Indice

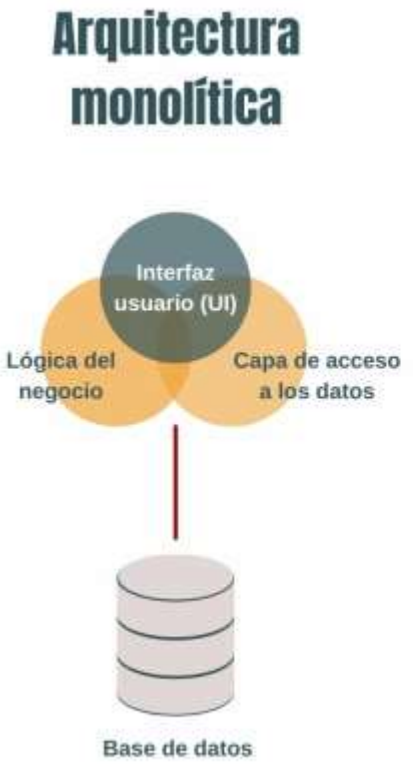
Arquitecturas monolíticas vs arquitectura de microservicios

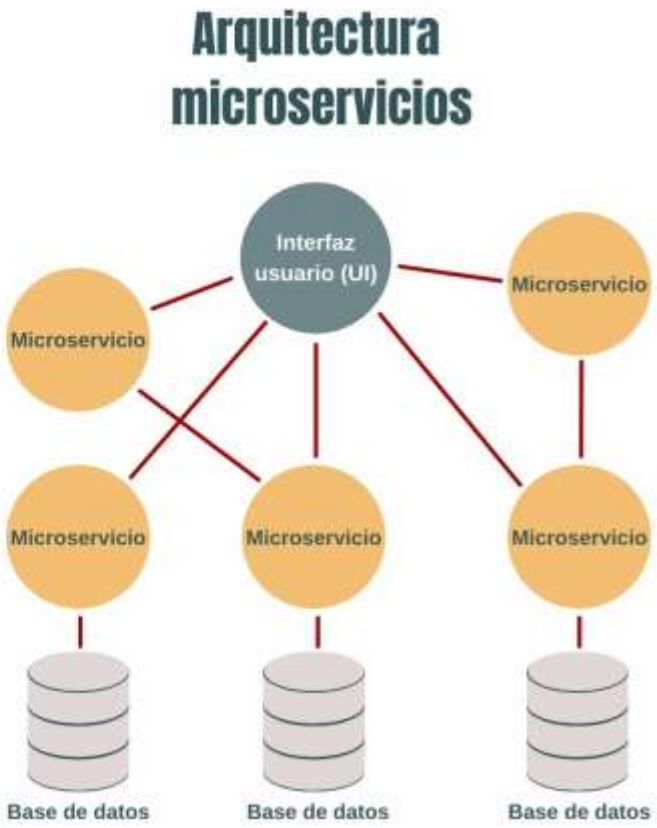
- El desarrollo de software empezó utilizando una **Arquitectura monolítica** que agrupaba todas sus funciones y servicios dentro de una base única y centralizada de código.
- Este tipo de Arquitectura ha ido quedando desfasada, sobre todo al crecer los proyectos en complejidad, número de desarrolladores, usuarios y cargas de trabajo.
- Por ello en la actualidad se tiende a utilizar una estructura basada en microservicios, que es escalable y facilita el trabajo colaborativo de los desarrolladores.

Arquitecturas monolíticas vs arquitectura de microservicios

➢La tendencia actual en el desarrollo de software adopta la cultura del uso de contenedores, haciendo necesario **construir una aplicación de forma distribuida**, permitiendo que los **distintos componentes puedan ser desplegados de forma independiente**.

Arquitecturas monolíticas vs arquitectura de microservicios





Arquitectura monolítica

Arquitectura monolítica

- El estilo arquitectónico monolítico consiste en crear una aplicación autosuficiente que contenga absolutamente toda la funcionalidad necesaria para realizar la tarea para la cual fue diseñada, **sin contar con dependencias externas que complementen su funcionalidad.**

- Los primeros programas informáticos utilizaban una arquitectura monolítica, agrupando todo lo relacionado con el sistema dentro del mismo proyecto. **Este tipo de arquitectura se caracteriza por:**
  - Los programas son fáciles de desarrollar.
  - El despliegue y la ejecución del software son muy sencillos.
  - El costo de desarrollo es bajo en comparación con otras arquitecturas.

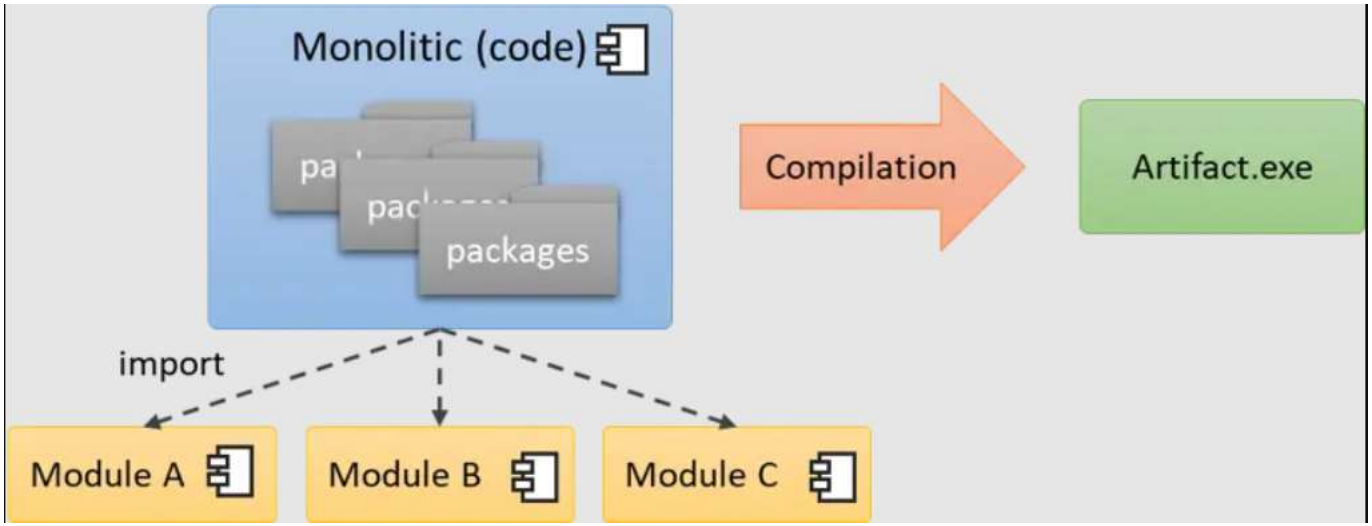
Arquitectura monolítica

Arquitectura monolítica

- Los problemas de este tipo de arquitectura, como la escalabilidad o la dificultad para los desarrolladores (necesitan entender todo el código de la aplicación) han hecho que **este tipo de desarrollo de software deje de ser utilizado en muchos proyectos** (aunque su sencillez y bajo costo hace que siga siendo interesante para ciertos proyectos con bajos requerimientos).

Arquitectura monolítica

Arquitectura monolítica



Arquitectura de microservicios

La Arquitectura de Microservicios o **MSA** (Micro Services Architecture) es un enfoque para el desarrollo de software que consiste en **construir una aplicación como un conjunto de pequeños servicios.**

Estos servicios se ejecutan en su propio proceso y se comunican con mecanismos ligeros (normalmente una API de recursos HTTP).

Arquitectura de microservicios

- Cada servicio se encarga de implementar una funcionalidad completa del negocio.
- Cada servicio:
  - Es desplegado de forma independiente.
  - Puede estar programado en distintos lenguajes.
  - Usa diferentes tecnologías de almacenamiento de datos.
- Se suele considerar la **MSA** como una forma específica de realizar una Arquitectura Orientada a Servicios.

Arquitectura de microservicios

Características

1- Los componentes son servicios: la principal manera de crear componentes (unidad de software independientemente reemplazable y actualizable) es mediante la inserción de un botón que automáticamente, gestione la descomposición en servicios en lugar de bibliotecas.

2- Organizada en torno a las funcionalidades del negocio: Cada servicio implementa una determinada funcionalidad del negocio que agrupa desde la interfaz de usuario, la persistencia en el almacenamiento y cualquiera de las colaboraciones externas.

3- Productos, no proyectos. En esta arquitectura normalmente se sigue la idea de que un equipo debe estar a cargo de un componente (servicio) durante todo el ciclo de vida del mismo, desde la etapa de diseño y construcción, la fase de producción y hasta la de mantenimiento.

Arquitectura de microservicios

Características

4- Gobierno descentralizado: permite usar tecnologías que se adapten mejor a cada funcionalidad. Con el sistema con múltiples servicios colaborativos, se puede decidir utilizar diferentes lenguajes de programación y tecnologías dentro de cada servicio.

5- Gestión de datos descentralizada. Los microservicios prefieren dejar que **cada servicio gestione su propia base de datos**. Por ejemplo se puede tener **Redis** para sesiones de usuarios (base de datos en memoria), **MySQL** (relacional) para los datos de pago, **MongoDB** (orientada a documentos) para el catálogo de productos**, Neo4j** (orientada a grafos) para las recomendaciones y **Apache Cassandra** (orientado a clave-valor) para el análisis de logs y analíticas.

6- Diseño tolerante a fallos. Las aplicaciones necesitan ser diseñadas de modo que puedan tolerar las fallas de los distintos servicios.

7- Diseño evolutivo. Cuando se divide el sistema en servicios hay que tener en cuenta que cada uno tiene que poder ser reemplazado o actualizado de forma independiente. Es decir, tiene que permitir una fácil evolución.

Arquitectura de microservicios

Integración de Servicios

Cuando tratamos con problemas complejos, es necesario determinar la forma de manejar y coordinar los **procesos de negocio** que involucran a varios servicios.
Hay dos formas de **integrar** los procesos de negocio :

1- Orquestación: lo que hacemos es tener un software que guíe y dirija el proceso, de forma similar a como lo hace un director de orquesta.

2- Coreografía: lo que hacemos es dejar que cada parte del sistema realice su trabajo y se los deja trabajar en los detalles. Es más adaptado a la MSA que cada servicio sepa cómo actuar en cada momento, e interactúe con otros, en lugar de tener a alguien que los coordine.

"Por eso para integrar se suele preferir tener coreografía"

Arquitectura de microservicios

Diferencias de MSA con SOA

- La MSA se basa en componentes pequeños y especializados, mientras que SOA se centra en servicios empresariales y la integración de sistemas más grandes.

- Es importante tener en cuenta las diferencias y no confundir los conceptos en el ámbito de la Arquitectura de software.

Arquitectura de microservicios

Diferencias de MSA con SOA

Características \ Arquitectura	SOA	Microservicios
Definición	Un enfoque arquitectónico que se basa en servicios empresariales para construir aplicaciones más grandes.	Un enfoque arquitectónico que divide una aplicación en componentes pequeños, autónomos y bien definidos llamados microservicios.
Tamaño del componente	Los servicios son más grandes y generalmente representan funcionalidades empresariales completas.	Los microservicios son pequeños y se centran en una funcionalidad específica del negocio.
Acoplamiento	Existe un acoplamiento más fuerte entre los servicios, lo que significa que los cambios en un servicio pueden afectar a otros servicios.	Hay un acoplamiento débil entre los microservicios, lo que permite realizar cambios en un microservicio sin afectar a los demás.

Arquitectura de microservicios

Diferencias de MSA con SOA

Arquitectura / Características	SOA	Microservicios
Escalabilidad	La escalabilidad se realiza a través de la replicación de servicios completos, lo que puede resultar costoso.	La escalabilidad se logra mediante la replicación de microservicios individuales, lo que permite un uso más eficiente de los recursos.
Mantenimiento	El mantenimiento de servicios puede ser más complejo debido al acoplamiento y la dependencia entre ellos.	El mantenimiento de microservicios es más sencillo, ya que se pueden actualizar e implementar de forma independiente.
Tecnología	SOA no está vinculado a una tecnología o protocolo específico.	Los microservicios generalmente se basan en tecnologías como contenedores, Docker y protocolos como HTTP/REST.
Enfoque organizacional	SOA se centra en la reutilización de servicios a nivel empresarial y puede requerir una coordinación y planificación centralizada.	Los microservicios promueven la descentralización y la autonomía de los equipos de desarrollo, lo que permite una mayor agilidad.

Arquitectura de microservicios

Beneficios de una arquitectura basada en microservicios

Trabajar con una arquitectura basada en microservicios **a la hora de desarrollar una aplicación** aporta una serie de beneficios como:

1. El **despliegue independiente** de cada uno de los componentes de la aplicación.
2. **Facilita las actualizaciones** al no tener que parar todo el proyecto, solo el módulo a actualizar.
3. Permite **desarrollar utilizando múltiples lenguajes de programación**, lo que aporta ventajas como velocidad, rendimiento, reducción de costos, elección de distintas herramientas de desarrollo, etc.
4. **Escalabilidad por módulos**, permitiendo incrementar las capacidades de cómputo del módulo que mayor carga soporte o mayor demanda tenga (al poderse instalar en distintos servidores los módulos de la aplicación).
5. Permite realizar una **entrega continu**a de software a los clientes de una forma mucho más ágil.
6. Reduce los errores de programación y **facilita los procesos de pruebas y test**.

Arquitectura de microservicios

DevOps+Contenedores para microservicios

- La MSA es un reto importante a la hora de ser implementada. **Las operaciones son más complejas** al tener un gran número de módulos y la creación y desarrollo de la infraestructura consume más tiempo y recursos.
- Para solucionar este problema, la **cultura DevOps(*) proporciona agilidad en el proceso de desarrollo**, con prácticas de desarrollo ágil y enfoque en el producto.
- El **aprovisionamiento rápido de recursos** se facilita con el uso de contenedores como Docker y orquestadores como Kubernetes o Swarm si se tienen que gestionar un número elevado de contenedores. Por ejemplo, gracias al uso de contenedores y orquestadores, los servidores web tradicionales se han cambiado por contenedores virtuales más pequeños y adaptables.

* **DevOps (Development Operations)** es un cambio de la cultura organizativa que **hace hincapié en el aprendizaje y la mejora continuos**, especialmente a través de la **autonomía del equipo**, el **feedback rápido**, el **alto grado de empatía y confianza** y la **colaboración entre equipos.**

Arquitectura de microservicios

Qué son los contenedores ?

- Los contenedores son unos **mecanismos de empaquetado de aplicaciones que permiten ejecutarlas en distintos entornos**.
- Los contenedores **separan las aplicaciones del entorno donde se ejecutan** permitiendo que se implementen de forma sencilla en distintos tipos de ambientes como servidores, nubes públicas, nubes privadas u ordenadores personales de los propios desarrolladores.
- Un contenedor es un **entorno de pruebas seguro** donde los desarrolladores pueden trabajar en una aplicación en su entorno original, sin tener que preocuparse por la portabilidad de la misma.
- Los contenedores se inician a partir de **imágenes** que incluyen la aplicación, así como los componentes requeridos para que se ejecuten (archivos de configuración, librerías, etc.).

Arquitectura de microservicios

Por qué usar contenedores

- **Ligereza.** Los contenedores no virtualizan el sistema operativo completo, ejecutándose directamente en el kernel del sistema operativo, lo que hace que sean mucho más livianos y rápidos a la hora de arrancar.
- **Uniformidad**. Los contenedores aportan a los desarrolladores un entorno completo y uniforme que contiene todas las dependencias de software que la aplicación requiere (bibliotecas, versiones de lenguajes de programación,etc).
- **Portabilidad.** El uso de contenedores brinda una **portabilidad y autosuficiencia** que permiten ejecutar la aplicación en cualquier entorno como distintos sistemas operativos (Windows, Linux y macOS), equipos físicos, máquinas virtuales, o en la nube.
- **Aislamiento.** Cada contenedor aísla la aplicación del sistema operativo y de otras aplicaciones. Este aislamiento permite que los desarrolladores puedan trabajar y testear aplicaciones sin que interfieran con otras.

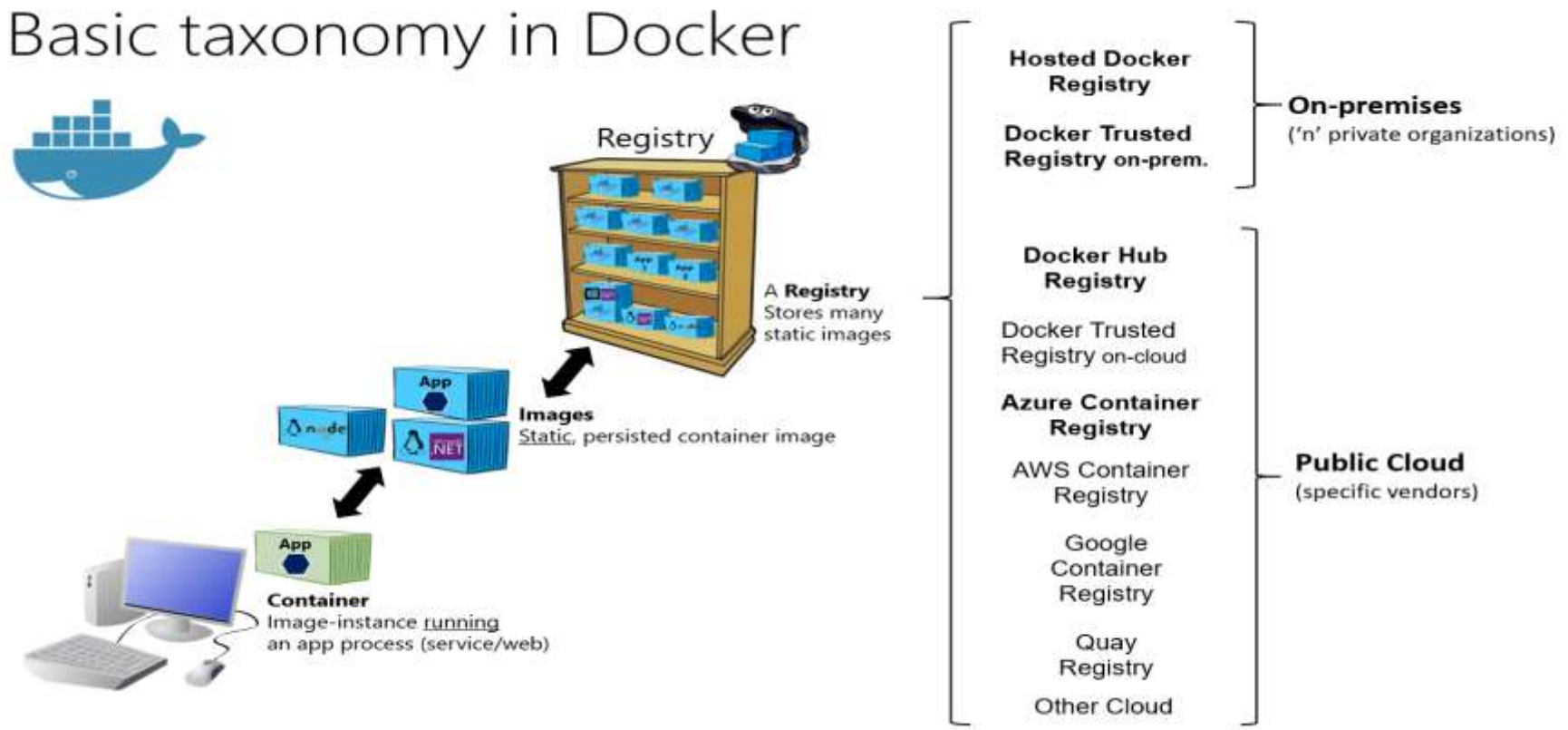
Arquitectura de microservicios

Qué son los contenedores Docker ?

- **Docker** es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de **virtualización de aplicaciones en múltiples sistemas operativos.**
- Con **Docker** es posible encapsular todo el entorno de trabajo **permitiendo trabajar en los distintos entornos de desarrollo** (local, producción, test, etc.) con la misma configuración.
- La tecnología de **Docker**, permite **utilizar el kernel de Linux** y sus funciones para separar los procesos (como Cgroups y Namespaces) y permitir que puedan **ejecutarse de forma independiente**. De esta forma **Docker** se convierte en una excelente herramienta para la gestión de contenedores individuales.

Arquitectura de microservicios

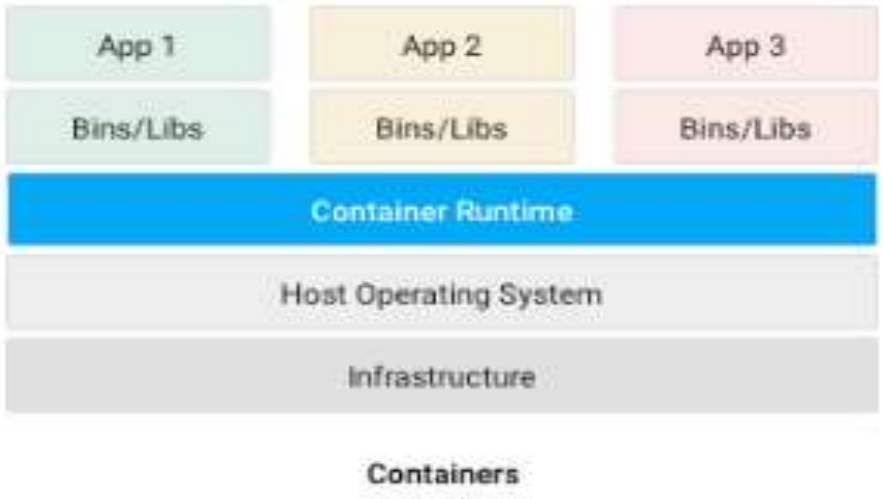
Qué son los contenedores Docker ?



Arquitectura de microservicios

- Qué son los contenedores Docker ?

- Cuando el número de contenedores con el que se trabaja empieza a ser elevado, su gestión comienza a complicarse, siendo necesario recurrir a una **plataforma de gestión de contenedores**, como pueden ser Docker Swarm o Kubernetes.



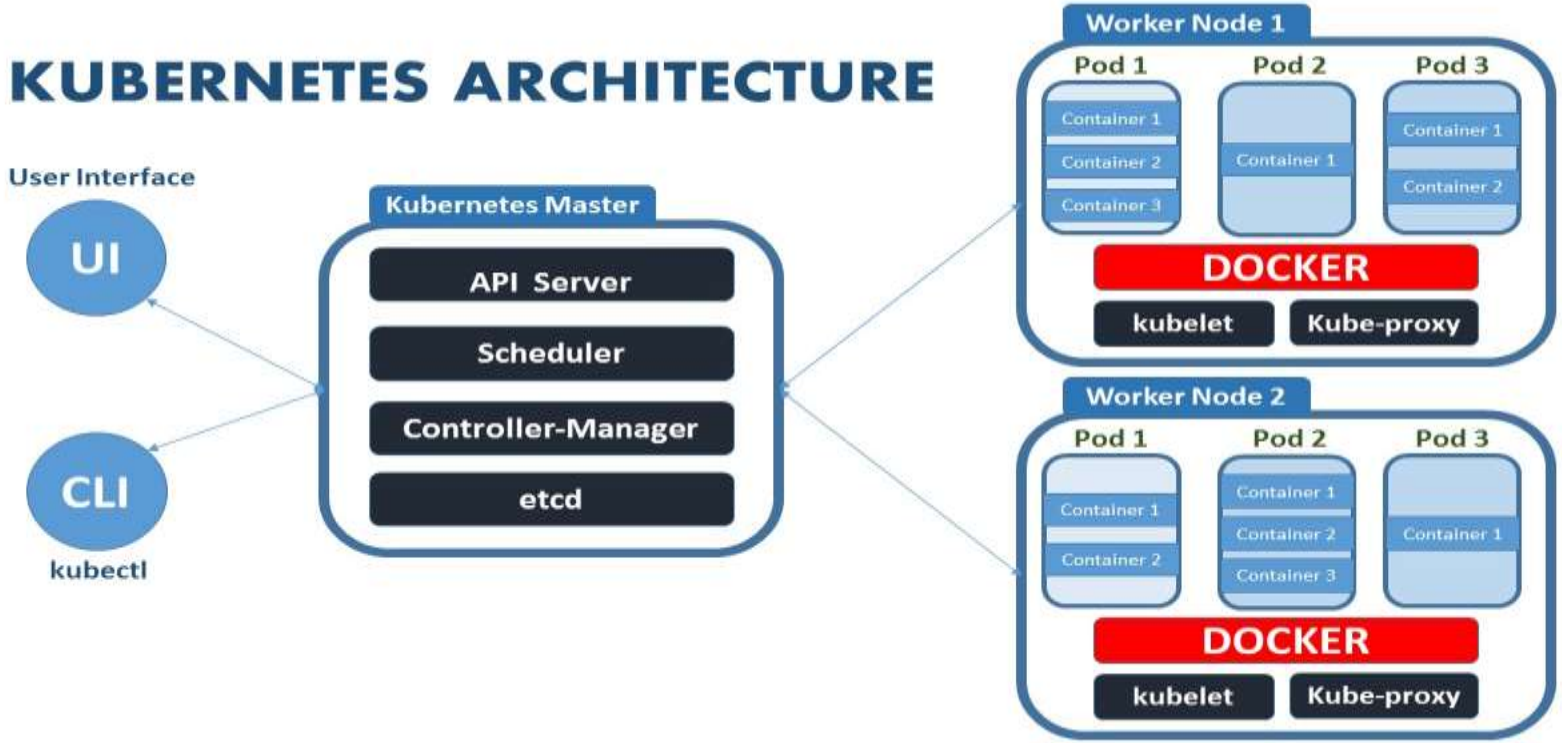
Arquitectura de microservicios

Qué es Kubernetes ?

- **Kubernetes** es una plataforma de sistema distribuido de código libre para la automatización del despliegue, ajuste de escala y manejo de aplicaciones en contenedores que fue originalmente diseñado por Google y donado a la Cloud Native Computing Foundation
- **Kubernetes** automatiza las tareas operativas de la administración de contenedores e incluye comandos integrados para implementar aplicaciones, actualizarlas y escalarlas.
- **Docker y Kubernetes** son dos tecnologías diferentes con casos de uso distintos. **Docker Desktop** se utiliza para ejecutar, editar y administrar el desarrollo de contenedores. **Kubernetes** se utiliza para ejecutar aplicaciones de producción a gran escala.

Arquitectura de microservicios

Arquitectura de Kubernetes



Arquitectura de microservicios

Arquitectura de Kubernetes → Componentes

Componentes de Kubernetes:

- **Web UI (Dashboard) - Interfaz de usuario web (panel de control)**

  Dashboard es una interfaz de usuario de Kubernetes basada en web.

- **Kubectl**

  Es una herramienta de configuración de línea de comandos (CLI) para Kubernetes que se utiliza para interactuar con el nodo maestro de Kubernetes.

- **Kubernetes Master**

  Es el nodo principal, responsable de gestionar todos los cluster´s de Kubernetes. Maneja la orquestación de los Worker Nodes. Tiene componentes que se encargan de la comunicación, almacenamiento, la programación y los controladores.

1- Servidor API: el servidor API de Kube interactúa con la API. Es una interfaz de control de Kubernetes.

2- Scheduler: el programador observa los pods y los asigna para que se ejecuten en hosts específicos.

3- Kube-Controller-Manager: el administrador de controladores ejecuta los controladores en segundo plano, lo que ejecuta diferentes tareas en el clúster de Kubernetes.

4- Etcd: es un almacén de valores clave de distribución simple. Kubernetes utiliza etcd como base de datos para almacenar todos los datos del clúster, información de programación de trabajos, pods, información de estado, etc.

Arquitectura de microservicios

Arquitectura de Kubernetes → Componentes

Componentes de Kubernetes:

- **Worker Nodes**

Los Worker Nodes son los nodos donde la aplicación realmente se ejecuta en el clúster de Kubernetes, también se conoce como minion. Cada uno de estos nodos trabajadores está controlado por el nodo maestro mediante el proceso kubelet.

- **Kubelet**

Kubelet es el agente que se ejecuta en cada Worker Nodes y lee los manifiestos del contenedor, lo que garantiza que los contenedores se estén ejecutando y en buen estado.

- **Kube proxy**

Kube-proxy es un proceso que nos ayuda a tener un proxy de red y un equilibrador de carga para los servicios en un solo Worker Nodes . Realiza el enrutamiento de red para paquetes tcp y udp y realiza el plegado de conexiones. Los Worker Nodes pueden exponerse a Internet a través de kubeproxy.

- **Pods**

Los pods son los objetos más pequeños y básicos que se pueden implementar en Kubernetes. Un pod representa una instancia única de un proceso en ejecución en tu clúster. Los pods contienen uno o más contenedores, como los contenedores de Docker