



UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

Introducción y Tipos de Arquitecturas

INF-343 Sistemas Distribuidos

Prof. Jorge Díaz M.

March 12, 2024





UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

Table of Contents

1 Introducción

- Introducción
- Arquitecturas
- Caso de Estudio



1 Introducción

Section 1.1

Definición y Ejemplos



¿Qué es un Sistema Distribuido?

1 Introducción

Tanenbaum et al. 2017

Conjunto de elementos de computación autónomos que se muestran al usuario como un sistema único y coherente.



¿Qué es un Sistema Distribuido?

1 Introducción

Tanenbaum et al. 2017

Conjunto de elementos de computación autónomos que se muestran al usuario como un sistema único y coherente.

Coulouris et al. 2011

Sistema en el cual componentes localizados en redes de computadores se comunican y coordinan sus acciones sólo por paso de mensajes.



¿Qué es un Sistema Distribuido?

Netflix

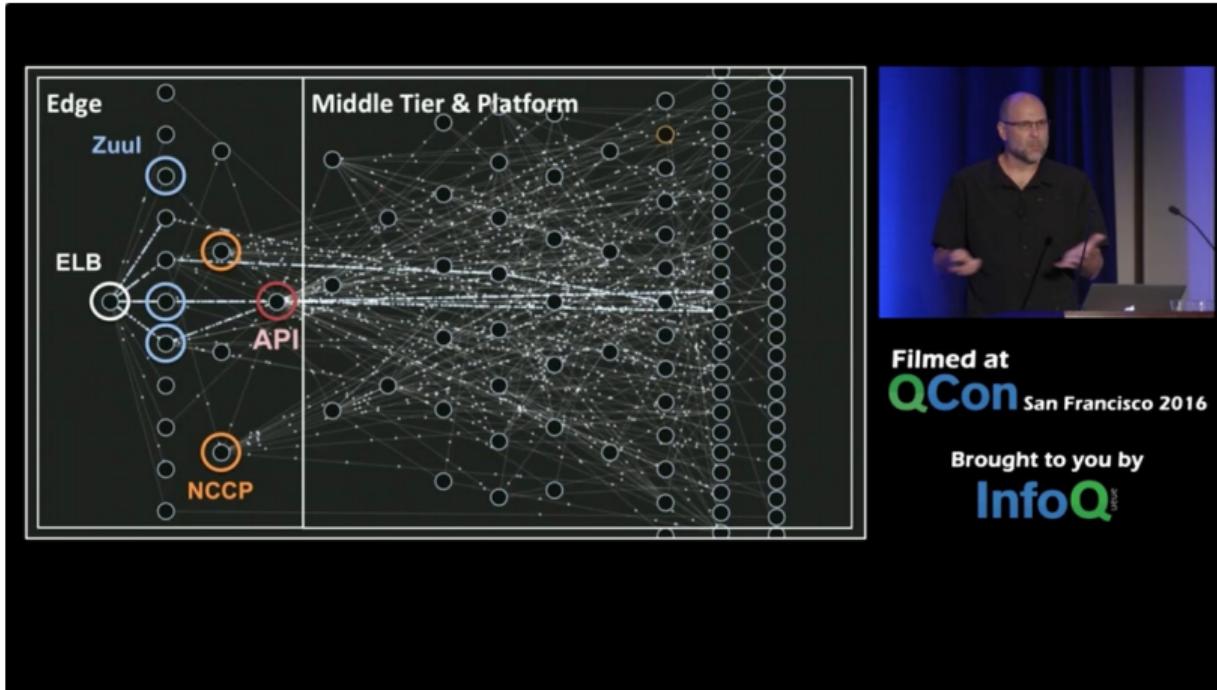


Figure: Source: Mastering Chaos - A Netflix Guide to Microservices



¿Qué es un Sistema Distribuido?

Bases de Datos





¿Qué es un Sistema Distribuido?

Vídeojuegos



¿Por qué es importante estudiar Sistemas Distribuidos?

1 Introducción

- Una gran cantidad de infraestructura crítica se basa en Sistemas Distribuidos
- Por lo general, usamos SD para:
 1. Obtener un alto rendimiento
 2. Tolerar fallas
 3. Para resolver problemas que están naturalmente separados físicamente
 4. Objetivos de seguridad



1 Introducción

Section 1.2

Objetivos de Diseño

Objetivos de Diseño

Rendimiento

Rendimiento → Escalabilidad

- Escenario perfecto: $2 \cdot PC \rightarrow 2 \cdot \text{rendimiento}$

Objetivos de Diseño

Rendimiento

Rendimiento → Escalabilidad

- Escenario perfecto: $2 \cdot PC \rightarrow 2 \cdot \text{rendimiento}$
- Escalabilidad en términos de...
 $\# \text{ Computadoras} \rightarrow \# \text{ Consultas (tráfico)} \rightarrow \# \text{Objetos (contenido)} \rightarrow \# \text{eventos de cálculo}$

Objetivos de Diseño

Rendimiento

Rendimiento → Escalabilidad

- Escenario perfecto: $2 \cdot PC \rightarrow 2 \cdot \text{rendimiento}$
- Escalabilidad en términos de...
 $\# \text{ Computadoras} \rightarrow \# \text{ Consultas (tráfico)} \rightarrow \# \text{Objetos (contenido)} \rightarrow \# \text{eventos de cálculo}$

Escalabilidad

Tener recursos disponibles para manejar una cantidad creciente de trabajo, esperado o inesperado.



Objetivos de Diseño

Tolerancia a Fallas

Tolerancia a fallos

Un sistema puede proporcionar sus servicios incluso en presencia de fallos.

Disponibilidad

Proporción de tiempo que el sistema está disponible para su uso.

Objetivos de Diseño

Tolerancia a Fallas

Manejo de Errores

- El error es la regla y no la excepción en SD
- Fallos parciales
- ¿Cómo lidiar con los fallos?
 - Detectar fallos y reparar o recuperar
 - Enmascarar los errores
 - Tolerar errores

Objetivos de Diseño

Seguridad

En SD los datos no son locales, están expuestos a amenazas.

- **CONFIDENCIALIDAD:** Nadie fuera de la comunicación puede ‘ver’ el contenido de un mensaje.
- **INTEGRIDAD:** Nadie fuera de la comunicación ha cambiado el contenido del mensaje.
- **SIN REPUDIO:** Nadie puede hacer una acción y luego decir que no lo hizo.

Objetivos de Diseño

Seguridad

En SD los datos no son locales, están expuestos a amenazas.

- **CONFIDENCIALIDAD:** Nadie fuera de la comunicación puede ‘ver’ el contenido de un mensaje.
- **INTEGRIDAD:** Nadie fuera de la comunicación ha cambiado el contenido del mensaje.
- **SIN REPUDIO:** Nadie puede hacer una acción y luego decir que no lo hizo.

Soluciones:

- Uso de técnicas criptográficas.
- Autenticación.



1 Introducción

Section 1.3

Desafíos de Diseño

Desafíos de Diseño

Concurrencia y Replicación

Enfoques para lograr disponibilidad, tolerancia a fallos y rendimiento:

- **REPLICACIÓN:** creación de nodos/objetos/cálculos redundantes en el sistema.
- **CONCURRENCIA:** dos o más eventos pueden ocurrir al mismo tiempo.
 - Uno a la vez → limita el rendimiento
 - Concurrencia → lidiar con conflictos
 1. Sincronización
 2. Consistencia



Desafíos de Diseño

Concurrencia y Replicación

Ejemplo de consistencia: Almacenamiento *Key-value*, donde tenemos las siguientes operaciones:

- `put (key, value)`: Agregar un elemento (`value`), asociado a una variable (`key`).
- `get(key) → value`: A partir de una variable (`key`), obtener su respectivo resultado (`value`).

Caso:

- Modificación del cliente 1: `put (A, 1)`.
- Modificación del cliente 2: `put (A, 2)`.
- **Coordinación requerida.**
- Se produce un error en el 2º servidor de réplica antes de que se produzca la actualización.



Desafíos de Diseño

Concurrencia y Replicación

Formas de tratar este problema:

- Consistencia fuerte
- Consistencia eventual



Desafíos de Diseño

Coordinación

Coordinación

Algoritmos para lograr consenso entre los participantes.

- Esto requiere comunicación → intercambio de mensajes.



Desafíos de Diseño

Coordinación

Coordinación

Algoritmos para lograr consenso entre los participantes.

- Esto requiere comunicación → intercambio de mensajes.

¿Cómo encontrar otros nodos/recursos para comunicarse con ellos? **NOMBRAMIENTO (NAMING)**



Desafíos de Diseño

Heterogeneidad y Transparencia

- **Heterogeneidad:** HW, OS, Network, Programas, Lenguajes.
- **Transparencia:** Parece que no es un sistema distribuido de almacenamiento o computación.

Desafíos de Diseño

Heterogeneidad y Transparencia

Middleware

Abstracción para enmascarar complejidad o heterogeneidad.

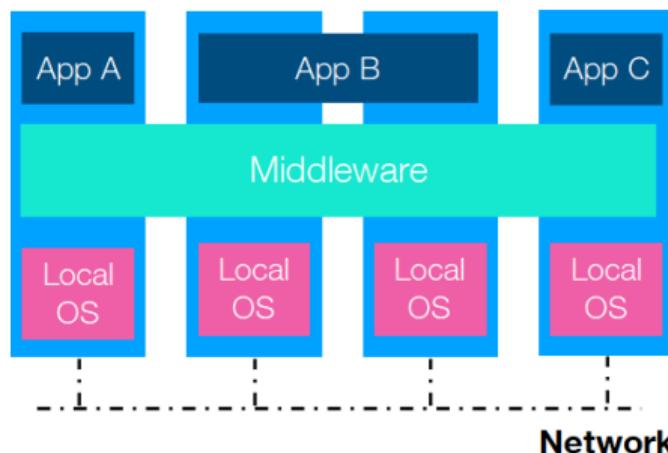


Figure: *Percibir el sistema como un todo en lugar de una colección de componentes.*



UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

Table of Contents

2 Arquitecturas

- Introducción
- Arquitecturas
- Caso de Estudio



Modelos arquitectónicos

2 Arquitecturas

Arquitectura

Estructura en términos de componentes especificados por separado y sus relaciones.

Modelos arquitectónicos

2 Arquitecturas

Arquitectura

Estructura en términos de componentes especificados por separado y sus relaciones.

A) Arquitectura de software

- En capas
- Basadas en objetos
- Centradas en recursos
- Basadas en eventos

B) Arquitectura del sistema

- Organizaciones centralizadas
- Organizaciones descentralizadas
- Arquitecturas híbridas

C) Modelos fundamentales

- Modelo de interacción
- Modelo de fallos
- Modelo de seguridad



2 Arquitecturas

Section 2.1

Arquitectura de Software

Arquitectura de Software

2 Arquitecturas

Patrones

Los patrones de diseño permiten a los arquitectos de software reutilizar diseños probados.

Frameworks

Permite a los desarrolladores de software reutilizar el código de trabajos previos:

- Esqueleto del programa que resuelve problemas relacionados.
- Implementa un patrón de diseño o una combinación.

Arquitectura de Software

2 Arquitecturas

Patrones

Los patrones de diseño permiten a los arquitectos de software reutilizar diseños probados.

Frameworks

Permite a los desarrolladores de software reutilizar el código de trabajos previos:

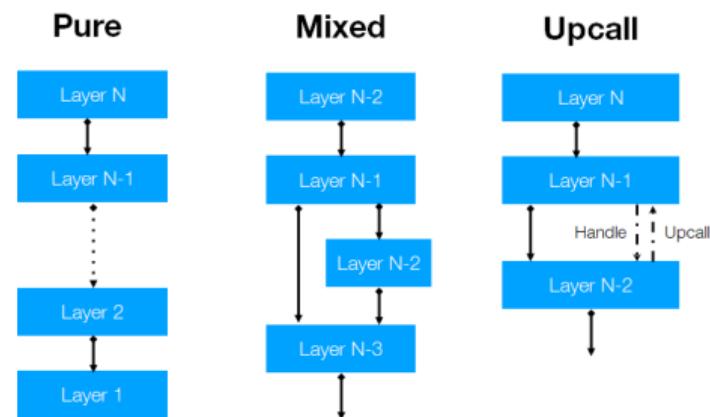
- Esqueleto del programa que resuelve problemas relacionados.
- Implementa un patrón de diseño o una combinación.

Los patrones, los *frameworks* y los *middleware* desempeñan un trabajo complementario.

Arquitecturas en Capas

2 Arquitecturas

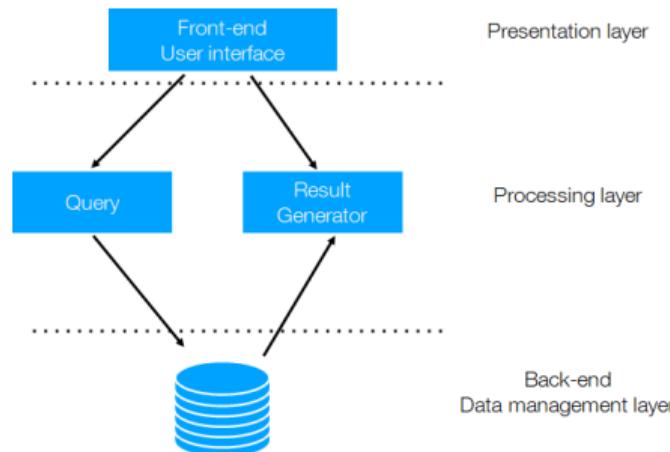
- Un sistema complejo se divide en varias capas. Una capa ofrece una abstracción SW.
- Una capa hace uso de los servicios ofrecidos por la capa de abajo.
- El componente en la capa puede realizar una llamada descendente a un componente en una capa de nivel inferior.



Arquitecturas en Capas

2 Arquitecturas

Ejemplo de capas de una aplicación:

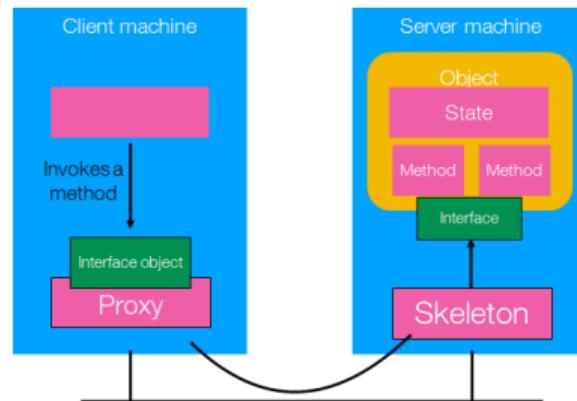


- Nivel de interfaz de aplicación: Controla la interacción con un usuario o una aplicación externa.
- Nivel de procesamiento: Contiene la funcionalidad principal de la aplicación.
- Nivel de datos: Opera en una base de datos o sistema de archivos.

Arquitecturas basadas en objetos

2 Arquitecturas

- Cada objeto corresponde a un componente, independiente de su entorno.
- Los componentes se conectan a través de un mecanismo de **llamada a procedimiento (RPC)**.
- Concentración de datos → estado del objeto.
- Encapsulamiento de operaciones → Métodos del objeto



Common organization of a
remote object

Arquitecturas basadas en recursos

2 Arquitecturas

- Un sistema distribuido se ve como una gran colección de recursos que son administrados individualmente por componentes.
- Las aplicaciones pueden agregar o eliminar recursos.
- Transferencia de Estado Representacional (REST):
 1. Los recursos se identifican con un esquema de nomenclatura único.
 2. Todos los servicios ofrecen la misma interfaz
 3. Los mensajes enviados hacia y desde un servicio se describen completamente a sí mismos.
 4. Después de ejecutar una operación, los componentes se olvidan de todo lo relacionado con la llamada → ejecución sin estado.

Arquitecturas basadas en recursos

Simple Storage Service (Amazon S3)

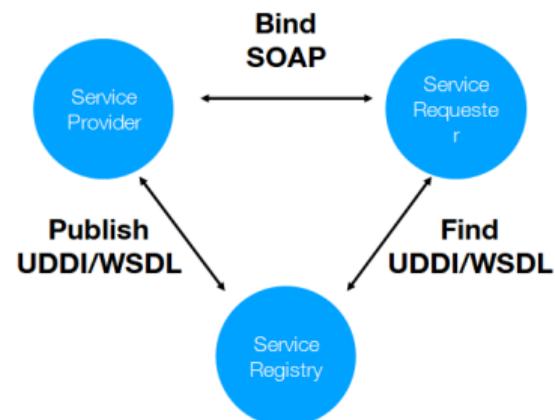
- Soporta objetos y buckets
- Un objeto (*ObjectName*) contenido en un bucket (*BucketName*) es referido por URI (Uniform Resource Identifier). Ejemplo:
`http://BucketName.s3.amazonaws.com/ObjectName`

| Operation | Description |
|-----------|--|
| POST | Create new resource |
| GET | Retrieve the state of a resource |
| DELETE | Delete a resource |
| PUT | Modify a resource by transferring a new state. |

Arquitecturas orientadas a servicios (SOA)

2 Arquitecturas

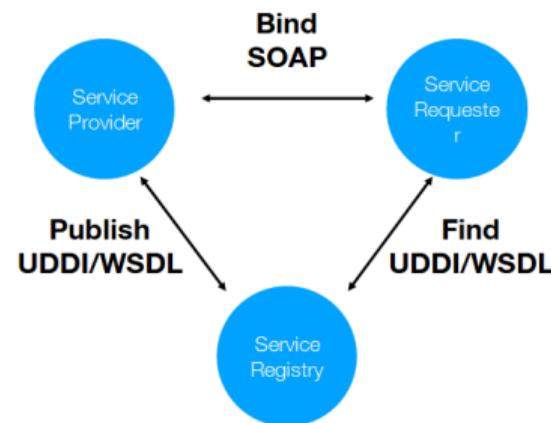
- Un sistema se construye como una composición de muchos servicios diferentes.
- Es posible que no todos los servicios pertenezcan a la misma organización.
- Service Provider: Implementación y ejecución de los servicios.
- Service Registry: Instancia donde un servicio notifica que existe.
- Service Requester/User: Quien usa los servicios.



Arquitecturas orientadas a servicios (SOA)

2 Arquitecturas

- UDDI: *Universal Description, Discovery and Integration.* Una forma de publicar y buscar información de servicios web.
- SOAP: Protocolo simple de acceso a objetos.
- XML: utilizado para la codificación de datos.
- WSDL: es un lenguaje XML utilizado para definir servicios Web y describir cómo tener acceso a ellos.



Arquitectura de microservicios

2 Arquitecturas

- Arquitectura emergente para el desarrollo distribuido de aplicaciones en la nube.
- Una sola aplicación grande (Monolito) se desacopla en un conjunto de aplicaciones muy pequeñas (microservicios).
- Cada componente del servicio se implementa como una unidad independiente.
- Todos los componentes están completamente desacoplados.

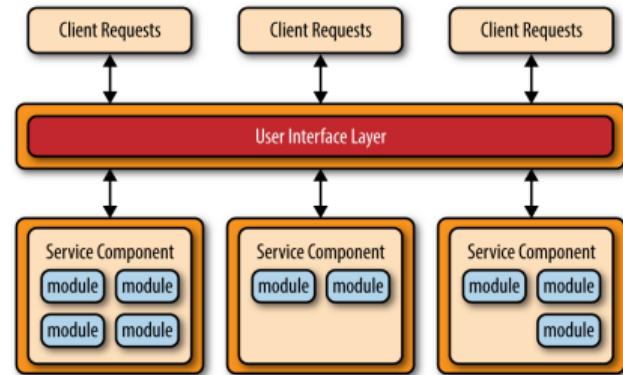


Figure: Source: Software Architecture Patterns (O'Reilly)

Arquitectura de microservicios

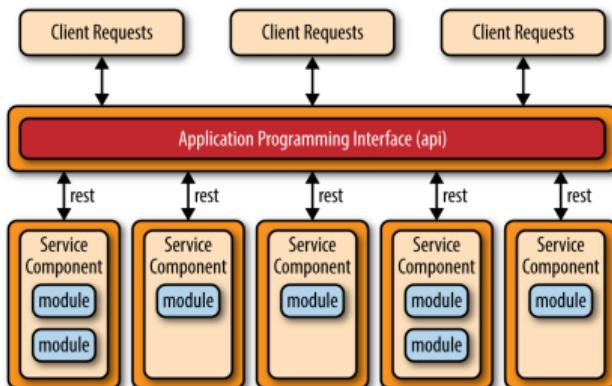
Ejemplos

- Cada servicio se puede implementar en su propio entorno: VM, contenedor.
- API basada en REST: servicios web RESTful basados en la nube encontrados por Yahoo!, Google y Amazon.
- Los microservicios se pueden descomponer por recursos, funciones o casos de uso.

Arquitectura de microservicios

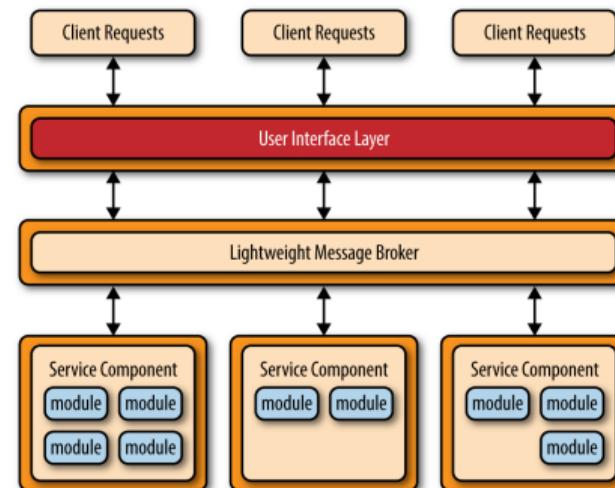
Ejemplos

El cliente se comunica con el sistema mediante una API basada en REST.



Source:
Software Architecture Patterns (O'Reilly)

El cliente se comunica indirectamente con el sistema mediante una capa de mensajes.



Source:
Software Architecture Patterns (O'Reilly)

Arquitectura de microservicios

Contenedores

Containers vs. VMs

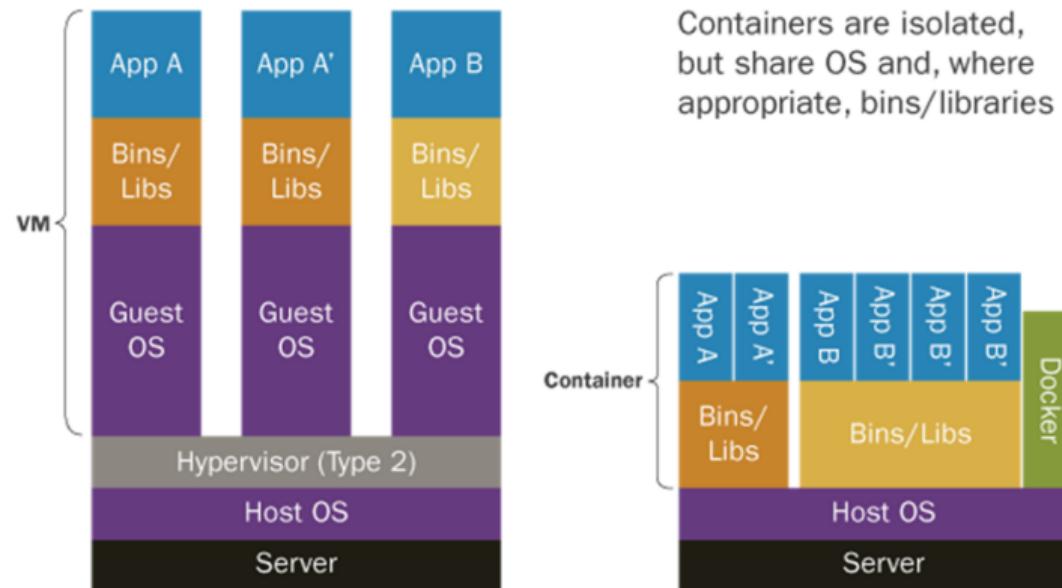
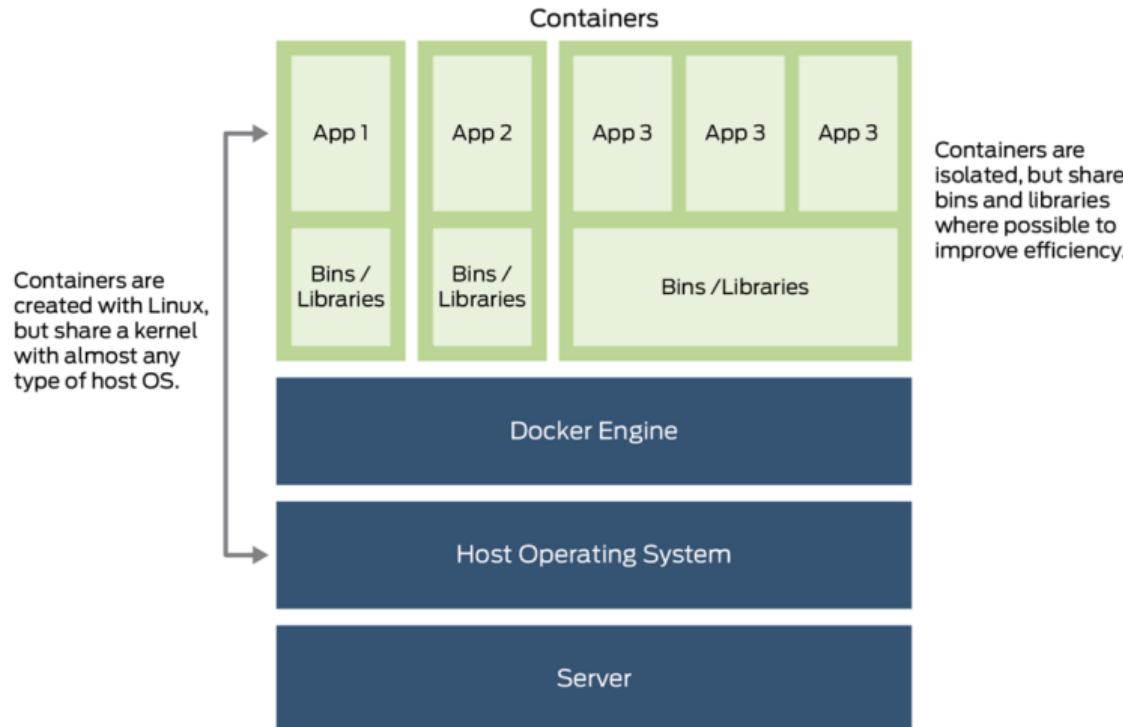


Figure: Source: Containerization and how it differs from Virtual Machines - Scott Thornton

Arquitectura de microservicios

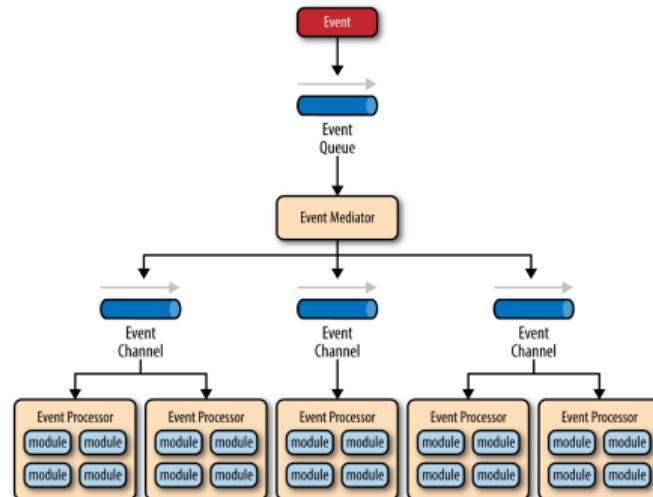
Contenedores: Docker



Arquitecturas basadas en eventos

Mediator Topology

- Dependencias sueltas, fuerte separación entre procesamiento y coordinación.
- Popular para sistemas distribuidos asincrónicos.
- Topología del mediador: entre docenas y varios cientos de colas de eventos.

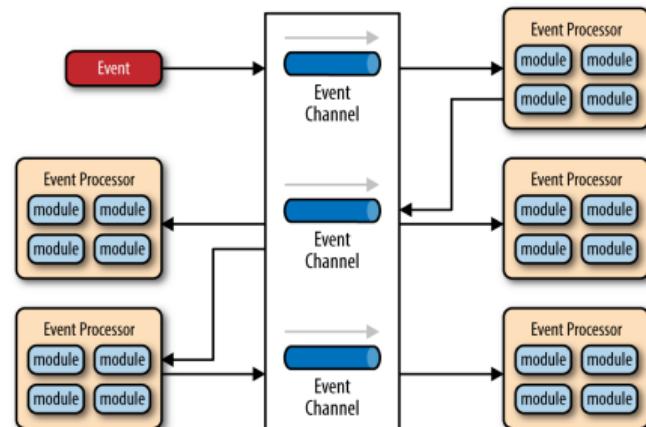


Source: Software Architecture Patterns
(O'Reilly)

Arquitecturas basadas en eventos

Broker Topology

- Sin mediador central, el flujo de mensajes se distribuye a través del componente del procesador de eventos en cadena a través de un agente de mensajes ligero (por ejemplo, ActiveMQ).
- Broker: un componente que controla todo el acceso entre diferentes aplicaciones.

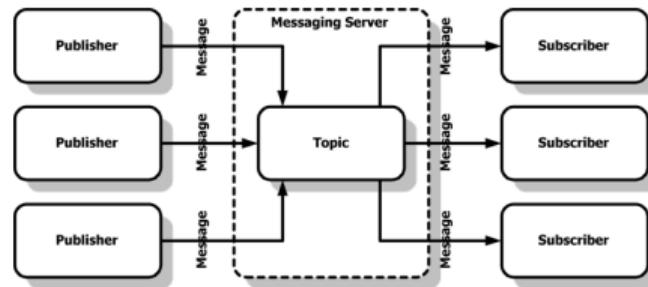


Source: Software Architecture Patterns
(O'Reilly)

Arquitecturas basadas en eventos

Publish-subscribe Pattern

- Arquitectura de publicación-suscripción.
- Un proceso puede publicar una notificación que describa la ocurrencia de un evento.
- El proceso puede suscribirse a un tipo específico de notificación.



Source: Software Architecture Patterns
(O'Reilly)



2 Arquitecturas

Section 2.2

Arquitectura del Sistema



Arquitectura del Sistema

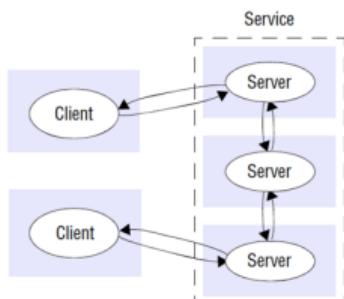
Organización Centralizada

- Arquitectura cliente-servidor simple
- Arquitecturas multinivel

Organización Centralizada

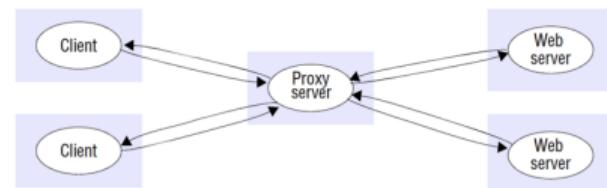
Cliente-Servidor simple

Asignación de servicios a múltiples servidores: Partición o réplicas.



Caching

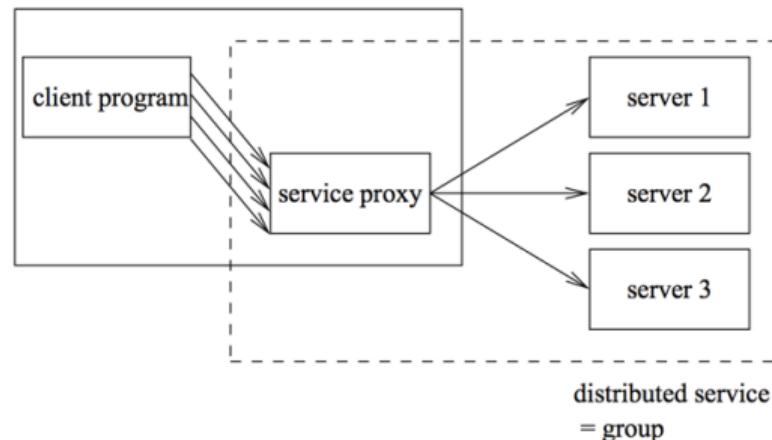
- Almacén de datos utilizados recientemente, que está más cerca de los clientes.
- Puede estar ubicado con el cliente o en un servidor proxy.



Organización Centralizada

Cliente-Servidor

- El **proxy** representa todo el conjunto de servidores.
- El cliente dirige toda su comunicación al proxy.
- Propiedades:
 - Encapsulación
 - Localidad
 - Protocolo de acceso
 - Capacidad
 - Sustituto (*Stub*)
 - Comunicación de confianza

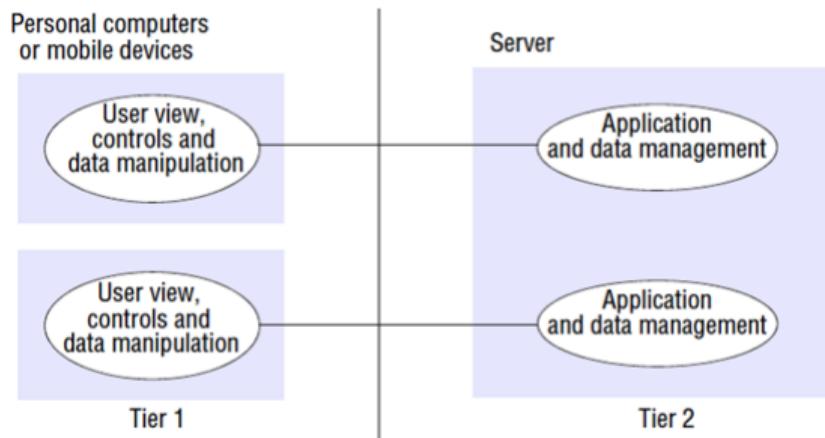


Source: Structured and Encapsulation in Distributed Systems: The proxy Principle, Marc Shapiro, In Proc 6th International Conf. On Distributed Computing Systems (ICDCS), Cambridge USA, 1986

Organización Centralizada

Arquitectura multinivel

La organización en niveles es una técnica para organizar la funcionalidad de una capa determinada y colocar esta funcionalidad en servidores, nodos físicos.

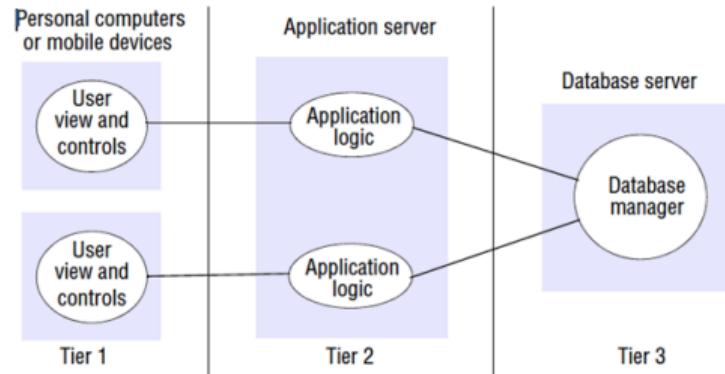


Organización Centralizada

Arquitectura multinivel

Ejemplo de arquitectura de tres niveles:

- Lógica de presentación: manejo de la interacción del usuario y actualización de la vista.
- Lógica de aplicación: manejo del procesamiento específico de la aplicación.
- Lógica de los datos: almacenamiento persistente de la aplicación.



Arquitectura del Sistema

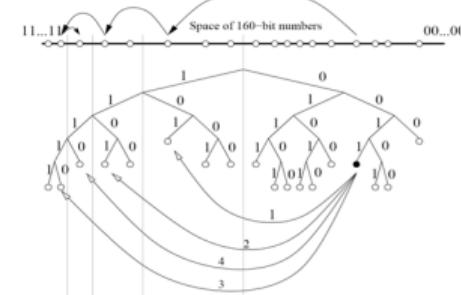
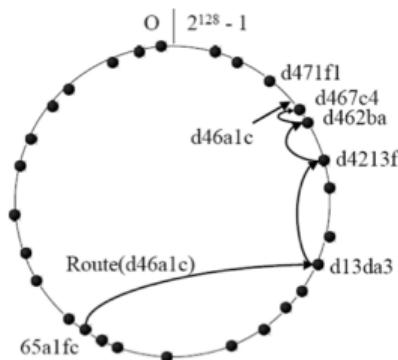
Organización Descentralizada

- Sistemas estructurados peer-to-peer
- Sistemas peer-to-peer no estructurados
- Redes peer-to-peer organizadas jerárquicamente

Organización Descentralizada

Sistemas estructurados peer to peer

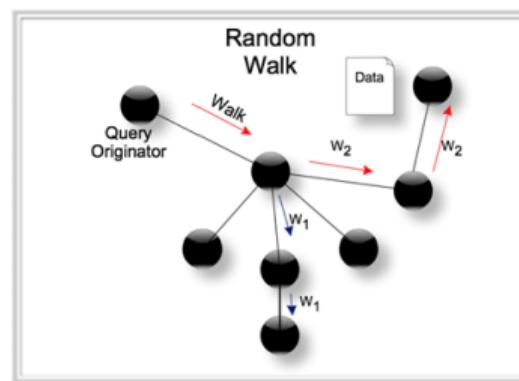
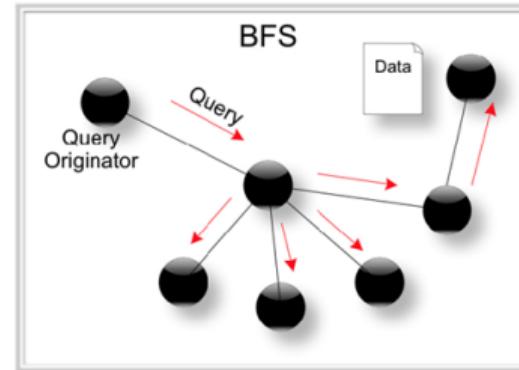
Los nodos de este tipo de sistema están organizados en estructuras conocidas, por ejemplo: arboles binarios, anillos, cuadrillas, etc.



Organización Descentralizada

Sistemas no estructurados peer to peer

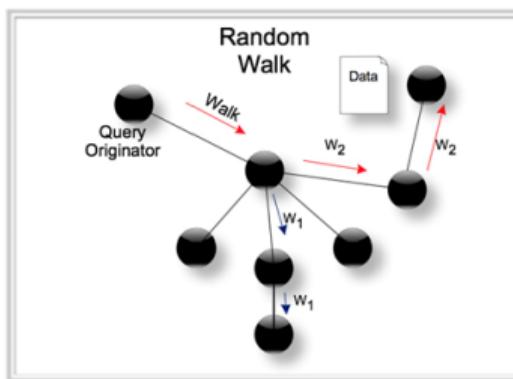
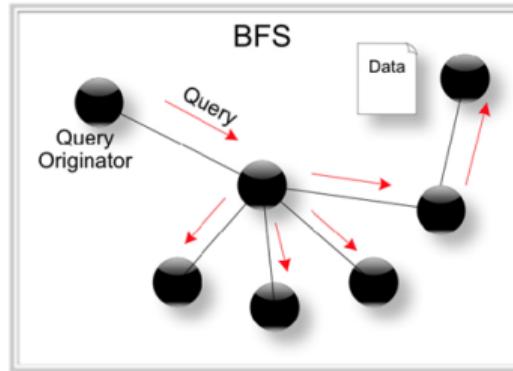
- Topología sin restricciones
- El almacenamiento de datos no tiene correlación con la topología
- Inundación controlada/Estilo de transmisión:
 - Búsqueda a lo ancho (*BFS*)
 - Caminata aleatoria (*Random Walk*)
 - Profundización iterativa



Organización Descentralizada

Sistemas no estructurados peer to peer

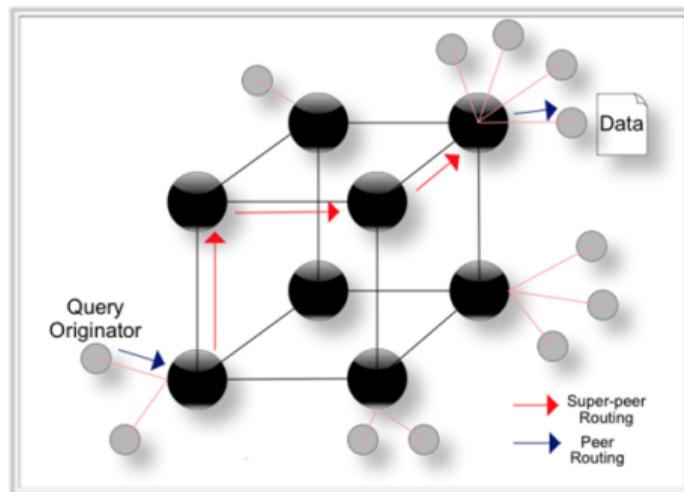
- No escalable cuando el número de consultas es alto.
- Ineficiente con elementos menos populares, la disponibilidad y persistencia de los datos no están garantizadas.



Organización Descentralizada

Sistemas jerárquicos peer to peer

Súper compañeros (*Super peers*): nodos que actúan como intermediarios (recopila datos sobre el uso de recursos y la disponibilidad para los nodos en las proximidades).





UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

Arquitectura del Sistema

Arquitecturas Híbridas

- Sistemas Edge-server

Arquitecturas Híbridas

Sistemas Edge-server

- Los servidores se colocan en el borde de la red.
- Los usuarios finales se conectan a Internet por medio de un servidor perimetral.
- El propósito principal del servidor perimetral es servir contenido.

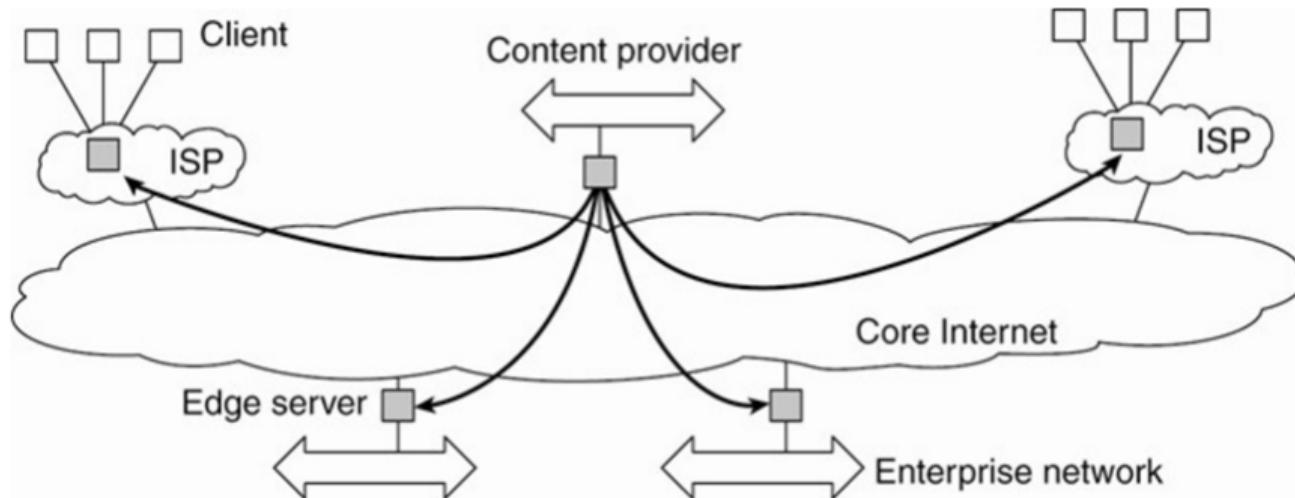
Ejemplos:

- Programa Netflix Open Connect.
- Propuesta de caché de motor de búsqueda web.



Arquitecturas Híbridas

Sistemas Edge-server





2 Arquitecturas

Section 2.3

Modelos fundamentales



Modelos Fundamentales

2 Arquitecturas

Contienen los ingredientes esenciales para entender las razones sobre algunos aspectos del comportamiento de un sistema.

- Hacer explícitos todos los supuestos relevantes.
- Hacer generalizaciones sobre lo que es posible dados los supuestos.
- Probar las propiedades del sistema utilizando técnicas matemáticas.

Modelos Fundamentales

2 Arquitecturas

Contienen los ingredientes esenciales para entender las razones sobre algunos aspectos del comportamiento de un sistema.

- Hacer explícitos todos los supuestos relevantes.
- Hacer generalizaciones sobre lo que es posible dados los supuestos.
- Probar las propiedades del sistema utilizando técnicas matemáticas.

Estudiaremos 3 tipos:

- Interacción: Comunicación (flujo de información) y coordinación (actividades de sincronización y ordenación) entre procesos.
- Errores: Definición y clasificación de fallos en los dispositivos o red.
- Seguridad: Definición y clasificación de las formas de ataques de agentes externos e internos.

Modelo de Interacción

2 Arquitecturas

- El comportamiento y el estado de un sistema distribuido se pueden describir mediante un algoritmo distribuido.
- Una definición de los pasos a seguir por cada uno de los procesos de los que se compone el sistema.

Modelo de Interacción

2 Arquitecturas

- El comportamiento y el estado de un sistema distribuido se pueden describir mediante un algoritmo distribuido.
- Una definición de los pasos a seguir por cada uno de los procesos de los que se compone el sistema.

Factores que afectan la interacción:

- El rendimiento de la comunicación es una característica limitante.
- Imposibilidad de mantener un único Estado global.
- Rendimiento de los canales de comunicación.

Modelo de Interacción

Rendimiento

- Retraso (Delay): entre el inicio de la transmisión de un mensaje y las recepciones → latencia.
- Ancho de Banda (Bandwidth): Cantidad total de información que se puede transmitir a través de ella en un tiempo determinado.
- Temblor (Jitter): es la variación en el tiempo necesario para entregar una serie de mensajes.

Modelo de Interacción

Reloj de computadora y eventos de tiempo

- Cada dispositivo tiene su propio reloj interno, utilizado localmente.
- La lectura del reloj de dos dispositivos diferentes puede proporcionar valores diferentes.
- Porque el reloj se aleja de la hora perfecta.
- Velocidad de deriva del reloj: la velocidad a la que el reloj de un dispositivo se desvía de un reloj de referencia perfecto

Modelo de Interacción

Reloj de computadora y eventos de tiempo

- Cada dispositivo tiene su propio reloj interno, utilizado localmente.
- La lectura del reloj de dos dispositivos diferentes puede proporcionar valores diferentes.
- Porque el reloj se aleja de la hora perfecta.
- Velocidad de deriva del reloj: la velocidad a la que el reloj de un dispositivo se desvía de un reloj de referencia perfecto

¿Qué hacer?

- Obtenga tiempo del Sistema de Posicionamiento Global (precisión 1 ms)
- Costoso, no funciona en interiores.
- Enviar mensaje de temporización a otros equipos. → acuerdo, sincronización!



Modelo de Interacción

Sincronismo

Sistemas distribuidos síncronos:

- El tiempo para ejecutar cada paso de un proceso tiene límites inferiores y superiores conocidos.
- Cada mensaje transmitido a través de un canal se recibe dentro de un tiempo acotado conocido.
- Cada proceso tiene un reloj local cuya tasa de deriva desde el tiempo real se ha conocido.
- Los tiempos de espera (timeouts) son posibles de usar para detectar fallas.



Modelo de Interacción

Sincronismo

Sistemas distribuidos síncronos:

- El tiempo para ejecutar cada paso de un proceso tiene límites inferiores y superiores conocidos.
- Cada mensaje transmitido a través de un canal se recibe dentro de un tiempo acotado conocido.
- Cada proceso tiene un reloj local cuya tasa de deriva desde el tiempo real se ha conocido.
- Los tiempos de espera (timeouts) son posibles de usar para detectar fallas.

Sistemas distribuidos asincrónicos:

- Sin límites en la velocidad de ejecución del proceso.
- No hay límites en los retrasos en la transmisión de mensajes.
- Sin límites en las tasas de deriva.
- Dificultad para llegar a un acuerdo en un sistema distribuido asíncrono.



Modelo de Interacción

Orden

Orden de eventos:

- ¿El evento en un proceso ocurrió antes, después o simultáneamente con otro evento en otro proceso?
- El retraso independiente de los mensajes puede producir desorden en la entrega de msg.

Modelo de Interacción

Orden

Orden de eventos:

- ¿El evento en un proceso ocurrió antes, después o simultáneamente con otro evento en otro proceso?
- El retraso independiente de los mensajes puede producir desorden en la entrega de msg.

Solución → Lamport: Modelo de tiempo lógico para proporcionar orden entre eventos en procesos en diferentes computadoras.

- Mensaje recibido después de su envío.
- Orden para un par de eventos
- Respuestas enviadas después de recibir mensajes.
- Asigne un número a cada evento.

Modelo de Error

2 Arquitecturas

| Clase de fallo | Afecta | Descripción |
|-----------------------|-----------------|---|
| Fail-stop | Proceso | El proceso se detiene y permanece detenido. Otros procesos pueden detectar este estado. |
| Crash | Proceso | El proceso se detiene y permanece detenido. Es posible que otros procesos no puedan detectar este estado. |
| Omission | Canal | Un mensaje insertado en un búfer de mensajes salientes nunca llega al búfer de mensajes entrantes del otro extremo. |
| Send-omission | Procesos | Un proceso completa una operación de envío, pero el mensaje no se coloca en su búfer de mensajes saliente. |
| Receive-omission | Proceso | Un mensaje se coloca en el búfer de mensajes entrantes de un proceso, pero ese proceso no lo recibe. |
| Arbitrary (Byzantine) | Proceso o canal | El proceso/canal exhibe un comportamiento arbitrario: puede enviar/transmitir mensajes arbitrarios en momentos arbitrarios o cometer omisiones. Un proceso puede detenerse o dar un paso incorrecto. |

Modelo de Error

Errores de tiempo

| Clase de fallo | Afecta | Descripción |
|----------------|---------|---|
| Clock | Proceso | El reloj local del proceso excede los límites de su velocidad de deriva (drift) desde el tiempo real. |
| Performance | Proceso | El proceso excede el límite en el intervalo entre dos pasos. |
| Performance | Canal | La transmisión de un mensaje tarda más que el límite indicado. |



Modelo de Seguridad

2 Arquitecturas

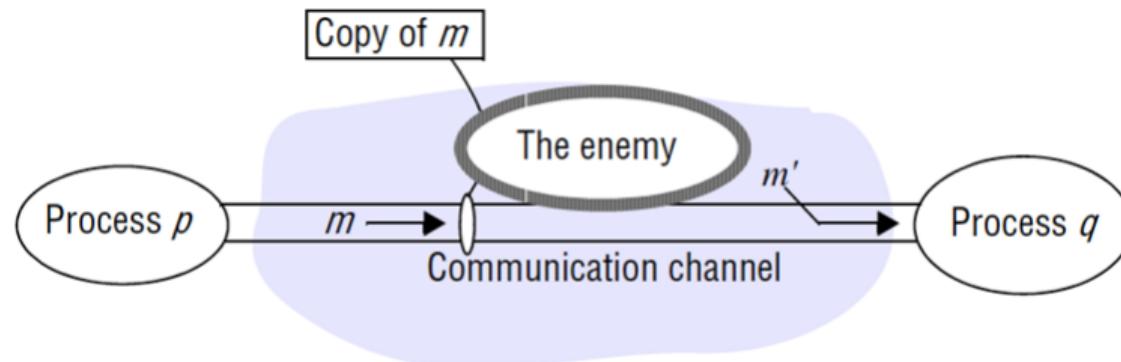
La seguridad de un sistema distribuido se puede lograr asegurando los procesos y los canales utilizados para sus interacciones y protegiendo los objetos que encapsulan contra el acceso no autorizado.

Modelo de Seguridad

2 Arquitecturas

Protección de objetos, procesos e interacciones:

- **Los derechos de acceso** especifican quién puede realizar las operaciones de un objeto.
- La integridad se ve amenazada por violaciones de seguridad.





Modelo de Seguridad

Ataques

El análisis implica la construcción de un modelo de amenazas que enumera todas las formas de ataques a las que está expuesto el sistema y una evaluación de los riesgos y consecuencias de cada uno.

| Afecta | Ataques |
|-------------------------|---|
| Procesos | Invocación con identidad falsa. |
| Canales de comunicación | Copiar, alterar o inyectar mensajes. |
| Canales de comunicación | Guardar y reproducir copias de un mensaje. |
| Procesos | Ataque de denegación de servicio: invocaciones excesivas e inútiles sobre los servicios. |
| Procesos | Rol de caballo de Troya: código que accede o modifica recursos. |

Modelo de Seguridad

Medidas de defensa

- Criptografía y secretos compartidos: La criptografía es la ciencia de mantener el mensaje seguro y el cifrado es el proceso de codificar un mensaje de tal manera que oculte su contenido. La criptografía se basa en algoritmos de cifrado que utilizan claves secretas (grandes números difíciles de adivinar).
- Autenticación: El uso de secretos compartidos y el cifrado proporciona la base para la autenticación de mensajes, proporcionando las identidades proporcionadas por sus remitentes. Conceptos básicos: incluir en un mensaje una porción cifrada que garantice su autenticidad.

Modelo de Seguridad

Medidas de defensa

- Canales seguros: El cifrado y la autenticación se utilizan para crear canales seguros como una capa de servicio sobre los servicios de comunicación existentes. Un canal seguro es un canal de comunicación que conecta un par de procesos, cada uno de los cuales actúa en nombre de un principal.



UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

Table of Contents

3 Caso de Estudio

- ▶ Introducción
- ▶ Arquitecturas
- ▶ Caso de Estudio



3 Caso de Estudio

Section 3.1

Escalar de cero a un millón de usuarios

Escalar de cero a un millón de usuarios

3 Caso de Estudio

Configuración inicial:

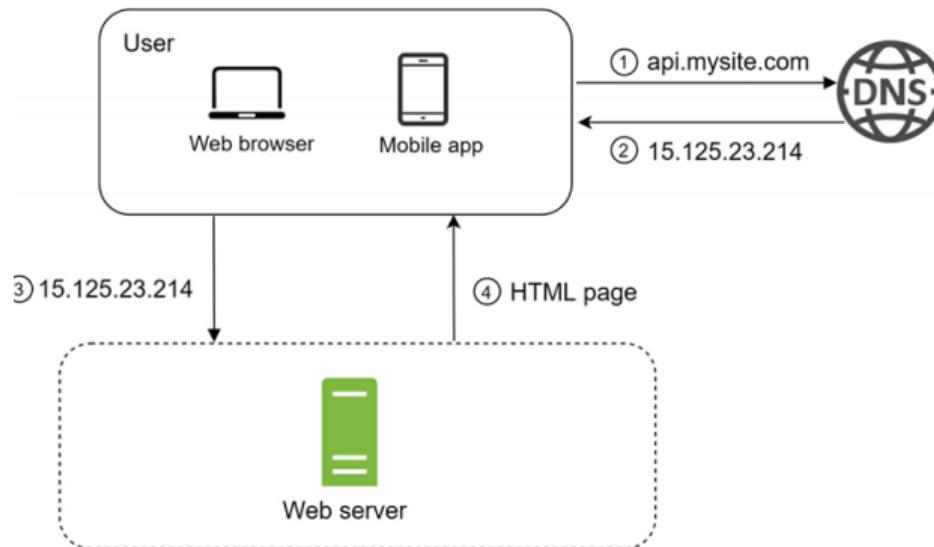


Figure: Source: System Design Interview, An insider's guide, Alex Xu, Second Edition.
Editor: Paul Solomon.

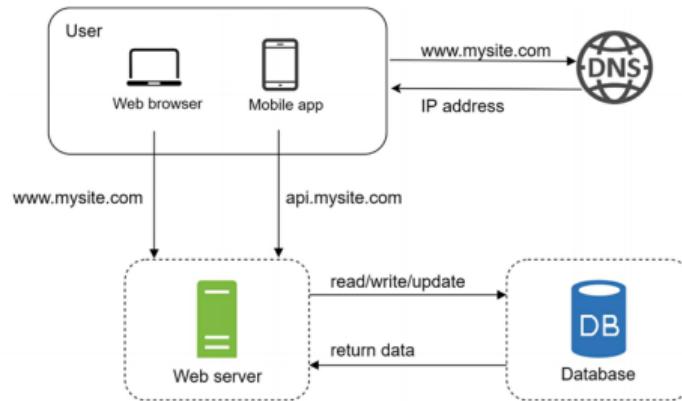
Base de Datos

3 Caso de Estudio

Separar el tráfico del Web tier y de DB tier les permite escalar de manera independiente.

¿Cuál BD elegir?

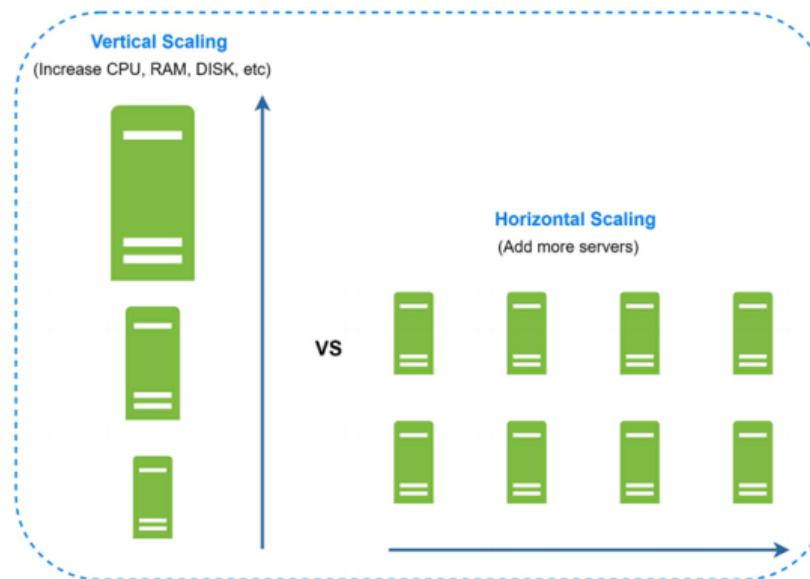
- Relacionales: transaccionales, permiten hacer consultas SQL.
- No-Relacionales o NoSQL.
 - No permiten hacer consultas SQL
 - Key-value, documentos, grafos, familia de columna.
 - Enfocadas a latencias submilisegundo, datos no estructurados, almacenamiento de big data.



Tipos de escalamiento

3 Caso de Estudio

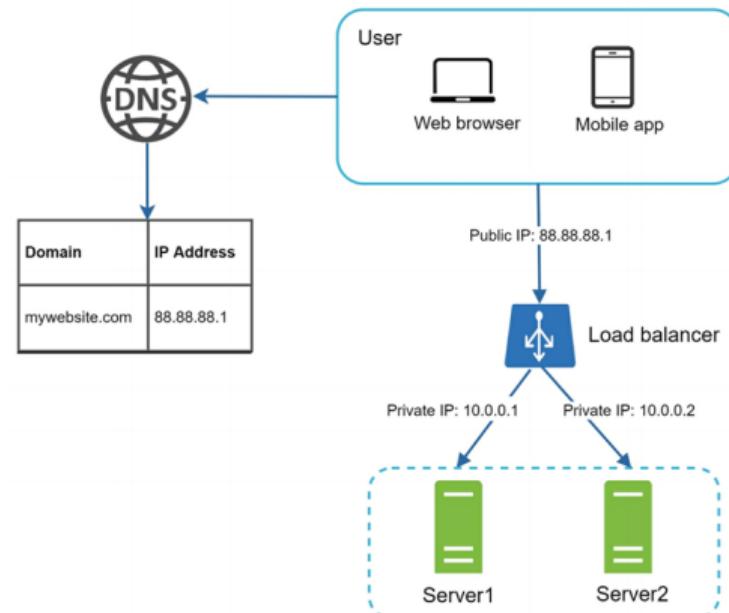
- Escalabilidad vertical si el tráfico es bajo.
- Escalabilidad horizontal es más deseable para aplicaciones a gran escala.



Balanceador de Carga

3 Caso de Estudio

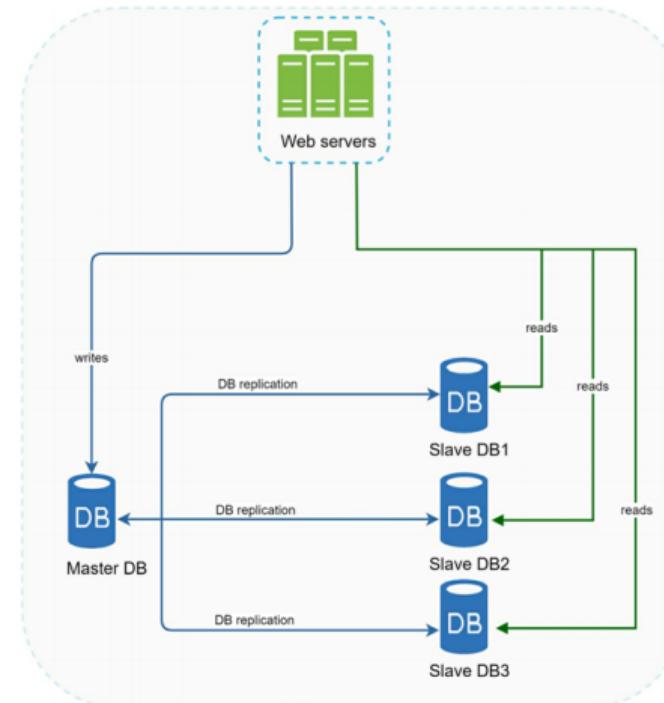
- Un balanceador de carga distribuye el tráfico entre servidores web que están definidos en un conjunto.
- Los clientes se conectan a la IP del balanceador de carga.
- Los servidores Web no son alcanzables por los usuarios.
- Ahora hay tolerancia a fallas y alta disponibilidad en los servidores.



Replicación en Base de datos

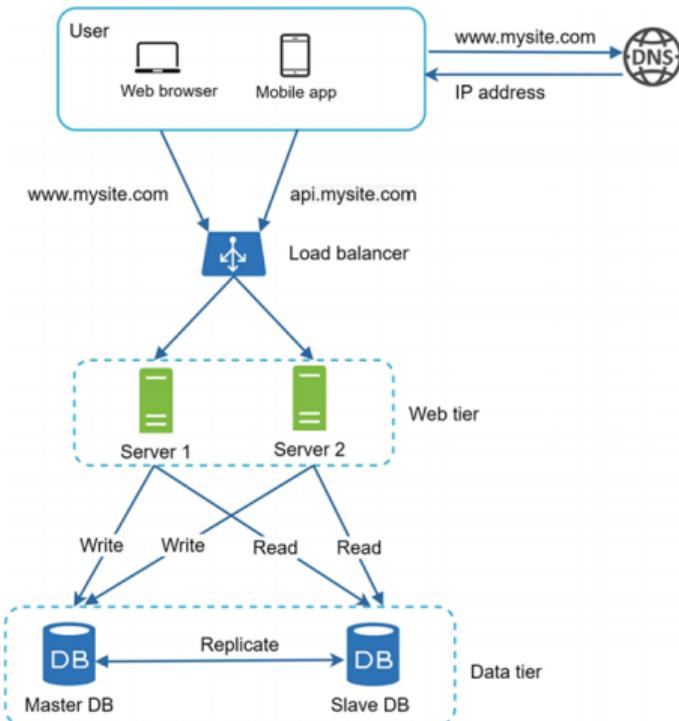
3 Caso de Estudio

- Generalmente se usa una relación maestro esclavo entre la BD original y las copias.
- Master: soporta operaciones de escritura.
- Las otras se ocupan para lectura, para evitar problemas de coordinación en escrituras.
- Mejor rendimiento, concurrencia, confiabilidad, alta disponibilidad.



Balanceador y replicación DB

3 Caso de Estudio



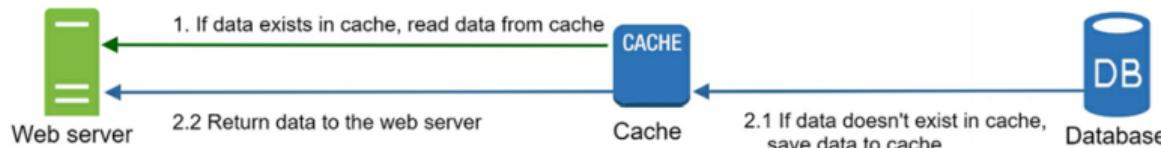
Caché

3 Caso de Estudio

- Almacenamiento temporal de respuestas que es más caro de computar nuevamente que almacenarlo o que son frecuentemente accedidas para mejorar latencia.
- Ejemplo de tecnologías: Memcached, Redis.
- Mejor cuando los datos son frecuentemente accedido pero no frecuentemente modificados (consistencia?).
- Políticas de reemplazo o expiración.

Caché

3 Caso de Estudio



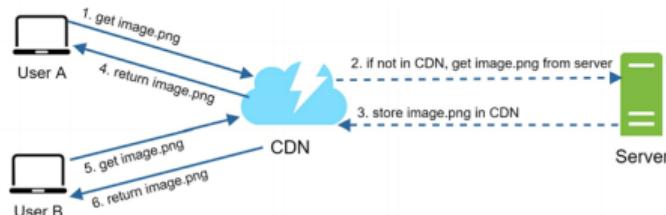
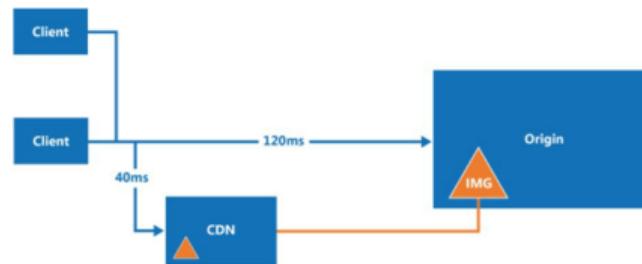
Ejemplo:

```
SECONDS = 1
cache.set('myKey', 'hello world', 3600 * SECONDS)
cache.get('myKey')
```

Content Delivery Network (CDN)

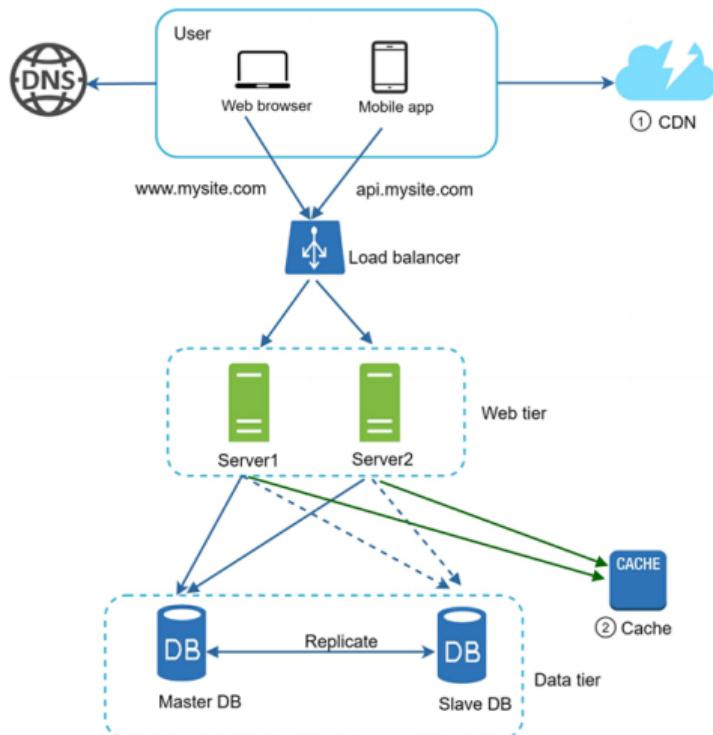
3 Caso de Estudio

- Un CDN es una red de servidores dispersa geográficamente usada para entregar contenido estático.
- Un servidor más cercano al cliente entrega el contenido.
- Ejemplo URL en CDN:
 - CloudFront:
`https://mysite.cloudfront.net/bigimage.jpg`
 - Akamai:
`https://mysitet.akamai.com/image-manager/img/bigimage.jpg`
- Consideración costo.



Arquitectura con caché y CDN

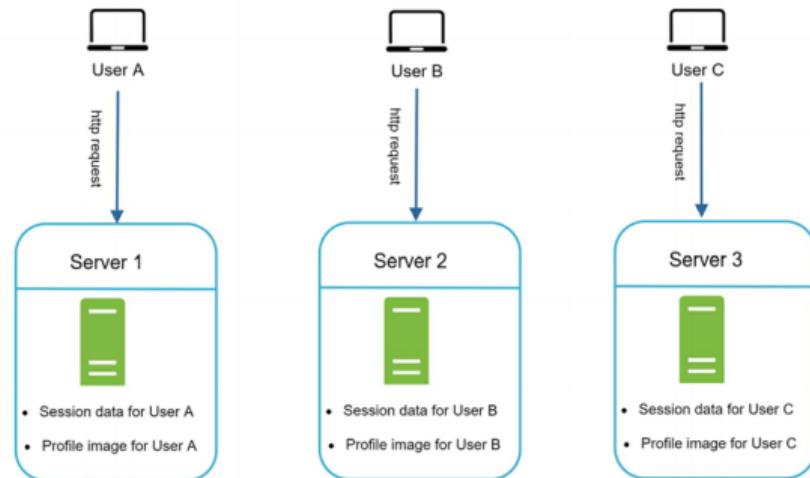
3 Caso de Estudio



Capas sin estado

3 Caso de Estudio

- Para escalar la capa web horizontalmente se necesita volverla sin estado.
- Mover estado (sesión de usuario) fuera de la capa Web.
- ¿Habilitar sticky session en balanceador? Overhead y dificultad en fallas.
- Mejor usar estado en BD.

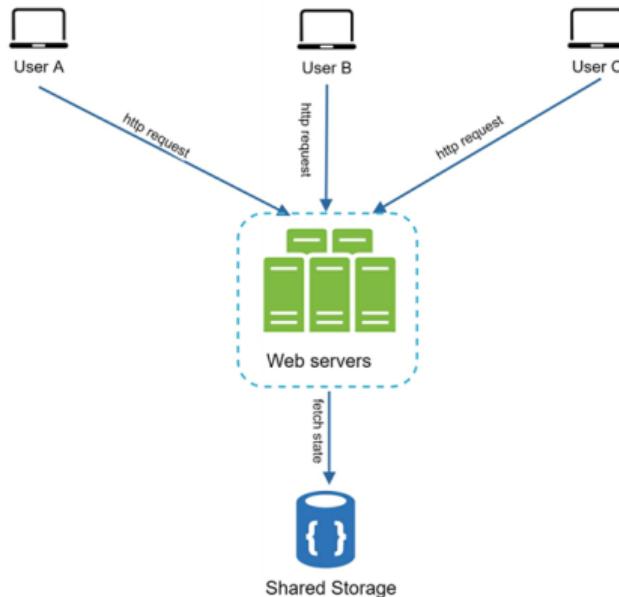




Capas sin estado

3 Caso de Estudio

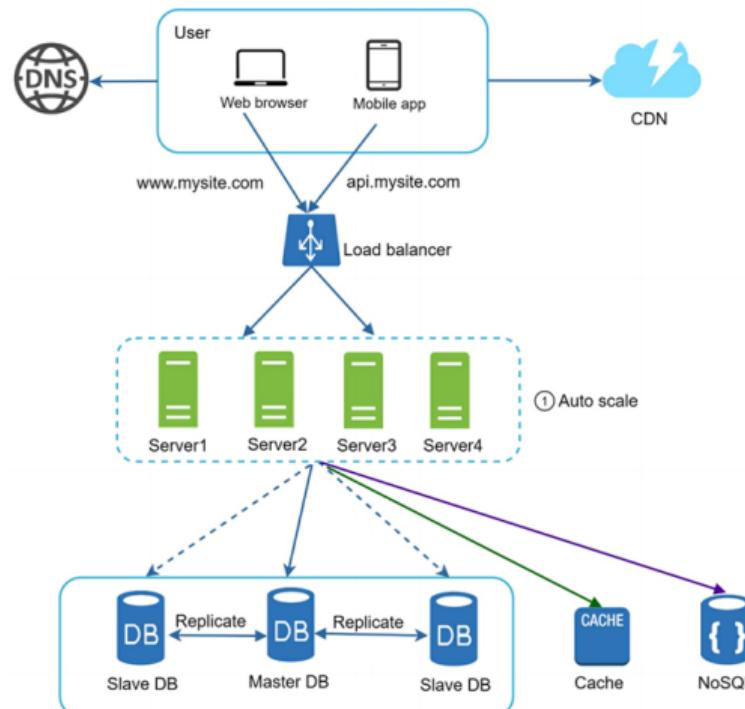
- Para escalar la capa web horizontalmente se necesita volverla sin estado.
- Mover estado (sesión de usuario) fuera de la capa Web.
- ¿Habilitar sticky session en balanceador? Overhead y dificultad en fallas.
- Mejor usar estado en BD.



Capas sin estado

3 Caso de Estudio

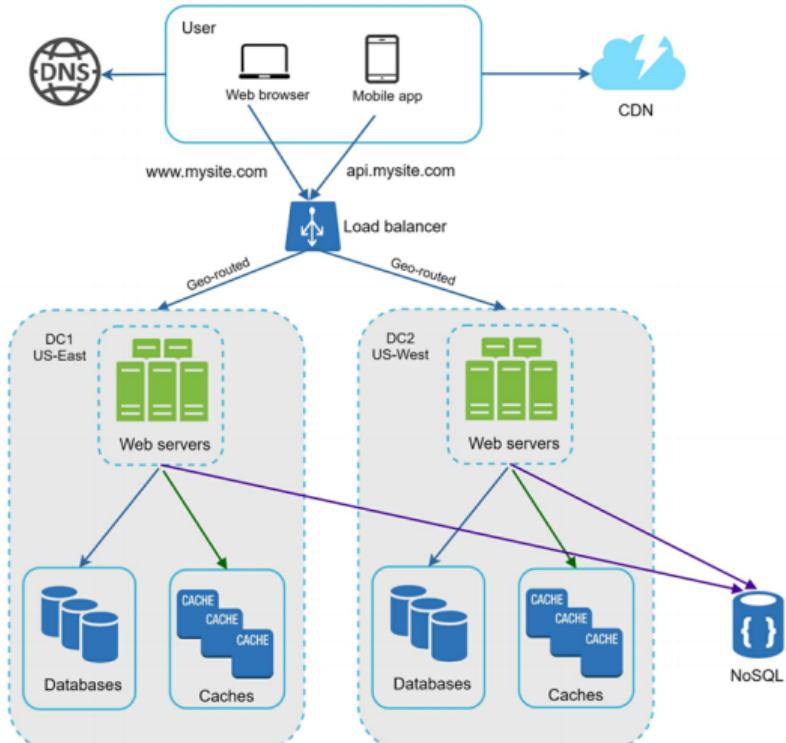
BD NoSQL estilo key-value son rápidas, escalables como para esta tarea.



Datacenters

3 Caso de Estudio

- Usuarios internacionales, acceso a datacenter más cercano.
- Soporta desastres naturales.
- Desafíos: redirección de tráfico, sincronización datos.

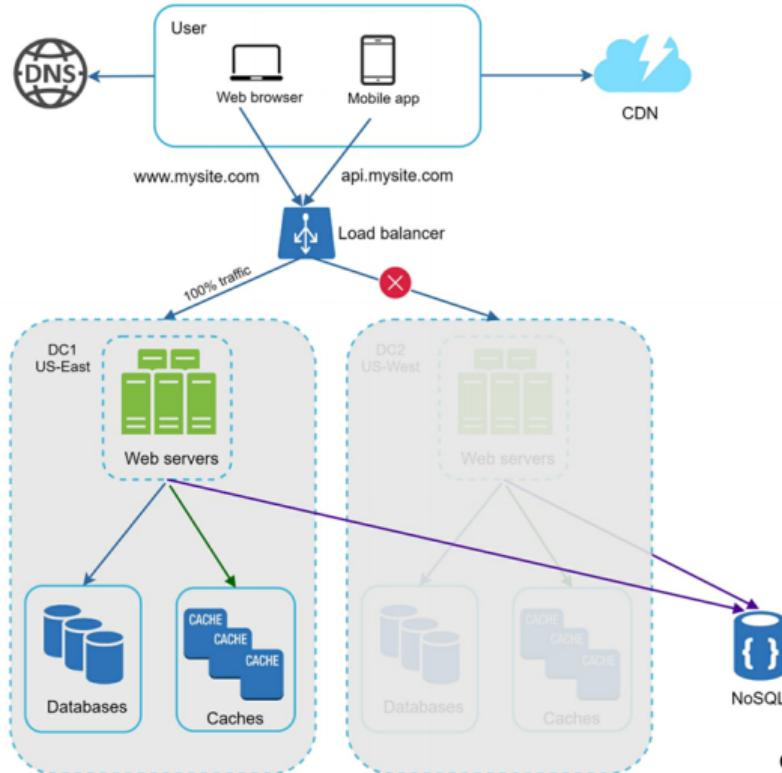




Datacenters

3 Caso de Estudio

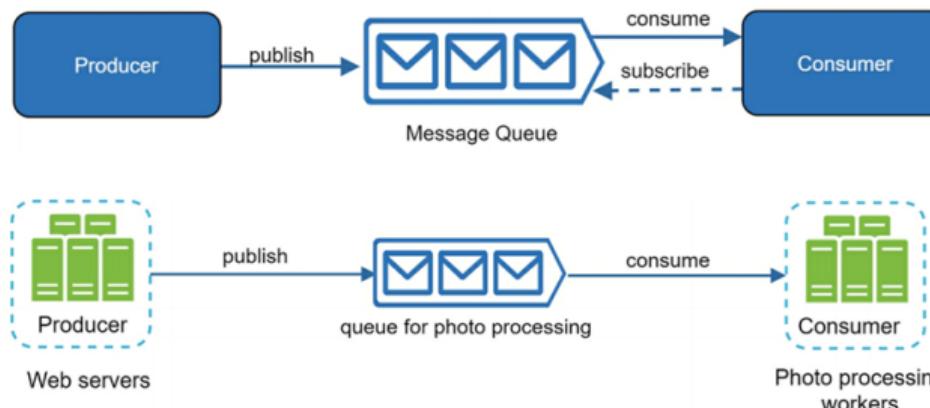
- Usuarios internacionales, acceso a datacenter más cercano.
- Soporta desastres naturales.
- Desafíos: redirección de tráfico, sincronización datos.



Colas de mensajes

3 Caso de Estudio

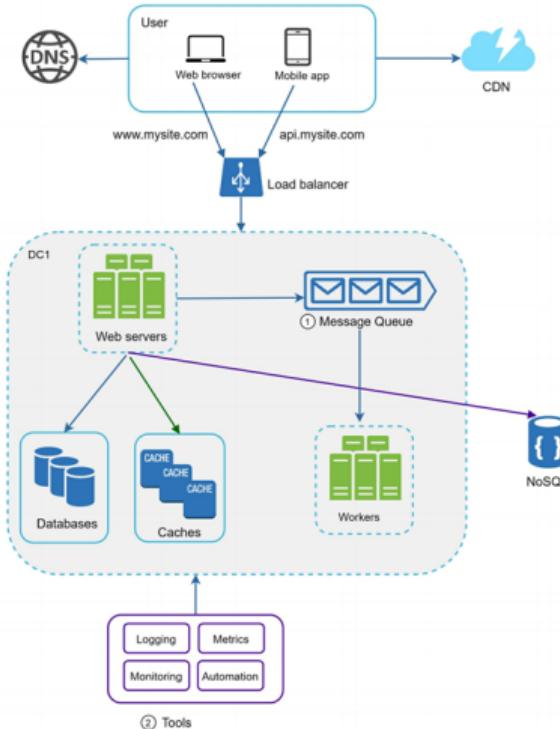
- Comunicación asíncrona para desacoplar diferentes componentes del sistema.
- Cola de mensajes: Almacenamiento en memoria, sirve como un buffer.
- Modelo productor/consumidor.
- Ejemplo: Procesamiento de imágenes, tareas lentas, escalan independientemente.



Registros, métricas y automatización

3 Caso de Estudio

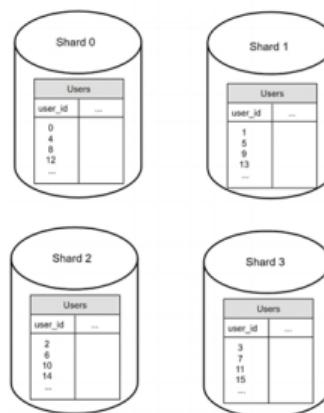
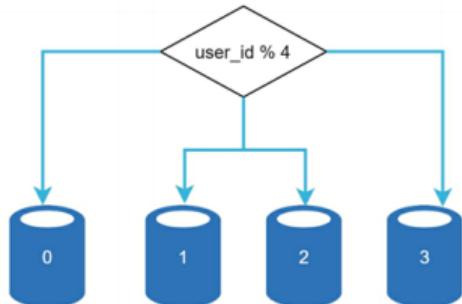
- Esencial cuando los sitios son grandes.
- Logging: hay que monitorear los logs.
- Recolectar métricas ayuda a entender la salud del sistema entre otros.
- Integración continua es una buena práctica. Automatizar build, test, deploy, process, etc.



Escalar base de datos

3 Caso de Estudio

- Sharding: separa la base de datos en shards más pequeños.
- Se comparte esquema, los datos son únicos en cada shard.
- Es importante elegir key que distribuya los datos uniformemente, e idealmente los accesos a esos datos.
- Problemas: ¿Cómo hacer re-sharding? Problema de celebridades, join y desnormalización.

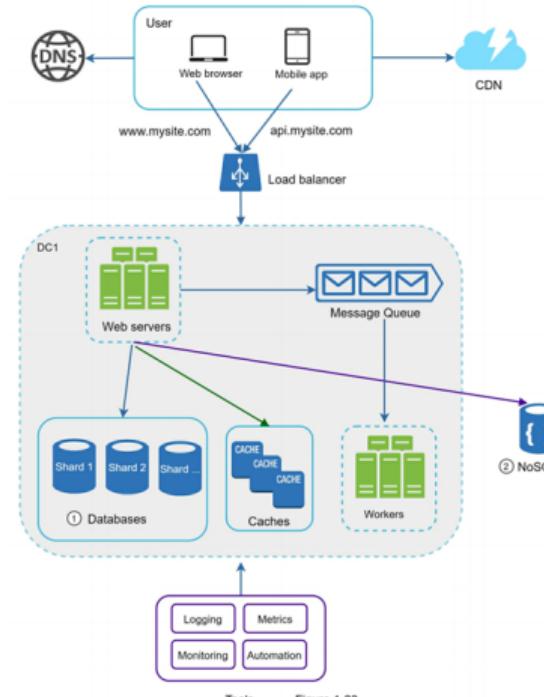


Arquitectura con Sharding

3 Caso de Estudio

Proceso iterativo. Ideas:

- Mantener capas de front-end sin estado.
- Implementar redundancia en cada capa.
- Soportar múltiples datacenters.
- Almacenar contenido estático en CDN.
- Escalar capa de datos con sharding.
- Dividir capas en servicios.
- Monitorear el sistema y usar herramientas de automatización.





UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

Introducción y Tipos de Arquitecturas

INF-343 Sistemas Distribuidos

Prof. Jorge Díaz M.

March 12, 2024

