|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **MATERIA** | Desarrollo Web Avanzado | **NRC** | 2296 | **TRABAJO No.** | 1 |
| **CARRERA** | Ing. de Software | **DOCENTE** | | Edison Lascano | |
| **PERIODO ACADÉMICO** | OCT 24 - MAR 25 | **FECHA** | | 29/12/2024 | |
| **TEMA** | Patrón de arquitectura de microservicios | | | | |
| **ESTUDIANTE(S)** | Matías Suárez | | | | |

|  |
| --- |
| **A.** **INTRODUCCION** |

La arquitectura de microservicios es una táctica de programación que organiza una aplicación como un grupo de servicios pequeños, autónomos y disponibles de manera individual. Este método facilita la implementación de modificaciones de manera rápida, regular y fiable, de acuerdo con los estándares de métricas de rendimiento como DORA. Es esencial para empresas que requieren ajustarse con rapidez a un ambiente de negocios en constante cambio.

|  |
| --- |
| **B.** **DESARROLLO** |

**Patrón de arquitectura de microservicios**

**Organización del grupo de trabajo:**

* Equipos reducidos, autónomos y multifacéticos (de acuerdo con "Topologías de Equipo").
* Implementación constante de prácticas DevOps.
* Modificaciones regulares y mínimas verificadas a través de líneas de producción automatizadas.

**Dominios Subdominios:**

* Cada subdominio representa una funcionalidad corporativa particular.
* Se compone de entidades corporativas (agregados DDD) y adaptadores que se relacionan con el entorno externo.
* Caso en Java: subdominios codificados en archivos JAR.

**Actividades del sistema:**

* Incluidas de manera síncrona, asíncrona o a través de sucesos.
* Cambian y revisan entidades de negocios en los subdominios.

**Problema**

¿Cómo organizar los subdominios en componentes desplegables y ejecutables, equilibrando simplicidad, autonomía de equipo y eficiencia?

**Fuerzas de Estructuración**

1. **Fuerzas internas de "energía oscura":**

* Elementos sencillos: Simples de comprender y preservar.
* Independencia del equipo: Independencia para crear, probar y poner en práctica.
* Rápido pipeline: Elementos de construcción y prueba rápidos y pequeños.
* Apoyo para diversas tecnologías: Capacitación en las baterías tecnológicas.
* Separación por atributos: Optimización de escalabilidad, protección y disponibilidad.

1. **Interacciones con fuerzas de "materia oscura":**

* Simples interacciones: Operaciones locales o con escasas interacciones entre los elementos.
* Interacciones productivas: Reducir transacciones de datos y ciclos de red.
* Preferencia por ACID en comparación con BASE: Desarrollar operaciones parecidas a las transacciones ACID.
* Reducir el acoplamiento durante el proceso de ejecución: Optimizada disponibilidad y reducida latencia.
* Reducir acoplamiento durante el diseño: Disminuir las modificaciones sincronizadas entre servicios.

**Solución**

1. **Organización de la aplicación:**

* Separar en servicios autónomos, cada uno vinculado a uno o varios subdominios.
* Cada servicio cuenta con su propia fuente de datos y cadena de despliegue.
* Empleo de un enlace API como entrada.

1. **Reparto de operaciones:**

* Operaciones a nivel local (dentro de un servicio) y dispersas (entre varios servicios).
* Desarrollo de operaciones distribuidas a través de esquemas de cooperación.

1. **Ejemplos de cooperación:**

* Crónica: Ciclo de operaciones locales.
* Reemplazos del lado de control: Información de lectura única replicada en servicios pertinentes.
* Estructura de API: Consultas distribuidas en un orden determinado de consultas locales.
* CQRS: División de instrucciones y consultas en diversos modelos.

**Ventajas**

* Sencillez: Servicios reducidos, sencillos de comprender y conservar.
* Autonomía: Grupos autónomos para el desarrollo y montaje.
* Rápido pipeline: Elementos rápidos para construir y examinar.
* Capacitación tecnológica: Apoyo para diversas tecnologías.
* Separación por atributos: Optimización de la escalabilidad y la seguridad.

**Desventajas**

* Inconvenientes: Restricciones Inconvenientes:
* Manejo distribuido complejo y complicado de depurar.
* Potencial ineficiencia en operaciones de distribución.
* Estructura complicada de las transacciones no-ACID.
* Ejecución en tiempo real que puede disminuir la disponibilidad.
* Danger de modificaciones coordinadas que impactan en la productividad.

**Casos de uso conocidos**

* **Netflix:** Arquitectura escalable basada en microservicios con más de mil millones de llamadas API diarias.
* **Amazon:** De arquitectura monolítica a cientos de servicios backend independientes.
* **eBay:** Aplicaciones independientes que gestionan funciones específicas como compra y venta.

|  |
| --- |
| **C.** **CONCLUCION** |

1. La estructura de microservicios es apropiada para sistemas complejos que demandan una elevada escalabilidad, adaptabilidad y entrega constante. El procedimiento de Assemblage estructura subdominios y enfrenta los retos fundamentales para construir una arquitectura eficaz y bien organizada.

|  |
| --- |
| **D.** **BIBLIOGRAFIA** |

Microservices Pattern: Microservice Architecture pattern. (s/f). microservices.io; Chris Richardson. Recuperado el 5 de enero de 2025, de <https://microservices.io/patterns/microservices.html>