UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CS2032 - Cloud Computing (Ciclo 2024-2)
Virtualización con contenedores
Semana 4 - Clase 1: Contenedores (Parte 2)

ELABORADO POR: GERALDO COLCHADO

Contenido

Contenedores

1. **Objetivo de la sesión**
2. Contenedores vs Máquinas virtuales
3. Arquitectura de Solución
4. Patrón de Arquitectura Monolítica
5. Patrón de Arquitectura de Microservicios
6. Arquitectura Monolítica vs Microservicios
7. Cierre

Objetivo de la sesión
Contenedores (Parte 2)

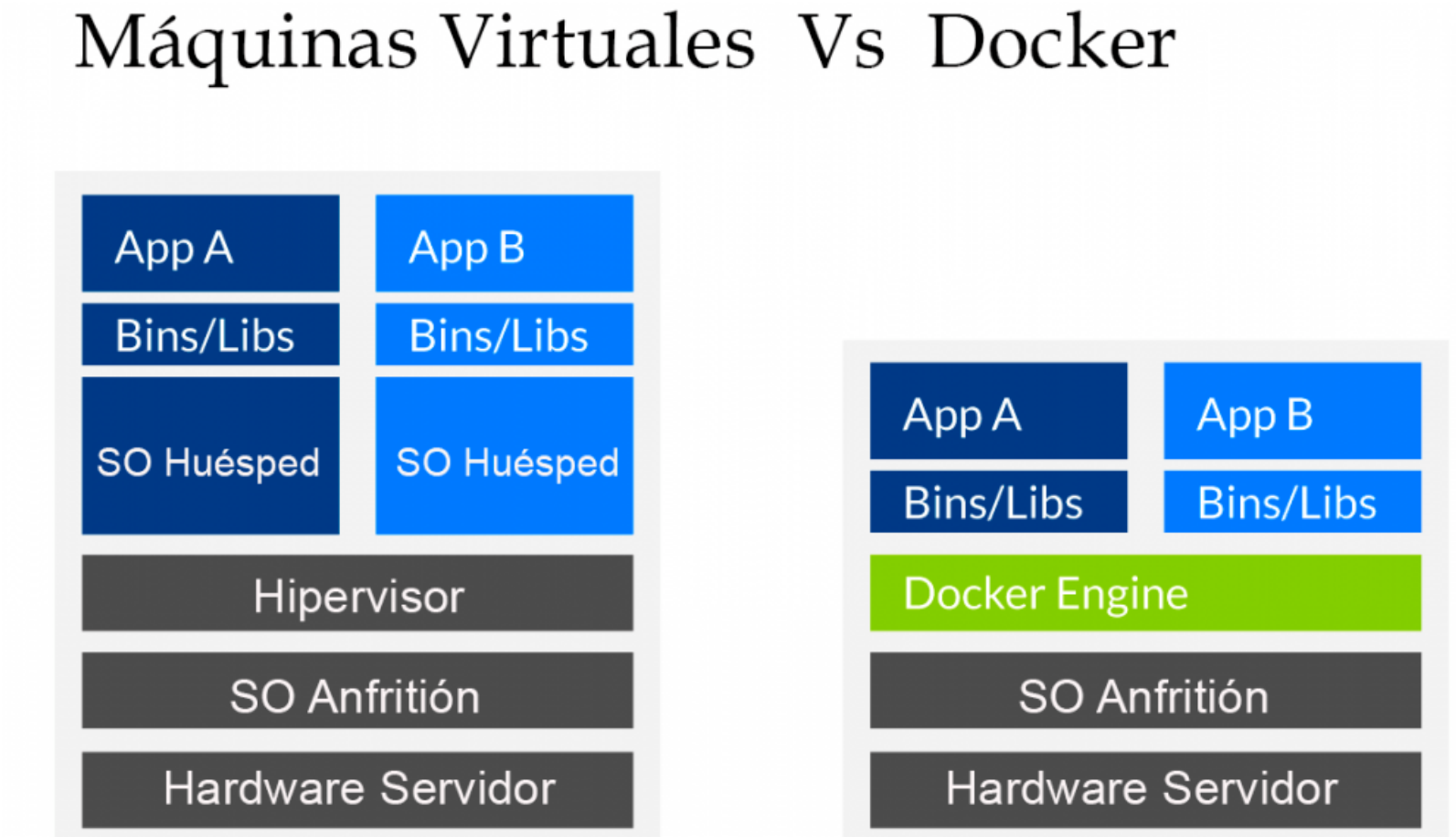
- Comparar Contenedores vs Máquinas virtuales
- Entender que es la Arquitectura de Solución
- Aprender el Patrón de Arquitectura Monolítica
- Aprender el Patrón de Arquitectura de Microservicios
- Comparar Arquitectura Monolítica vs Microservicios

Contenido

Contenedores

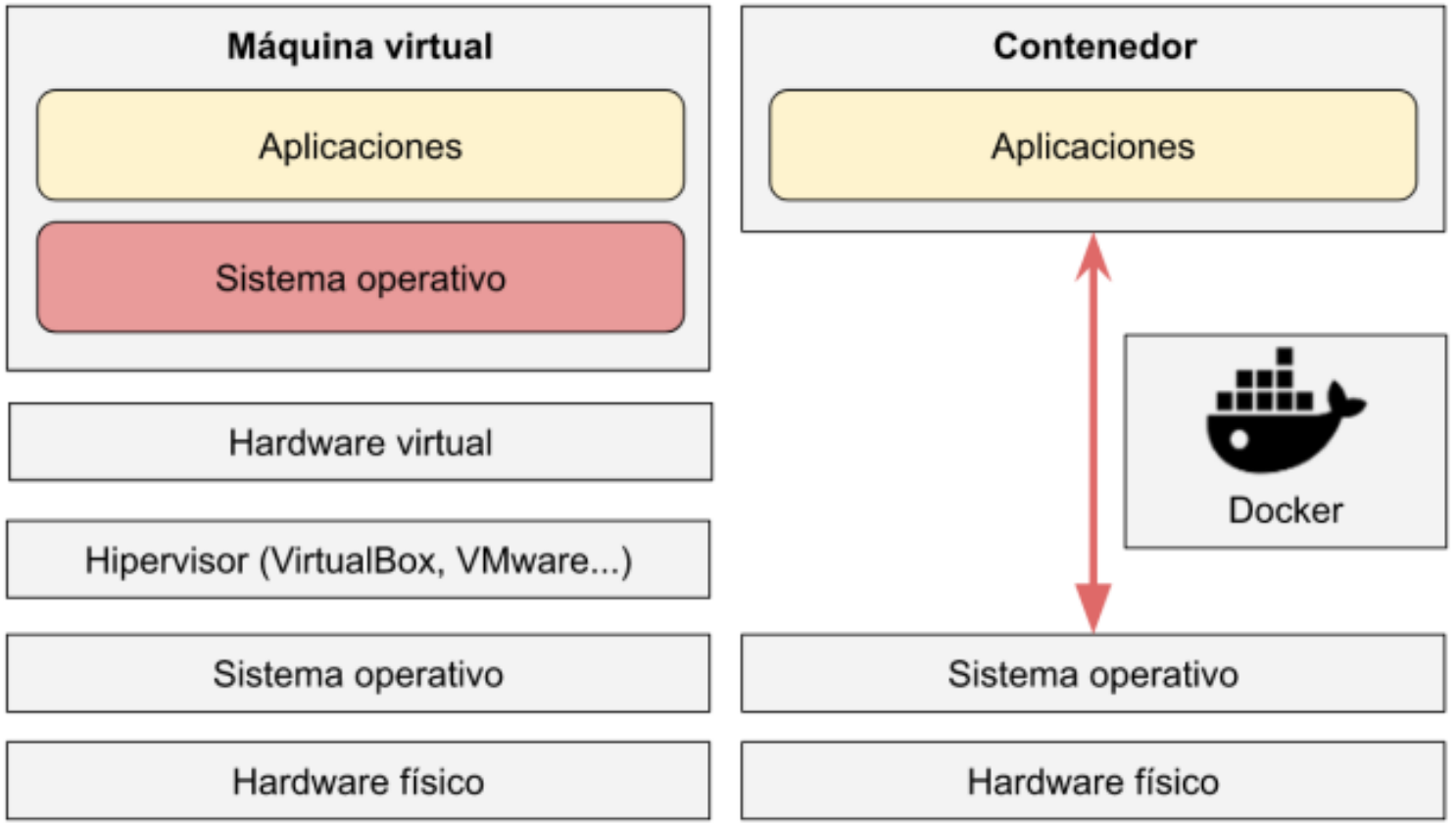
1. Objetivo de la sesión
2. **Contenedores vs Máquinas virtuales**
3. Arquitectura de Solución
4. Patrón de Arquitectura Monolítica
5. Patrón de Arquitectura de Microservicios
6. Arquitectura Monolítica vs Microservicios
7. Cierre

Contenedores vs Máquinas virtuales
Comparación



Referencia: http://unforoinformatico.emiweb.es/blog/seguridad/que-es-dockers.html

Contenedores vs Máquinas virtuales
Comparación



Referencia: https://danielme.com/2021/04/27/curso-jakarta-ee-docker-introduccion/

Contenedores vs Máquinas virtuales (VM)
Comparación

Característica	Máquina Virtual	Contenedor
Isolation (Aislamiento)	Completo aislamiento del sistema operativo host y de otras máquinas virtuales (VM).	Aislamiento ligero del host y otros contenedores, pero no proporciona un límite de seguridad tan fuerte como una VM.
Operating system (Sistema Operativo)	Ejecuta un sistema operativo completo, incluido el kernel, por lo que requiere más recursos del sistema (CPU, memoria y almacenamiento).	Ejecuta la parte del modo de usuario de un sistema operativo y se puede adaptar para contener solo los servicios necesarios para su aplicación, utilizando menos recursos del sistema.
Guest compatibility (SO de máquina virtual o de contenedor)	Ejecuta casi cualquier sistema operativo dentro de la máquina virtual.	Se ejecuta en la misma versión del sistema operativo que el host.
Deployment (Despliegue)	Se puede automatizar la creación de una o más máquinas virtuales con comandos.	Se puede automatizar la creación de un contenedor con comandos de Docker y varios contenedores, a demanda, con un orquestador como Kubernetes.
Operating system updates and upgrades (Actualización y Mejora del Sistema Operativo)	Debe descargar e instalar las actualizaciones del sistema operativo en cada VM.	También se puede actualizar el sistema operativo de una imagen actualizando el Dockerfile para usar una imagen base más reciente y haciendo build.

Referencia: https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm

Contenedores vs Máquinas virtuales
Comparación

Característica	Máquina Virtual	Contenedor
Persistent storage (Almacenamiento persistente)	Se puede asignar un almacenamiento propio para la VM o almacenamiento compartido entre varias VM.	El contenedor tiene almacenamiento propio y también puede tener almacenamiento compartido entre varios contenedores.
Load balancing (Balanceo de carga)	El balanceador de carga balancea el tráfico entre todas las VM en ejecución.	El orquestador (Ej. Kubernetes) puede iniciar o detener automáticamente los contenedores en los nodos del clúster para administrar los cambios en la carga y la disponibilidad.
Fault tolerance (Tolerancia a fallos)	Las máquinas virtuales pueden conmutar por error a otro servidor en un clúster y el sistema operativo de la máquina virtual se reinicia en el nuevo servidor.	Si falla un nodo de clúster, el orquestador vuelve a crear rápidamente cualquier contenedor que se ejecute en él en otro nodo de clúster.
Networking (Redes)	Utiliza adaptadores de red virtual.	Utiliza una vista aislada de un adaptador de red virtual, lo que proporciona un poco menos de virtualización.

Referencia: https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm

Contenido

Contenedores

1. Objetivo de la sesión
2. Contenedores vs Máquinas virtuales
3. **Arquitectura de Solución**
4. Patrón de Arquitectura Monolítica
5. Patrón de Arquitectura de Microservicios
6. Arquitectura Monolítica vs Microservicios
7. Cierre

Arquitectura de Solución
Arquitecto de Soluciones

- Profesional con la experiencia necesaria para saber “de todo un poco” pero teniendo una visión end-to-end (extremo a extremo) de la solución.
- Es el encargado de escuchar las necesidades del cliente y diseñar una solución, mapeando los requerimientos funcionales hacia tecnologías.
- El arquitecto es el puente entre el cliente con sus necesidades, y los ingenieros que implementarán la solución.
- La arquitectura de la solución en sí misma, abarca la arquitectura de negocios, sistemas, información, seguridad, aplicaciones y tecnología.
- El arquitecto debe tener habilidades de liderazgo sólidas y también habilidades comunicativas. Además, claro, debe tener conocimientos técnicos y comerciales.

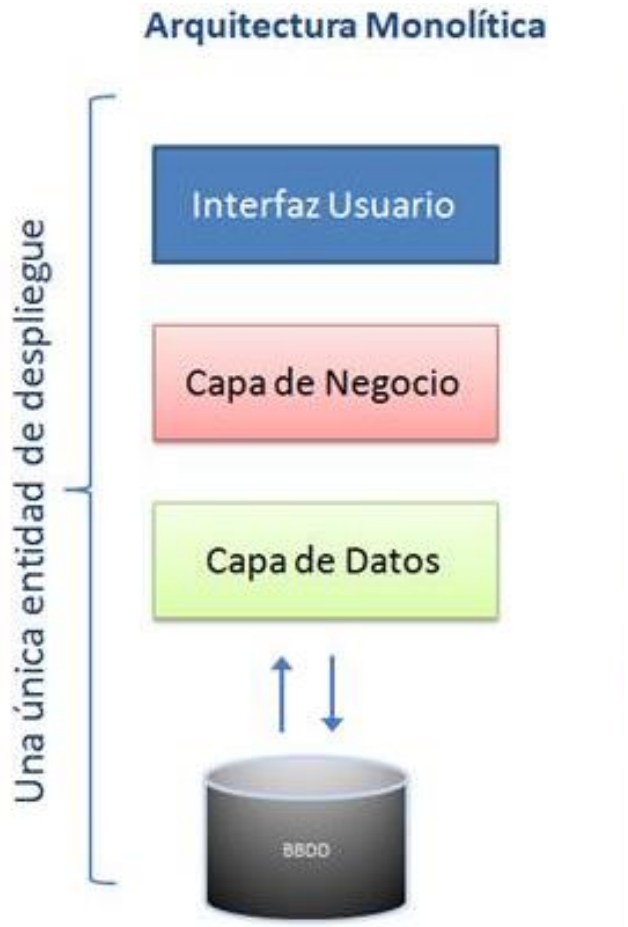
Fuentes:
- https://www.cloudmasters.es/que-es-un-arquitecto-de-soluciones-sabias-que/
- https://www.rackspace.com/es-pe/blog/what-solution-architect
- https://sg.com.mx/revista/46/la-importancia-la-arquitectura-soluciones

Contenido

Contenedores

1. Objetivo de la sesión
2. Contenedores vs Máquinas virtuales
3. Arquitectura de Solución
4. **Patrón de Arquitectura Monolítica**
5. Patrón de Arquitectura de Microservicios
6. Arquitectura Monolítica vs Microservicios
7. Cierre

Patrones de Arquitectura
Patrón de Arquitectura Monolítica

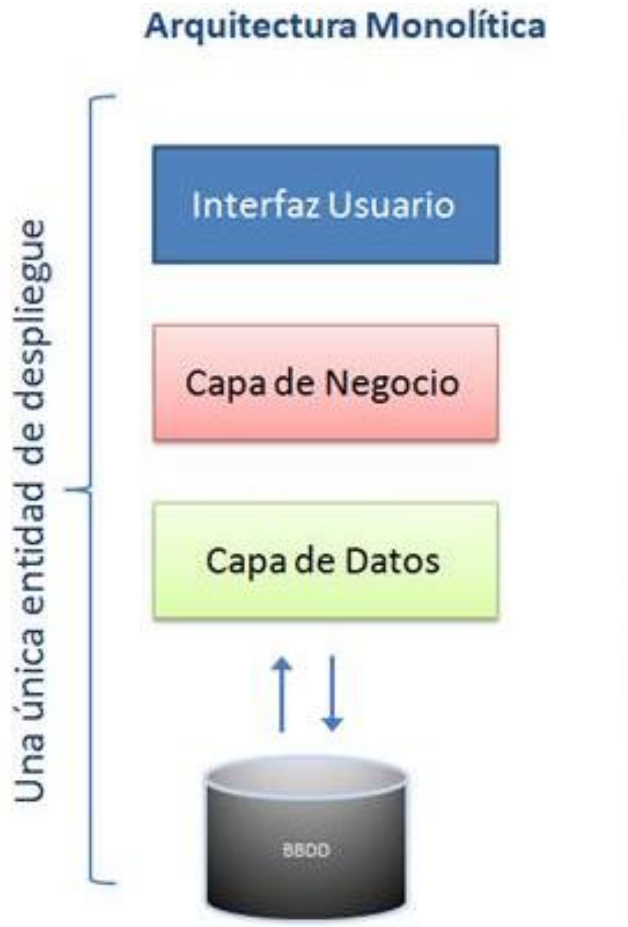


Características:

- Son aplicaciones **autosuficientes** (no requieren de nada para funcionar).
- Realizan de **punta a punta** todas las operaciones para terminar una tarea.
- Son por lo general **aplicaciones grandes**, aunque no es un requisito.
- Son por lo general **silos de datos privados**, es decir, cada instalación administra su propia base de datos.
- Todo el sistema corre sobre **una sola plataforma**.

Referencias: https://unpocodejava.com/2015/11/13/microservicios-buenas-practicas-y-stacks-tecnologicos/
https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/monolitico

Patrones de Arquitectura
Patrón de Arquitectura Monolítica

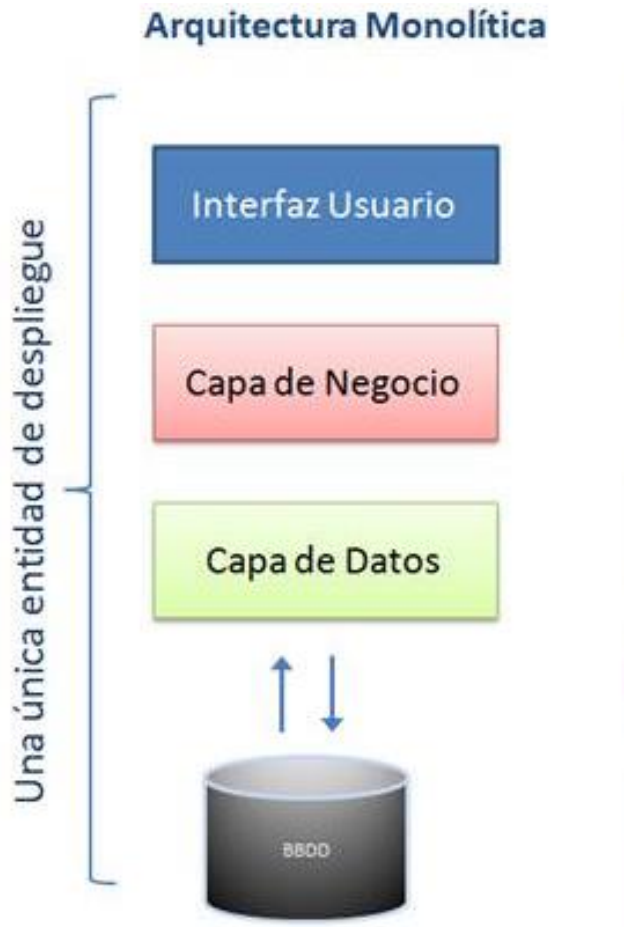


Ventajas:

- **Fácil de desarrollar**: Debido a que solo existe un componente, es muy fácil para un equipo pequeño de desarrollo iniciar un nuevo proyecto y ponerlo en producción rápidamente.
- **Fácil de escalar**: Solo es necesario instalar la aplicación en varios servidores y ponerlo detrás de un balanceador de carga.
- **Pocos puntos de fallo**: El hecho de no depender de nadie más, mitiga gran parte de los errores de comunicación, red, integraciones, etc.
- **Autónomo**: Las aplicaciones Monolíticas se caracterizan por ser totalmente autónomas (auto suficientes), lo que les permite funcionar independientemente del resto de aplicaciones.
- **Performance**: Son más rápidas debido que todo el procesamiento lo realizan localmente y no requieren consumir procesos distribuidos para completar una tarea.
- **Fácil de probar**: Debido a que es una sola unidad de código, toda la funcionalidad está disponible desde el inicio de la aplicación, por lo que es posible realizar todas las pruebas necesarias sin depender de nada más.

Referencias: https://unpocodejava.com/2015/11/13/microservicios-buenas-practicas-y-stacks-tecnologicos/
https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/monolitico

Patrones de Arquitectura
Patrón de Arquitectura Monolítica



Desventajas:

- **Anclado a un Stack tecnológico**: Debido a que todo el software es una sola pieza, implica que utilicemos el mismo Stack tecnológico para absolutamente todo, lo que impide que aprovechemos todas las tecnologías disponibles.
- **Escalado Monolítico**: Escalar una aplicación Monolítica implica escalar absolutamente toda la aplicación, gastando recursos para funcionalidad que quizás no necesita ser escalada (en el estilo de Microservicios analizaremos como solucionar esto).
- **El tamaño sí importa**: Las aplicaciones Monolíticas son fácil de operar con equipo pequeños, pero a medida que la aplicación crece y con ello el equipo de desarrollo, se vuelve cada vez más complicado dividir el trabajo sin afectar funcionalidad que otro miembro del equipo también está moviendo.
- **Versión tras versión**: Cualquier mínimo cambio en la aplicación implicará realizar una compilación de todo el artefacto y con ello una nueva versión que tendrá que ser administrada.
- **Si falla, falla todo**: A menos que tengamos alta disponibilidad, si la aplicación Monolítica falla, falla todo el sistema, quedando totalmente inoperable.
- **Es fácil perder el rumbo**: Por la naturaleza de tener todo en un mismo módulo es fácil caer en malas prácticas de programación, separación de responsabilidades y organización de las clases del código.
- **Puede ser abrumador**: En proyectos muy grandes, puede ser abrumador para un nuevo programador hacer un cambio en el sistema.

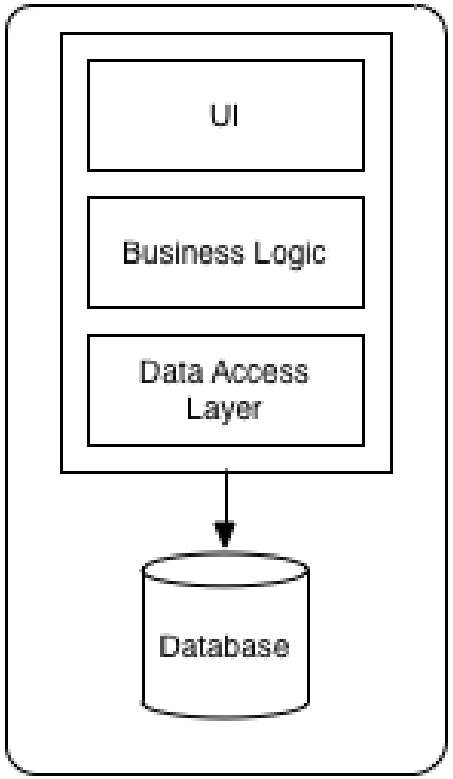
Referencias: https://unpocodejava.com/2015/11/13/microservicios-buenas-practicas-y-stacks-tecnologicos/
https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/monolitico

Contenido

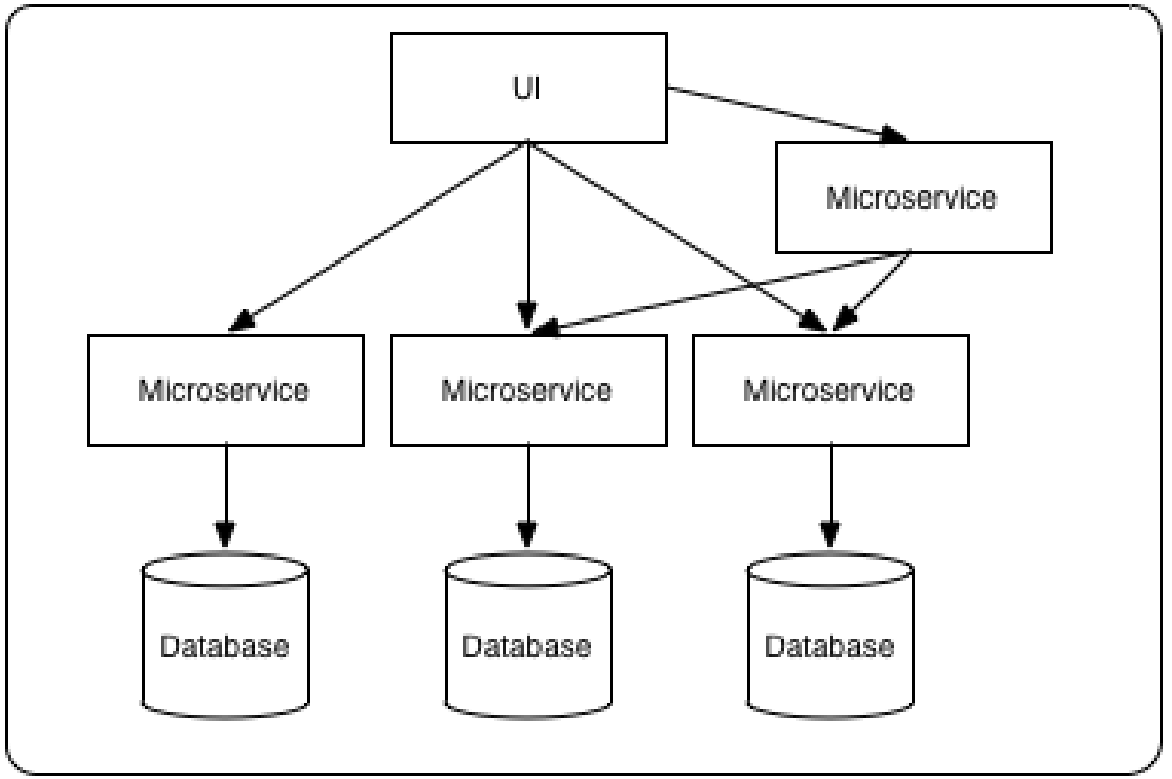
Contenedores

1. Objetivo de la sesión
2. Contenedores vs Máquinas virtuales
3. Arquitectura de Solución
4. Patrón de Arquitectura Monolítica
5. **Patrón de Arquitectura de Microservicios**
6. Arquitectura Monolítica vs Microservicios
7. Cierre

Patrones de Arquitectura
Patrón de Arquitectura de Microservicios



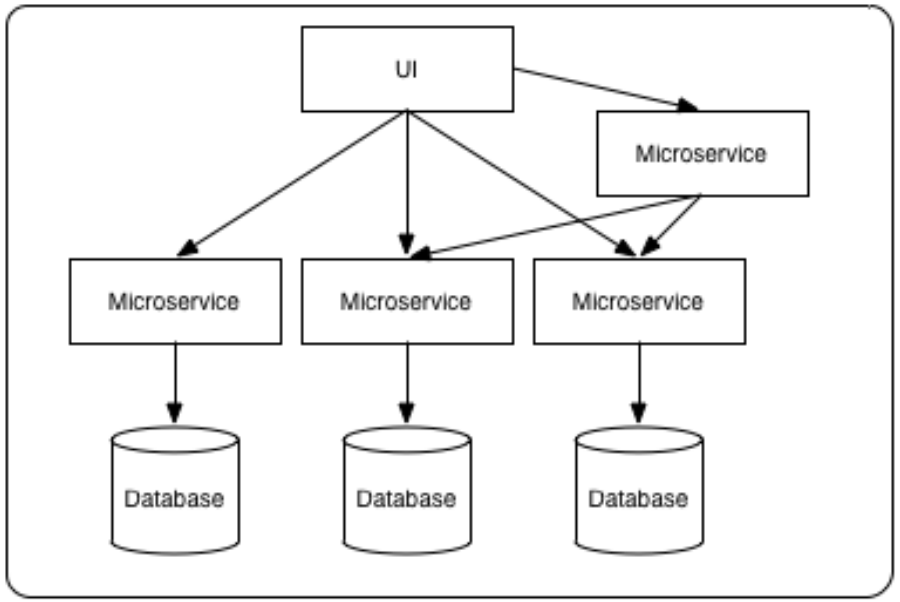
Monolithic Architecture



Microservices Architecture

Referencias: https://neuronet.cl/consultoria/arquitectura-de-microservicios/

Patrones de Arquitectura
Patrón de Arquitectura de Microservicios



Microservices Architecture

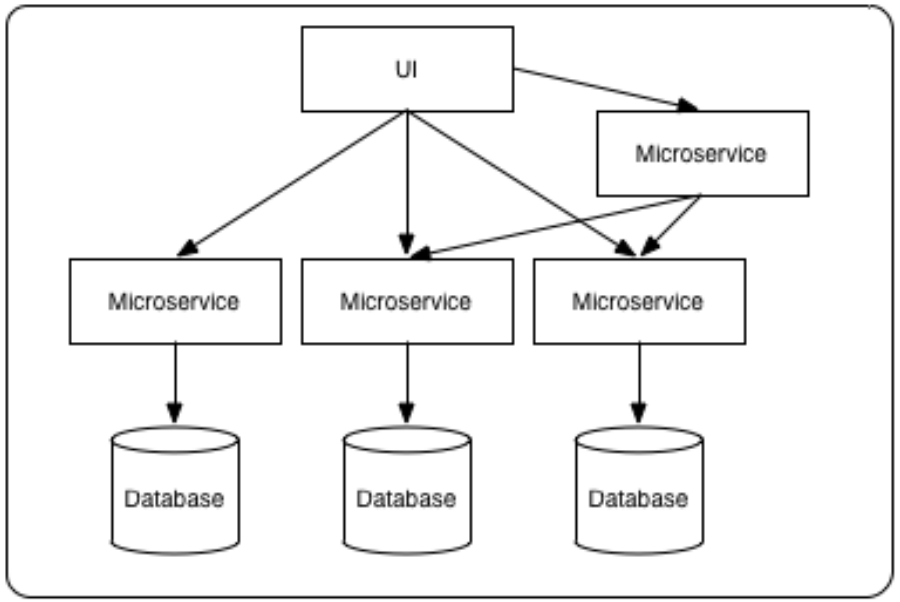
¿Qué es una arquitectura de microservicios?

La arquitectura de microservicios es un método de desarrollo de aplicaciones software que funciona como un **conjunto de pequeños servicios** que se ejecutan de manera independiente y autónoma, proporcionando una funcionalidad de negocio completa. En ella, **cada microservicio es un código que puede estar en un lenguaje de programación diferente, y que desempeña una función específica**. Los microservicios se comunican entre sí a través de APIs, y **cuentan con sistemas de almacenamiento propios**, lo que evita la sobrecarga y caída de la aplicación.

Los microservicios han creado infraestructuras IT más adaptables y flexibles. Porque si se quiere modificar solamente un servicio, no es necesario alterar el resto de la infraestructura. **Cada uno de los servicios se puede desplegar y modificar sin que ello afecte a otros servicios** o aspectos funcionales de la aplicación.

Referencias: https://neuronet.cl/consultoria/arquitectura-de-microservicios/
https://decidesoluciones.es/arquitectura-de-microservicios/

Patrones de Arquitectura
Patrón de Arquitectura de Microservicios



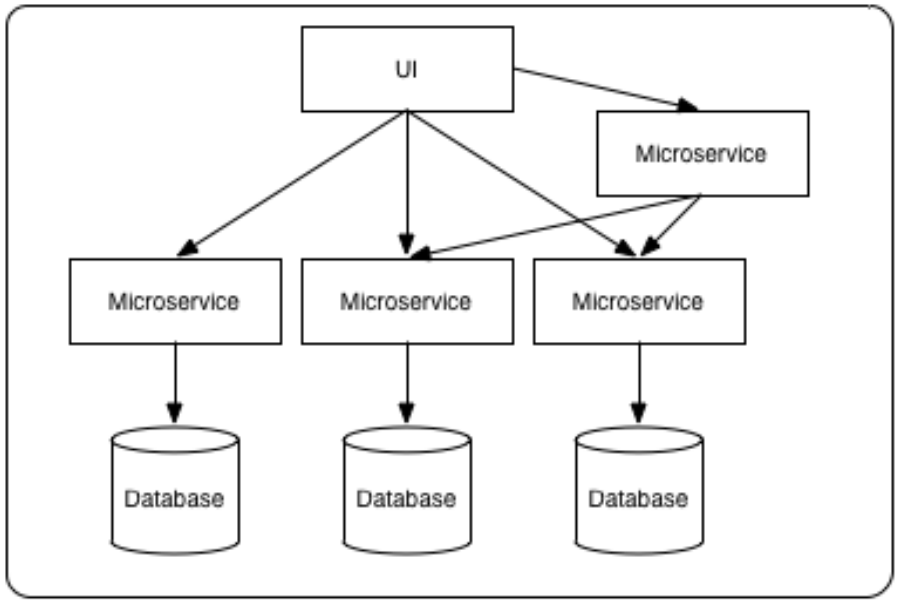
Microservices Architecture

Ventajas:

- **Modularidad**: al tratarse de servicios autónomos, se pueden desarrollar y desplegar de forma independiente. Además, un error en un servicio no debería afectar la capacidad de otros servicios para seguir trabajando según lo previsto.
- **Escalabilidad**: como es una aplicación modular, se puede escalar horizontalmente cada parte según sea necesario, aumentando el escalado de los módulos que tengan un procesamiento más intensivo.
- **Versatilidad**: se pueden usar diferentes tecnologías y lenguajes de programación. Lo que permite adaptar cada funcionalidad a la tecnología más adecuada y rentable.
- **Rapidez de actuación**: el reducido tamaño de los microservicios permite un desarrollo menos costoso, así como el uso de "contenedores de software" permite que el despliegue de la aplicación se pueda llevar a cabo rápidamente.
- **Mantenimiento simple y barato**: al poder hacerse mejoras de un solo módulo y no tener que intervenir en toda la estructura, el mantenimiento es más sencillo y barato que en otras arquitecturas.
- **Agilidad**: se pueden utilizar funcionalidades típicas (autenticación, trazabilidad, etc.) que ya han sido desarrolladas por terceros, no hace falta que el desarrollador las cree de nuevo.

Referencias: https://neuronet.cl/consultoria/arquitectura-de-microservicios/
https://decidesoluciones.es/arquitectura-de-microservicios/

Patrones de Arquitectura
Patrón de Arquitectura de Microservicios



Microservices Architecture

Desventajas:

- **Alto consumo de memoria**: al tener cada microservicio sus propios recursos y bases de datos, consumen más memoria y CPU.
- **Inversión de tiempo inicial**: al crear la arquitectura, se necesita más tiempo para poder fragmentar los distintos microservicios e implementar la comunicación entre ellos.
- **Complejidad en la gestión**: si contamos con un gran número de microservicios, será más complicado controlar la gestión e integración de los mismos. Es necesario disponer de una centralización de trazas y herramientas avanzadas de procesamiento de información que permitan tener una visión general de todos los microservicios y orquesten el sistema.
- **Perfil de desarrollador**: los microservicios requieren desarrolladores experimentados con un nivel muy alto de experiencia y un control exhaustivo de las versiones. Además de conocimiento sobre solución de problemas como latencia en la red o balanceo de cargas.
- **No uniformidad**: aunque disponer de un equipo tecnológico diferente para cada uno de los servicios tiene sus ventajas, si no se gestiona correctamente, conducirá a un diseño y arquitectura de aplicación poco uniforme.
- **Dificultad en la realización de pruebas**: debido a que los componentes de la aplicación están distribuidos, las pruebas y test globales son más complicados de realizar.
- **Coste de implantación alto**: una arquitectura de microservicios puede suponer un alto coste de implantación debido a costes de infraestructura y pruebas distribuidas.

Referencias: https://neuronet.cl/consultoria/arquitectura-de-microservicios/
https://decidesoluciones.es/arquitectura-de-microservicios/

Contenido

Contenedores

1. Objetivo de la sesión
2. Contenedores vs Máquinas virtuales
3. Arquitectura de Solución
4. Patrón de Arquitectura Monolítica
5. Patrón de Arquitectura de Microservicios
6. **Arquitectura Monolítica vs Microservicios**
7. Cierre

Patrones de Arquitectura
Arquitectura Monolítica vs Microservicios

	Monolithic	Microservice
Architecture	Built as a single logical executable (typically the server-side part of a three tier client-server-database architecture)	Built as a suite of small services, each running separately and communicating with lightweight mechanisms
Modularity	Based on language features	Based on business capabilities
Agility	Changes to the system involve building and deploying a new version of the entire application	Changes can be applied to each service independently
Scaling	Entire application scaled horizontally behind a load-balancer	Each service scaled independently when needed
Implementation	Typically written in one language	Each service implemented in the language that best fits the need
Maintainability	Large code base intimidating to new developers	Smaller code base easier to manage
Transaction	ACID	BASE

ACID: 4 key properties that define a transaction: **A**tomicity, **C**onsistency, **I**solation, and **D**urability
BASE: **B**asically **A**vailable, **S**oft State, **E**ventually Consistent

(SQL: MySQL, PostgreSQL, Oracle)
(NoSQL: MongoDB, DynamoDB)

Referencias: https://medium.com/@goodrebels/microservicios-ventajas-y-contras-de-la-arquitectura-descentralizada-a3b7fc814422
https://phoenixnap.com/kb/acid-vs-base

Contenido

Contenedores

1. Objetivo de la sesión
2. Contenedores vs Máquinas virtuales
3. Arquitectura de Solución
4. Patrón de Arquitectura Monolítica
5. Patrón de Arquitectura de Microservicios
6. Arquitectura Monolítica vs Microservicios
7. **Cierre**

Cierre
Explique con sus propias palabras

- Qué diferencias hay entre Contenedores vs Máquinas virtuales ?
- Qué es la Arquitectura de Solución ?
- Describa el Patrón de Arquitectura Monolítica
- Describa el Patrón de Arquitectura de Microservicios
- Compare Arquitectura Monolítica vs Microservicios

Gracias

Docente: Geraldo Colchado