejemplo de despliegue escalable con Kubernetes: GitHub - k8s-examples

Sesión 1: Fundamentos de Arquitectura y Microservicios

Conceptos de Microservicios

Descripción: Los microservicios son un estilo arquitectónico donde una aplicación se construye como un conjunto de servicios pequeños e independientes que se ejecutan en sus propios procesos y se comunican entre sí a través de interfaces bien definidas (generalmente HTTP/REST o mensajes).
Referencias:

- Microservices (Martin Fowler)
- What are Microservices? (AWS)

Código Ejemplo:

- Ejemplo de aplicación Spring Boot basada en microservicios: GitHub - spring-microservices-example

Sesión 1: Fundamentos de Arquitectura y Microservicios

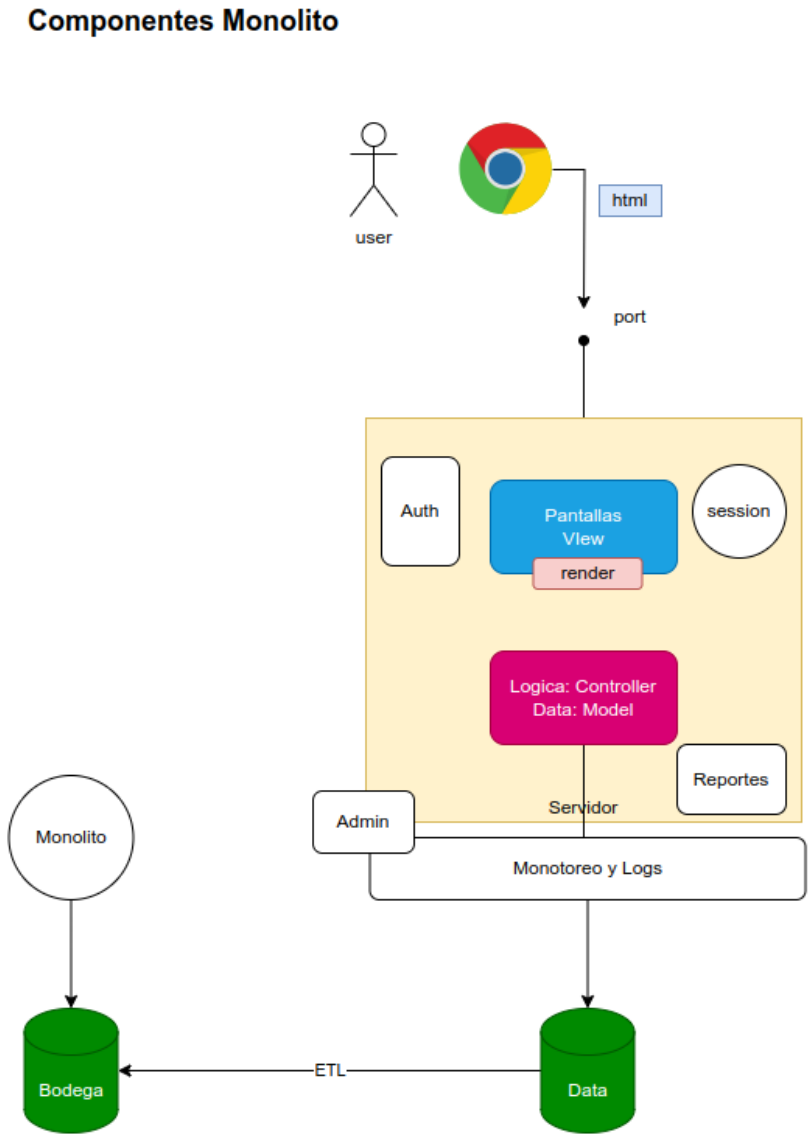
De Monolito a Microservicios

Descripción: La transición de una arquitectura monolítica a una basada en microservicios es un proceso complejo que implica dividir el monolito en servicios más pequeños y autónomos, manteniendo la funcionalidad existente mientras se mejoran aspectos como la escalabilidad y la flexibilidad.
Referencias:

- Refactoring: Improving the Design of Existing Code (Martin Fowler)
- Monolithic to Microservices Architecture (Google Cloud)

Ejemplo:

- Ejemplo de migración de monolito a microservicios: GitHub - monolith-to-microservices



Sesión 1: Fundamentos de Arquitectura y Microservicios

De Microservicios a una Aplicación Distribuida

Descripción: Una aplicación distribuida basada en microservicios escala más allá de un conjunto básico de servicios, integrando características como gestión de transacciones distribuidas, comunicación asíncrona, y la gestión avanzada de estado.

Referencias:

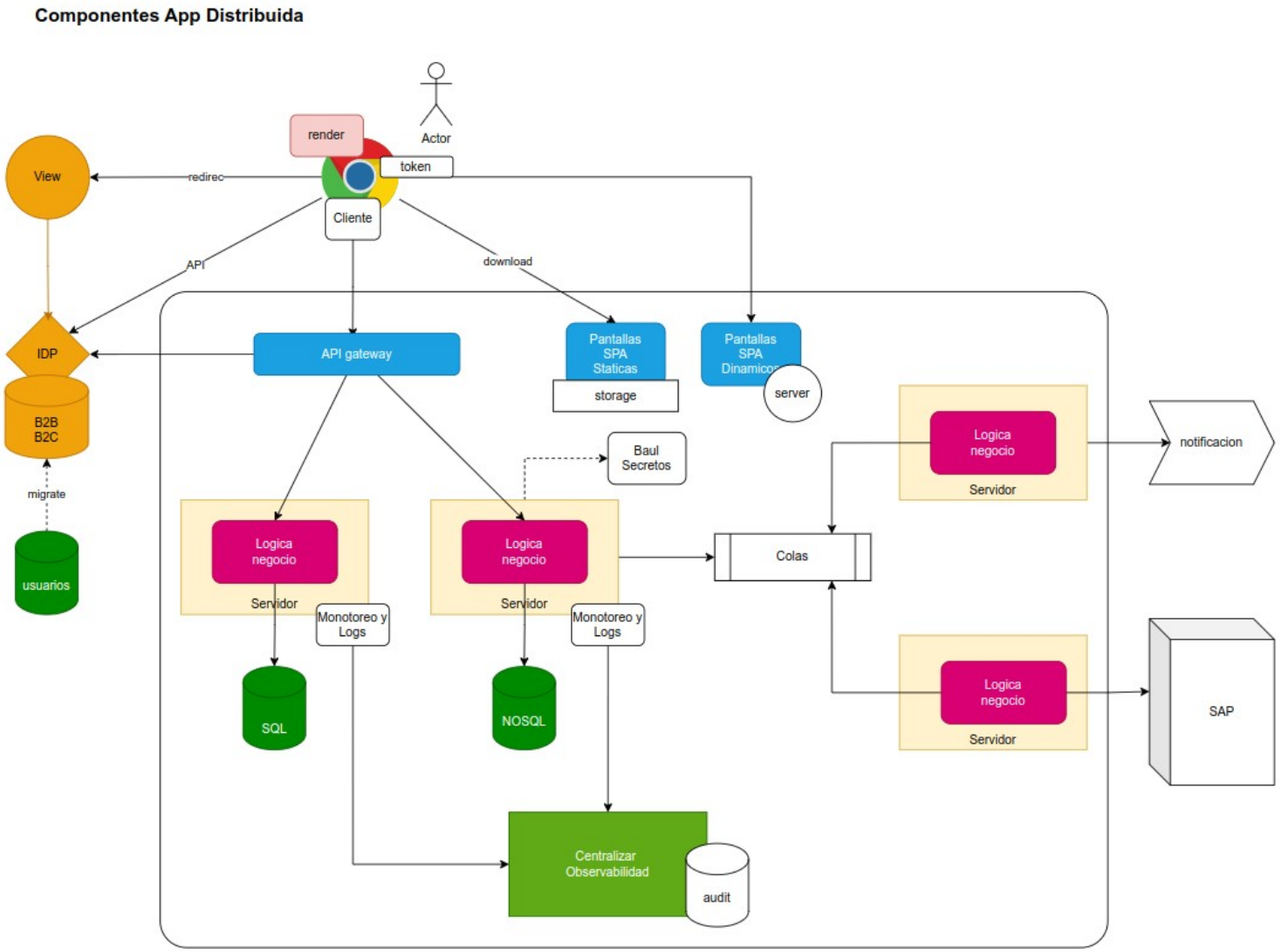
- Build cloud-native applications in Azure

- Distributed Systems (Maarten van Steen, Andrew S. Tanenbaum)

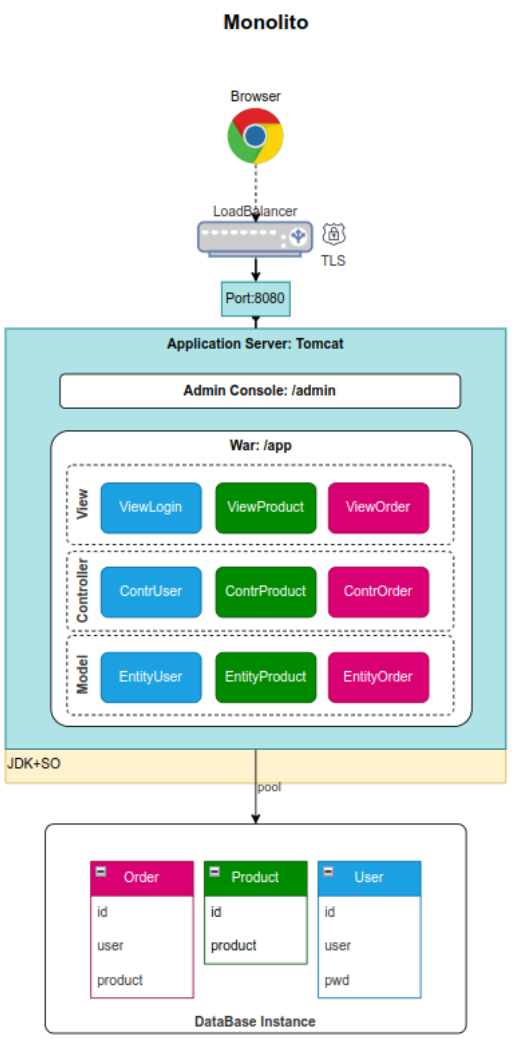
Código Ejemplo:

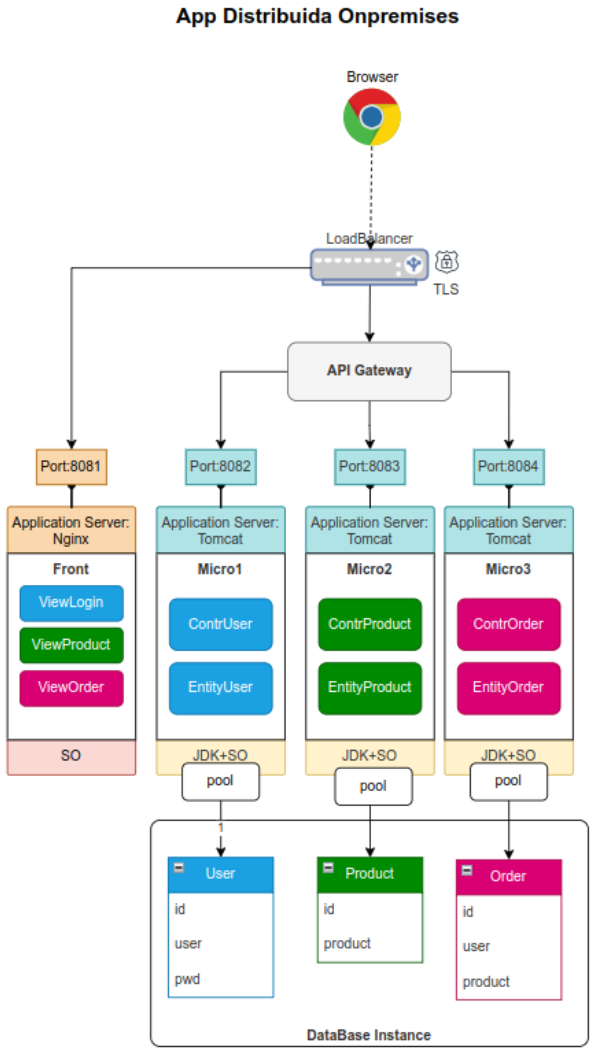
- Ejemplo de sistema distribuido usando Kubernetes y microservicios: GitHub - k8s-distributed

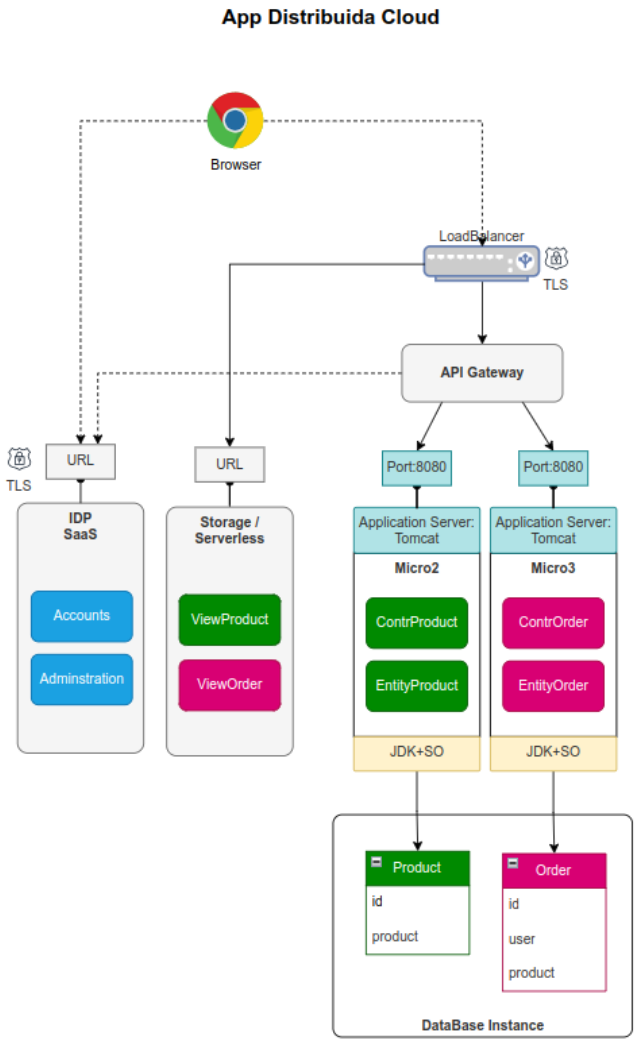
Sesión 1: Fundamentos de Arquitectura y Microservicios



Sesión 1: Fundamentos de Arquitectura y Microservicios







Sesión 1: Fundamentos de Arquitectura y Microservicios

Microservicios en On-premises y Cloud

Descripción: Implementar microservicios en entornos on-premises y en la nube presenta desafíos diferentes. En la nube, se aprovechan servicios gestionados que facilitan el escalado y la resiliencia, mientras que on-premises puede requerir una mayor inversión en infraestructura y herramientas.

Referencias:

- Migrating to Cloud-Native Application Architectures (Matt Stine)
- Hybrid Cloud Strategy (Microsoft Azure)

Código Ejemplo:

- Ejemplo de despliegue de microservicios en AWS: GitHub - aws-microservices

Sesión 2: Desarrollo de Microservicios con Spring Boot

Objetivo: Aprender a desarrollar microservicios utilizando Spring Boot 3 y aplicar buenas prácticas en su construcción.

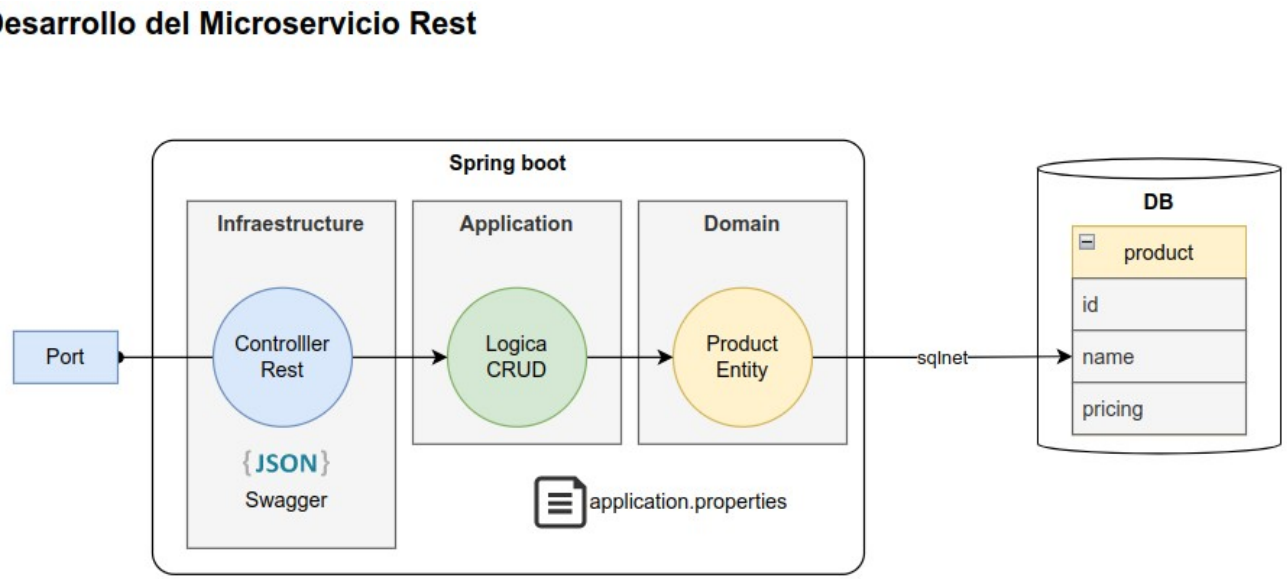
Sesión 2: Desarrollo de Microservicios con Spring Boot

Microservicios en Spring Boot 3
Descripción: Spring Boot 3 ofrece un conjunto de herramientas y bibliotecas optimizadas para el desarrollo de microservicios. Con soporte mejorado para Java 17 y GraalVM, Spring Boot 3 permite construir aplicaciones más rápidas y eficientes.
Referencias:

- Spring Boot 3 Documentation
- Migrating to Spring Boot 3

Código Ejemplo:

- Ejemplo básico de microservicio con
- Spring Boot 3: GitHub - spring-boot-3-example



Sesión 2: Desarrollo de Microservicios con Spring Boot

Framework Spring Boot

Spring Boot Initializr:

Descripción: Spring Initializr es una herramienta que simplifica la configuración inicial de proyectos Spring Boot, permitiendo a los desarrolladores seleccionar dependencias y versiones de Java adecuadas para comenzar rápidamente.

Referencias:

- Spring Initializr

Código Ejemplo:

- Generación de un proyecto básico de microservicios: GitHub - spring-initializr-example

Arquetipos:

Descripción: Los arquetipos en Spring Boot ayudan a estructurar proyectos de manera consistente y eficiente, promoviendo la reutilización de componentes y patrones de diseño comunes.

Referencias:

- Maven Archetypes
- Spring Boot Maven Plugin

Código Ejemplo:

- Ejemplo de uso de arquetipos en un proyecto Spring Boot: GitHub - spring-archetype-example

Sesión 2: Desarrollo de Microservicios con Spring Boot

Buenas Prácticas de Construcción de Microservicios

Frameworks:

Descripción: Selección de frameworks adecuados como Spring Cloud para gestionar la comunicación entre microservicios y resiliencia, y Spring Security para la gestión de autenticación y autorización.

Referencias:

- Spring Cloud Documentation
- Spring Security Documentation

Código Ejemplo:

- Configuración de un microservicio con Spring Cloud y Spring Security: GitHub - spring-cloud-security-example

Logging:

Descripción: Implementación de logs eficaces utilizando frameworks como Logback o SLF4J, y técnicas para la centralización y análisis de logs en entornos distribuidos.

Referencias:

- Spring Boot Logging Documentation
- Logback Project

Código Ejemplo:

- Configuración de Logback en Spring Boot: Spring-logging-example

Sesión 2: Desarrollo de Microservicios con Spring Boot

Buenas Prácticas de Construcción de Microservicios

Variables de Ambiente:

Descripción: Gestión de configuraciones dinámicas mediante perfiles de Spring y la inyección de variables de entorno para la adaptación de la aplicación a diferentes entornos (desarrollo, staging, producción).

Referencias:

- Spring Boot External Configuration

Código Ejemplo:

- Uso de perfiles en Spring Boot: GitHub - spring-profiles-example

Testing:

Descripción: Pruebas unitarias y de integración utilizando frameworks como JUnit 5, Mockito, y Spring Test, asegurando la calidad y fiabilidad de los microservicios.

Referencias:

- Testing in Spring Boot
- JUnit 5 Documentation

Código Ejemplo:

- Pruebas unitarias y de integración en un microservicio: GitHub - spring-testing-example

Sesión 2: Desarrollo de Microservicios con Spring Boot

Buenas Prácticas de Construcción de Microservicios

Gobierno:
Descripción: Gestión y control de microservicios en producción utilizando herramientas como Spring Boot Admin para el monitoreo y Zipkin para el rastreo de llamadas distribuidas.

Referencias:
- Spring Boot Admin Documentation
- Zipkin Documentation

Código Ejemplo:
- Implementación de monitoreo y rastreo: GitHub - spring-monitoring-example

Calidad de Código:
Descripción: Herramientas y técnicas para mantener código limpio y mantenible, como SonarQube para análisis estático y Checkstyle para seguir normas de estilo de código.

Referencias:
- SonarQube Documentation
- Checkstyle Documentation

Código Ejemplo:
- Integración de SonarQube en un proyecto Spring Boot: Spring-sonarqube-example

Sesión 2: Desarrollo de Microservicios con Spring Boot

Conceptos de Contenerización

Descripción: Contenerizar aplicaciones permite empaquetar software en contenedores portables, asegurando que se ejecuten de manera consistente en cualquier entorno. Esto es clave en microservicios para facilitar el despliegue y la escalabilidad.

Referencias:

- Docker for Java Developers
- Introduction to Containerization

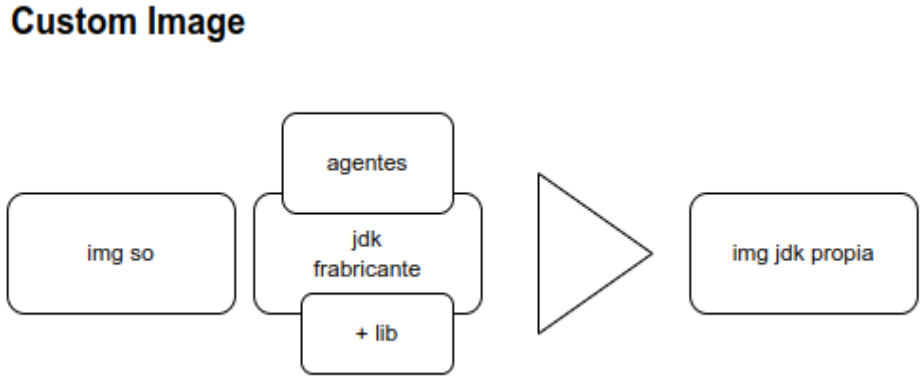
Código Ejemplo:

- Contenerización básica de una aplicación Spring Boot: GitHub - spring-boot-docker

Sesión 2: Desarrollo de Microservicios con Spring Boot

Docker, Imágenes y Contenedores

Descripción: Docker es una plataforma que permite crear, desplegar y ejecutar aplicaciones en contenedores. Los contenedores son ligeros y se ejecutan en cualquier entorno que tenga Docker instalado, lo que es ideal para microservicios.



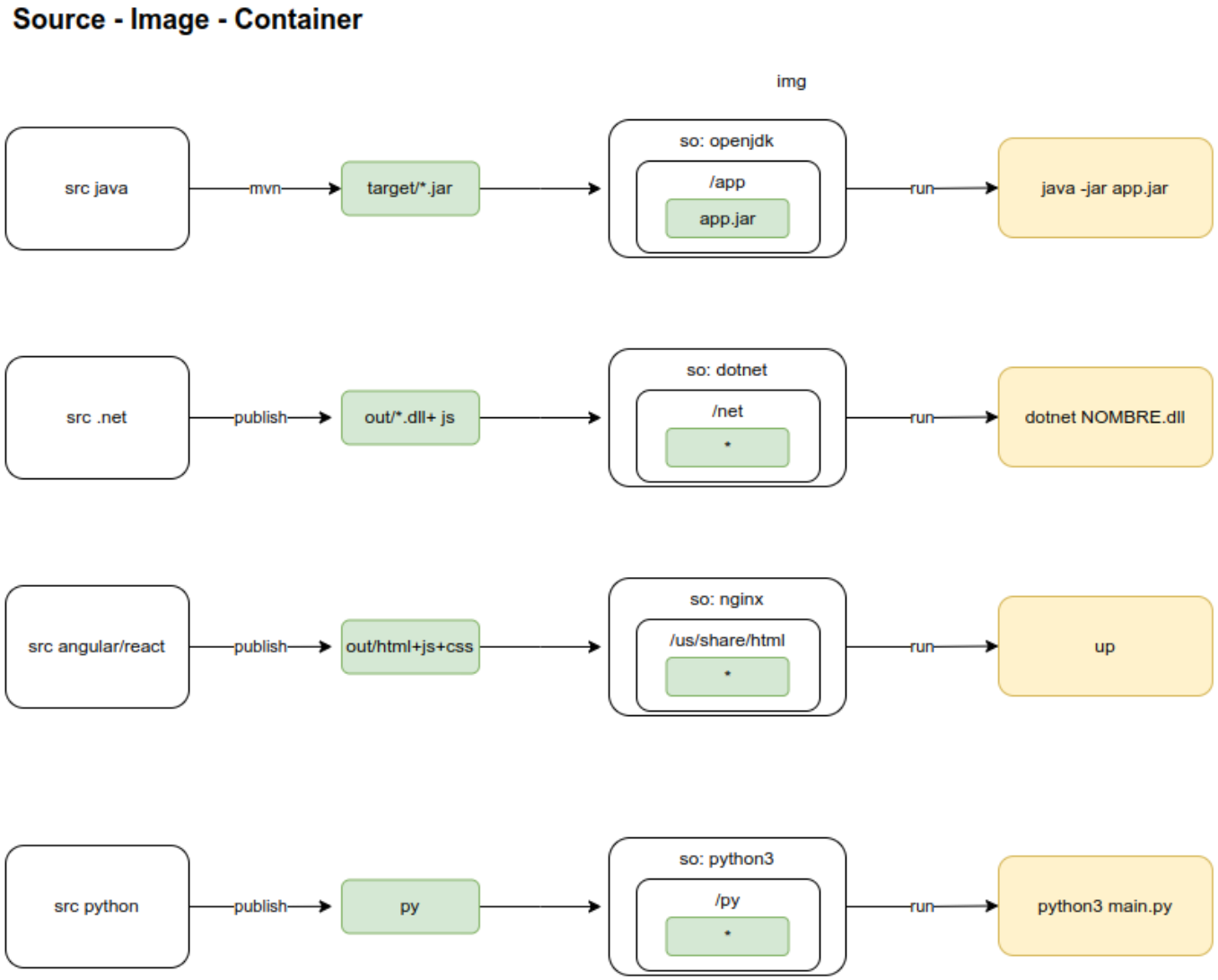
Referencias:

- Docker Documentation
- Dockerfile Reference

Código Ejemplo:

- Creación de una imagen Docker para un microservicio Spring Boot: GitHub - docker-spring-boot-example

Sesión 2: Desarrollo de Microservicios con Spring Boot



Sesión 3: Construcción y Testing de Microservicios

Objetivo: Desarrollar y probar un microservicio completo con acceso a datos, aplicando buenas prácticas en la construcción de imágenes Docker.

Sesión 3: Construcción y Testing de Microservicios

Paso a Paso: Construcción de un Microservicio con Acceso a Datos

Descripción: Esta sección se enfoca en guiar a los participantes en la creación de un microservicio desde cero que interactúe con una base de datos, utilizando Spring Data JPA para la gestión de datos.

- Conexión a la Base de Datos: Configuración de la conexión a una base de datos relacional (como MySQL o PostgreSQL) utilizando Spring Data JPA.
- Operaciones CRUD: Implementación de operaciones básicas (Crear, Leer, Actualizar, Eliminar) en el microservicio.
- Controladores y Servicios: Creación de controladores REST y servicios para gestionar la lógica de negocio y exposición de datos.

Referencias:

- Spring Data JPA Documentation
- RESTful Web Services with Spring Boot

Código Ejemplo:

- Ejemplo de microservicio con acceso a base de datos: GitHub - spring-data-example

Sesión 3: Construcción y Testing de Microservicios

Construcción de un Test en Postman

Descripción: Postman es una herramienta popular para el desarrollo y pruebas de API. Esta sección cubre cómo crear y ejecutar pruebas para el microservicio desarrollado.

- Colecciones de Postman: Creación de colecciones que agrupen las solicitudes relacionadas y permiten pruebas organizadas y repetibles.
- Pruebas Automatizadas: Configuración de pruebas automatizadas en Postman, incluyendo validación de respuestas, tiempos de respuesta, y pruebas de escenarios negativos.

Referencias:

- Postman Documentation
- API Testing with Postman

Código Ejemplo:

- Ejemplo de colección Postman para pruebas de un microservicio: GitHub - postman-api-tests

Sesión 3: Construcción y Testing de Microservicios

Construcción de Test en k6

Descripción: k6 es una herramienta moderna para pruebas de carga y rendimiento de APIs. En esta sección, aprenderás a escribir y ejecutar pruebas de rendimiento contra el microservicio.

- Script de k6: Creación de scripts de prueba de carga con k6 utilizando JavaScript.
- Ejecución de Pruebas de Carga: Simulación de múltiples usuarios concurrentes y análisis del rendimiento del microservicio.

Referencias:

k6 Documentation

Load Testing with k6

Código Ejemplo:

- Ejemplo de script de k6 para pruebas de carga: GitHub - k6-load-test

Sesión 3: Construcción y Testing de Microservicios

Construcción de Imágenes Docker (Dockerfile)

Descripción: Un Dockerfile es un script que contiene una serie de instrucciones para construir una imagen Docker. En esta sección, se detalla cómo crear un Dockerfile eficiente para empaquetar el microservicio.

- Instrucciones Básicas: Uso de comandos como FROM, COPY, RUN, EXPOSE, y CMD para crear un Dockerfile básico.
- Multistage Builds: Optimización del Dockerfile utilizando construcciones de múltiples etapas para reducir el tamaño de la imagen final.

Referencias:

- Dockerfile Reference
- Docker Best Practices

Código Ejemplo:

- Ejemplo de Dockerfile para un microservicio Spring Boot: Spring-boot-dockerfile

Sesión 3: Construcción y Testing de Microservicios

Buenas Prácticas de Construcción de Imágenes

Descripción: La eficiencia y seguridad de las imágenes Docker son fundamentales. Esta sección explora técnicas para optimizar Dockerfiles y reducir el tamaño y vulnerabilidades de las imágenes.

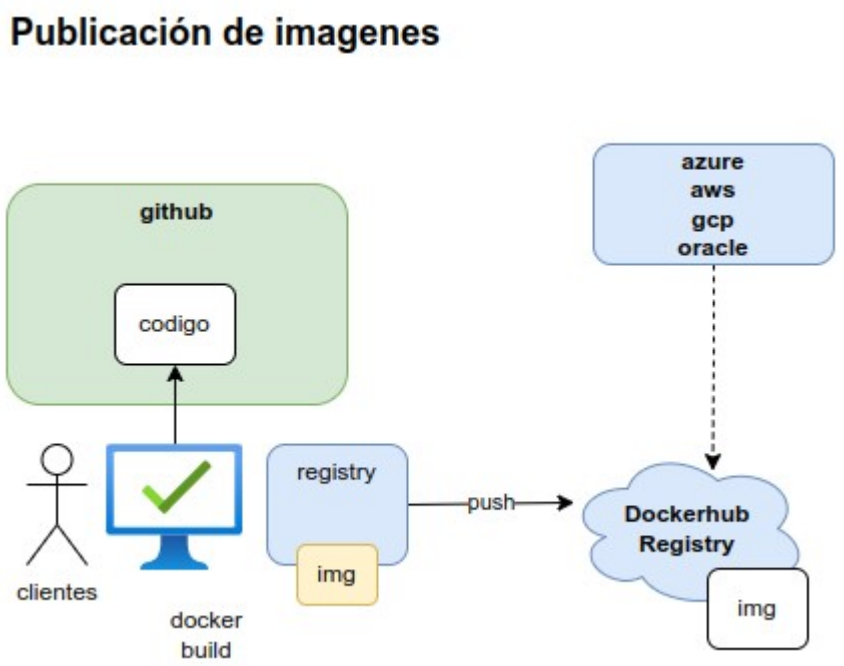
- Reducir Tamaño de Imágenes: Uso de imágenes base ligeras como alpine, eliminación de capas innecesarias y limpieza de cachés.
- Seguridad: Integración de herramientas para escaneo de vulnerabilidades en imágenes Docker.

Referencias:

- Docker Security Best Practices

Código Ejemplo:

- Ejemplo de Dockerfile optimizado: GitHub - optimized-dockerfile



Sesión 3: Construcción y Testing de Microservicios

Ejecución de Contenedores con Docker Run

Descripción: docker run es el comando principal para ejecutar contenedores Docker. Esta sección cubre cómo iniciar y gestionar contenedores utilizando este comando.

- Comandos Básicos: Uso de opciones como -d (detached), -p (puerto), --env (variables de entorno) para ejecutar contenedores.
- Volúmenes y Redes: Configuración de volúmenes para persistencia de datos y redes para la comunicación entre contenedores.

Referencias:

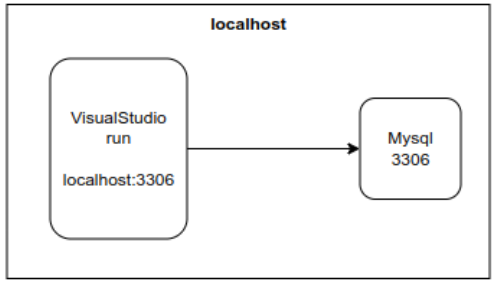
- Docker Run Reference
- Docker Networking

Código Ejemplo:

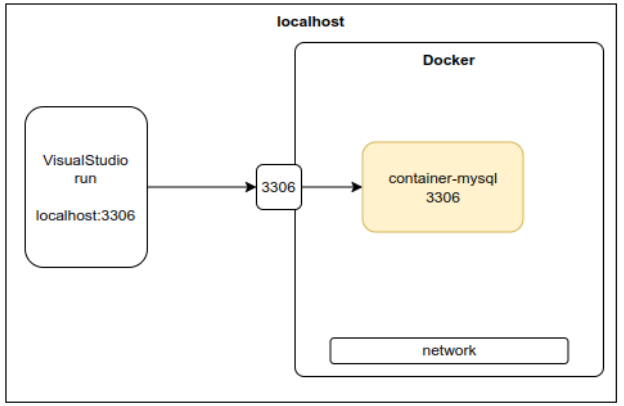
- Ejemplo de ejecución de un contenedor con Docker Run: GitHub - docker-run-examples

Sesión 3: Construcción y Testing de Microservicios

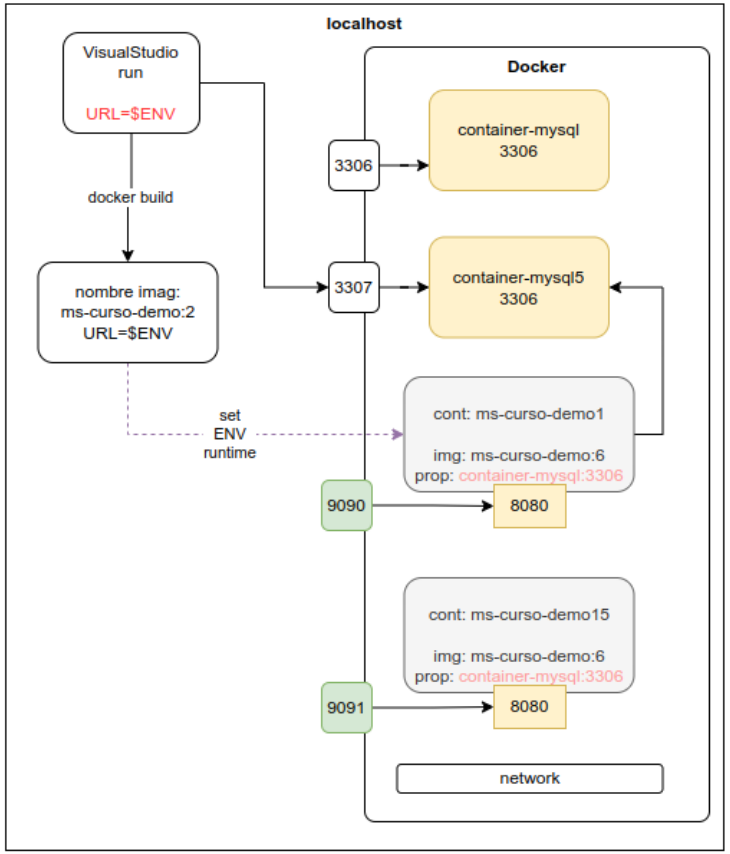
Ambiente Desarrollo Tradicional



Ambiente Desarrollo con Docker



Ambiente Desarrollo Multiples Contenedores



Sesión 4: Gestión y Despliegue de Contenedores

Objetivo: Profundizar en la gestión de contenedores y en su despliegue en diferentes entornos

Sesión 4: Gestión y Despliegue de Contenedores

Ejecución de Contenedores con Docker-Compose

Descripción: Docker Compose es una herramienta que permite definir y gestionar aplicaciones multicontenedor. En esta sección, aprenderás a orquestar múltiples contenedores, configurando redes, volúmenes, y dependencias entre servicios.

- Definición de Servicios: Uso del archivo docker-compose.yml para definir servicios, redes y volúmenes.
- Comandos Básicos: Ejecución de comandos como docker-compose up, docker-compose down, y docker-compose logs para gestionar el ciclo de vida de los contenedores.

Referencias:

- Docker Compose Documentation
- Get Started with Docker Compose

Código Ejemplo:

- Ejemplo de configuración con Docker Compose: GitHub - docker-compose-example

Sesión 4: Gestión y Despliegue de Contenedores

Sizing de Contenerización

Descripción: El dimensionamiento de contenedores es crucial para optimizar el uso de recursos y mejorar la eficiencia. Esta sección cubre cómo asignar CPU, memoria y otras restricciones de recursos de manera óptima.

- Configuración de Recursos: Uso de opciones como --memory, --cpus, y --cpu-shares para limitar y gestionar los recursos asignados a contenedores.
- Monitoreo y Ajuste: Herramientas y técnicas para monitorear el uso de recursos y ajustar la configuración de contenedores para un rendimiento óptimo.

Referencias:

- Resource Constraints in Docker
- Monitoring Docker Performance

Código Ejemplo:

- Ejemplo de limitación de recursos en un Dockerfile: Docker-resource-limits

Sesión 4: Gestión y Despliegue de Contenedores

Herramientas de Gestión de Contenedores

Descripción: Existen diversas herramientas para la administración de contenedores que facilitan el monitoreo, escalado y gestión de aplicaciones en producción. Esta sección presenta algunas de las más utilizadas.

- Portainer: Una interfaz gráfica para gestionar contenedores Docker, permitiendo la gestión de stacks, imágenes, volúmenes y redes de manera visual.
- Rancher: Una plataforma para la gestión de Kubernetes que simplifica la implementación y operación de clústeres de contenedores.

Referencias:

- Portainer Documentation
- Rancher Documentation

Código Ejemplo:

- Integración de Portainer en un entorno Docker: GitHub - portainer-docker-example

Sesión 4: Gestión y Despliegue de Contenedores

Ejecución en Ambientes On-premises y Cloud/Serverless

Descripción: El despliegue de contenedores en diferentes entornos presenta desafíos únicos. Esta sección explora las diferencias clave y las mejores prácticas para implementar contenedores tanto en entornos locales (on-premises) como en la nube o plataformas serverless.

- On-premises: Configuración de entornos locales utilizando soluciones como Docker Swarm o Kubernetes.
- Cloud: Uso de servicios como Amazon ECS, Google Kubernetes Engine (GKE), y Azure Kubernetes Service (AKS) para el despliegue en la nube.
- Serverless: Implementación de contenedores en plataformas serverless como AWS Fargate o Google Cloud Run.

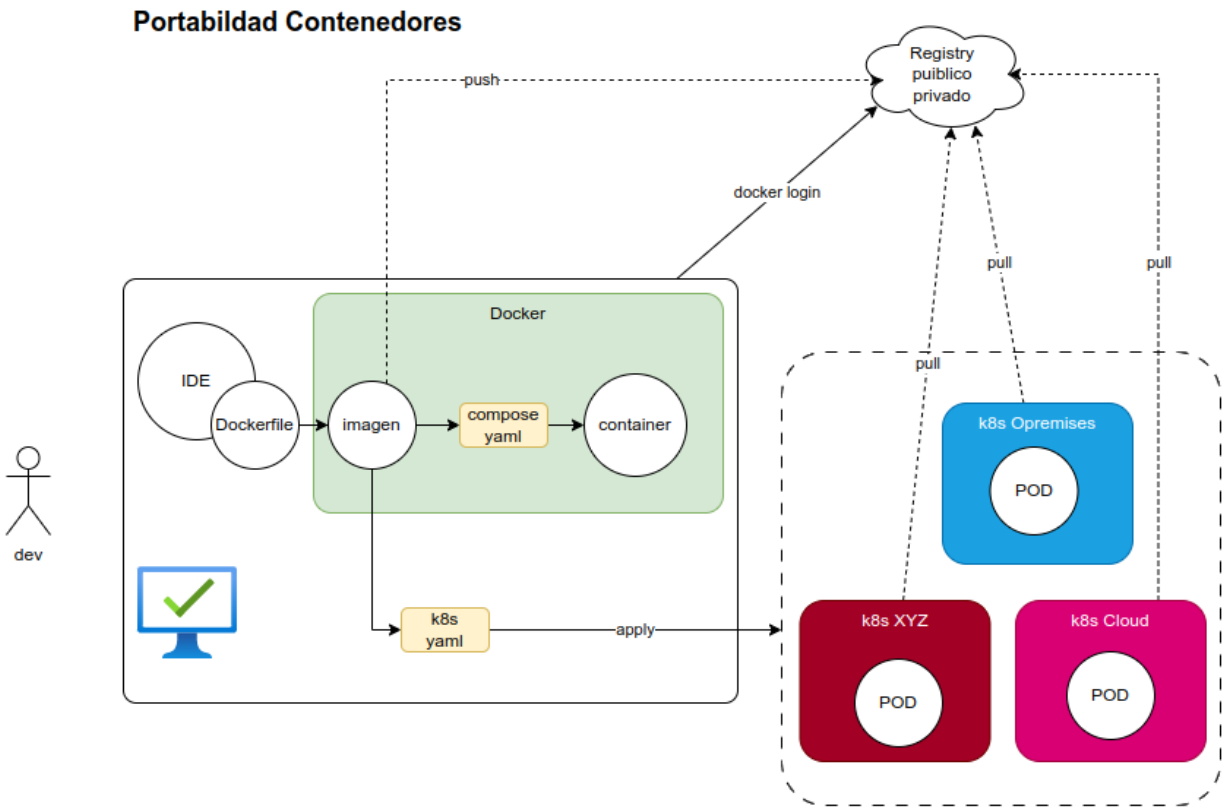
Referencias:

- Docker Swarm vs Kubernetes
- Serverless Containers with AWS Fargate
- Deploying Docker on GCP

Código Ejemplo:

- Ejemplo de despliegue en AWS Fargate: GitHub - aws-fargate-example

Sesión 4: Gestión y Despliegue de Contenedores



Sesión 5: Gobierno y Documentación de Microservicios

Objetivo: Implementar estrategias de gobierno, documentación y monitorización de microservicios.

Sesión 5: Gobierno y Documentación de Microservicios

Gobierno de Microservicios

Descripción: El gobierno de microservicios es esencial para mantener la cohesión, seguridad y eficiencia de un sistema distribuido. Esta sección aborda prácticas y herramientas clave para la gobernanza, incluyendo políticas de versionado, descubrimiento de servicios, y autenticación/autorización.

- Versionado de APIs: Estrategias para gestionar versiones de APIs y asegurar compatibilidad entre servicios.
- Service Mesh: Introducción a Service Meshes como Istio o Linkerd para gestionar la comunicación, seguridad y monitoreo de microservicios.
- Autenticación y Autorización: Implementación de OAuth2 y OpenID Connect para gestionar la autenticación en microservicios.

Referencias:

- Building Evolutionary Architectures
- Istio Service Mesh Documentation
- OAuth 2.0 and OpenID Connect

Código Ejemplo:

- Ejemplo de implementación de Service Mesh con Istio: GitHub - istio-example

Sesión 5: Gobierno y Documentación de Microservicios

Documentación con Spring Doc-OpenAPI y Markdown

Descripción: La documentación es clave para la mantenibilidad y colaboración en proyectos de microservicios. Esta sección explora cómo generar documentación de APIs automáticamente con Spring Doc-OpenAPI y cómo utilizar Markdown para mantener documentación técnica.

- Spring Doc-OpenAPI: Integración de Spring Doc-OpenAPI en microservicios Spring Boot para generar documentación de API en formato OpenAPI.
- Markdown: Uso de Markdown para escribir y mantener documentación técnica, incluyendo la estructura de archivos y mejores prácticas.

Referencias:

- Spring Doc-OpenAPI Documentation
- Markdown Guide

Código Ejemplo:

- Ejemplo de proyecto con Spring Doc-OpenAPI: GitHub - springdoc-openapi-example

Sesión 5: Gobierno y Documentación de Microservicios

Healthcheck de Contenedores con Spring Boot Actuator

Descripción: Los healthchecks permiten monitorear el estado de los microservicios y contenedores. En esta sección, aprenderás a utilizar Spring Boot Actuator para exponer endpoints de healthcheck y monitorear el estado de las aplicaciones.

- Endpoints de Actuator: Configuración de endpoints de salud (/actuator/health) y personalización de verificaciones de estado.
- Integración con Kubernetes: Configuración de liveness y readiness probes en Kubernetes utilizando los endpoints de Actuator.

Referencias:

- Spring Boot Actuator Documentation
- Kubernetes Probes

Código Ejemplo:

- Ejemplo de uso de Spring Boot Actuator con Kubernetes: GitHub - actuator-kubernetes-example

Sesión 6: Kubernetes

Objetivo: Implementar aplicaciones en Kubernetes

Sesión 6: Kubernetes

Arquitecturas On-premises vs Cloud

Descripción: Esta sección proporciona una comparativa detallada entre la implementación de microservicios en entornos on-premises y en la nube, destacando las diferencias en términos de infraestructura, escalabilidad, costos, y mantenimiento.

- On-premises: Ventajas y desventajas de mantener la infraestructura en local, incluyendo control total y costos fijos.
- Cloud: Beneficios de la escalabilidad automática, modelos de pago por uso, y alta disponibilidad en la nube.

Referencias:

- Cloud vs On-Premises: The Pros and Cons
- Understanding Cloud Native Applications

Código Ejemplo:

- No se requiere código para esta sección, enfoque en comparativas y decisiones estratégicas.

Sesión 6: Kubernetes

Conceptos de Kubernetes: Arquitectura e Implementación On-premises

Descripción: Kubernetes es la plataforma de orquestación de contenedores más popular. Esta sección introduce su arquitectura básica, componentes principales, y cómo desplegar un clúster en un entorno on-premises.

- Componentes de Kubernetes: Explicación de componentes clave como el API Server, etcd, Controller Manager, y kubelet.
- Despliegue On-premises: Instalación y configuración de un clúster Kubernetes en un entorno local utilizando herramientas como kubeadm o k3s.

Referencias:

- Kubernetes Documentation
- k3s Documentation for Lightweight Kubernetes

Código Ejemplo:

- Ejemplo de despliegue de un clúster con k3s: GitHub - k3s-example

Sesión 6: Kubernetes

Arquitectura e Implementación Kubernetes en Cloud

Descripción: Kubernetes es una plataforma de orquestación de contenedores ampliamente utilizada en entornos cloud. En esta sección, aprenderás cómo desplegar un clúster Kubernetes en proveedores cloud como AWS, Azure, o Google Cloud.

- Provisión de Clúster en Cloud: Uso de herramientas como EKS (AWS), AKS (Azure), o GKE (Google Cloud) para desplegar Kubernetes.
- Configuración y Escalabilidad: Configuración de nodos, networking, y almacenamiento. Estrategias para escalar automáticamente en función de la demanda.

Referencias:

- Amazon EKS Documentation
- Azure Kubernetes Service Documentation
- Google Kubernetes Engine Documentation

Código Ejemplo:

- Ejemplo de despliegue de un clúster en AWS EKS usando Terraform: GitHub - eks-terraform-example

Sesión 6: Kubernetes

Manifiestos de Despliegue en Kubernetes

Descripción: Los manifiestos de Kubernetes son archivos YAML que definen la configuración de recursos como Pods, Services, y Deployments. Esta sección detalla cómo crear y gestionar estos manifiestos para desplegar y escalar aplicaciones en Kubernetes.

- Creación de Manifiestos: Guía para escribir manifiestos básicos de Kubernetes, incluyendo Pods, Services, y Deployments.
- Gestión y Actualización: Uso de herramientas como kubectl para aplicar, actualizar y gestionar recursos en un clúster Kubernetes.

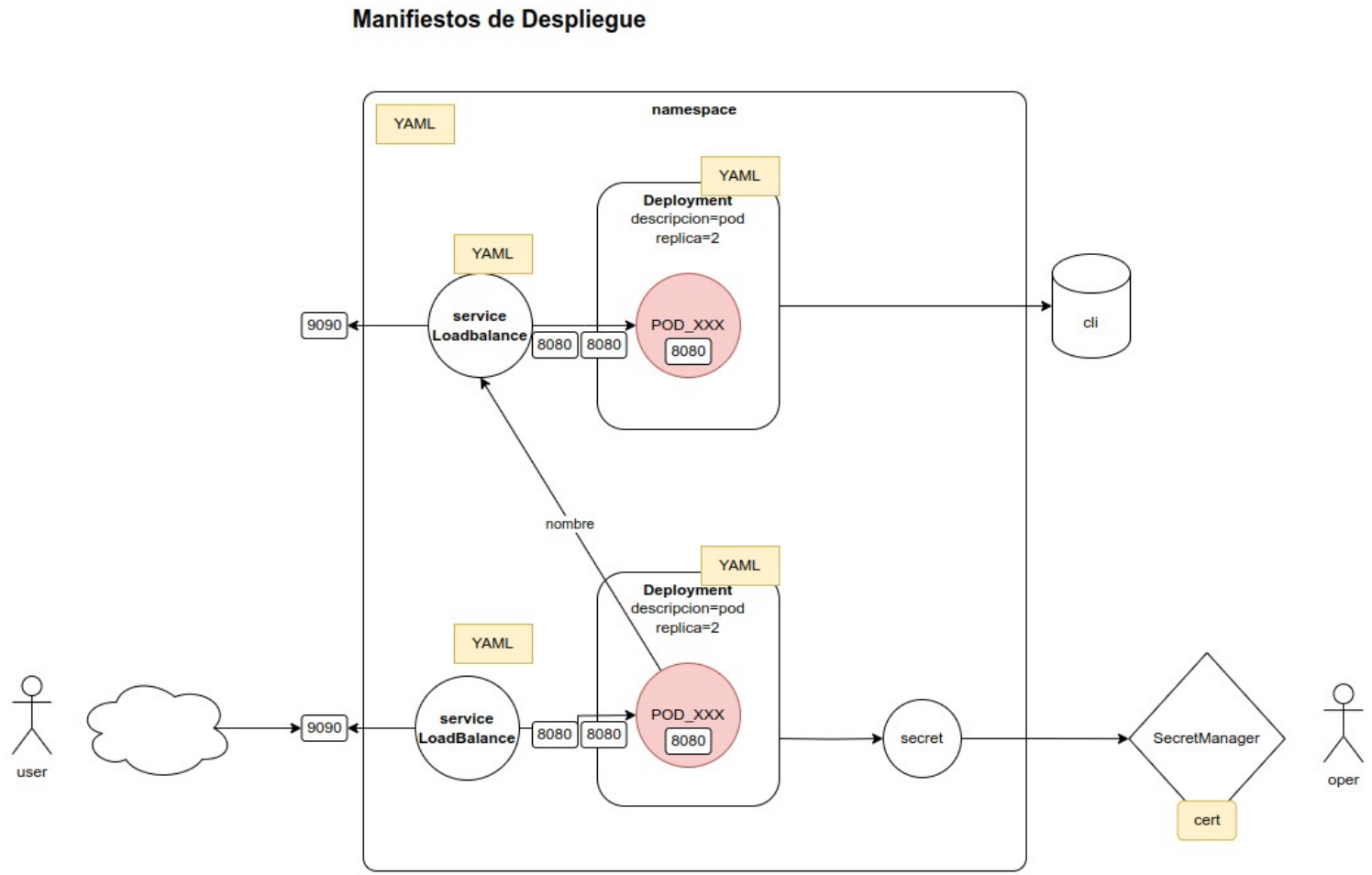
Referencias:

- Kubernetes Deployment Documentation
- Kubernetes Production Best Practices

Código Ejemplo:

- Ejemplo de manifiestos YAML para Kubernetes: GitHub - kubernetes-yaml-example

Sesión 6: Kubernetes



Sesión 7: DevSecOps

Objetivo: Comprender los conceptos y prácticas de automatización y control en el ciclo de vida de desarrollo.

Sesión 6: DevSecOps

Conceptos de DevSecOps

Descripción: DevSecOps es una práctica que integra la seguridad en cada fase del ciclo de vida del desarrollo. Esta sección explora cómo incluir controles de seguridad en pipelines CI/CD.

- Integración de Seguridad: Uso de herramientas como Snyk, Aqua Security, y Trivy para análisis de vulnerabilidades en contenedores e imágenes Docker.
- Compliance y Policies: Implementación de políticas de seguridad y cumplimiento en pipelines.

Referencias:

- DevSecOps Practices and Tools
- Snyk Documentation

Código Ejemplo:

- Pipeline de Jenkins con integración de seguridad usando Sonarqube: Jenkins-Sonarqube-example)

Sesión 6: DevSecOps

Pipelines de CI/CD

Descripción: La integración continua y el despliegue continuo son fundamentales para la entrega rápida y confiable de software. Aquí aprenderás a construir pipelines CI/CD optimizados para Kubernetes.

- Configuración de Pipelines: Creación de pipelines utilizando herramientas como Jenkins, GitLab CI/CD, y GitHub Actions.
- Despliegue en Kubernetes: Automatización del despliegue de aplicaciones en Kubernetes directamente desde el pipeline.

Referencias:

- Jenkins Pipeline Documentation
- GitLab CI/CD Documentation
- GitHub Actions Documentation
- Azure DevOps Documentation

Código Ejemplo:

- Ejemplo de pipeline CI/CD para despliegue en Kubernetes: GitHub - kubernetes-cicd-pipeline-example

Sesión 6: DevSecOps

Logging

Uso de Suite ELK para Gestión de Logs

Descripción: ELK (Elasticsearch, Logstash, Kibana) es una solución completa para la centralización, búsqueda, y visualización de logs. Aquí aprenderás a implementar la suite ELK en un clúster Kubernetes.

- Elasticsearch y Logstash: Configuración para recoger y almacenar logs de aplicaciones y sistemas.
- Kibana: Configuración para visualización de logs y generación de dashboards.

Referencias:

- Elastic Stack Documentation
- ELK Stack on Kubernetes

Código Ejemplo:

- Ejemplo de despliegue de ELK Stack en Kubernetes: GitHub - elk-stack-kubernetes-example

Ajuste de Logging en Spring Boot con Logback

Descripción: Logback es el framework de logging por defecto en Spring Boot. Esta sección detalla cómo ajustar la configuración de Logback para optimizar la captura y almacenamiento de logs en aplicaciones Spring Boot.

- Configuración de Logback: Personalización de logback-spring.xml para definir appender, logger, y patrones de logging.
- Integración con ELK: Cómo enviar logs de Spring Boot a Logstash para su posterior almacenamiento en Elasticsearch.

Referencias:

- Logback Project
- Spring Boot Logging Documentation

Código Ejemplo:

- Ejemplo de configuración avanzada de Logback: GitHub - spring-boot-logback-example

Sesión 6: DevSecOps

Estrategias de Despliegue en Kubernetes con GitOps

Descripción: GitOps es un enfoque que utiliza repositorios Git como fuente única de verdad para la automatización de despliegues en Kubernetes. Aquí explorarás cómo implementar GitOps para despliegues continuos y confiables.

- Herramientas GitOps: Uso de herramientas como ArgoCD y Flux para automatizar el despliegue y la gestión del ciclo de vida de aplicaciones en Kubernetes.
- Flujos de Trabajo GitOps: Implementación de pipelines GitOps para aplicar cambios automáticamente a través de repositorios Git.

Referencias:

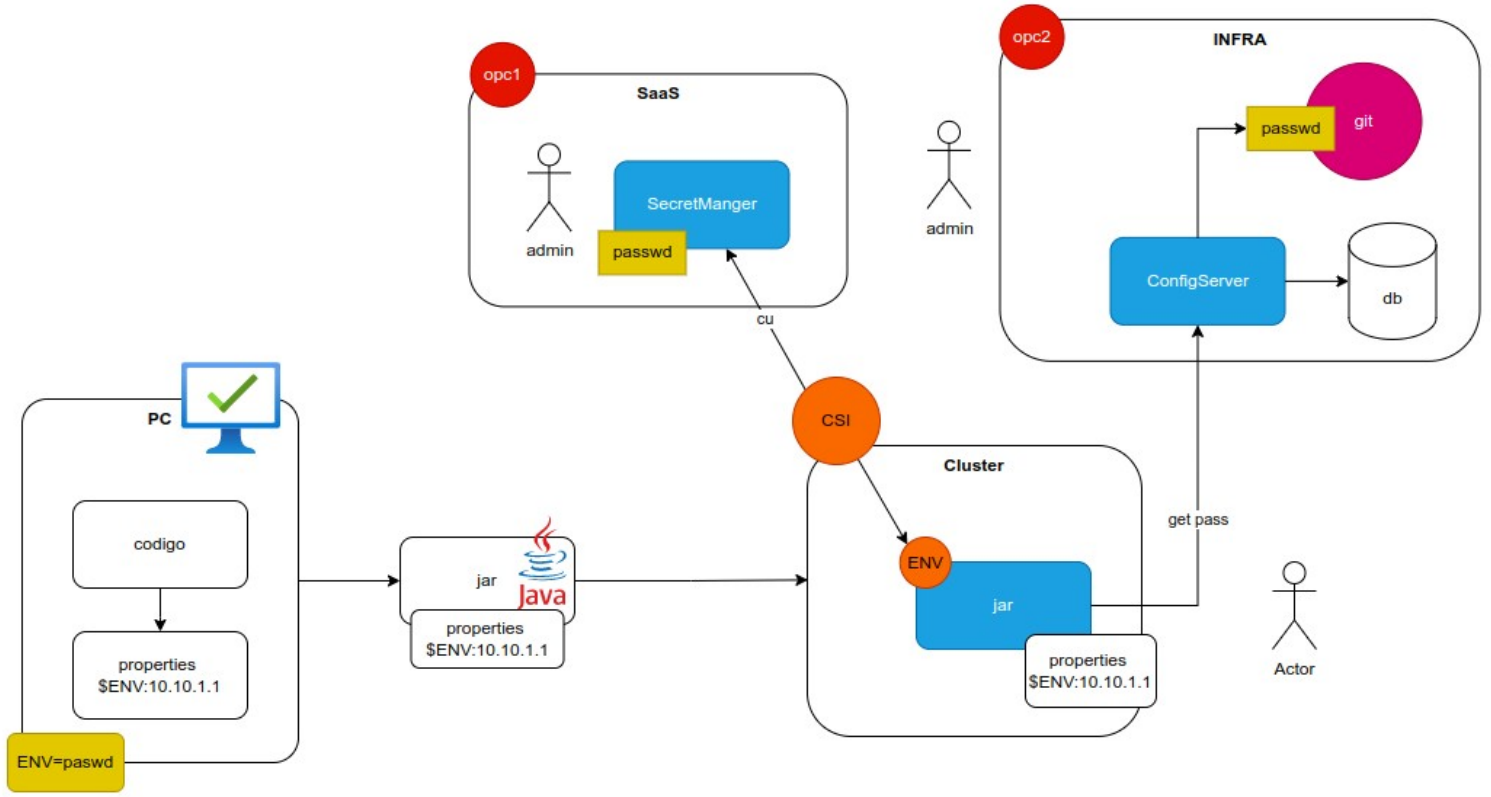
- What is GitOps?
- ArgoCD Documentation
- Flux CD Documentation

Código Ejemplo:

- Ejemplo de despliegue de aplicaciones con GitOps usando ArgoCD: GitHub - argocd-gitops-example

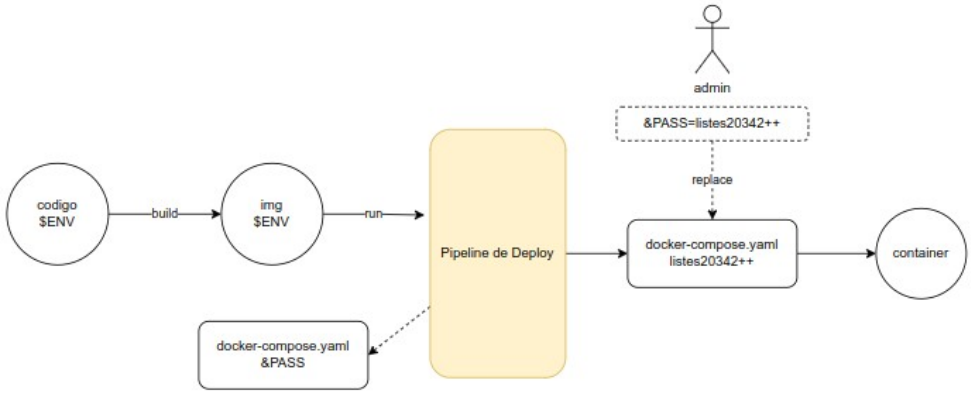
Sesión 6: DevSecOps

Opciones de gestión passwords

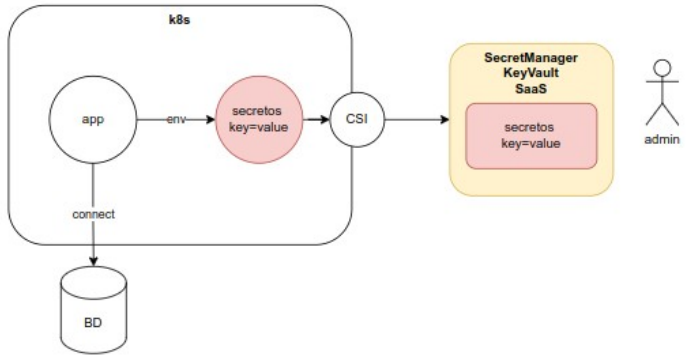


Sesión 6: DevSecOps

Gestión Secretos Pipeline



Gestión Secretos K8S



Sesión 8: Observabilidad en Microservicios

Objetivo: Integrar observabilidad y arquitecturas orientadas a eventos en microservicios.

Sesión 8: Observabilidad en Microservicios

Observabilidad

Collectores de OpenTelemetry

Descripción: OpenTelemetry es una plataforma de código abierto para la observabilidad, que proporciona APIs y herramientas para la recopilación y monitorización de trazas, métricas, y logs. En esta sección, aprenderás a integrar OpenTelemetry en tus microservicios para obtener visibilidad completa del sistema.

- Configuración de Collectores: Configurar collectores para capturar y exportar trazas y métricas a sistemas de almacenamiento o análisis.
- Integración con Microservicios: Instrumentación de microservicios para generar trazas y métricas utilizando OpenTelemetry.

Referencias:

- OpenTelemetry Documentation
- Spring Boot and OpenTelemetry Integration

Código Ejemplo:

- Ejemplo de integración de OpenTelemetry en una aplicación Spring Boot: GitHub - opentelemetry-spring-boot-example

Sesión 8: Observabilidad en Microservicios

Grafana y Prometheus

Descripción: Grafana y Prometheus son herramientas populares para la visualización y monitoreo de sistemas. Prometheus actúa como un sistema de monitoreo y almacenamiento de series temporales, mientras que Grafana se usa para la visualización de datos.

- Configuración de Prometheus: Implementación de Prometheus para recopilar métricas desde aplicaciones y sistemas.
- Dashboards en Grafana: Creación de dashboards personalizados en Grafana para visualizar métricas y trazas recolectadas.

Referencias:

- Prometheus Documentation
- Grafana Documentation

Código Ejemplo:

- Ejemplo de configuración de Prometheus y Grafana para monitorear un clúster Kubernetes: GitHub - prometheus-grafana-k8s-example

Sesión 9: Arquitecturas EDA

Objetivo: Comprender los conceptos de las arquitecturas orientadas a eventos

Sesión 9: Arquitecturas EDA (Event-Driven Architecture)

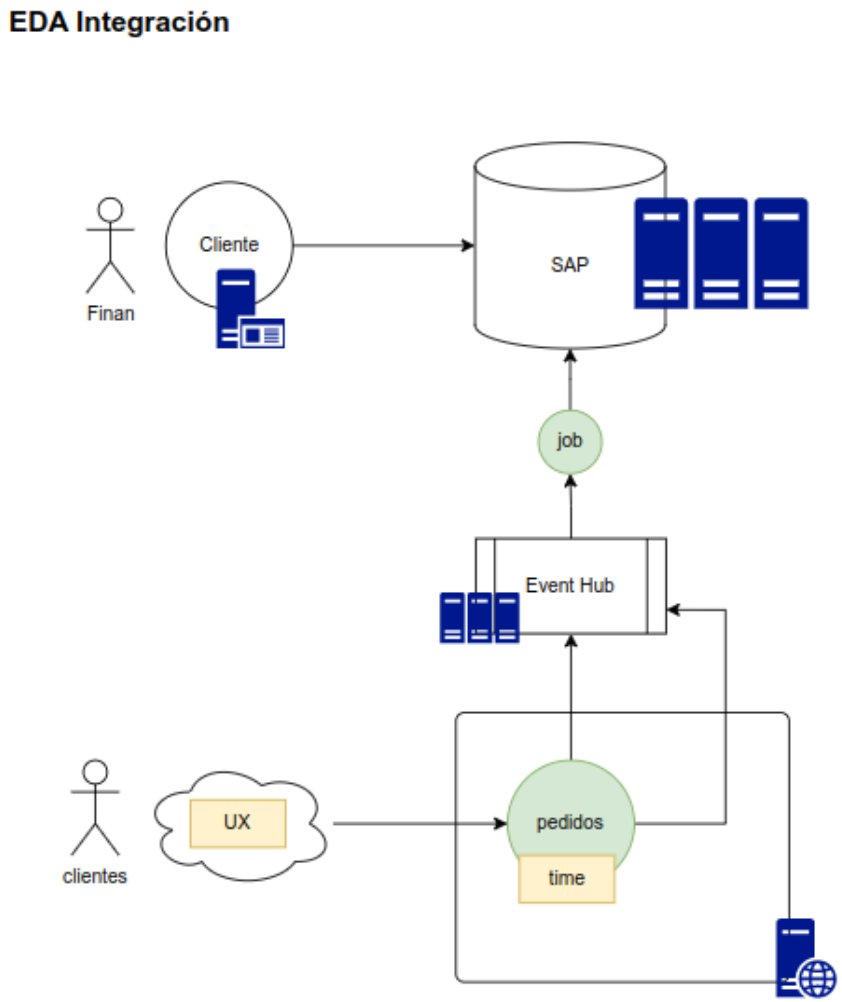
Implementación de Arquitecturas Orientadas a Eventos

Descripción: Las arquitecturas orientadas a eventos (EDA) permiten la construcción de sistemas reactivos y desacoplados mediante la emisión y consumo de eventos. En esta sección, exploraremos cómo implementar EDA en microservicios.

- Diseño de EDA: Principios básicos del diseño de sistemas orientados a eventos, incluyendo la modelación de eventos y flujos de datos.
- Integración de Microservicios en EDA: Cómo los microservicios pueden interactuar mediante eventos en una arquitectura distribuida.

Referencias:

- Event-Driven Architecture Overview
- Spring Boot Event-Driven Microservices



Sesión 9: Arquitecturas EDA (Event-Driven Architecture)

Broker de Mensajería RabbitMQ

Descripción: RabbitMQ es un broker de mensajería ampliamente utilizado en arquitecturas orientadas a eventos. Facilita la comunicación entre microservicios mediante el intercambio de mensajes asíncronos.

- Configuración de RabbitMQ: Instalación y configuración de RabbitMQ para gestionar colas y enrutamiento de mensajes.
- Integración con Microservicios: Uso de RabbitMQ como broker de mensajería para enviar y recibir mensajes entre microservicios.

Referencias:

- RabbitMQ Documentation
- Spring AMQP with RabbitMQ

Código Ejemplo:

Ejemplo de implementación de RabbitMQ en microservicios Spring Boot: GitHub - spring-rabbitmq-example

Sesión 9: Arquitecturas EDA (Event-Driven Architecture)

Message-Driven Beans (MDB)

Descripción: Message-Driven Beans (MDB) son un componente de la arquitectura Java EE para recibir mensajes de forma asíncrona. En este contexto, se integran con brokers como RabbitMQ para manejar la lógica de procesamiento de eventos.

- Configuración de MDB: Configuración de beans que manejan mensajes asíncronos en aplicaciones Java EE.
- Integración con RabbitMQ: Uso de MDB en conjunto con RabbitMQ para procesar mensajes en una arquitectura orientada a eventos.

Referencias:

- Using MDB with RabbitMQ

Código Ejemplo:

- Ejemplo de implementación de MDB con RabbitMQ: GitHub - rabbitmq-mdb-example

Muchas gracias