Introducción a entranamiento Cloud (GCP)

Segunda sesión

Fundamentos Cloud

Group

Scotiabank

GRUPO CORPORIS
Capacitación | Consultoría | OTEC

Valor centrado en el desarrollo de las personas

Resumen del curso

Plan de Estudios de Google Cloud Platform (GCP)

Bloque 2: *Fundamentos Cloud*

Fundamentos de Arquitectura Cloud

Principios de diseño de arquitecturas resilientes, escalables y de alta disponibilidad en la nube

Patrones de diseño comunes en la nube (microservicios, serverless, contenedores)

AWS

Visión general de los servicios principales de AWS (EC2, S3, Lambda, RDS)

Google Cloud Platform

Servicios fundamentales de GCP: Compute Engine (máquinas virtuales), Cloud Storage (almacenamiento de objetos), VPC (redes virtuales), Cloud SQL (bases de datos gestionadas)

Resumen del curso

Plan de Estudios de Google Cloud Platform (GCP)

Bloque 2: *Fundamentos Cloud*

Google Cloud Platform

Servicios fundamentales de GCP: Compute Engine (máquinas virtuales), Cloud Storage (almacenamiento de objetos), VPC (redes virtuales), Cloud SQL (bases de datos gestionadas). Gestión de recursos en GCP: proyectos, facturación, IAM (Identity and Access Management)

Azure

Visión general de los servicios principales de Azure (Virtual Machines, Blob Storage, Azure Functions, Azure SQL Database)

Ventajas y desventajas Cloud (Recapitulación)

Escalabilidad

Alta disponibilidad

Costo eficiente *

Seguridad robusta

Velocidad de despliegue

Dependencia del proveedor

*Seguridad compartida

*Costos impredecibles

Implementaciones no regulares

Ventajas y desventajas Cloud (Recapitulación)

Escalabilidad

Alta disponibilidad

Resiliente

Motivación

Group

Scotiabank

GRUPO CORPORIS
Capacitación | Consultoría | OTEC

Valor centrado en el desarrollo de las personas

Escalabilidad

Soy dueño de un bar y mis clientes no paran de crecer

El problema:

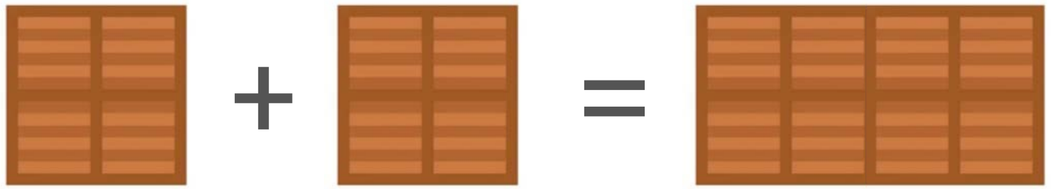
Al principio, en mi bar venían grupos pequeños. Una mesa para 4 personas era suficiente. Pero ahora, cada semana vienen más y más personas en grupos más grandes: de 6, de 8, de 10.

Tengo que pensar cómo adaptar mi espacio para atender a más gente sin que se vayan a otro lugar.

¿Qué opciones tengo?

Escalabilidad

Opción 1 (Horizontal)



Juntar mesas pequeñas

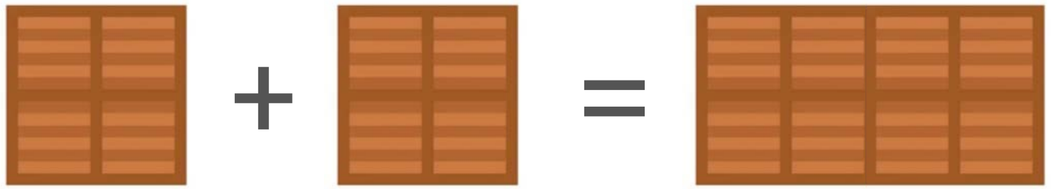
¿Por qué sirve?

Puedo atender cualquier número de personas y distribuir las mesas

Puedo utilizar mesas pequeñas que ya tengo

Escalabilidad

Opción 1 (Horizontal)



¿Dificultades?

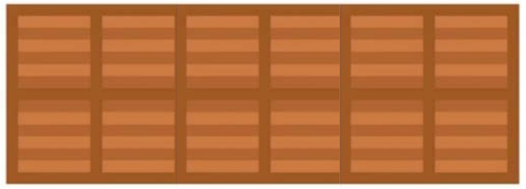
Necesito cambiar distribuciones, y coordinar al personal

Podrían considerarlas pequeñas para la celebración

La distribución podría no ser ideal

Escalabilidad

Opción 2 (Vertical)



Mesas más grandes

¿Por qué sirve?

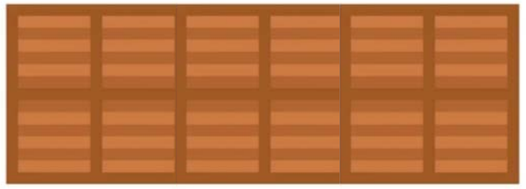
Tiene más capacidad para atender más gente

Es más fácil de implementar

Logistica más simple

Escalabilidad

Opción 2 (Vertical)



Mesas más grandes

¿Dificultades?

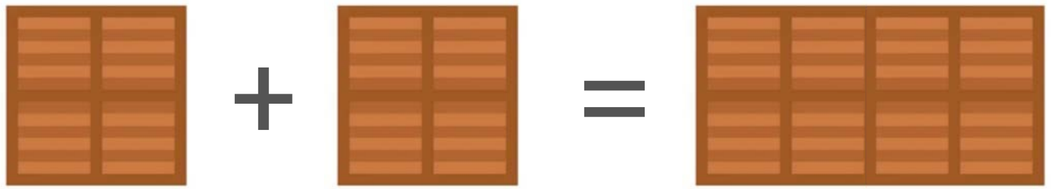
No puedo atender más personas que los que permite la mesa

Son más costosas

Muchas mesas quedarán con asientos vacios

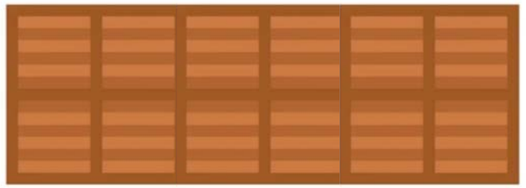
Escalabilidad

Opción 1 (Horizontal)



u

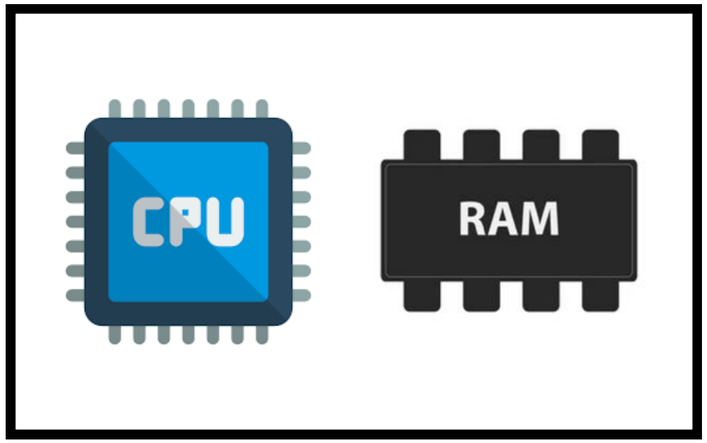
Opción 2 (Vertical)

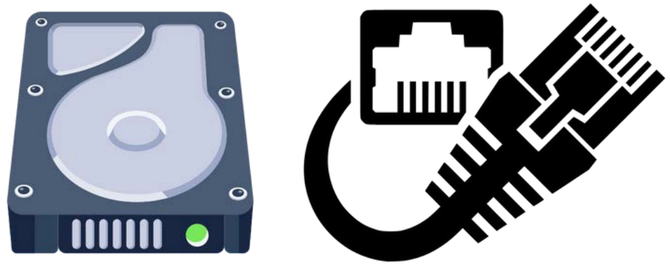


Valor centrado en el desarrollo de las personas

Escalabilidad

¿Y en la nube?





¿Qué conceptos importantes considerar?

Stateless y Event Driven

Load Balancers

Autoscaling

¿Qué nos podría ayudar a mejorar rendimiento?

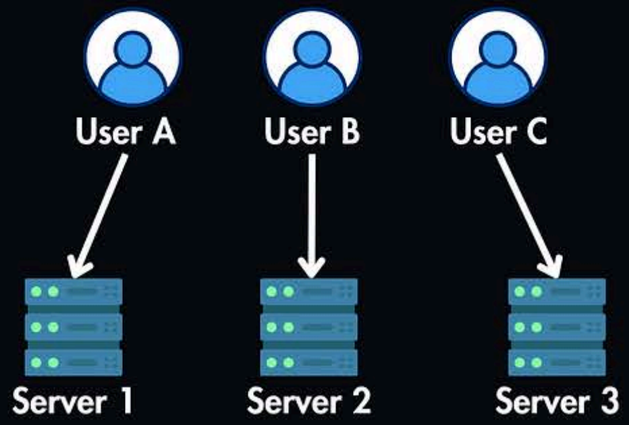
Caché

Sharding / Partitioning

Wrap Up

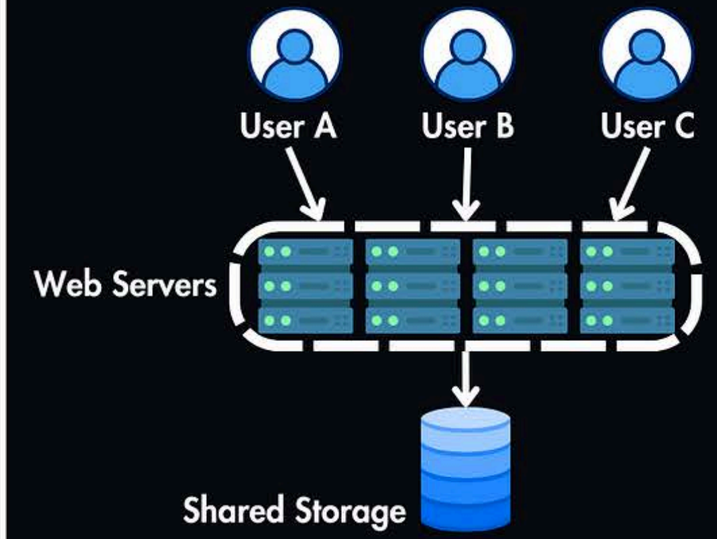
Stateful

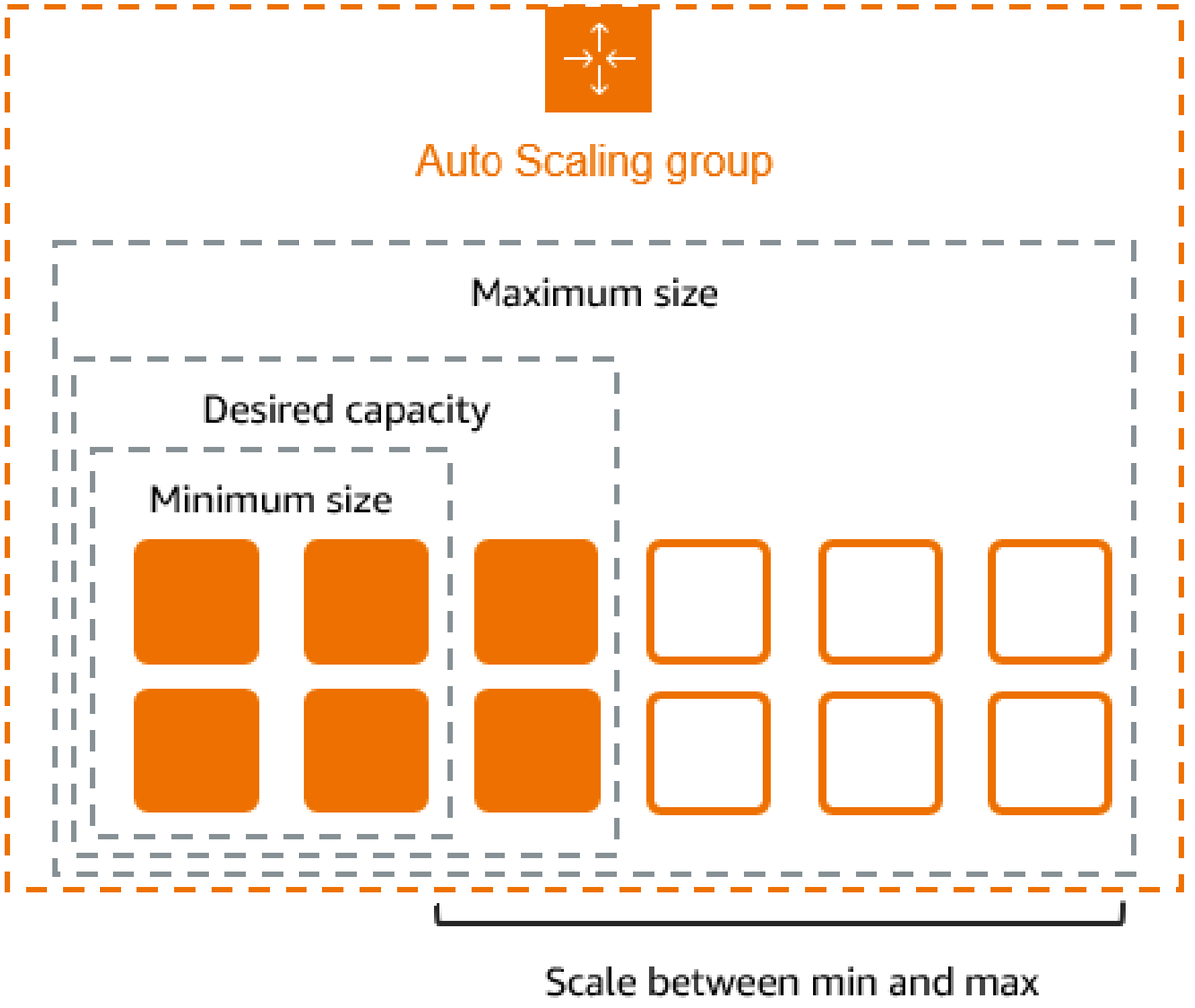
a server remembers who you are

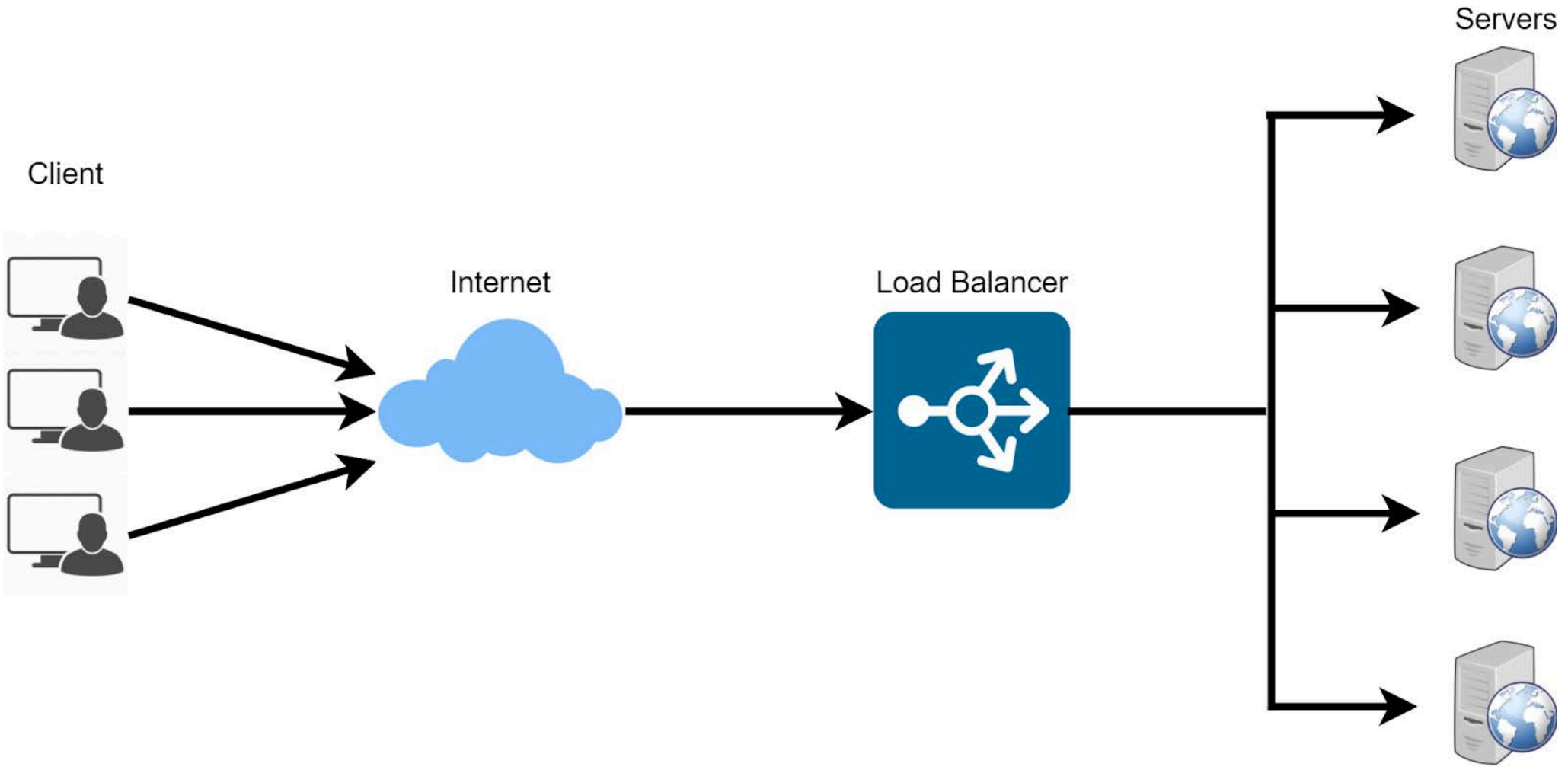


Stateless

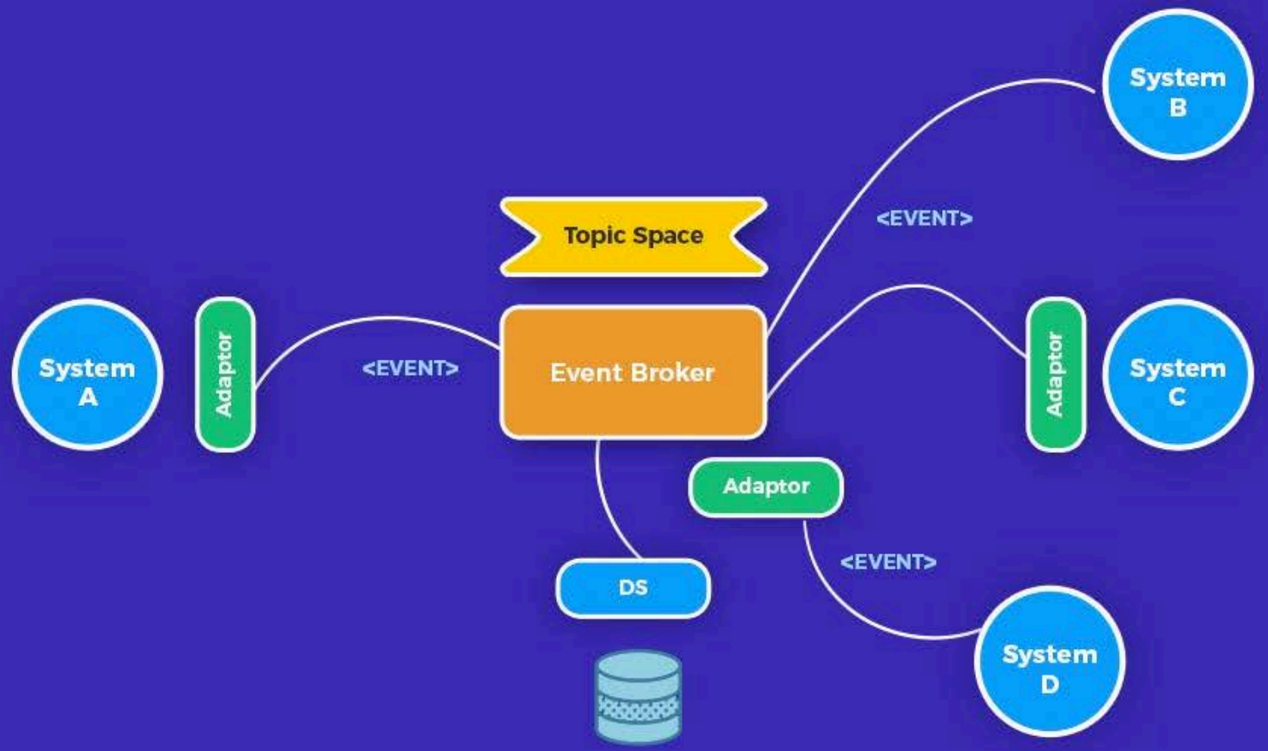
no server remembers who you are







EVENT DRIVEN ARCHITECTURE



Escalabilidad

Caso Práctico

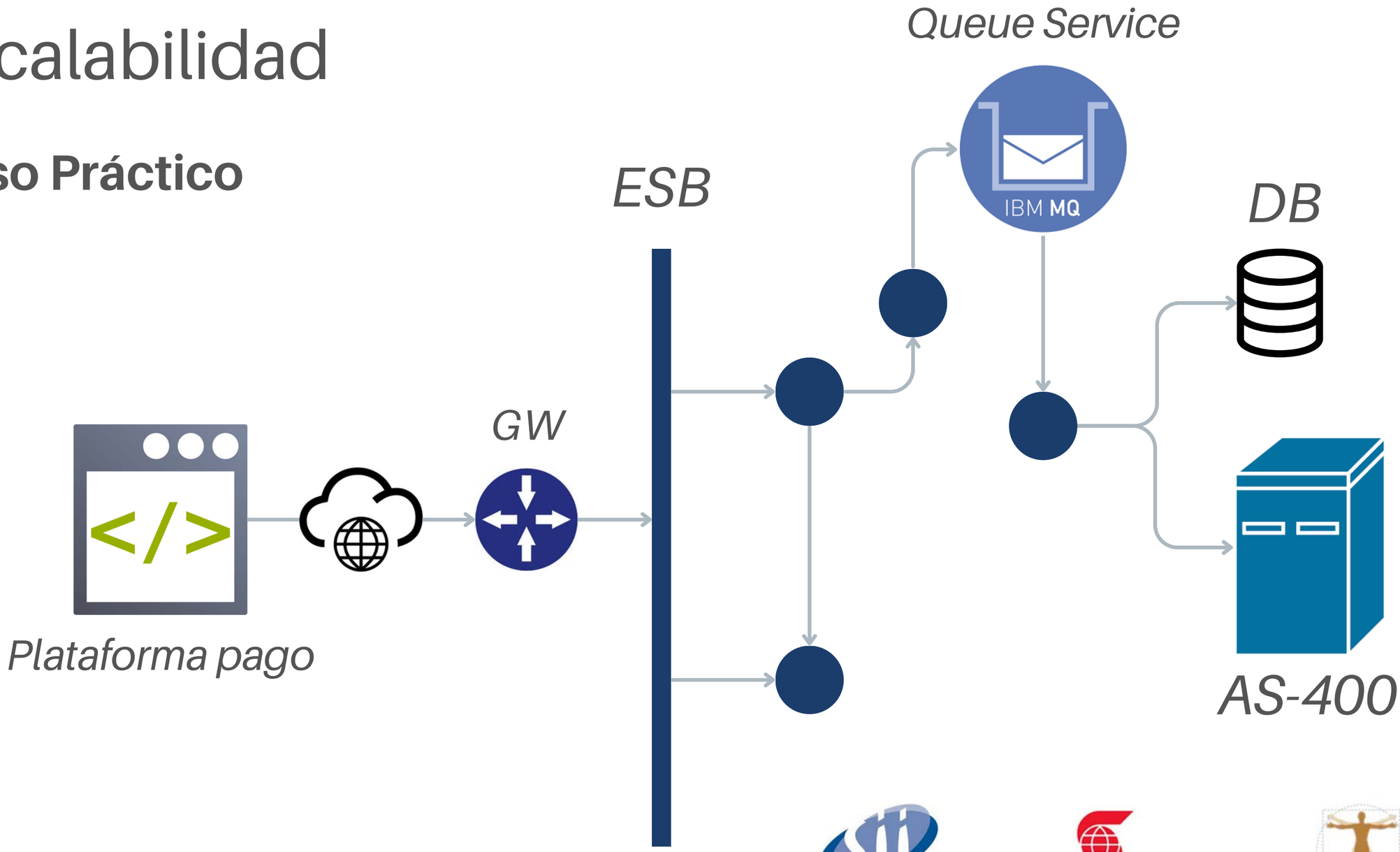
Una empresa de tecnología financiera (fintech) opera una plataforma de pagos online similar a Mercado Pago. Los usuarios realizan transacciones en tiempo real desde múltiples canales (web, app, API de terceros).

En los últimos meses, el sistema de procesamiento de transacciones ha experimentado cuellos de botella, especialmente en horarios de alta demanda, lo que ha provocado retrasos en la confirmación de pagos y en la visualización de los movimientos por parte de los usuarios.

Actualmente, el servicio de carga y validación de transacciones se ejecuta en un servidor centralizado, lo que ha demostrado ser insuficiente frente al crecimiento del volumen de operaciones.

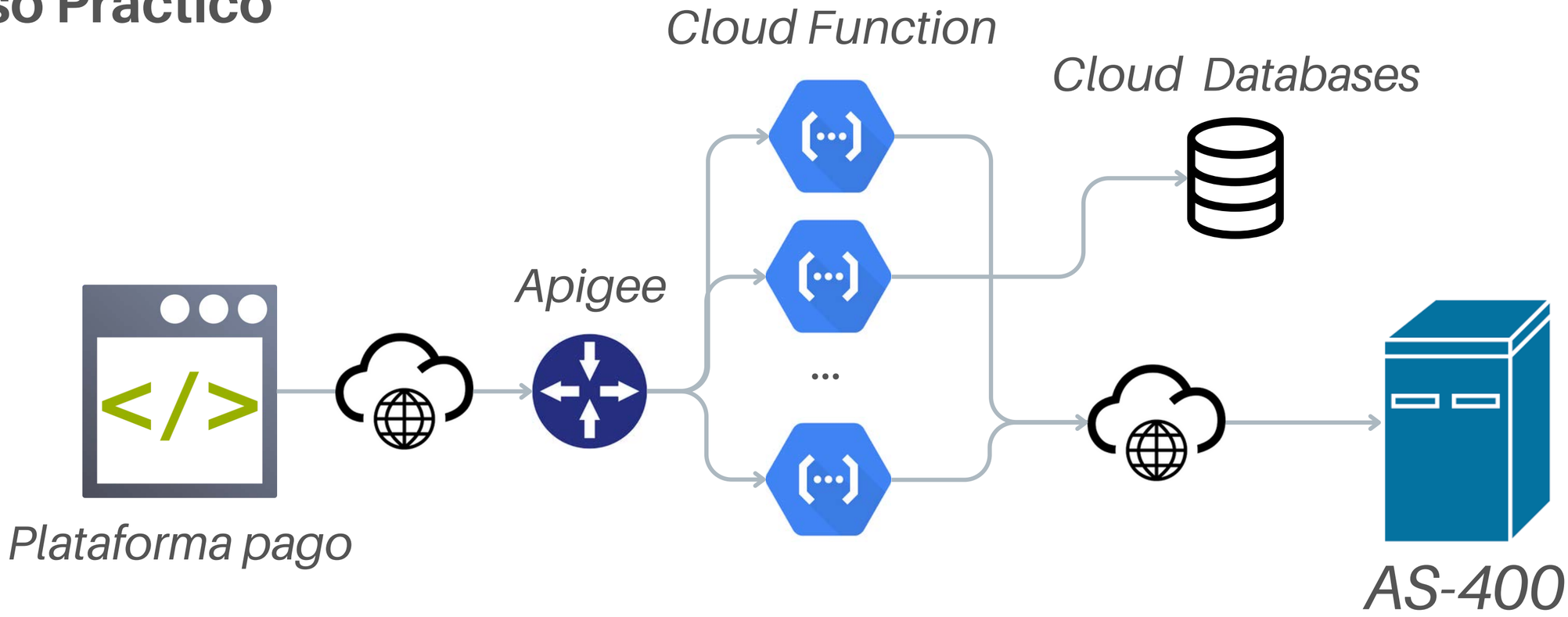
Escalabilidad

Caso Práctico



Escalabilidad

Caso Práctico



Alta Disponibilidad (HA)

Imagina que tienes una reunión muy importante y necesitas llegar sí o sí a tiempo usando Transantiago.

Y se suspende el recorrido de la 506, que es el que te sirve

Alta Disponibilidad (HA)

Sin alta disponibilidad

Te acercas a tu paradero y esperas el bus que tomas siempre.
Pero ese bus no llega.
Preguntas o revisas en la app, y te dicen:
"Ese recorrido está suspendido por hoy."
No hay buses alternativos cerca, ni otra forma rápida de llegar.
Te quedaste sin servicio.

Todo deja de funcionar!!!!

Alta Disponibilidad (HA)

¿Cómo mejoramos la disponibilidad del servicio?

- *La app de Transantiago te sugiere otras combinaciones de buses o incluso Metro.*
- *Si un bus no aparece, puedes tomar otro recorrido alternativo que conecta con tu destino.*

Redundancia

- *Incluso si hay un desvío en la ruta, el sistema reorganiza las opciones para que no te quedes sin transporte.*
- *Monitoreo de recorrido y estado de servicios de transporte (tomando acciones rápidas)*

Tolerancia a fallos

Alta Disponibilidad (HA)

Redundancias

Cantidad

N+1

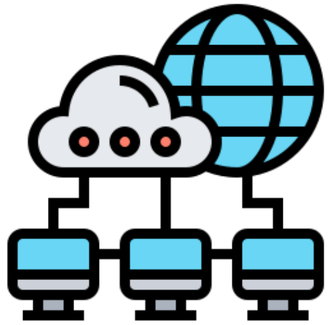
2N

Modo

Activo-Activo

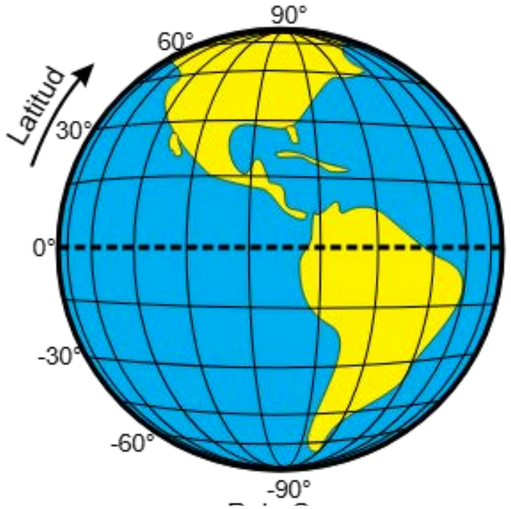
Activo-Pasivo

¿Qué normalmente requiere redundancia?



Alta Disponibilidad (HA)

Consideraciones



AZ

Posibles desastres naturales

Problemas de conectividad

Cortes de luz

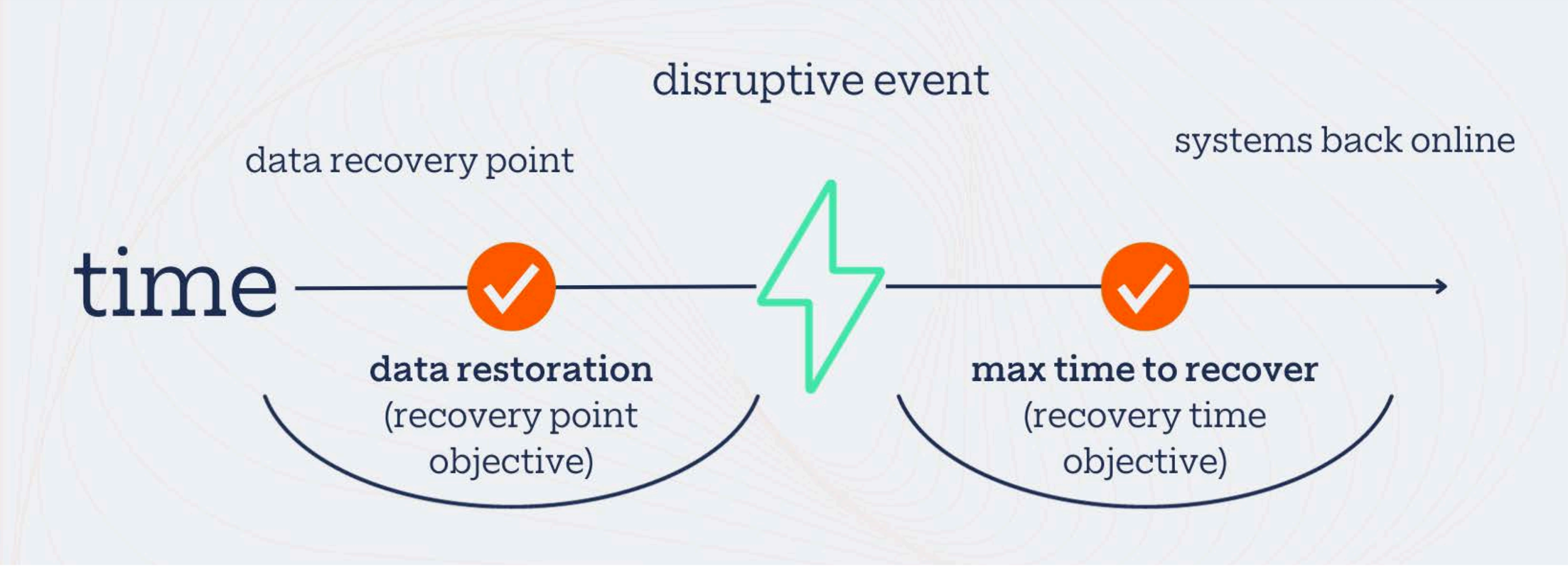
Regulaciones locales

Costo de sistemas

Alta Disponibilidad (HA)

Consideraciones

RPO y RTO



Alta Disponibilidad (HA)

Caso Práctico

El banco quiere lanzar un nuevo servicio digital para sus clientes (por ejemplo, una plataforma para gestionar inversiones). Dado que maneja información sensible y está sujeto a regulaciones, requiere:

Alta disponibilidad

- *Continuidad operacional ante fallos*
- *Cumplimiento con normativas locales (datos sensibles deben estar on-premise)*
- *Escalabilidad rápida en caso de alta demanda*

Objetivo del caso:

- *Diseñar una arquitectura de alta disponibilidad híbrida (on-premise + cloud) para este nuevo servicio, que asegure resiliencia, failover automático y cumplimiento.*

Alta Disponibilidad (HA)

Caso Práctico

Arquitectura Simplificada de Sandbox (On-Premise)

Elementos:

- *1 servidor Apache*
- *1 servidor Spring Boot*
- *1 HAProxy como balanceador*
- *1 PostgreSQL primario*
- *1 Redis para caché*
- *Switch L3 para redundancia de red*

Alta Disponibilidad (HA)

Caso Práctico

¿Cómo añadimos HA?

Alta Disponibilidad (HA)

Caso Práctico

¿Cómo sería en la nube?

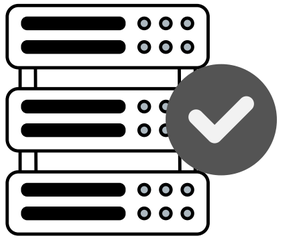
Resilencia

Capacidad de un sistema para tolerar fallos parciales sin afectar su operación global, y recuperarse automáticamente para seguir funcionando con mínima intervención humana.

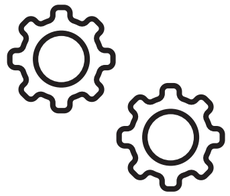
No sólo disponible, también responder a fallas

Resilencia

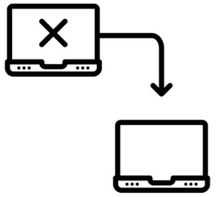
Health checks para detectar problemas en tiempo real.



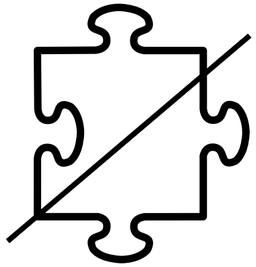
Failover controlado: si una región falla, enrutar a otra (DNS, Route 53).



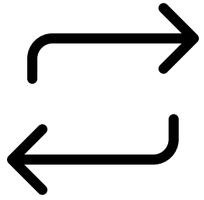
Desacoplamiento con colas/eventos, para que no dependan uno del otro en tiempo real.



Failover controlado: si una región falla, enrutar a otra (DNS, Route 53).



Circuit breakers: evitan cascadas de fallos entre servicios.



Retries con backoff: reintentos inteligentes sin saturar recursos.

Resilencia

Caso Práctico 1

El banco tiene un microservicio llamado auth-service desplegado en Kubernetes (EKS, GKE o on-prem). Este servicio:

- *Valida usuarios*
- *Emite tokens JWT*
- *Consume una base de datos y Redis para sesión temporal*

Dado que es crítico para operaciones, debe estar siempre disponible.

Resilencia

Caso Práctico 2

Un banco ofrece un portal web transaccional donde los clientes pueden revisar saldos, movimientos y realizar transferencias.

El frontend se comunica con una API REST interna para obtener los datos del cliente (/**api/account/summary**).

⚠️ Problema:
Durante un despliegue, se introdujo una lógica de reintentos automáticos en el frontend si la API falla que colapso el sistema.

¿Cómo diseñar sistemas escalables y menos acoplados?

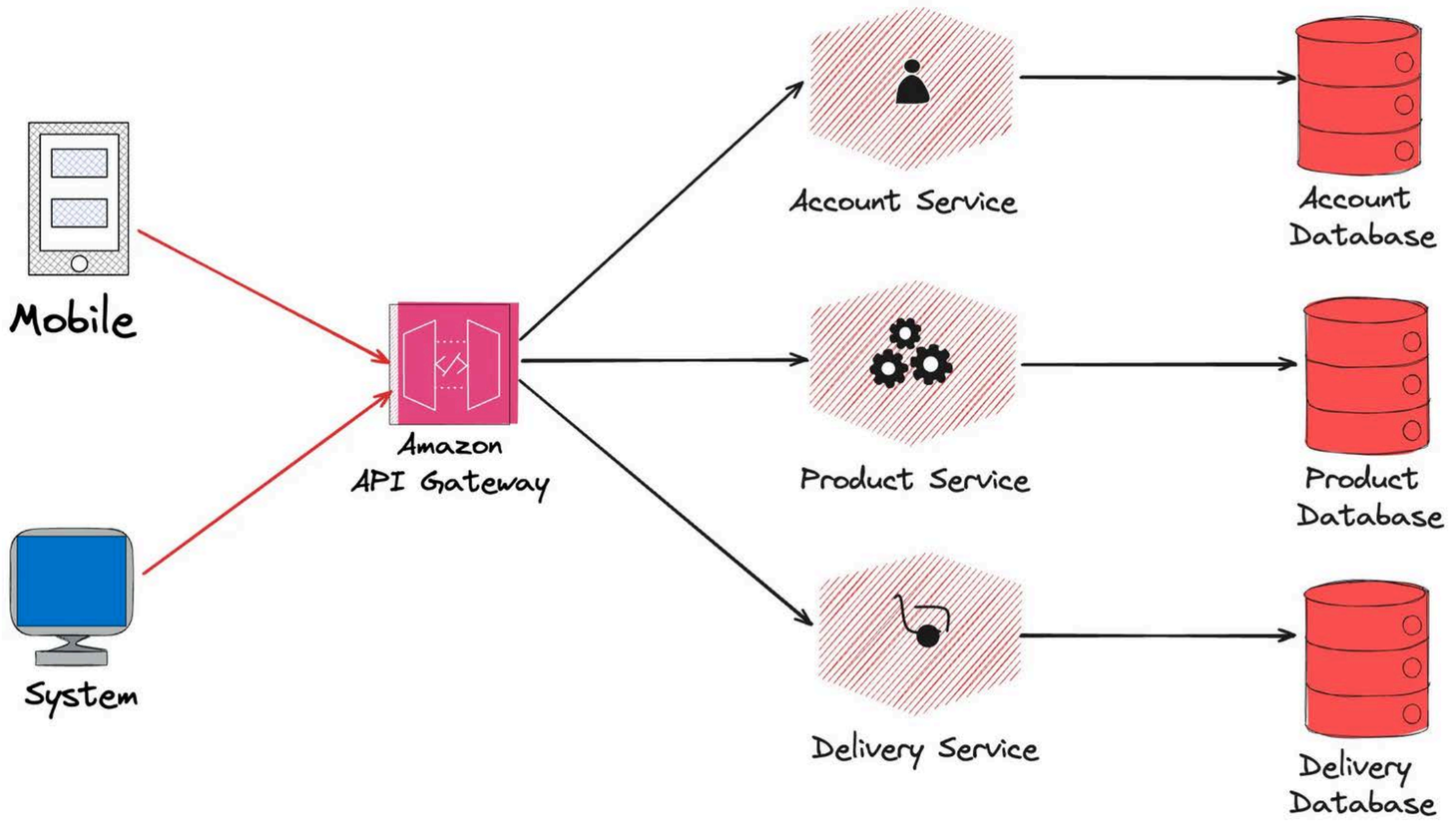
- Muchos compartimientos
- Poco espacio de trabajo (límite físico)
- Cuesta encontrar lo que necesito
- Muy probable tener muchas herramientas duplicadas

¿Cómo diseñar sistemas escalables y menos acoplados?

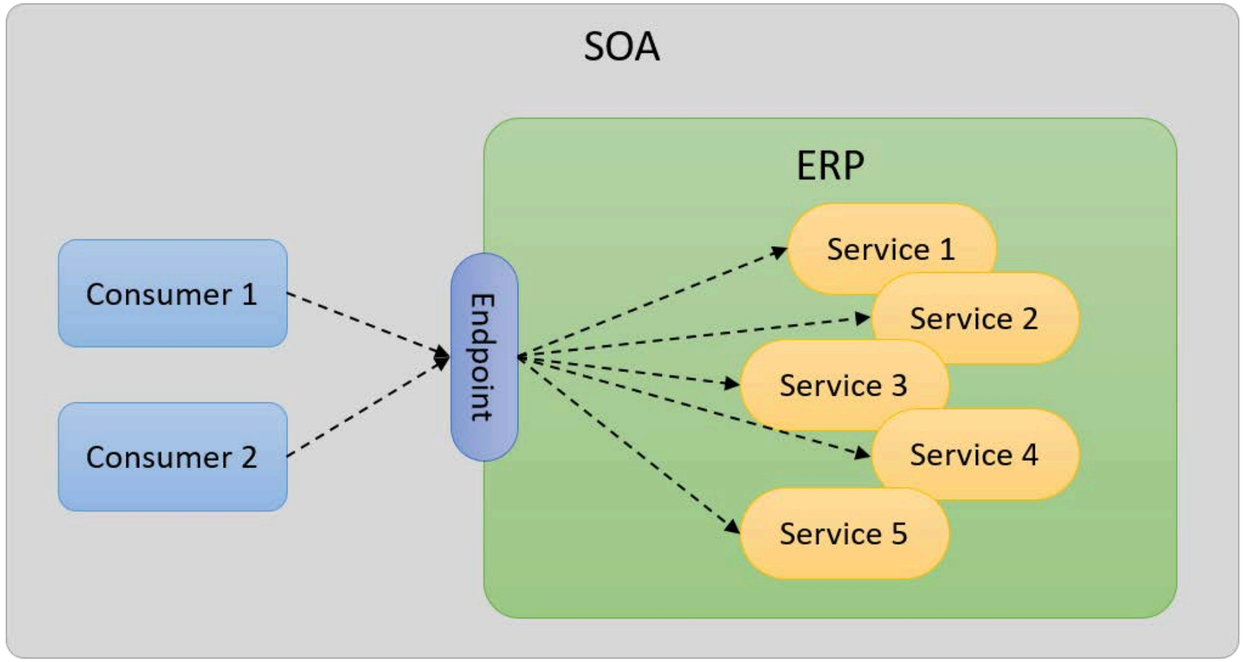
En software que es el símil a esto:

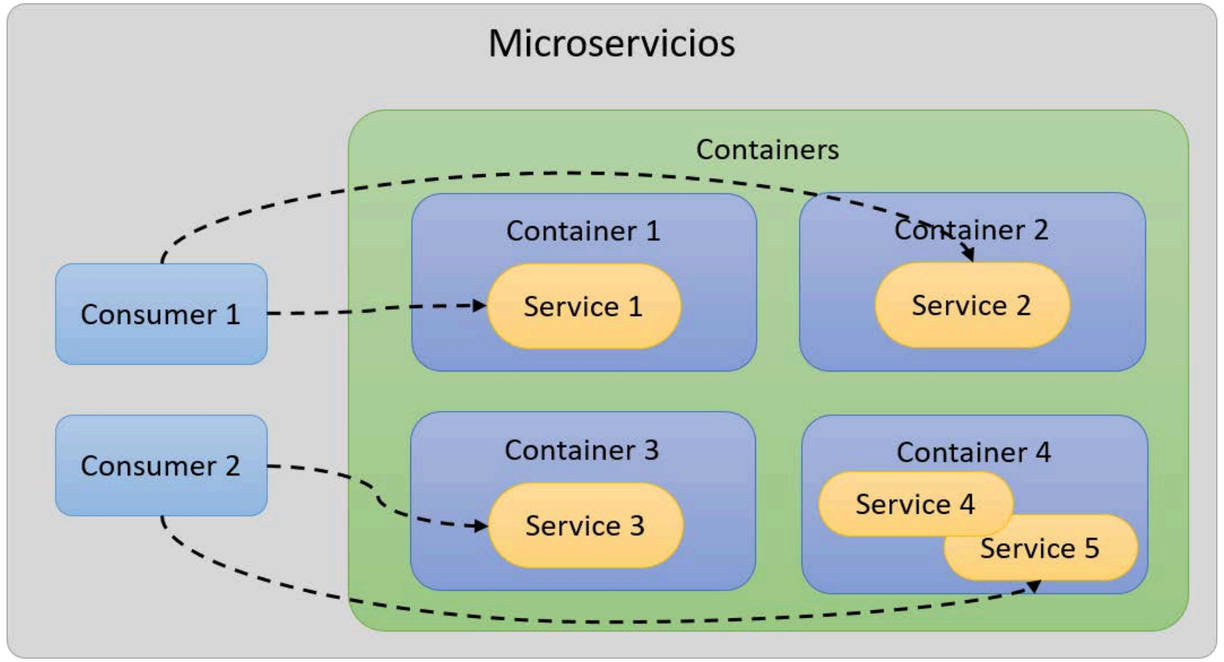
- Microservicios
- Serverless

Microservicios



Microservicios

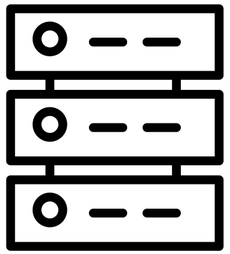




Microservicios

¿Cómo se conforma normalmente un Microservicio?

Interfaz



Backend

Base de datos

- Acotado
- Proposito claro

Microservicios

¿Cómo defino que microservicios modelar?

Criterios de negocio

Domain Driven Design
Event Storming
Bussiness Capability Mapping

Criterios técnicos

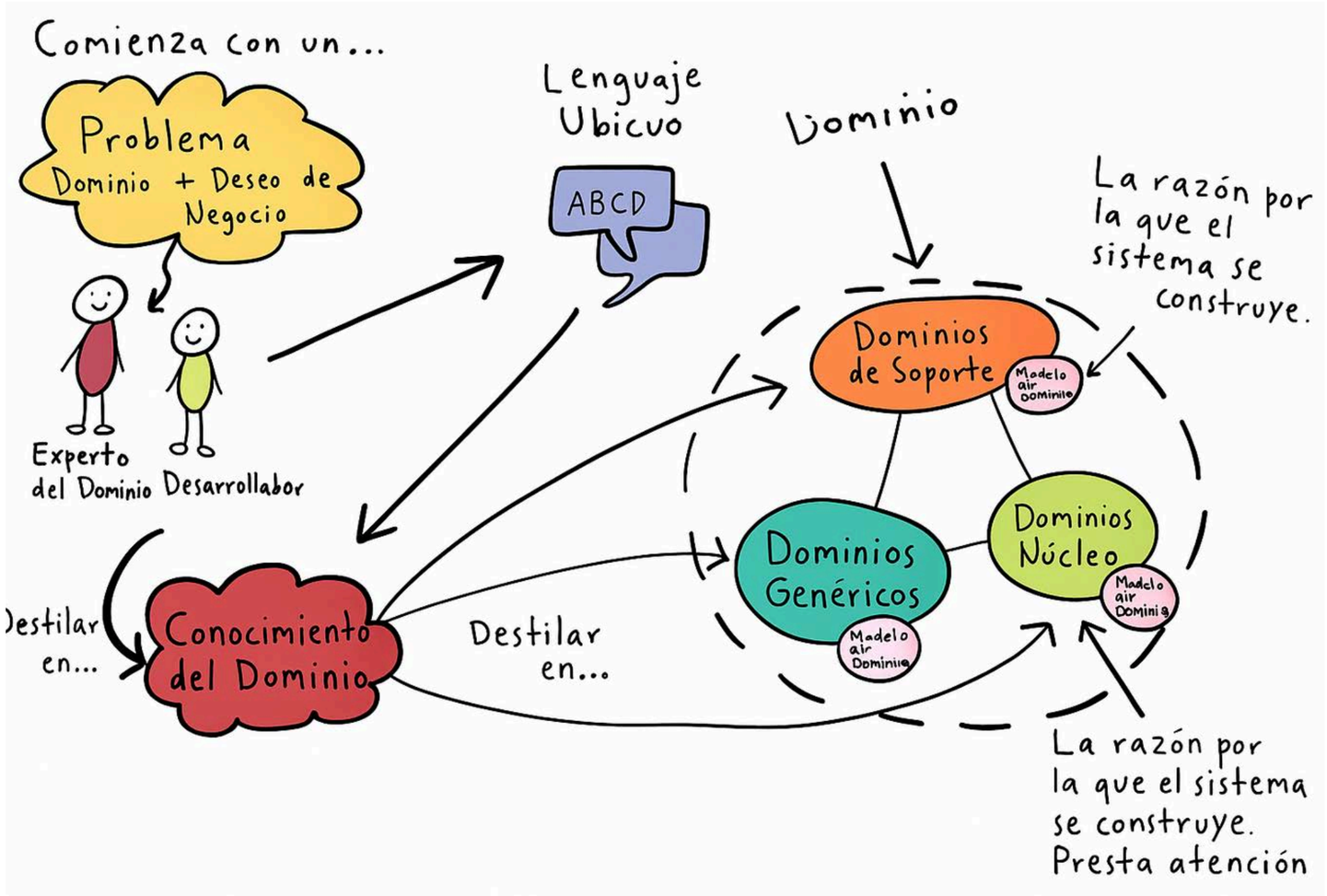
Hexagonal Architecture
Clean Architecture
Service Mesh

Recursos disponibles

Criterios de negocio

- **Dominio funcional claro**: cada microservicio debe alinearse con una función específica del negocio (por ejemplo: facturación, gestión de inventario, atención al cliente).
- **Independencia en la evolución del negocio**: si una parte del negocio cambia con frecuencia o de forma autónoma, puede justificar un microservicio separado.
- **Diferentes prioridades de desarrollo**: algunas funcionalidades pueden requerir cambios rápidos o tener objetivos distintos (como innovación vs. estabilidad).

Domain Driven Design



Bounded Contexts

Aggregates y Entities

Ubiquitous Language

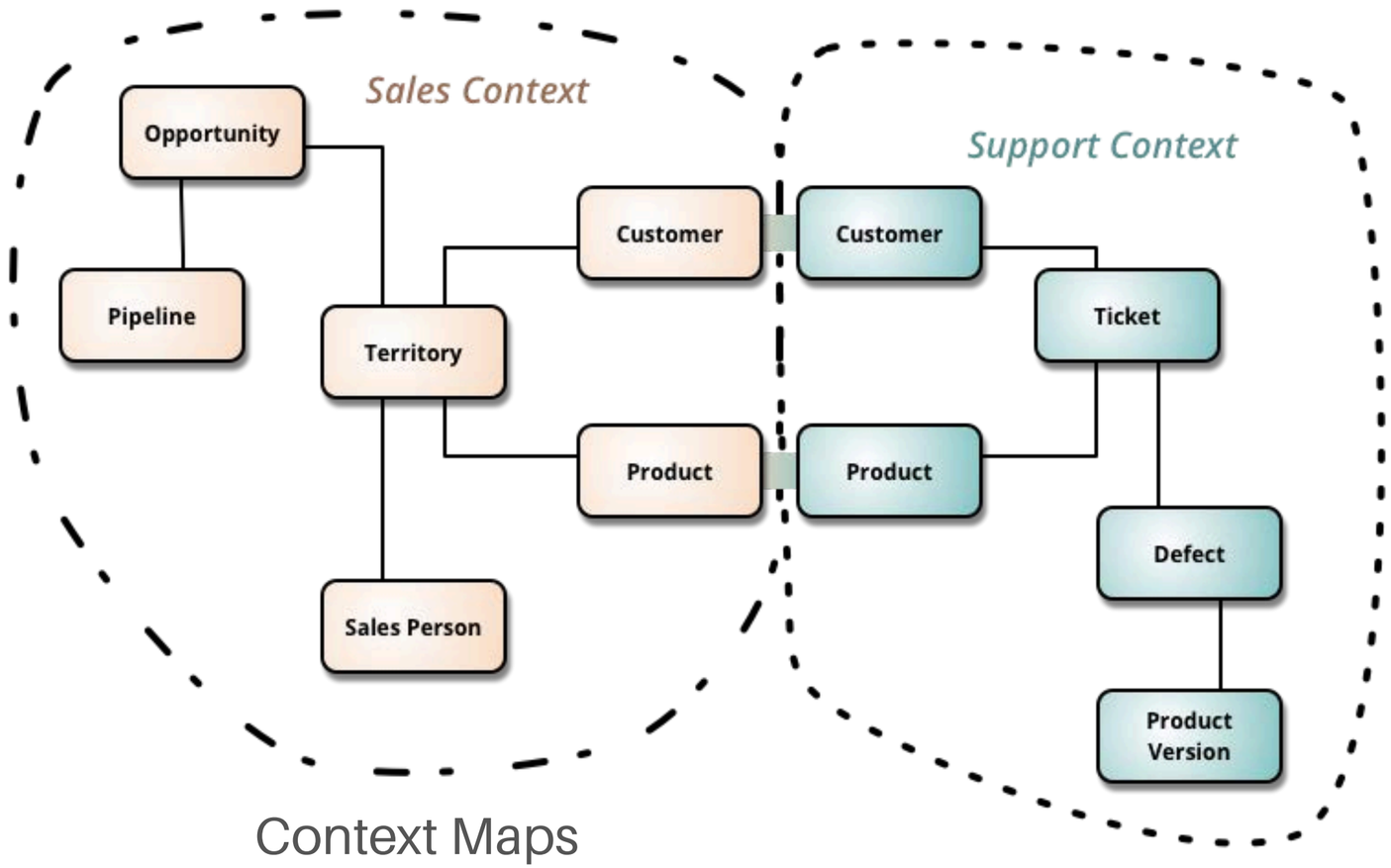
Group

Scotiabank

GRUPO CORPORIS
Capacitación | Consultoría | OTEC

Valor centrado en el desarrollo de las personas

Domain Driven Design



Context Maps

Bounded Contexts

- Entrevistas con expertos
- Listado de conceptos
- Ver que se traslapa conceptualmente

Domain Driven Design

DDD Táctico: Entidades, Value Objects y Agregados

Entidades

```
class CuentaBancaria {
  constructor(readonly id: string, private saldo: number) {}

  depositar(monto: number) { this.saldo += monto; }
  retirar(monto: number) { if (this.saldo >= monto) this.saldo -= monto; }
}
```

Tienen identidad

Mutables

Domain Driven Design

DDD Táctico: Entidades, Value Objects y Agregados

Value Objects

```
class Monto {
  constructor(readonly valor: number, readonly moneda: string) {
    if (valor < 0) throw new Error('El monto debe ser positivo');
  }
}
```

No tienen ID

Son comparables

Domain Driven Design

DDD Táctico: Entidades, Value Objects y Agregados

Agregados

Conjunto de objetos que se pueden considerar uno para interactuar con operaciones y transacciones

Un cliente del banco

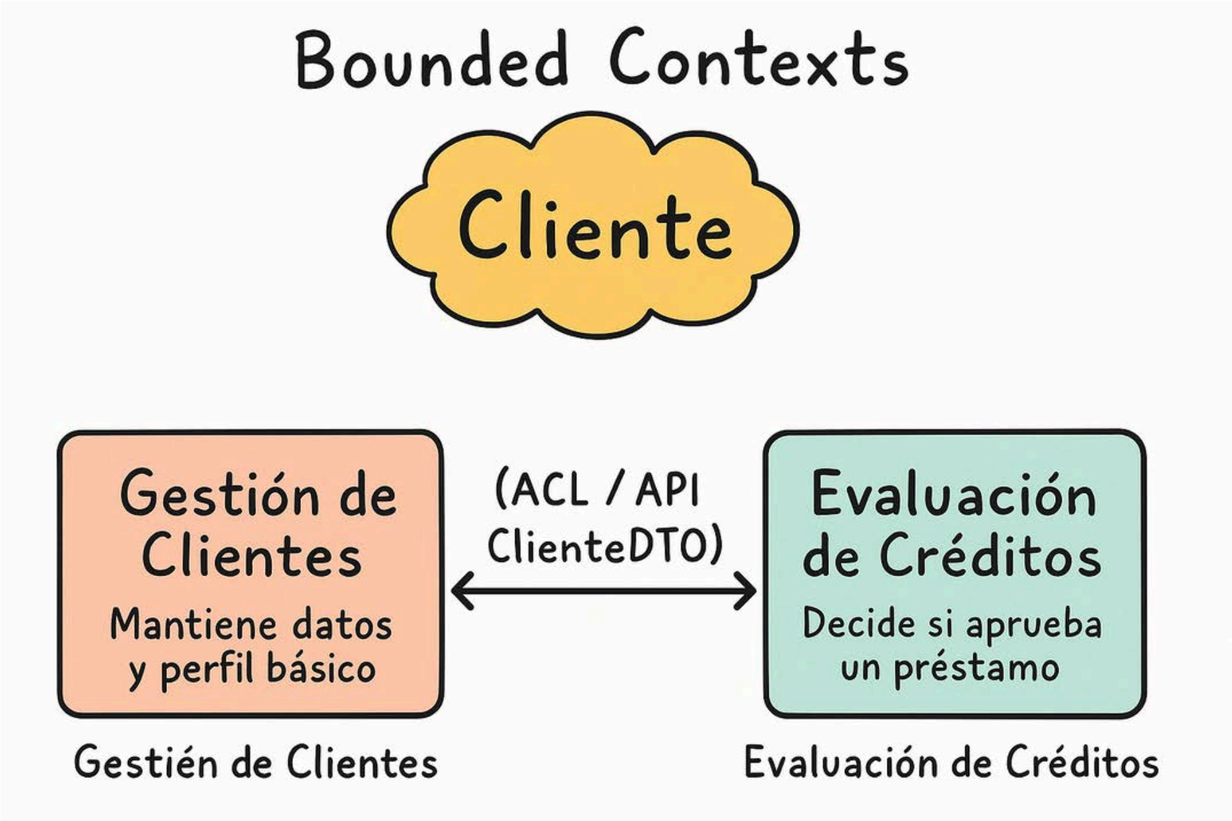
Domain Driven Design

Caso Práctico

Una institución financiera necesita modernizar su plataforma de otorgamiento de préstamos. Actualmente, el sistema es monolítico y presenta dificultades para escalar y adaptarse a nuevas reglas de negocio.
Para abordar este problema, se plantea un rediseño basado en microservicios. Se identifican al menos dos Bounded Contexts clave:
Gestión de Clientes: Responsable de almacenar y administrar la información personal y de contacto de los clientes.
Evaluación de Créditos: Encargado de procesar solicitudes de préstamo, validando antecedentes financieros y tomando decisiones de aprobación.
Ambos contextos comparten el concepto de “cliente”, pero desde perspectivas distintas.

Domain Driven Design

Caso Práctico



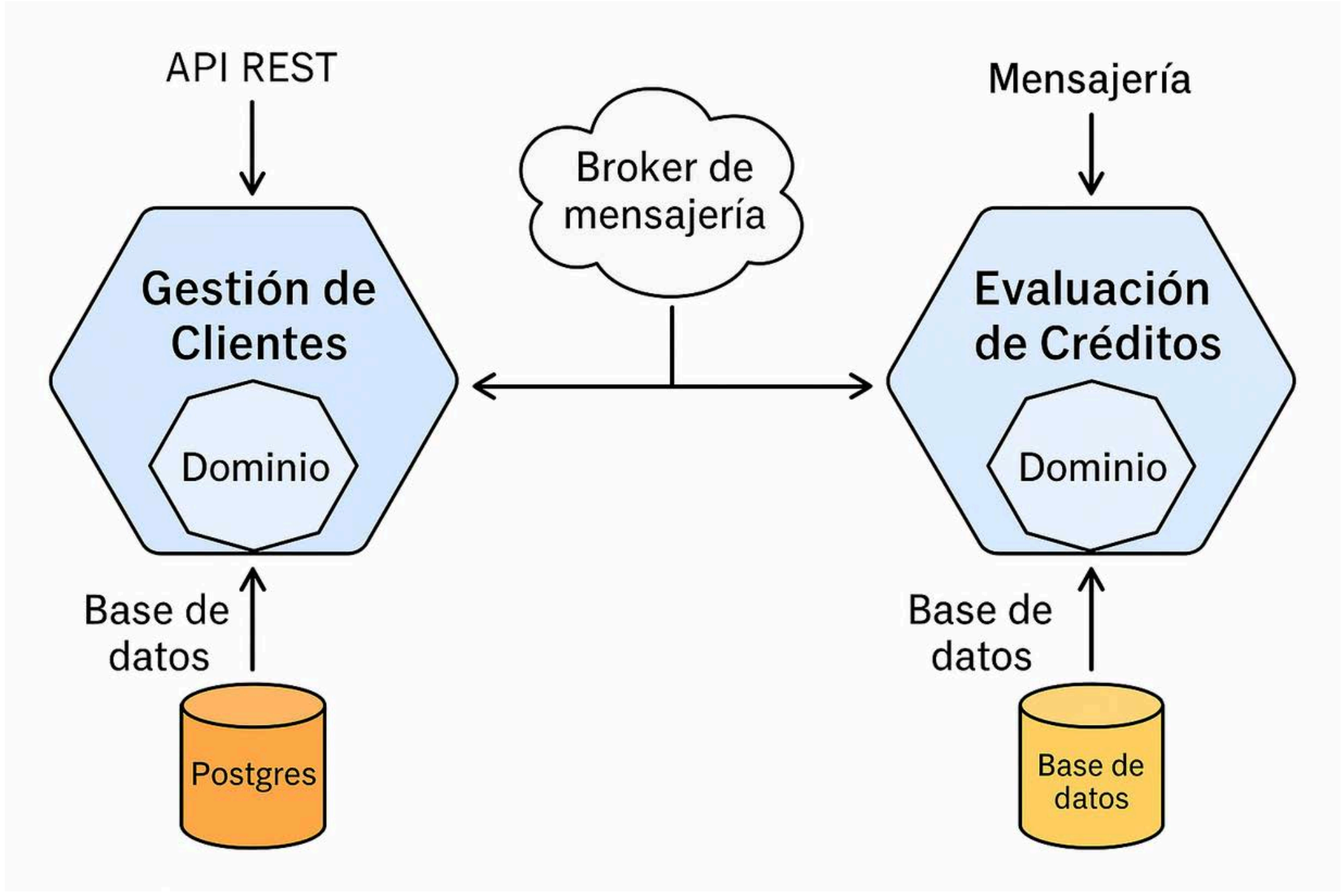
¿Cuáles serían mis candidatos a microservicios?

Gestión de Clientes

Evaluación de Créditos

Domain Driven Design

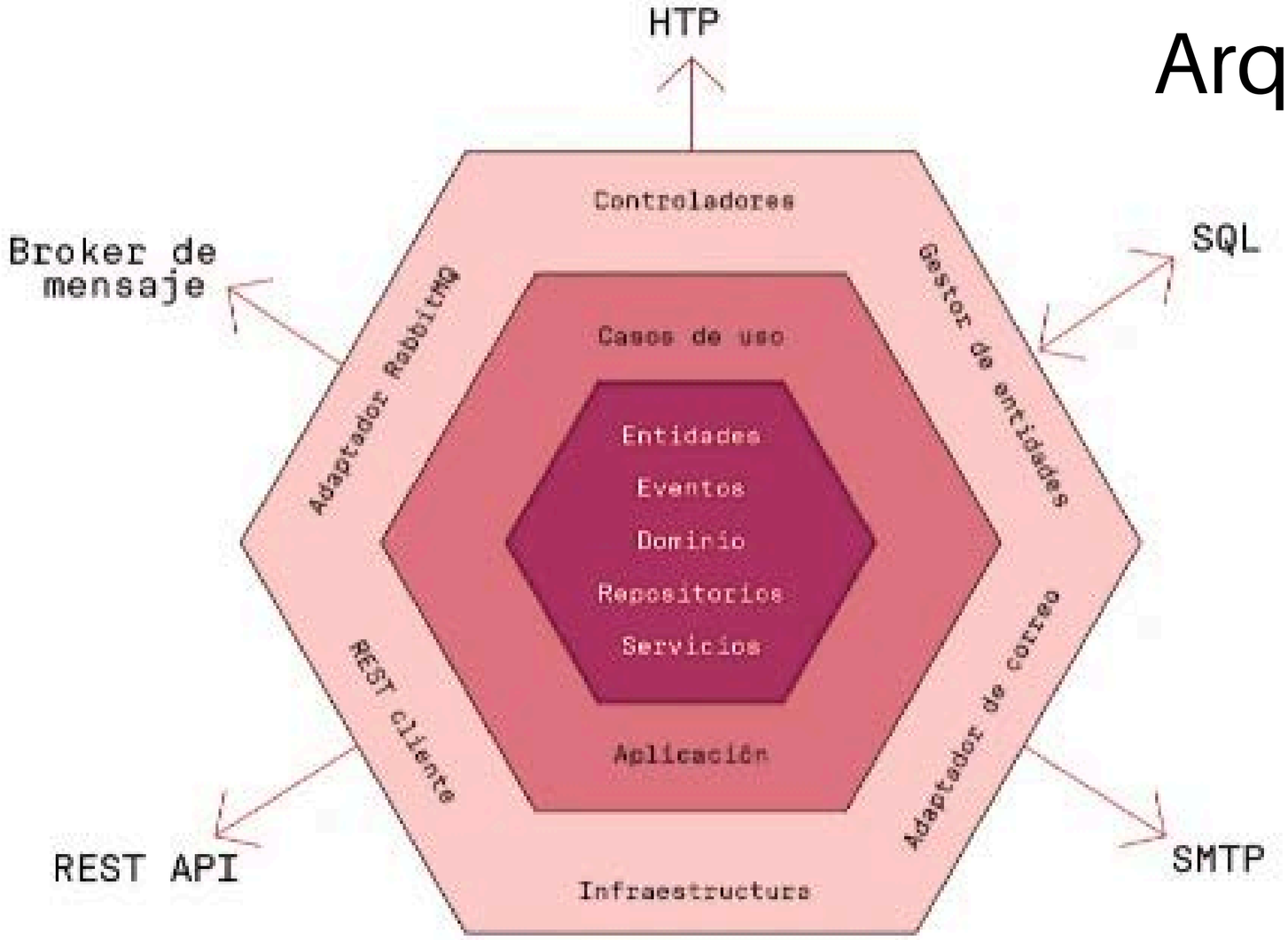
Caso Práctico



Criterios de técnicos

- **Escalabilidad independiente**: componentes que necesitan escalar de forma diferente (por ejemplo, el módulo de búsqueda puede requerir más recursos que el de autenticación).
- **Aislamiento de fallos**: evitar que un problema en un componente afecte a todo el sistema.
- **Requerimientos tecnológicos distintos**: por ejemplo, un servicio que necesita procesamiento en tiempo real puede estar mejor implementado en un lenguaje diferente o con una arquitectura distinta (como serverless).
- **Límites naturales de datos**: cuando una funcionalidad puede tener su propia base de datos sin necesidad de joins con otros dominios.

Arquitectura Hexagonal



Puertos y adaptadores

Dominio (Core)

Aplicación (casos de uso)

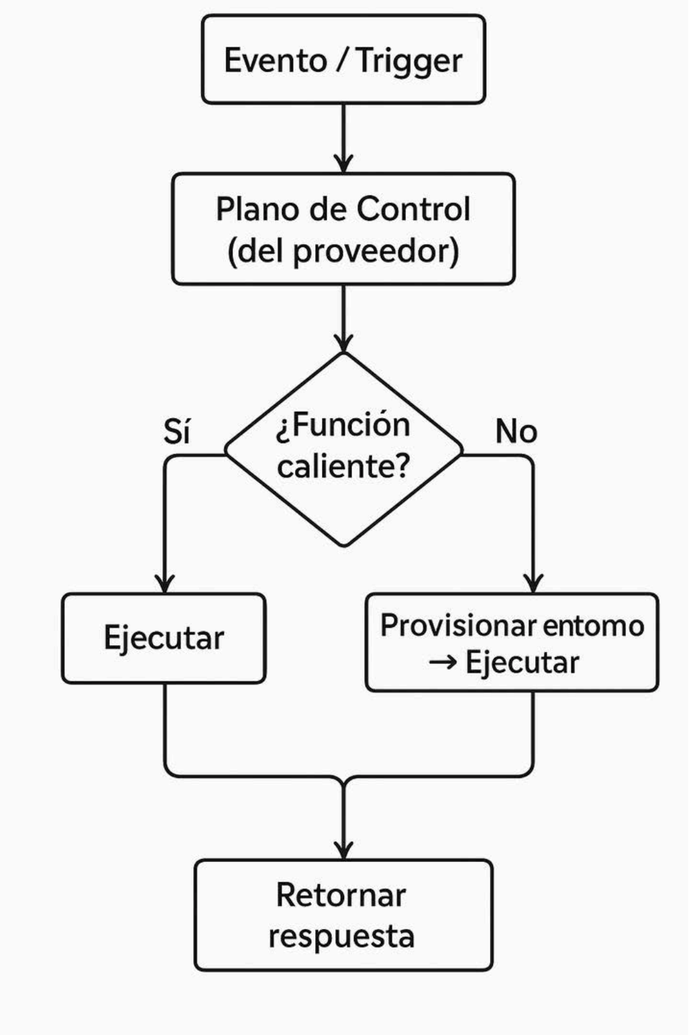
Serverless

¿Cómo corre un código si no hay servidor?

Sí lo hay!!!!!

Pero se disponibilizan por requerimiento

Serverless



Las máquinas se encienden

Cold Start

Warm

GRUPO CORPORIS
Capacitación | Consultoría | OTEC
Valor centrado en el desarrollo de las personas

Serverless

¿Qué debo considerar al diseñar?

Serverless

Cold Start Friendly

Ejecuciones rápidas

Concurrencia

Uso eficiente de recursos

Event-driven

Escalabilidad

Idempotencia

Serverless

¿Qué debo considerar al diseñar?

Desacoplado

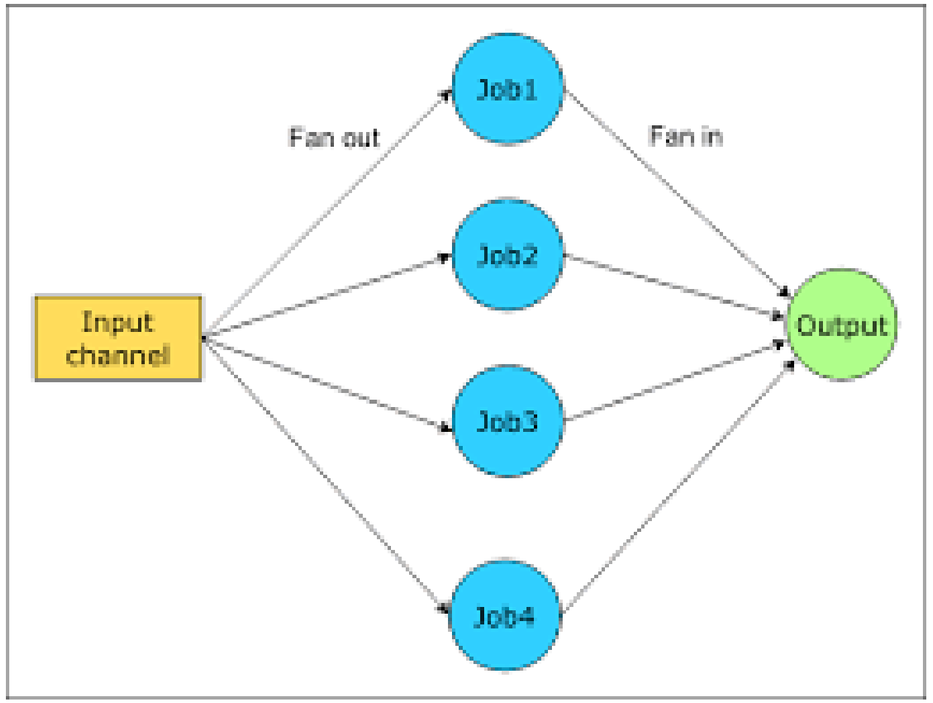
Observabilidad

Seguridad mínimo acceso

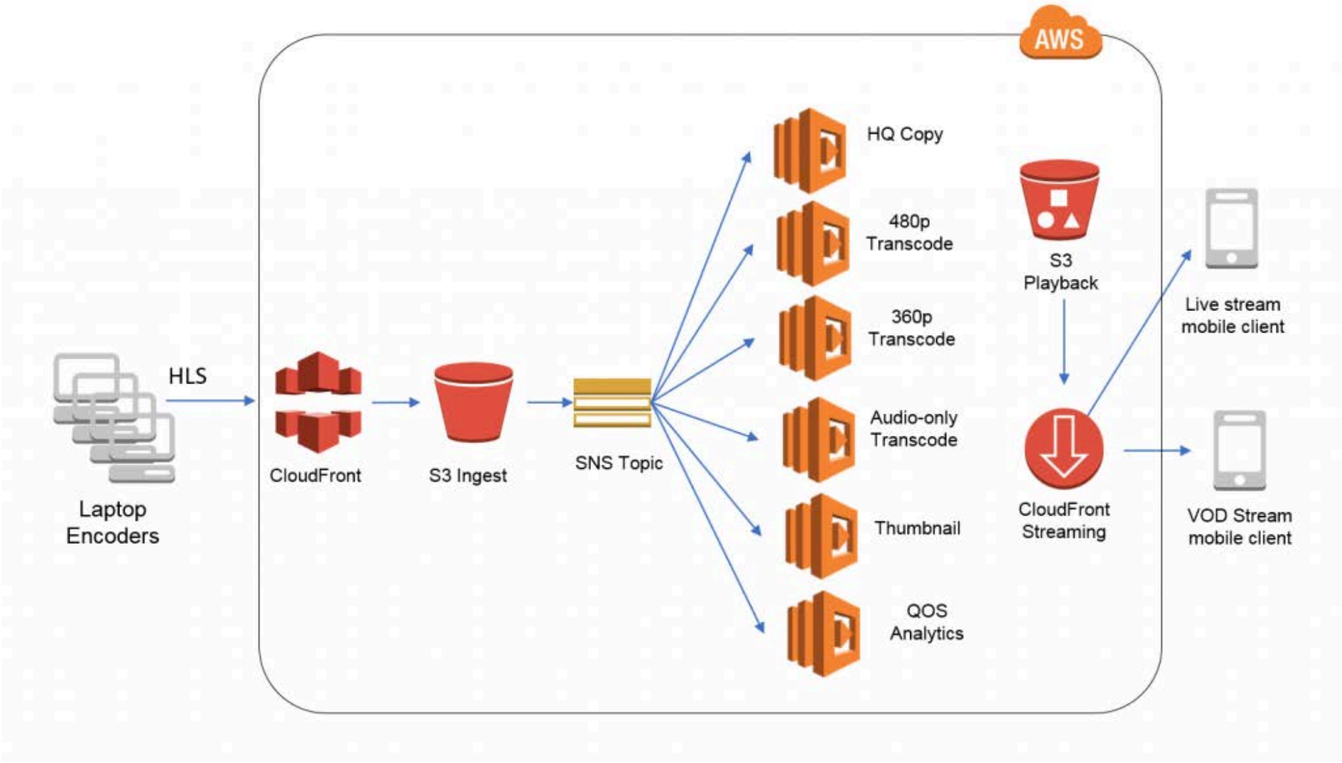
Reintentos

Patrones de diseño Serverless

Fan In/Fan Out



Messenger / Subscriber



Patrones de diseño Serverless

Caso Práctico

Evaluación Comercial de Clientes – ETL Diario (Serverless)

Contexto:

El banco necesita consolidar diariamente la información comercial de cada persona (ingresos, comportamiento financiero, deudas, etc.) desde múltiples fuentes externas e internas. Esta información se usará para analítica avanzada y futuras decisiones de crédito.

Patrones de diseño Serverless

Caso Práctico

Consideraciones:

Fuentes de datos:

- *Servicio_AFP*
- *Servicio_SII*

Información:

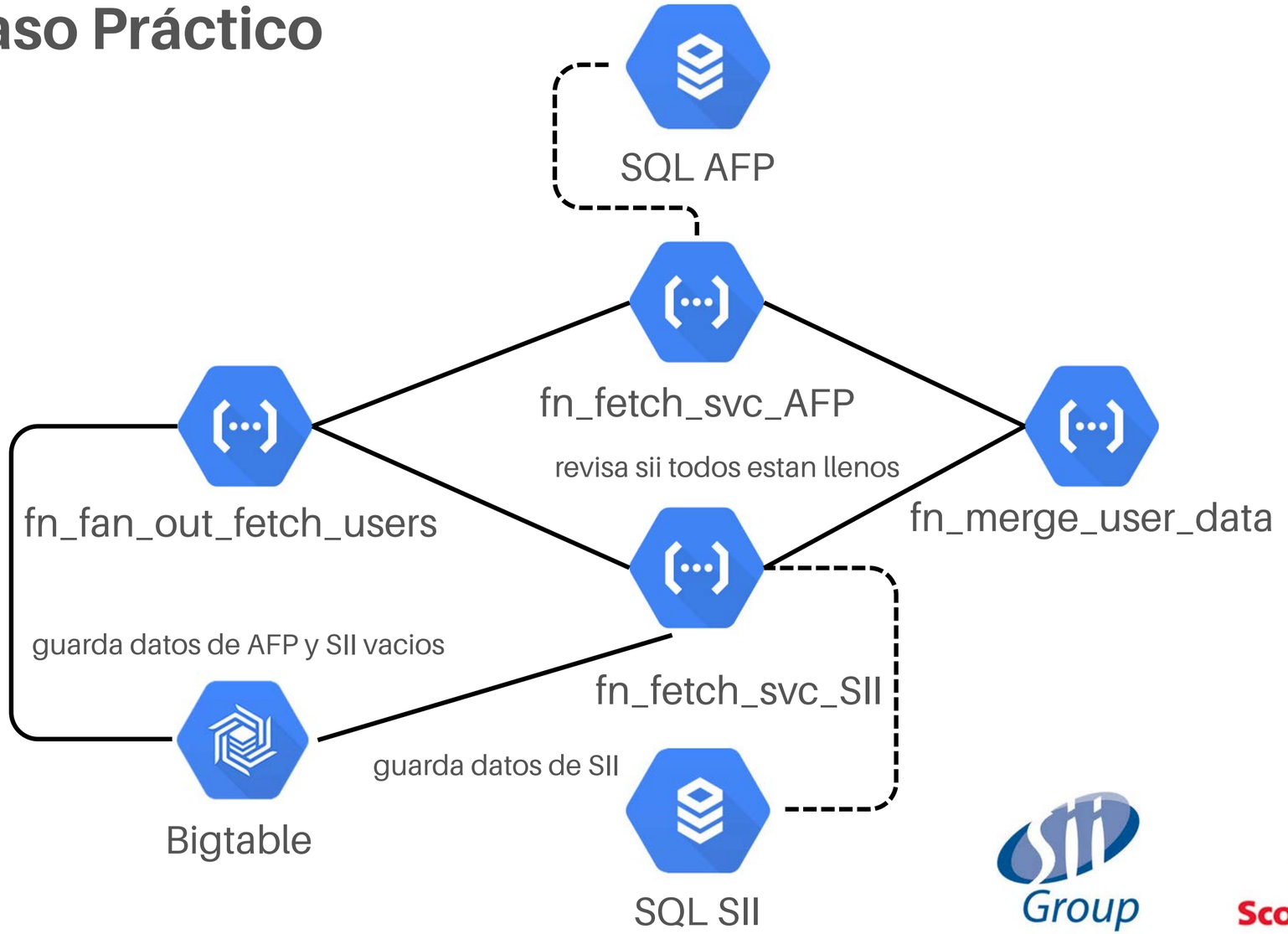
- *Bigtable*
  - *Llave primaria (user_id_svc)*

Topicos Colas:

- *data_ready*

Patrones de diseño Serverless

Caso Práctico



Pub/Sub

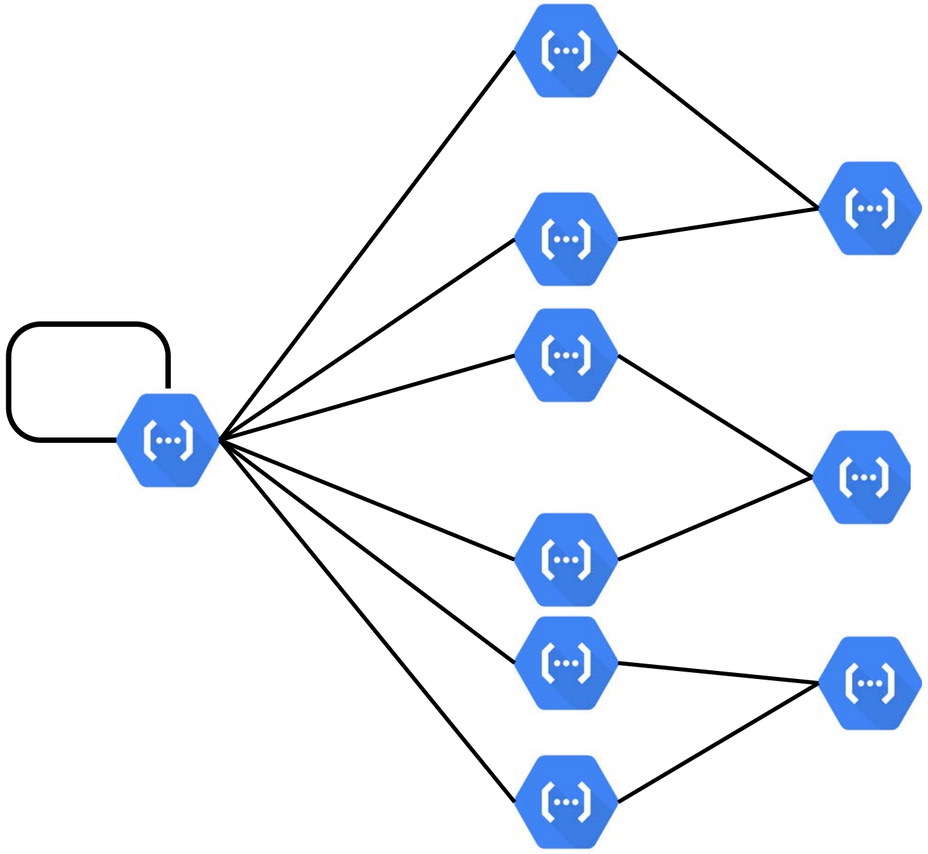
Group

Scotiabank

GRUPO CORPORIS
Capacitación | Consultoría | OTEC
Valor centrado en el desarrollo de las personas

Patrones de diseño Serverless

Caso Práctico



Patrones de diseño Serverless

Caso Práctico: Smart ATM

Sensores en ATM reportan datos (nivel de efectivo, errores, estado) y mandan la información via HTTP a la nube.

Se requiere alertar a las distintas áreas de mantenimiento de forma automática para mejorar la calidad de servicio al cliente.

¿Cómo lo harían? (veamos opciones y sus ventajas)

Patrones de diseño Serverless

Event Driven Design

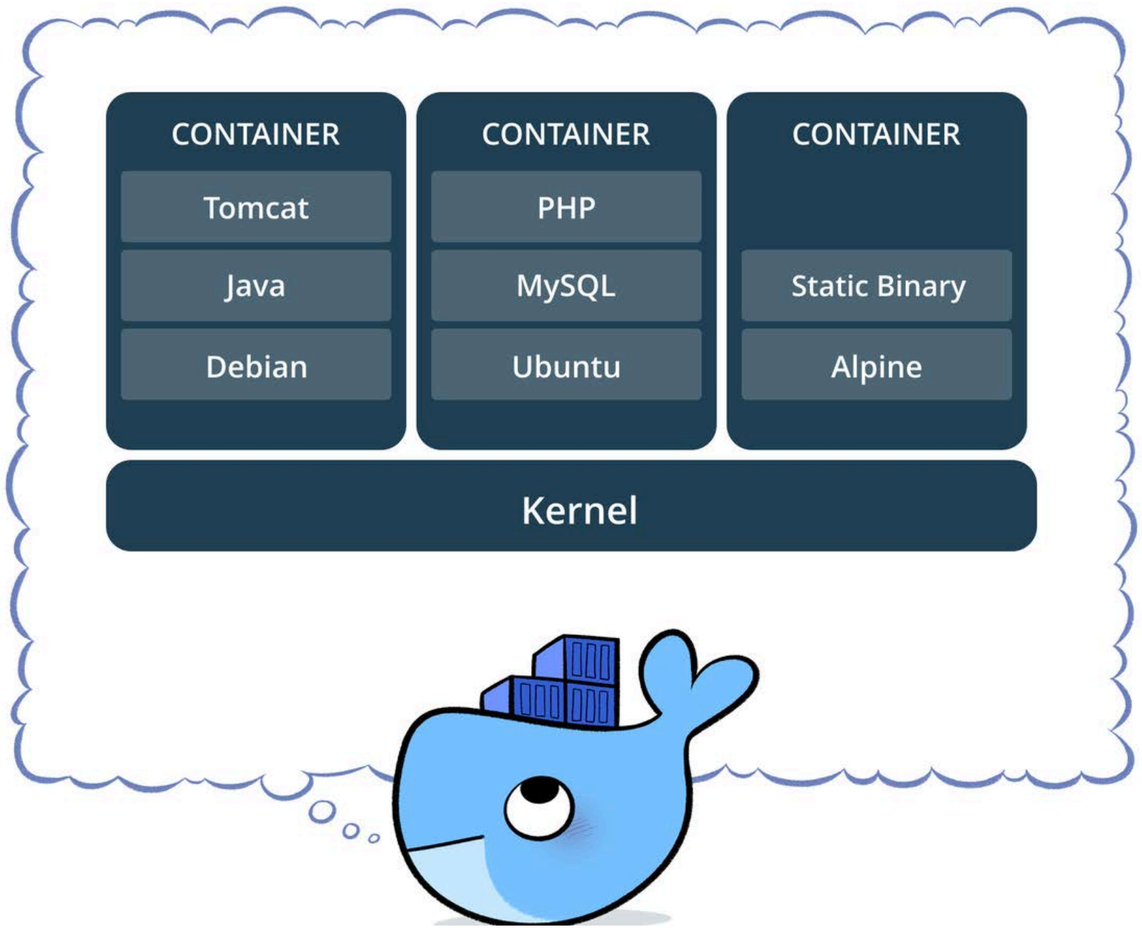
Caso Práctico: Smart ATM

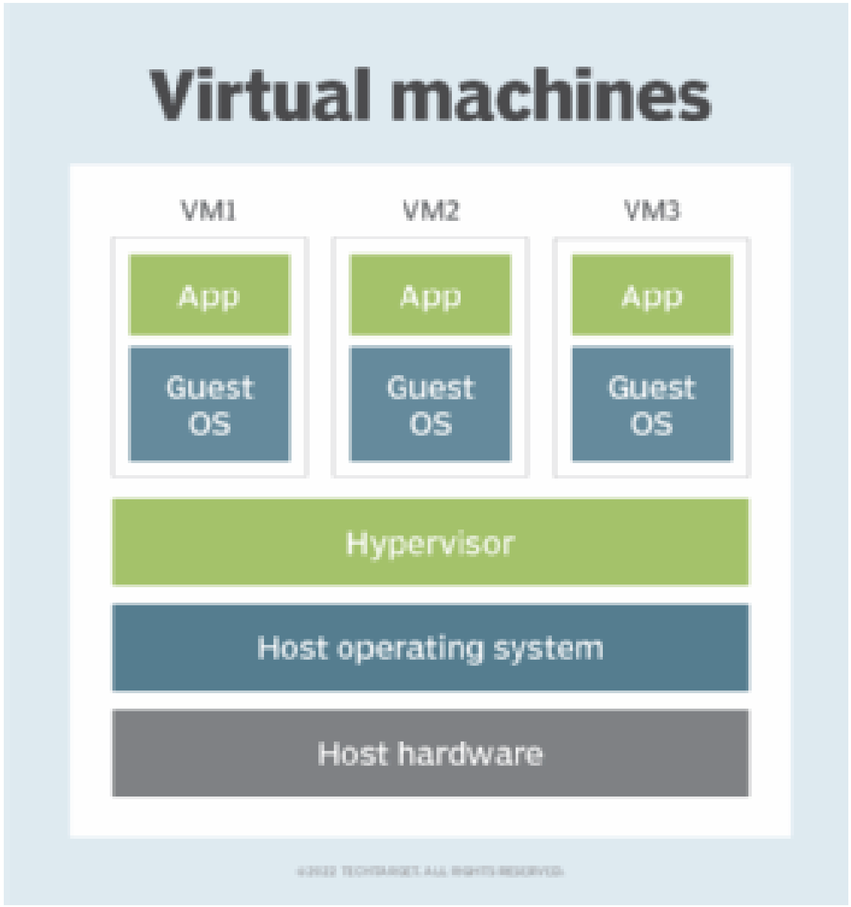
¿Cómo lo harían? (veamos opciones y sus ventajas)

- Cron para chequear los errores **Batch o lote** *Reactivo* **Eventos** *Proactivo*
- Multicloud, replica **HA**
- Computo **Server** **Serverless**
  1 instancia **1000 instancia**

GRUPO CORPORIS
Capacitación | Consultoría | OTEC
Valor centrado en el desarrollo de las personas

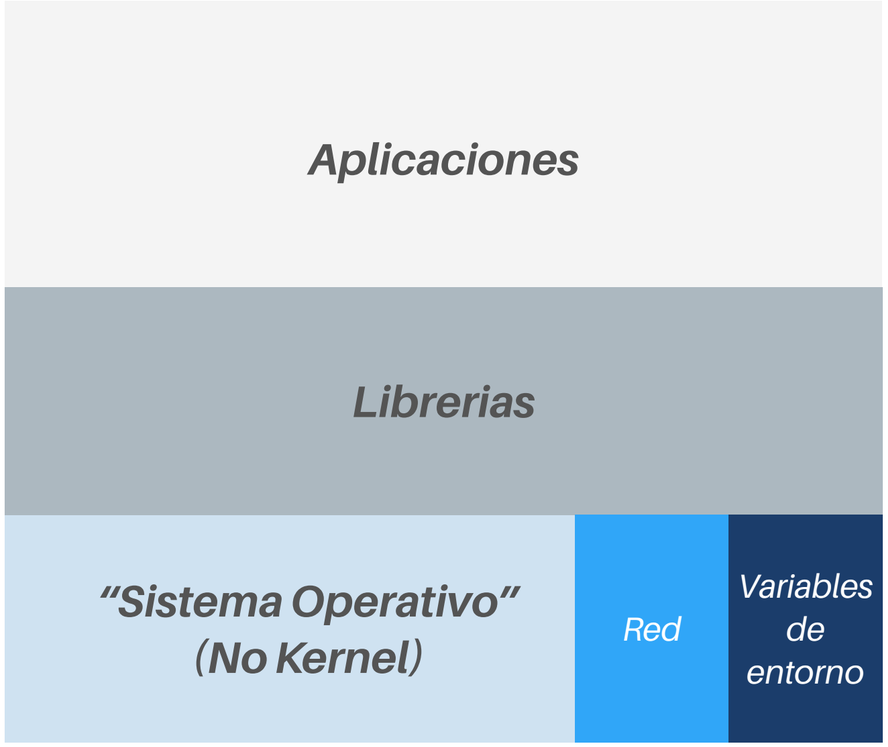
Contenedores





¿Qué son los contenedores?

Unidad funcional de software



No son:

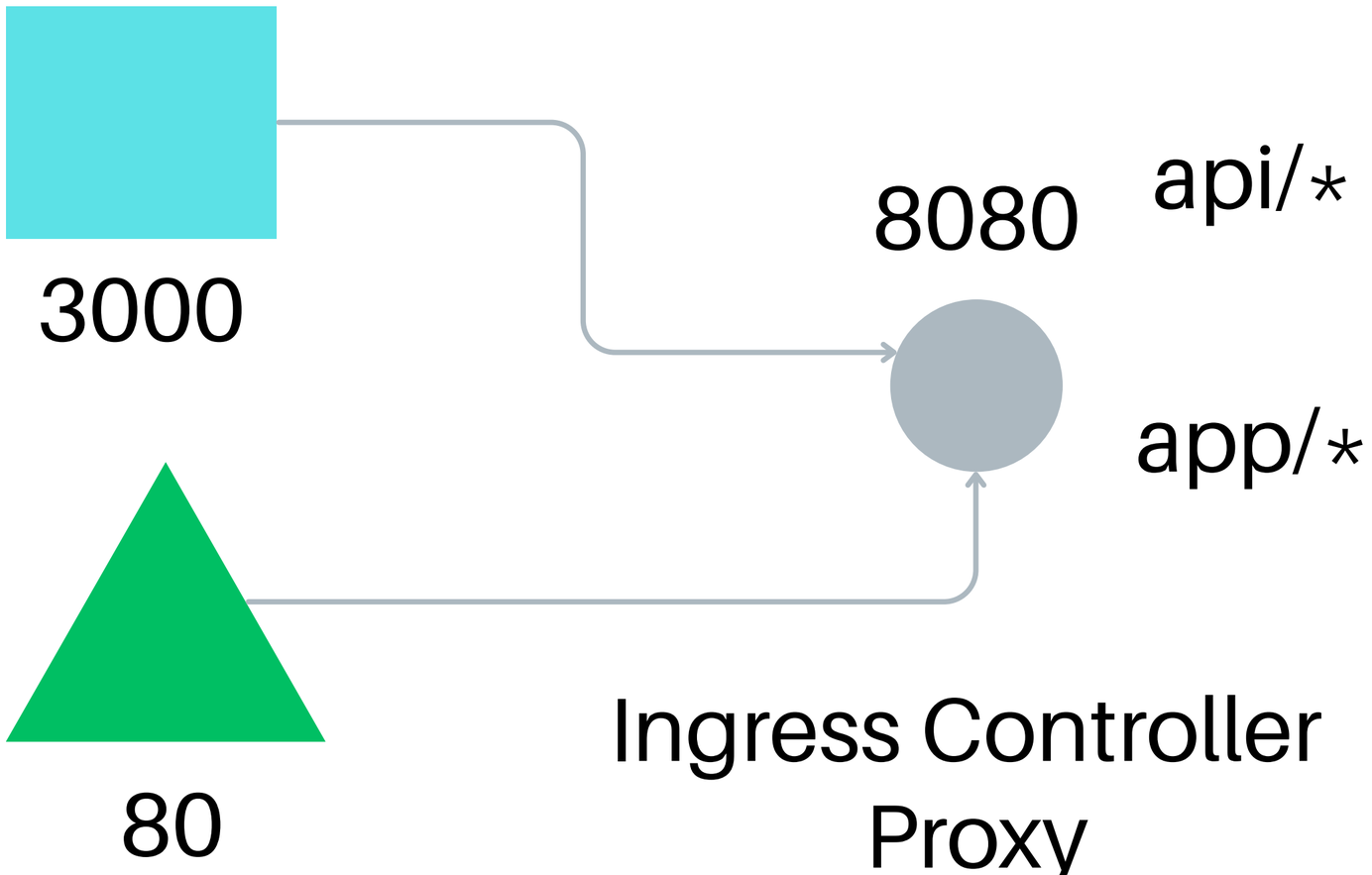
- Máquinas Virtuales
- Sistemas Operativos

¿Para que sirven?

¿Qué son los contenedores?

¿Para que sirven?

- Microservicio (descoplamiento: librerias, dependencias, red)
- Desarrollo (mismo ambiente) Debuguear



¿Qué son los contenedores?

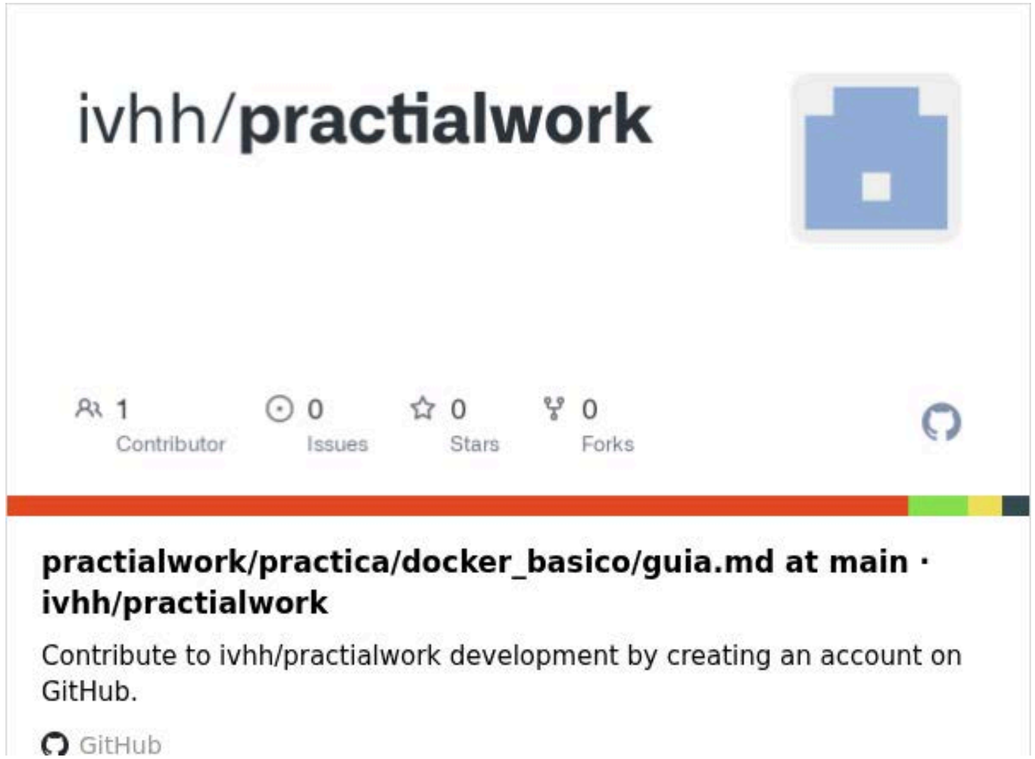
Característica	Contenedor	Máquina Virtual
Kernel	Comparte el kernel del host	Tiene su propio kernel
Peso	Ligero (MBs)	Pesado (GBs)
Velocidad de inicio	Segundos	Minutos
Aislamiento	Aislamiento de procesos	Aislamiento completo (hardware sim)
Uso de recursos	Más eficiente	Más demandante

¿Qué es Docker?

- Herramienta que permite crear contenedore
- Funciona nativamente en Linux*
- Se puede usar en cualquier SO

¿Cómo instalar Docker?

Creando mi primer contenedor

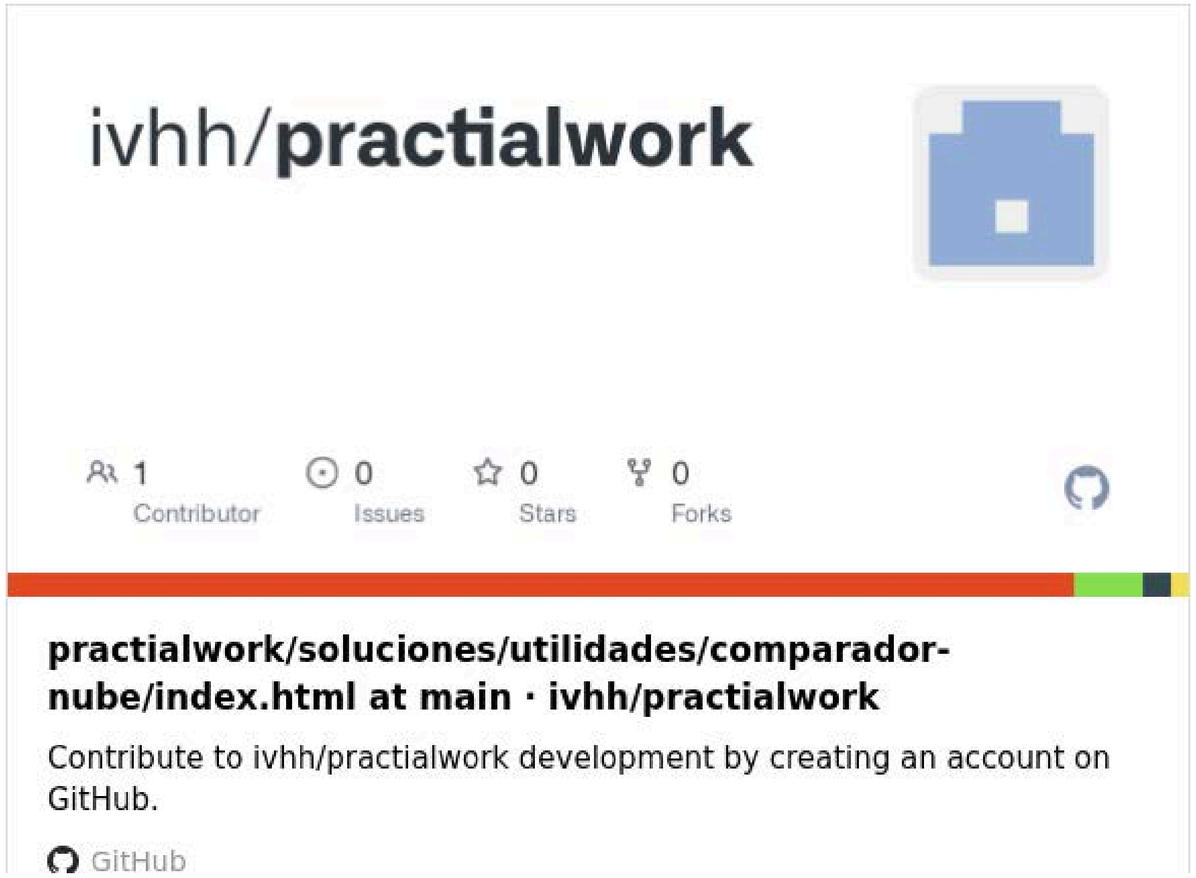


¿Por qué usar contenedores?

- Portabilidad y consistencia
- Desacoplamiento
- Aislamiento ligero
- Escalabilidad y replicabilidad
- Ecosistema maduro

¿Tiene alguna desventaja?

Servicios en la nube



Desafio práctico

Group

Scotiabank

GRUPO CORPORIS
Capacitación | Consultoría | OTEC

Valor centrado en el desarrollo de las personas

Mini-lab 1: Crear una máquina virtual

Mini-lab 2: Base de datos gestionada

Group

Scotiabank

GRUPO CORPORIS
Capacitación | Consultoría | OTEC

Valor centrado en el desarrollo de las personas

Mini-lab 3: Serverless function

Mini-lab 4: Desplegar contenedor