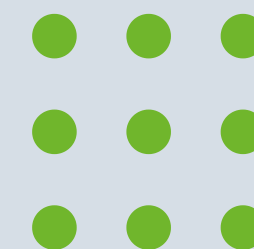


building your ideas

Tecnología OTC

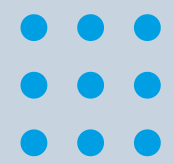




3

+

+



**Tecnología OTC**



# Objetivos

- Conocer las tecnologías que se usaron en OTC





# Agenda

- Microservicios
- Plataformas
- Framework y Lenguajes







# Microservicios



# Que son Microservicios

El término "arquitectura de microservicio" ha surgido en los últimos años para describir una forma particular de diseñar aplicaciones de software como conjuntos de servicios de implementación independiente. Si bien no existe una definición precisa de este estilo arquitectónico, existen ciertas características comunes en torno a la organización en torno a la capacidad empresarial, la implementación automatizada, la inteligencia en los puntos finales y el control descentralizado de lenguaje y datos.

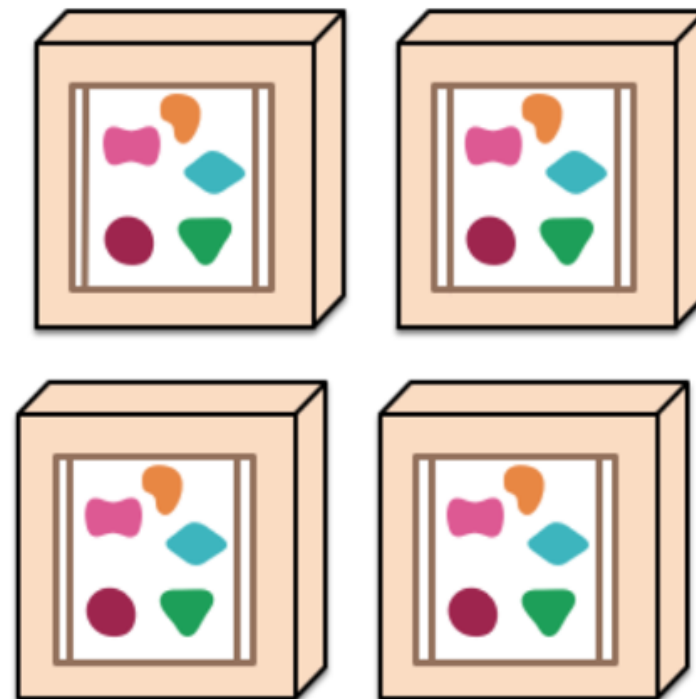


# Microservicios

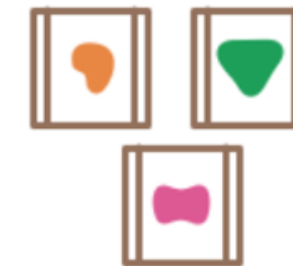
*A monolithic application puts all its functionality into a single process...*



*... and scales by replicating the monolith on multiple servers*



*A microservices architecture puts each element of functionality into a separate service...*



*... and scales by distributing these services across servers, replicating as needed.*

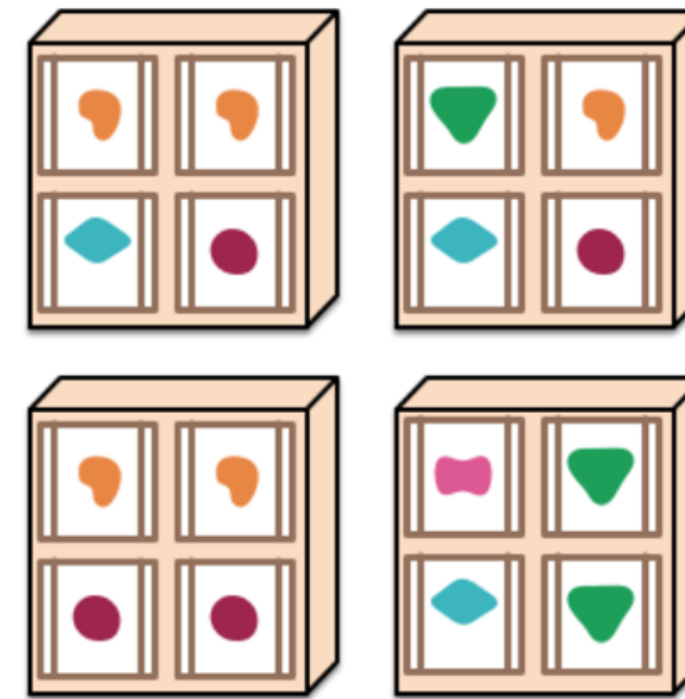



Figure 1: Monoliths and Microservices



# Características

## Componentes a través de Servicios

Las arquitecturas de microservicio utilizarán bibliotecas, pero su principal forma de componentes de su propio software es dividirse en servicios. Definimos bibliotecas como componentes que están vinculados en un programa y llamados mediante llamadas a función en memoria, mientras que los servicios son componentes fuera de proceso que se comunican con un mecanismo como una solicitud de servicio web o una llamada a procedimiento remoto.







# Características

## **Organizado en torno a las capacidades Empresariales**

“Cualquier organización que diseña un sistema (definida ampliamente) producirá un diseño cuya estructura es una copia de la estructura de comunicación de la organización.”

Melvyn Conway, 1967

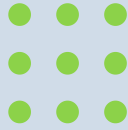


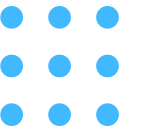


# Características

## Productos no Proyectos

La mentalidad del producto se relaciona con la vinculación con las capacidades empresariales. En lugar de considerar el software como un conjunto de funcionalidades para completarse, existe una relación continua en la que la pregunta es cómo puede el software ayudar a sus usuarios a mejorar la capacidad empresarial.

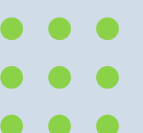




# Características

## Puntos finales inteligentes y tuberías tontas

La comunidad de microservicios favorece un enfoque alternativo: puntos finales inteligentes y tuberías tontas. Las aplicaciones creadas a partir de microservicios apuntan a estar lo más desacopladas y cohesivas posible (poseen su propia lógica de dominio y actúan más como filtros en el sentido clásico de Unix), reciben una solicitud, aplican lógica según corresponda y producen una respuesta. Estos son coreografiados usando simples protocolos RESTish en lugar de protocolos complejos como WS-Choreography o BPEL o orchestration por una herramienta central.





# Características

## Gobiernos descentralizados

Los equipos que crean microservicios también prefieren un enfoque diferente a los estándares. En lugar de utilizar un conjunto de estándares definidos escritos en algún lugar del papel, prefieren la idea de producir herramientas útiles que otros desarrolladores puedan usar para resolver problemas similares a los que enfrentan. Estas herramientas generalmente se obtienen de implementaciones y se comparten con un grupo más amplio, a veces, pero no exclusivamente, utilizando un modelo interno de código abierto. Ahora que git y github se han convertido en el sistema de control de versiones de facto, las prácticas de código abierto son cada vez más comunes en la empresa.





# Características

## Gestión de datos descentralizados

Descentralizar la responsabilidad de los datos en los microservicios tiene implicaciones para administrar las actualizaciones. El enfoque común para tratar las actualizaciones ha sido utilizar las transacciones para garantizar la coherencia al actualizar múltiples recursos. Este enfoque se usa a menudo dentro de monolitos.








# Características

## **Automatización de Infraestructura**

Otra área en la que vemos que los equipos usan una amplia automatización de la infraestructura es cuando gestionan microservicios en producción. En contraste con nuestra afirmación anterior, mientras el despliegue sea aburrido no hay mucha diferencia entre monolitos y microservicios, el paisaje operativo para cada uno puede ser notablemente diferente.





# Características

## Diseño para fallos

Una consecuencia del uso de los servicios como componentes es que las aplicaciones deben diseñarse para que puedan tolerar a fallos de los servicios. Cualquier llamada de servicio podría fallar debido a la falta de disponibilidad del proveedor, el cliente tiene que responder a esto de la forma más elegante posible. Esto es una desventaja en comparación con un diseño monolítico ya que introduce una complejidad adicional para manejarlo. La consecuencia es que los equipos de microservicio constantemente reflexionan sobre cómo las fallas del servicio afectan la experiencia del usuario. Simian Army de Netflix induce fallas en los servicios e incluso en los centros de datos durante el día de trabajo para probar tanto la capacidad de recuperación como la supervisión de la aplicación






# Características

## Diseño Evolutivo

Los profesionales de microservicio, por lo general, provienen de un fondo de diseño evolutivo y ven la descomposición del servicio como una herramienta adicional para permitir a los desarrolladores de aplicaciones controlar los cambios en su aplicación sin ralentizar el cambio. El control de cambios no significa necesariamente la reducción de cambios: con las actitudes y herramientas correctas, puede realizar cambios frecuentes, rápidos y bien controlados en el software.

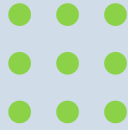






# Contenedores

Un contenedor es una instancia de tiempo de ejecución de una imagen: la imagen se convierte en memoria cuando se ejecuta. Se ejecuta completamente aislado del entorno de host de forma predeterminada, solo accediendo a los archivos de host y puertos si está configurado para hacerlo.






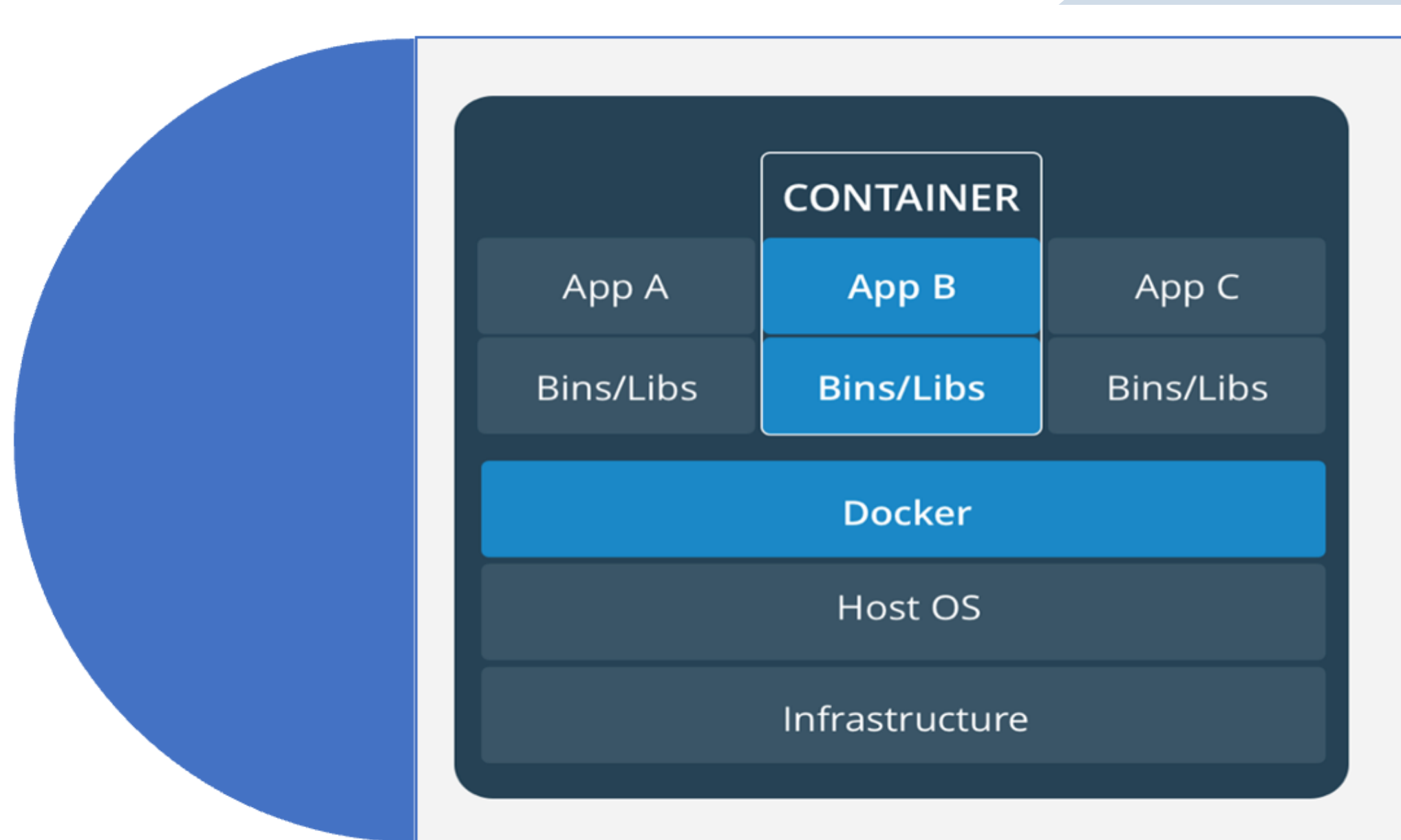


# Contenedores

Los contenedores ejecutan aplicaciones de forma nativa en el kernel de la máquina host. Tienen mejores características de rendimiento que las máquinas virtuales que solo obtienen acceso virtual a los recursos del host a través de un hipervisor. Los contenedores pueden obtener acceso nativo, cada uno ejecutándose en un proceso discreto, sin tomar más memoria que cualquier otro ejecutable.



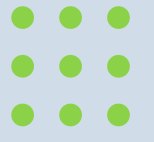
# Contenedores





# Kubernetes


Kubernetes, también conocido como K8s, es un sistema de código abierto para automatizar la implementación, el escalado y la gestión de aplicaciones en contenedores.

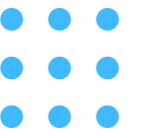




# Openshift

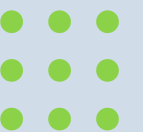
Esta plataforma se desarrolla a partir de contenedores kubernetes que los desarrolladores utilizan para desplegar apps en diferentes lenguajes de programación. Su principal objetivo es mejorar la productividad de los desarrolladores y promover la innovación.





# Docker / Kubernetes / Openshift

	OpenShift	Kubernetes	Docker
Sistema operativo	Linux, Fedora, CentOS	Cualquier sistema operativo	Cualquier sistema operativo
Seguridad	Altos estándares de seguridad	La seguridad está en manos del usuario	La seguridad está en manos del usuario
CI/CD	Parte integral del software	Posible usando Jenkins	Combina Jenkins
Facilidad de uso	Pensado para ser fácil de usar	Uso algo más complejo	fácil uso
Interfaz de usuario	Interfaz de usuario sencilla	Puede instalarse un cuadro de mando adicional	Sencilla
Escalabilidad	En principio para el nivel empresarial, pero puede escalarse	Puede usarse en proyectos de cualquier magnitud	facil escalabilidad
Plantillas	Menos intuitivas	El uso de Helm permite una gran flexibilidad	Docker hub
Networking	Open vSwitch ofrece posibilidades de networking	El networking es posible mediante plug-ins de terceros	Docker ofrece posibilidades de networking








# Redis - Cache


Redis es un almacén de estructura de datos de valores de clave en memoria rápido y de código abierto. Redis incorpora un conjunto de estructuras de datos en memoria versátiles que le permiten crear con facilidad diversas aplicaciones personalizadas. Entre los casos de uso principales de Redis se encuentran el almacenamiento en caché, la administración de sesiones, pub/sub y las clasificaciones.





# IBM MQ

IBM MQ da soporte al intercambio de información entre aplicaciones, sistemas, servicios y archivos enviando y recibiendo datos de mensajes a través de colas de mensajería. Esto simplifica la creación y el mantenimiento de las aplicaciones empresariales. IBM MQ funciona con una amplia gama de plataformas informáticas, y se puede desplegar en una gama de distintos entornos, incluidos los despliegues en local, en cloud y en cloud híbrido. IBM MQ da soporte a diversas API, incluidas la Interfaz de cola de mensajes (MQI), Servicio de mensajes Java (JMS), REST, .NET, IBM MQ Light y MQTT.





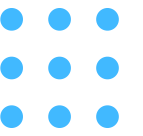


# Spring Boot

Spring Boot facilita la creación de aplicaciones autónomas de grado de producción basadas en Spring, que usted puede "simplemente ejecutar". La mayoría de las aplicaciones Spring Boot necesitan muy poca configuración de Spring.

Puede utilizar Spring Boot para crear aplicaciones Java que se pueden iniciar utilizando `java -jar` o más implementaciones war tradicionales.





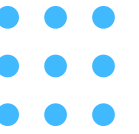
# Apache Camel

Apache Camel es un potente marco de integración de código abierto basado en conocidos patrones de integración empresarial con una poderosa integración de beans.

Camel le permite crear Patrones de Integración Empresarial para implementar reglas de enrutamiento y mediación en un Lenguaje Específico de Dominio basado en Java (o API Fluent), a través de archivos de Configuración Xml basados en Spring o Blueprint, o mediante el DSL de Scala. Esto significa que puede completar de manera inteligente las reglas de enrutamiento en su IDE, ya sea en su editor de Java, Scala o XML.







# Quarkus

Quarkus es un marco de Java integral creado en Kubernetes para las compilaciones originales y las máquinas virtuales Java (JVM), el cual permite optimizar esta plataforma especialmente para los contenedores y para los entornos sin servidor, de nube y de Kubernetes.

Se diseñó para que funcione con las bibliotecas, los marcos y los estándares de Java conocidos, como Eclipse MicroProfile y Spring (ambos se presentaron como parte de una sesión en el evento Red Hat Summit 2020), así como también Apache Kafka, RESTEasy (JAX-RS), Hibernate ORM (JPA), Infinispan, Camel y muchos más.



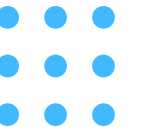




# Anti-corruption layer

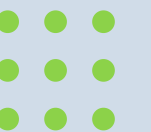
Implementa una fachada entre aplicaciones nuevas y las heredadas, para garantizar que el diseño de una nueva aplicación no esté limitado por las dependencias de los sistemas heredados.





# Backends for Frontends

Crea servicios backend independientes para diferentes tipos de clientes, como computadoras de escritorio y dispositivos móviles. De esta forma, un único servicio de backend no necesita manejar los requisitos conflictivos de varios tipos de clientes. Este patrón puede ayudar a mantener cada microservicio simple, separando las preocupaciones específicas del cliente.





# Request-Replay

Desacople el procesamiento de componente a otro, donde el procesamiento de back-end debe ser asíncrono, pero el front-end todavía necesita una respuesta clara.





# ¡Gracias!

**Búscanos como:**

**Visita nuestra web:**

**Correo:**

**Teléfono:**

@gizlosoftware

[www.gizlocorp.com](http://www.gizlocorp.com)

[info@gizlocorp.com](mailto:info@gizlocorp.com)

+1(207)900-8663

