

MICROSERVICIOS CON C/C++ UTILIZANDO CONTENEDORES Y KUBERNETES

Versión 2:

1. Introducción a los Microservicios con C/C++

- Fundamentos de los microservicios:

- Ventajas frente a arquitecturas monolíticas.
- Desafíos específicos al trabajar con lenguajes de bajo nivel como C/C++.
- Casos de uso en sistemas embebidos, sistemas de alto rendimiento, y telecomunicaciones.

- Comparación con otros lenguajes de programación en arquitecturas de microservicios:

- Cuando elegir C/C++ frente a lenguajes como Java, Python o Go.

2. Creación de APIs RESTful con C/C++

- Introducción a los protocolos HTTP, HTTPs y WebSockets

- Peticiones http. Ejemplo de un servidor Http con la librería boost::asio

- Implementación de APIs en C/C++:

- Uso del framework crow
- Creación de rutas para GET, POST, PUT, DELETE.
- Manejo de respuestas HTTP y serialización/deserialización de datos en JSON.

- Serialización / deserialización de datos con JSON y la librería nlohmann/json

- Comentar los problemas con: Cpp-REST-SDK y Pistache.

- Introducción a la librería: boost.beast

- Comparativa de las librerías: crow vs boost.beast

- Gestión de errores y excepciones:

- Manejo de códigos de error y optimización del flujo de datos.
- Implementación de registros de errores y mensajes de diagnóstico.

3. Comunicación entre Microservicios en C/C++

- Implementación de patrones de mensajería:
 - Uso de ZeroMQ, gRPC y RabbitMQ para comunicación asincrónica.
 - Comparativa entre REST y gRPC en sistemas de alto rendimiento.
 - Implementación de WebSockets para la comunicación en tiempo real.
- Gestión de la concurrencia y el multithreading:
 - Uso de `std::thread` y `boost::asio` para manejar múltiples solicitudes.
 - Estrategias de sincronización y control de acceso concurrente

4A. Docker (se adelanta en el temario para verlo antes de la persistencia)

- Introducción a Docker, diferencias entre los contenedores y la virtualización.
 - Comandos, herramientas, el dockerhub
 - Contenedores, imágenes, volúmenes y redes
 - Los ficheros dockerfile
 - Despliegues con Docker-compose
 - Despliegue de contenedores con bases de datos

4B. Gestión del Estado y Persistencia en C/C++

- Conexión a bases de datos relacionales y no relacionales:
 - Integración con MySQL, PostgreSQL y bases de datos NoSQL como MongoDB.
 - Uso de librerías ORM como `soci` o `libpqxx`.
- Estrategias de persistencia y acceso rápido a datos:
 - Caching con Redis o Memcached en aplicaciones C++.
 - Implementación de acceso en tiempo real a grandes volúmenes de datos.

5. Optimización y Gestión de Recursos en Microservicios C/C++

- Optimización de rendimiento en microservicios de alto rendimiento:
 - Uso de técnicas de gestión de memoria eficiente (malloc/free, RAII).
 - Minimización de la latencia en sistemas distribuidos con C++.
- ~~Profiling y benchmarking de microservicios:~~
- ~~Uso de herramientas como Valgrind, gprof o perf.~~
- ~~Estrategias para mejorar la eficiencia del CPU y la memoria.~~

6. Seguridad y Autenticación en Microservicios con C/C++

- Autenticación con JWT (JSON Web Tokens):
- Generación y validación de tokens JWT en C++.
- Uso de bibliotecas criptográficas como OpenSSL para manejar tokens seguros.
- Protección de la comunicación entre microservicios:
- Implementación de encriptación con SSL/TLS.
- Configuración de políticas de seguridad para prevenir ataques como CSRF y XSS.

7. Despliegue de Microservicios C/C++ con Docker y Kubernetes

- Contenerización de microservicios en C/C++ con Docker:
 - Creación de Dockerfiles optimizados para aplicaciones C/C++.
 - Gestión de dependencias y bibliotecas externas dentro de los contenedores.
- Orquestación de microservicios con Kubernetes:
 - Creación de Pods, Deployments, y Services en Kubernetes.
 - Estrategias de despliegue y escalabilidad en Kubernetes.
- Despliegue automatizado con Helm:
 - Uso de Helm Charts para la automatización de despliegues y actualizaciones.
 - Gestión de configuraciones de microservicios en entornos de producción.

8. Monitorización y Logging en Microservicios C/C++

- Monitorización del rendimiento en tiempo real:

- Uso de Prometheus y Grafana para la recolección y visualización de métricas.
- Integración con cAdvisor para monitorizar contenedores.

- Implementación de tracing distribuido:

- Uso de Jaeger o OpenTelemetry para tracing distribuido.
- Análisis de la latencia y tiempos de respuesta en microservicios C++.

- Gestión de logs en entornos distribuidos:

- Uso de Fluentd y Elasticsearch para la centralización de logs.
- Búsqueda y análisis de logs en sistemas distribuidos."