Arquitectura dirigida a eventos

¿Qué es?

es un modelo y una arquitectura de software que sirve para diseñar aplicaciones. En un sistema como este, la captura, la comunicación, el procesamiento y la permanencia de los eventos son la estructura central de la solución.

Ventajas

- **Desacoplo de servidor y clientes:** Los servidores y los clientes únicamente deberán usar la misma estructura en los mensajes de los eventos, pero se pueden programar en distintos lenguajes de programación y ser totalmente independientes
- **Desacoplo entre dispositivos:** cada dispositivo solo tiene que comunicarse con el servidor. De esta manera, se puede relacionar cualquier dispositivo entre sí sin límite
- **Altamente escalable:** Gracias al desacoplo entre dispositivos se puede añadir fácilmente cualquier nuevo servicio al sistema.

Desventajas

- El servidor nunca podrá saber si un servicio este caído, ya que no se cuenta con ninguna conexión real entre ellos.
- Un dispositivo tampoco sabrá si el servidor este caído
- No se puede comprobar si un evento ha sido atendido

Características

- Se utiliza para dispararse y comunicarse entre servicios desacoplados y es común en aplicaciones modernas construidas con microservicios
- Consiste en creadores de eventos, gestores de eventos y consumidores de eventos
- No tiene un tiempo de respuesta determinado para procesar eventos input, pero es mucho más rápido adaptándose a cambios

¿Por qué es importante?

Varias aplicaciones modernas se basan en eventos, como los marcos de interacción con los clientes, que deben utilizar los datos de los clientes de formas inmediata.

Las aplicaciones con esta arquitectura opueden crearse con cualquier lenguaje de programación ya que la expresión basado en eventos se refiere a un enfoque de programación y no a un lenguaje especifico.

Herramientas

Creately. Edraw.

Zen Flowchart.	Diagrams.
SmartDraw.	Canva.
Lucidchart.	StarUML

N capas

Descripción

Una arquitectura de n niveles divide una aplicación en capas lógicas y niveles físicos. Las capas son una forma de separar responsabilidades y administrar dependencias.

Cada capa tiene una responsabilidad específica. Una capa superior puede utilizar los servicios de una capa inferior, pero no al revés.

Los niveles están físicamente separados y se ejecutan en máquinas diferentes. Un nivel puede llamar a otro nivel directamente o usar mensajería asincrónica (cola de mensajes).

Ventajas

- Portabilidad entre la nube y entornos locales y entre plataformas en la nube
- Menor curva de aprendizaje para la mayoría de los desarrolladores
- Evolución natural desde el modelo de aplicaciones tradicional
- Abiertas a entornos heterogéneos (Windows o Linux)

Desventajas

- Es fácil de terminar con un nivel intermedio que solo realiza operaciones CRUD en base de datos, agregando latencia adicional sin hacer ningún trabajo útil.
- El diseño monolítico impide la implementación independiente de características
- La administración de una aplicación laas es más trabajo que una aplicación que utiliza solo los servicios administrados
- Puede ser difícil administrar la seguridad de red en un sistema grande

Características

- Es un estilo para definir el despliegue de las capas en una instalación
- Se obtiene una descomposición funcional de la aplicación, servicios e instalación
- Mejora la escalabilidad, disponibilidad, administración y utilización de recursos
- Tiene al menos tres capas separados o partes, cada una de ellas con su responsabilidad y está localizada en diferentes servidores

¿Por qué es importante?

Mejorando en las posibilidades de mantenimiento. Debido a que cada capa es independiente de la otra los cambios o actualizaciones pueden ser realizados sin afectar la aplicación como un todo.

Herramientas

- Capa de presentación (Interfaz gráfica)
- Modelo vista controlador (MVC)
- Atom.
- Notepad++
- Sublime Text.
- Adobe Dreamweaver CC.
- Capa de negocio o dominio
- Entidad de dominio, modelo de dominio u objeto de negocio
- Objeto de valor
- Ruta agregada.
- Script de transacción
- Tabla modular
- OK Builder.
- WordPress.
- Adobe Dreamweaver.
- Adobe Muse.

Cliente servidor

Descripción

es uno de los estilos arquitectónicos distribuidos más conocidos, el cual está compuesto por dos componentes, el proveedor y el consumidor. El proveedor es un servidor que brinda una serie de servicios o recursos los cuales son consumido por el Cliente.

Ventajas

- Centralización: El servidor fungirá como única fuente de la verdad, lo que impide que los clientes conserven información desactualizada.
- Seguridad: El servidor por lo general está protegido por firewall o subredes que impiden que los atacantes pueden acceder a la base de datos o los recursos sin pasar por el servidor.
- Fácil de instalar (cliente): El cliente es por lo general una aplicación simple que no tiene dependencias, por lo que es muy fácil de instalar.
- Separación de responsabilidades: La arquitectura cliente-servidor permite implementar la lógica de negocio de forma separada del cliente.

Desventajas

- Actualizaciones (clientes): Una de las complicaciones es gestionar las actualizaciones en los clientes, pues puede haber muchos terminales con el cliente instalado y tenemos que asegurar que todas sean actualizadas cuando salga una nueva versión.
- Concurrencia: Una cantidad no esperada de usuarios concurrentes puede ser un problema para el servidor, quien tendrá que atender todas las peticiones de

forma simultánea, aunque se puede mitigar con una estrategia de escalamiento, siempre será un problema que tendremos que tener presente.
- Todo o nada: Si el servidor se cae, todos los clientes quedarán totalmente inoperables.

Caracteristicas

- El Cliente y el Servidor pueden actuar como una sola entidad y también pueden actuar como entidades separadas, realizando actividades o tareas independientes.
- Las funciones de Cliente y Servidor pueden estar en plataformas separadas, o en la misma Plataforma
- Su representación típica es un centro de trabajo (PC), en donde el usuario dispone de sus propias aplicaciones de oficina y sus propias bases de datos, sin dependencia directa del sistema central de información de la organización.

¿Por qué es importante?

es que permite conectar a varios clientes a los servicios que provee un servidor y como sabemos hoy en día, la mayoría de las aplicaciones y servicios tienen como gran necesidad que puedan ser consumidos por varios usuarios de forma simultánea.

Herramientas

- Easel
- SQLNetwork de Gupta
- Extra de Attachmate
- Visual Basic de Microsoft
- ObjectVision de Borland
- ObjectView de KnowledgeWare
- SQLWindows de Gupta
- PowerBuilder de PowerSoft
- Easel Workbench de Easel
- CA-dBFast de Computer Associates
- Uniface.

Microkernel

Descripción

Se dividen en dos tipos de componentes, en **Sistema Core y los Plugin**, el sistema Core contiene los elementos mínimos, por otra parte, los módulos con componentes periféricos que se añaden o instalan al componente Core para extender su funcionalidad.

Ventajas

- Reducción de la complejidad, la descentralización de los fallos y el depurar controladores de dispositivos
- Es posible reutilizar los módulos en varias instancias, incluso, es posible comercializarlas de forma independiente

Desventajas

- Sus principales dificultades son la complejidad en la sincronización de todos los módulos lo que componen.
- Sus detractores le achacan, fundamentalmente, mayor complejidad en el código, menor rendimiento, o limitaciones en diversas funciones.

Caracteristicas

- Manipulación de las interrupciones en el sistema desde dispositivos físicos.
- Manipula excepciones del procesador.
- Proporciona soporte para la recuperación de un sistema con alguna falla de caída de energía.

¿Por qué es importante?

El microkernel tiene la particularidad de que las funciones centrales **son controladas por un núcleo o kernel** quien interactúa directamente con el hardware y la interfaz del usuario. La arquitectura de microkernel **es complejo, sofisticado, pero más centrado** en su quehacer para el SO, toda acción pasa por el microkernel, lo cual hace a un SO.

Microservicios

Descripción

Conjunto de pequeños servicios que se ejecutan de manera independiente y autónoma, Se comunican entre sí a través de Apis, y cuentan con sistemas de almacenamiento propios.

Cada microservicio es código.

Ventajas

- Modularidad
- Escalabilidad
- Versatilidad
- Rapidez de actuación
- Mantenimiento simple y barato

Desventajas

- Alto consumo de memoria
- Inversión de tiempo inicial
- Complejidad de la gestión

- Dificultad en la realización de pruebas

Caracteristicas

- Alto nivel de desacoplamiento
- Resiliencia
- Escenario profesional
- Independencia

¿Por qué es importante?

- Elimina los desafíos asociados con las aplicaciones monolíticas
- Implementa y beneficia de las tecnologías más modernas a escala web
- Reducir la utilización de recursos
- Añadir rápidamente nuevas funciones sin interrumpir el servicio
- Integrar fácilmente las soluciones desarrolladas por terceros.

Herramientas

- Eureka
- Ribbon
- Zuul
- Hystrix
- Turbine

Serveless

Descripcion

es un modelo de desarrollo nativo de la nube que permite a los desarrolladores crear y ejecutar aplicaciones sin tener que administrar servidores

Ventajas

- La informática sin servidor puede aumentar la productividad del desarrollador y reducir los costos operativos.
- 2. Serverless ayuda a habilitar la adopción de DevOps.
- 3. Es posible agilizar aún más el desarrollo de aplicaciones mediante la incorporación de componentes completos de ofertas BaaS de terceros.
- 4. Los costos operativos se reducen en un modelo sin servidor.

Desventajas

- No ejecutar su propio servidor o controlar su propia lógica del lado del servidor puede tener inconvenientes.
- 2. Los proveedores de la nube pueden tener restricciones estrictas sobre cómo se puede interactuar con sus components.

- 3. Ceder el control de estos aspectos de su pila de TI también lo abre al bloqueo del proveedor.

Características

- Ejecución bajo demanda, Se ejecutan bajo demanda cuando se invocan y permanecen inactivos hasta que los eventos los invocan.
- Elástico, En pocas palabras, el escalado no puede ser controlado por usted, y usted depende de la plataforma subyacente para que la solución en la nube se amplíe.
- Sin anfitrión, En el caso de las soluciones sin servidor, el proveedor de la nube abstrae del servidor subyacente y del entorno de alojamiento.
- Distribuido, Cada uno de los componentes de la solución sin servidor debe realizar la tarea que necesita.
- Menor tiempo de ejecución, El código de aplicación en un servicio sin servidor se ejecuta cada vez que un evento invoca el servicio.

¿Por qué es importante?

- . Coste: la ventaja principal de utilizar informática sin servidor es el hecho de que ésta te proporciona servicios solo por lo que realmente utilizas.
- 2. Mantenimiento: estos sistemas no presentan servidores o sistemas operativos que mantener.
- 3. Escalamiento fácil y eficiente: las aplicaciones sin servidor se pueden escalar automáticamente o como máximo con unos pocos clics para elegir la capacidad deseada.
- 4. Alta disponibilidad: las aplicaciones sin servidor tienen disponibilidad incorporada.
- 5. Autonomía para la empresa: La informática sin servidor brinda a las organizaciones la libertad de concentrarse en sus ofertas comerciales principales y olvidarse de los problemas de servidores de bajo nivel.

Herramientas

- The Serveless framework
- Localstack
- Openfaas
- Firecracker
- Chromeless
- Zappa
- Up
- Chalice
- AWS Serverless Application Model
- Kubeless