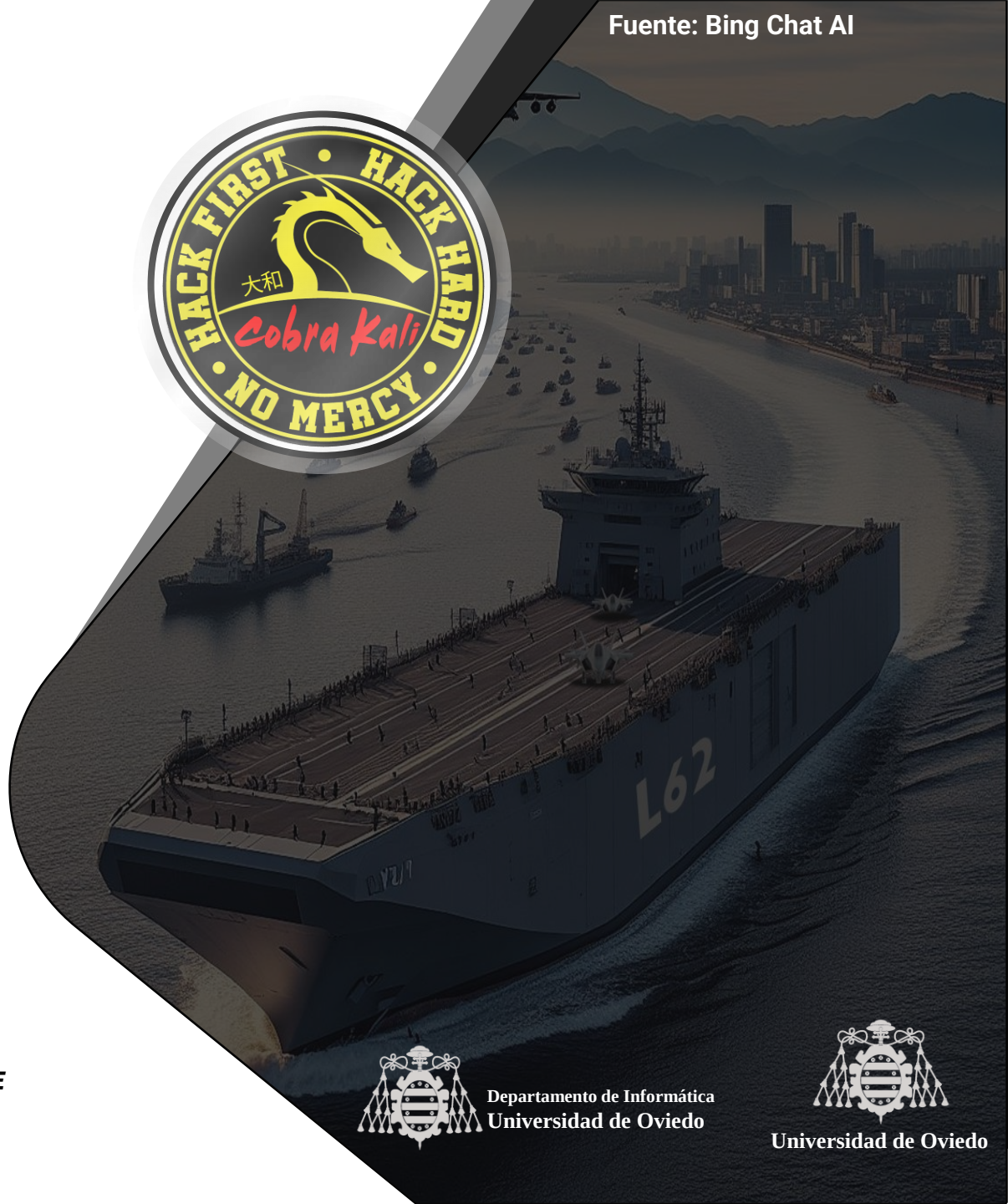
PRINCIPIOS DE DISEÑO DE INFRAESTRUCTURAS

Diseñando sistemas según nuestras necesidades

José Manuel Redondo López Proyecto "*L-62 Princesa de Asturias*" v1.2




Fuente: Bing Chat AI



Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿QUÉ DEBERÍAS SABER PARA ENTENDERLO CORRECTAMENTE?

José Manuel
Redondo López

- **Para entender este tema hace falta saber lo siguiente**
  - Principios de administración de sistemas Linux a nivel de usuario / usuario avanzado
  - Conocimiento de los conceptos básico de redes de ordenadores
  - Conocimientos básicos de desarrollo de webs dinámicas modernas
  - Saber qué es una máquina virtual y la función del software de virtualización
- **Gran parte de los conocimientos requeridos deberías ya tenerlos de tus estudios anteriores**
  - Si hay algo que quieres repasar, puedes usar la "S-81 Isaac Peral" en: https://ocw.uniovi.es/course/view.php?id=109

BACK

Cómo diseñar una infraestructura

José Manuel
Redondo López

- **Hay dos principios para diseñar una infraestructura con éxito**
  - **Divide y vencerás**: descomponer el problema de arriba a abajo en distintos módulos
  - **Compromisos**: **NO HAY SOLUCIÓN PERFECTA**, hay que asumir compromisos
    - Asume tus restricciones y calcula el impacto que pueden tener en el rendimiento global de todo
- **Procedimiento**
  1. Entrevista a tus usuarios y hazte una idea clara del problema a resolver según sus requisitos
  2. Diseña el sistema primero "en papel" (¡o en diagrama! :D)
     - Es decir, **en abstracto**
  3. Analiza lo que has diseñado, identifica cuellos de botella y haz ajustes (y vuelve a 2)
- **¡Es como diseñar software!**

¿Cómo que es como diseñar software?

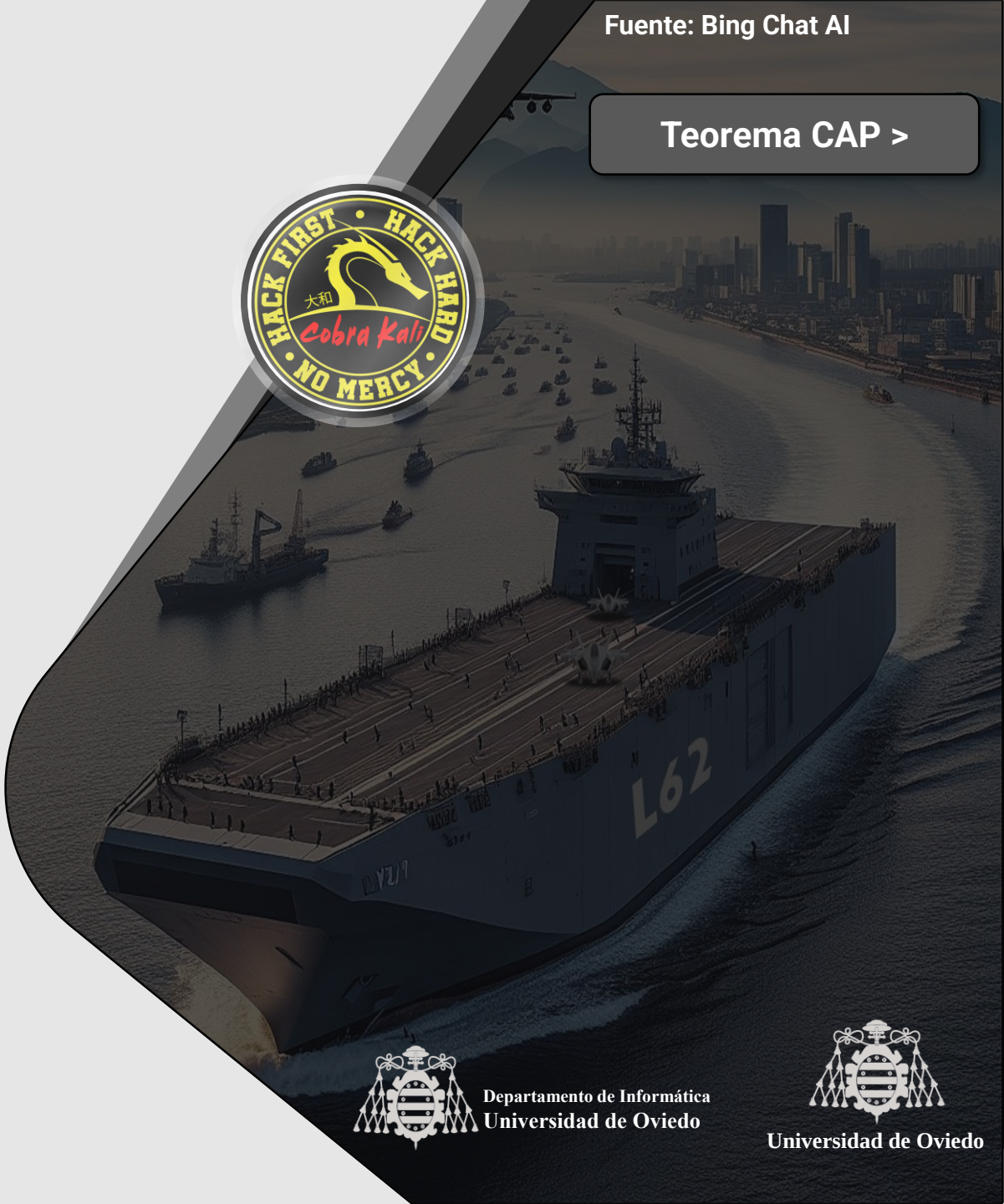
José Manuel
Redondo López

- **¡Claro!, pero en lugar de diagramas de clases UML puedes utilizar diagramas de componentes**
  - UML no tiene elementos específicos para arquitecturas
- **No habrá herencia entre sistemas (componentes), pero sí…**
  - Relaciones entre algunos de ellos
    - Conexiones de red
  - Dependencias entre los mismos
    - Ej.: la base de datos necesaria por un backend, el frontend de ese backend...
  - Habrá restricciones de conexión entre sistemas
    - Distintos segmentos de red, firewalls...
- **Y, sobre todo, porque hoy en día la infraestructura se crea como código**
  - Infraestructure as Code o IaC
  - Esto es en lo que nos centraremos en esta asignatura
  - Pero ahora haremos una introducción a aspectos de construcción de infraestructuras e IaC

ÍNDICE

< Ir al Índice

Teorema CAP >



Fuente: Bing Chat AI

Infraestructuras de Sistemas

Esquema general de como tomarse el diseño de una infraestructura en serio

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿QUÉ VAMOS A VER EN ESTE APARTADO?

José Manuel
Redondo López

- Los **principios básicos** que rigen el diseño de una buena infraestructura de sistemas
- El **aspecto general** de una infraestructura de máquinas segura
- **Conceptos** involucrados en la construcción de infraestructuras seguras
- Lo importante que es integrar la construcción del **software** con la infraestructura (**DevOps**)

Principios de diseño de una infraestructura

José Manuel
Redondo López

- **En la siguiente transparencia podéis ver el diagrama de una infraestructura de un sistema basado en estos principios**
  - **Especialización de servidores**: Cada tipo de servidor tiene un único propósito: front-end, back-end, SGBD…
  - **Segmentación**: Cada "capa" de la infraestructura está totalmente aislada del resto
    - Solo algunos sistemas deberán ser capaces de acceder a servicios de las capas adyacentes
  - Preparada para incorporar seguridad avanzada
    - Si bien esto será objeto de estudio en otra asignatura
    - **Administración de Servidores Web "*F-103 Blas de Lezo*"**

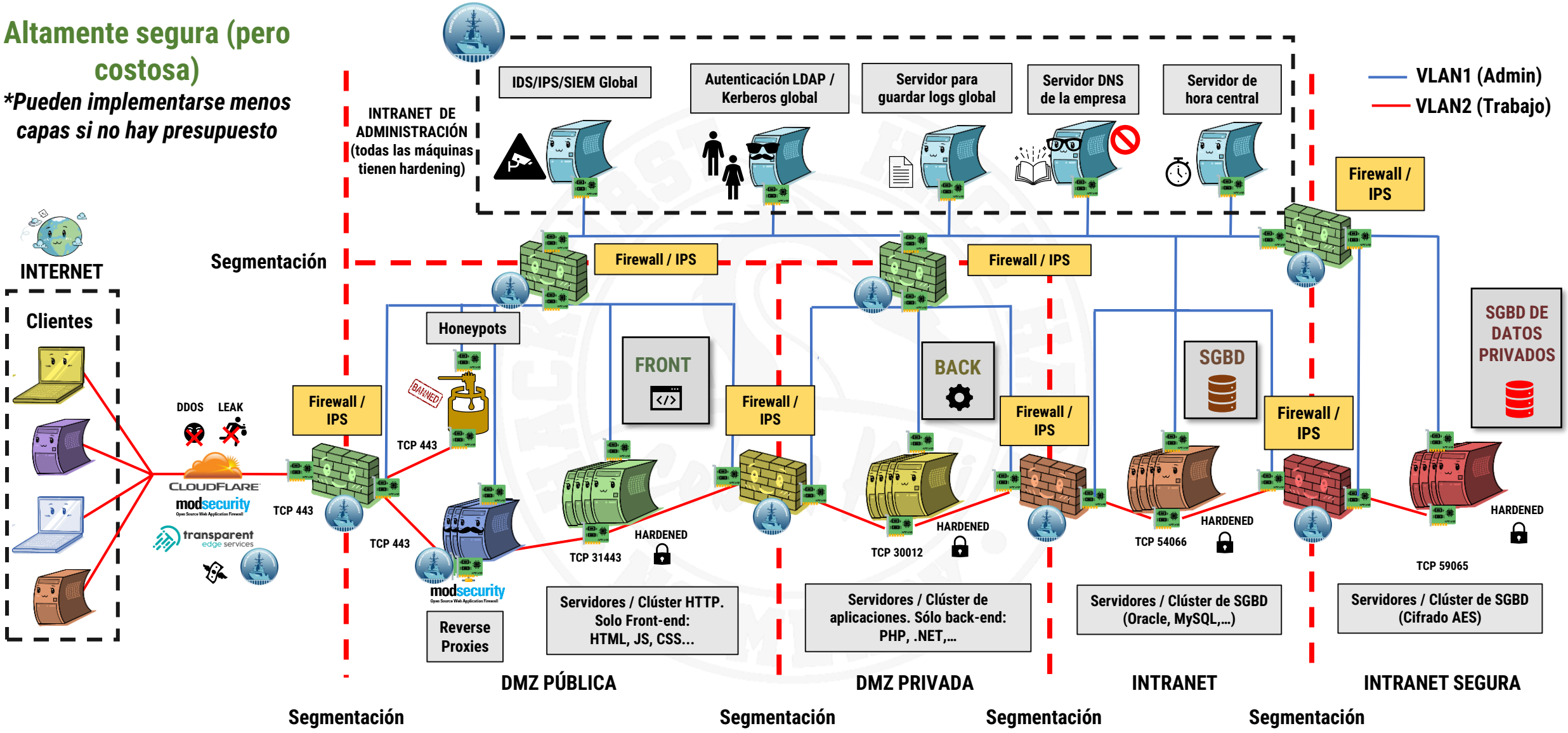
- **En esta asignatura nos limitaremos a los elementos que forman la infraestructura y no a específicos de seguridad**
  - Pero podréis incorporarlos fácilmente si cursáis la asignatura complementaria
  - ¡Las dos asignaturas están integradas!

INFRAESTRUCTURA DE RED ALTAMENTE SEGURA

José Manuel
Redondo López

Altamente segura (pero costosa)

Pueden implementarse menos capas si no hay presupuesto



* Los logos son propiedad de sus compañías correspondientes. Iconos de *PowerPoint*. *Esta infraestructura está derivada de:* https://pciguru.files.wordpress.com/2013/12/200511-theultrasecurenetworkarchitecture.pdf

Elementos importantes de la infraestructura

José Manuel
Redondo López

- Veremos detalles de distintos tipos de **firewalls** en "*F-103 Blas de Lezo*"
  - Solo necesitamos saber que aíslan una red de otra, sólo permitiendo tráfico de/hacia IPs/puertos concretos
- Dos DMZ con 2 **firewalls**: **pública** (cara a internet) y otra **privada**
  - **DMZ pública**
    - Contienen **únicamente la lógica de la interfaz de usuario de la aplicación** (front-end)
    - Están aislados de los sistemas de la DMZ privada
  - **DMZ privada**
    - Contienen la **lógica de negocio/aplicación** (back-end, código del lado del servidor, PHP, ASP, ...)
    - Y enlaces a sistemas internos con funciones adicionales
  - Esto permite reglas más detalladas para acceder a la lógica de la aplicación y **mitiga ataques**
- Dos LAN internas con doble cortafuegos
  - **LAN interna:** contiene los servidores y sistemas accesibles para los empleados, que no almacenan información confidencial, así como los datos no confidenciales de las aplicaciones
  - **LAN segura**: contiene servidores con información cifrada y confidencial que podría utilizarse para el robo de identidad u otros fraudes
    - Números de tarjeta de crédito, números de cuenta de cheques, etc.

ELEMENTOS IMPORTANTES DE LA INFRAESTRUCTURA

José Manuel
Redondo López

- Se deben configurar **reglas** en los **firewalls** para **restringir y controlar** las comunicaciones entrantes y salientes
  - El **firewall de la red DMZ pública** se configuraría solo para permitir el tráfico TCP 443 entrante y saliente a las IP de los servidores HTTPS orientados al público (proxy inversos / honeypots)
  - **Entre la DMZ pública y la DMZ privada**, sólo el puerto 30012 debe estar abierto, restringiendo sus comunicaciones a las direcciones IP de los servidores involucrados en nuestro ejemplo
  - **Entre la DMZ privada y la LAN interna**, sólo los puertos necesarios para comunicarse con los distintos servidores deben estar abiertos, y restringidos a direcciones IP específicas
  - **Entre la LAN interna y la LAN segura**, sólo deben estar abiertos los puertos necesario para obtener acceso al servidor SGBD cifrado, y restringido a las direcciones IP permitidas en la LAN interna
  - **Todos los demás puertos y protocolos estarían cerrados**, ya que no son necesarios
- Esto requiere un mecanismo fiable para identificar cada servidor que cumple con un rol crítico (**asignación de IPs estáticas**)

Elementos importantes de la infraestructura

José Manuel
Redondo López

- **Una política común de ubicación de los puertos de servicios**
  - Los servicios en la web **deben utilizar un cifrado adecuado SIEMPRE** (`tcp/443`)
    - **En producción, ya NUNCA se usa HTTP** (`tcp/80`) en la actualidad
  - El puerto HTTPS por defecto (`tcp/443`) sólo se usa en la DMZ pública
    - Queremos que nuestros servicios públicos sean accesibles en los lugares habituales
  - Todas las demás conexiones a los servicios **usan puertos TCP y UDP no estándar**
    - Reduce la posibilidad de que atacantes identifiquen accidentalmente servicios e información examinando puertos estándar con ataques de inyección de código o automatizados
- **Cada capa de la infraestructura usa una subred distinta a la del resto**
  - Firewalls con varias NICs (tarjetas de red) aíslan las diferentes subredes de las capas
  - Cada DMZ debe tener sus propias subredes **con IPs privadas** que se usen en ninguna otra subred
    - Recuerda que se definen IPs privadas como IPs de redes que no deben usarse en Internet: https://es.wikipedia.org/wiki/Red_privada
    - Es decir, si subredes en el rango `10.x.x.x` se usan en las dos LANs y la subred privada `192.168.x.x` se usa para la LAN de administración, las DMZ deben utilizar direcciones IP en las subredes `172.16-31.x.x`

¿Entonces que debo tener en cuenta?

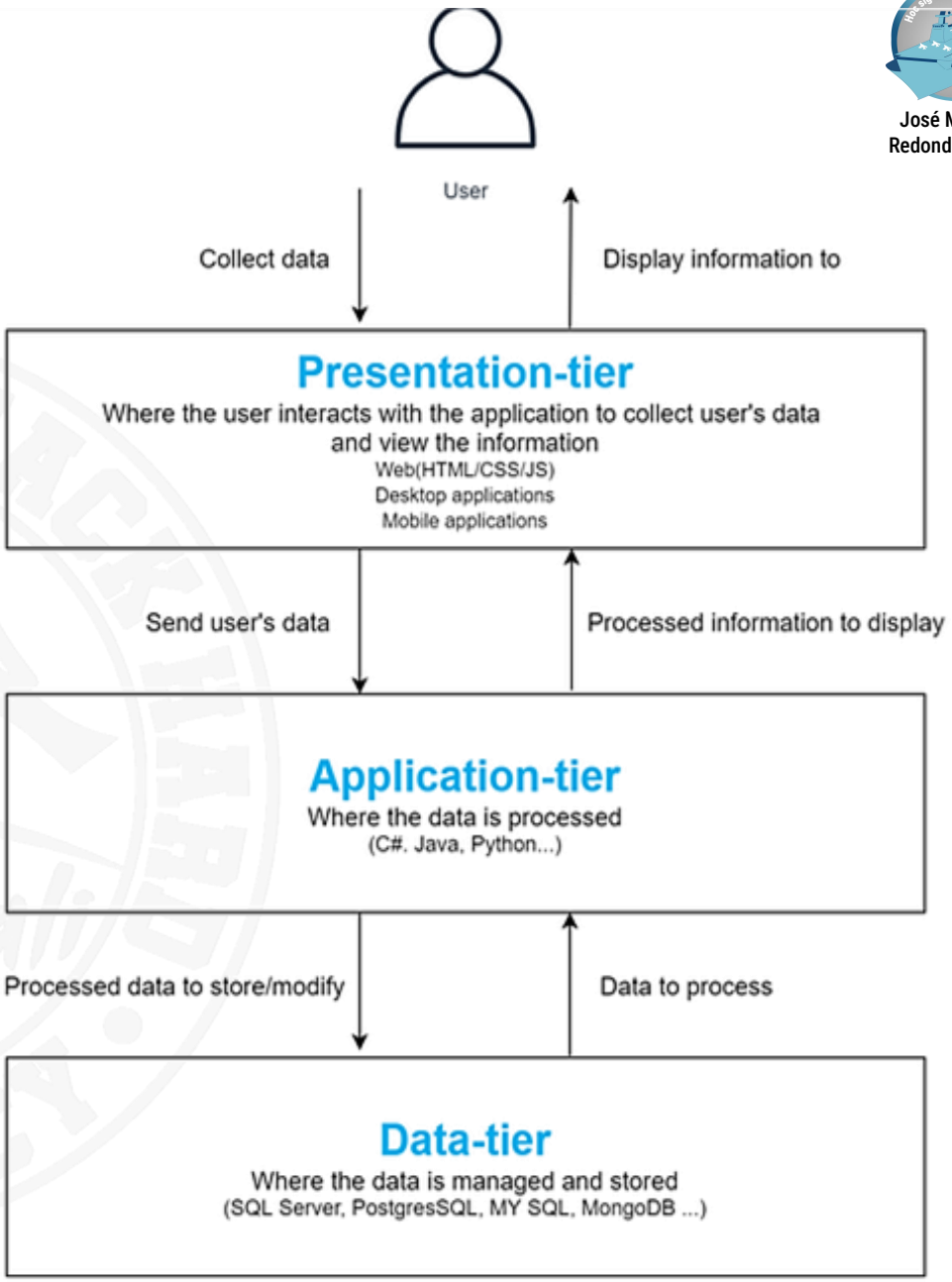
José Manuel
Redondo López

- **Las "piezas" de la arquitectura, los “sistemas”**
  - Proxys, firewalls, servidores web, servidores de aplicaciones, SGBD...
- **Cómo van a trabajar esas piezas juntas**
- **Qué uso van a tener y qué "sacrificios" habrá que hacer**
  - Normalmente para mantener los costes en unos límites razonables...
  - Veremos que **siempre hay que sacrificar algo** luego (teorema CAP)
- **Cuáles son nuestras “unidades de servicio” y los protocolos que usan**
  - Páginas web, aplicaciones, servicios web…
  - Esta asignatura pretende ser lo más independiente posible de las mismas, para que se pueda adaptar a todas las necesidades
  - Por eso usaremos el término “**unidad del servicio**” para denominar a una instancia de una aplicación de cualquier tipo que atienda peticiones

¿ENTONCES QUÉ DEBO TENER EN CUENTA?

José Manuel
Redondo López

- Cuando se monta una infraestructura como esta **hay que diseñar software que se adapte a ella**
- Las aplicaciones deben desplegarse en tantas capas (**Tiers**) como tenga la infraestructura
- Lo cual es sencillo si se hace así desde un principio
  - De ahí que sea tan importante integrar **DEV**elopment y **OP**eration**S** ☺
- De esta forma se fomenta la modularidad y se facilita el despliegue

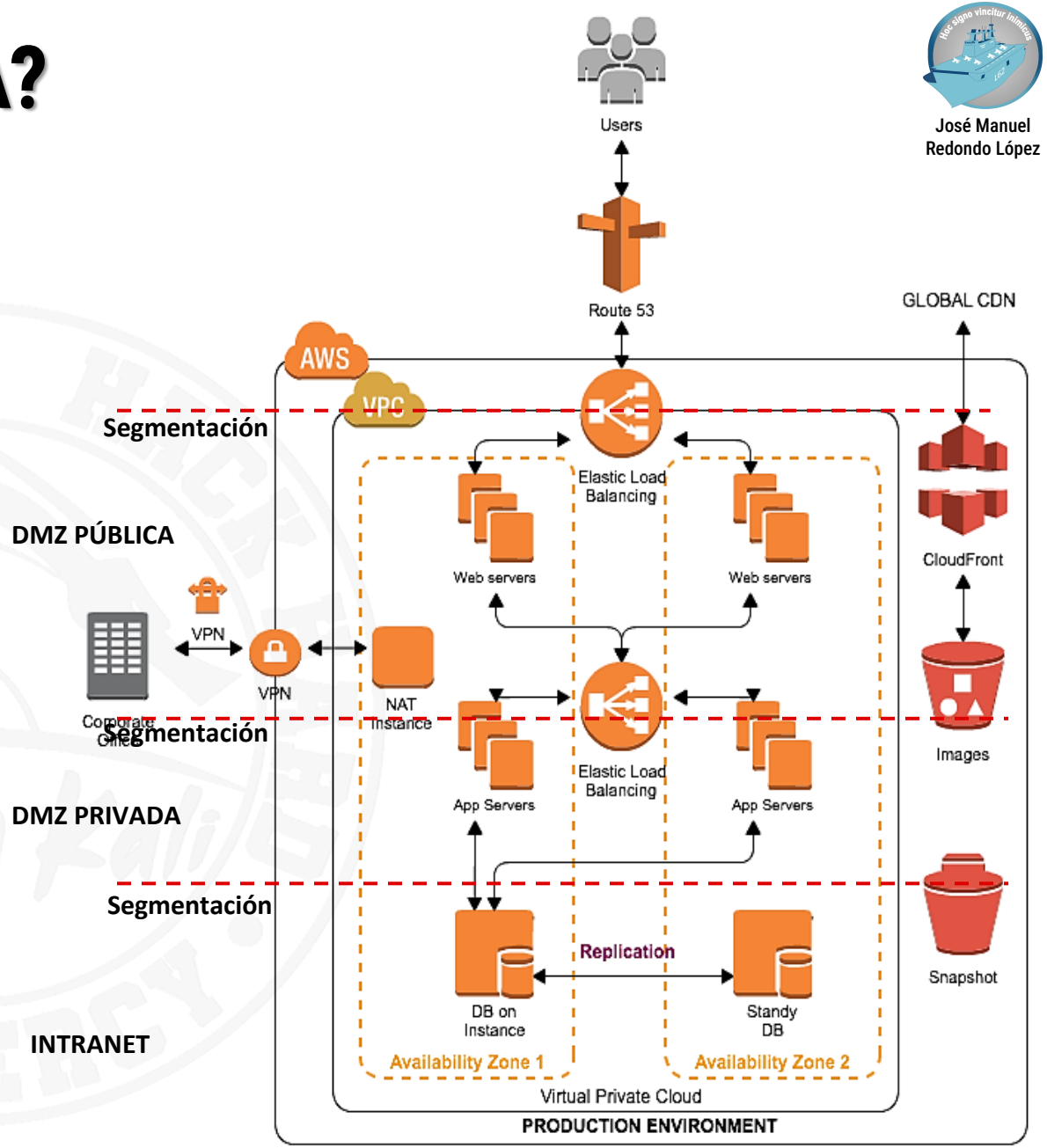


Fuente: https://enlabsoftware.com/development/how-to-build-and-deploy-a-three-tier-architecture-application-with-c-sharp.html

¿Entonces qué debo tener en cuenta?

José Manuel
Redondo López

- Llevar un esquema así a la nube también es más sencillo
- Cada capa puede tener incorporados balanceadores de carga que la optimicen
  - Y seguridad embebida (Anti-DDoS, WAF…)
- Usando las funcionalidades que dan los distintos proveedores
- El esquema de despliegue no debería depender de ninguna feature particular de un proveedor de nube concreto
  - Esto facilitaría su migración a otro proveedor
  - Mitiga el vendor locking



Fuente: https://tkssharma-devops.gitbook.io/devops-training/aws-architecture-example-lab/application-3-tier-architecture

Teorema CAP (Consistency, Availability, Partition Tolerance)

Compromisos a la hora de diseñar sistemas

¿QUÉ PASA CUANDO ACTUALIZAMOS LOS NODOS DE UN SISTEMA?

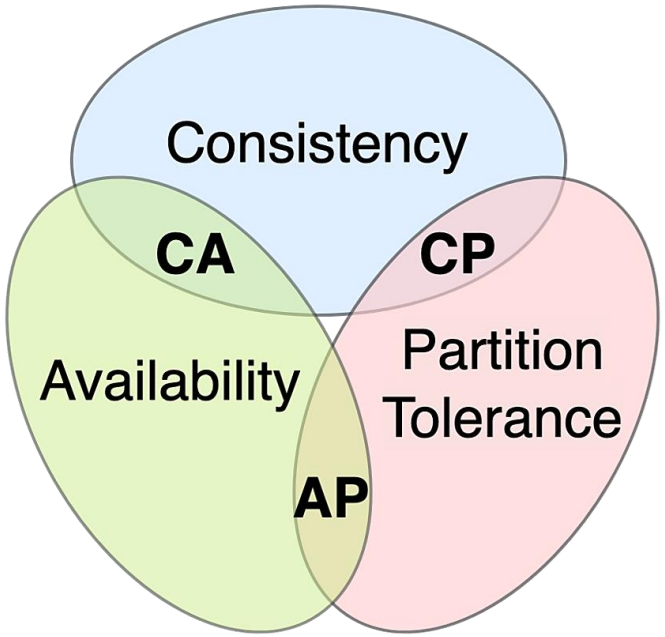
José Manuel
Redondo López

- **Este teorema se suele aplicar a SGBD**
  - Pero también debemos considerarlo cuando se actualizan servidores de front/back
  - Ej.: Nuevas releases de aplicaciones web
- **Consistency: Todos los nodos ven los mismos datos al mismo tiempo**
  - Se consigue **actualizando los nodos** antes de permitir leer de los mismos
- **Availability: Todas las peticiones obtienen respuesta**
  - Con éxito o error, pero alguna respuesta ☺
  - Se consigue **replicando la información** en diferentes sistemas
- **Partition Tolerance: El sistema sigue funcionando a pesar de que se pierdan mensajes o haya un fallo parcial del mismos**
  - **Una partición** es un evento por el cual aparecen sistemas con los que un determinado cliente se puede comunicar y sistemas con los que no
  - Es decir, **puede soportar cualquier fallo de red parcial** sin que por ello falle toda la red
  - Se consigue **replicando datos** en grupos de sistemas o redes para mantener el sistema activo

¿QUÉ OCURRE CON LOS ELEMENTOS CAP?

José Manuel
Redondo López

- **ES IMPOSIBLE CONSEGUIR LAS TRES A LA VEZ** (como máximo 2) ¿Por qué?
  - Para ser consistente, **todos los nodos deben tener las mismas actualizaciones** en el mismo orden
  - Pero si la red se particiona en distintas "islas" por un fallo...
    - ¡Las actualizaciones que lleguen a una de esas particiones pueden no llegar al mismo tiempo a las otras!
  - ¿Qué pasa si un cliente lee datos de una partición desactualizada, tras haberlos leído de una que si lo está?
    - ¡Los datos se vuelven incoherentes!
    - **Esto es un riesgo grave**, debido a que aumenta la probabilidad de corromper o incluso destruir información
    - Deberíamos parar de servir peticiones desde los sistemas que están desactualizados
    - Pero entonces...el servicio ¡ya no está 100% disponible para todos!
      - Los clientes que solo puedan ver la partición no actualizada perderán el acceso al mismo



Fuente:
https://en.wikipedia.org/wiki/CAP_theorem

¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

Sobre el diseño general de infraestructuras...

- *¿Entiendes qué es la especialización de servidores?*
- *¿Puedes definir qué significa la segmentación?*
- *¿Entiendes la diferencia entre una DMZ pública y una DMZ privada en una infraestructura?*
- *¿Entiendes la diferencia entre una LAN interna y una LAN segura en una infraestructura?*
- *¿Puedes recordar las normas básicas para restringir comunicaciones entre redes vía **firewall**?*
- *¿Podrías recomendarle a alguien en qué puertos debe situar servicios públicos? ¿y privados?*
- *¿Podrías recomendarle a alguien qué esquema de direccionamiento debe usar en las distintas redes de la infraestructura?*
- *¿Entiendes qué es el teorema CAP y las limitaciones a la hora de aplicarlo?*

Sobre la creación de aplicaciones web sobre infraestructuras...

- *¿Eres consciente de que la "perfección" en el diseño de una infraestructura no existe y siempre vas a tener que sacrificar algo para obtener otra cosa?*
- *¿Comprendes que, si no se diseñan las aplicaciones para ser alojadas e integrarse en una infraestructura segura, no se logrará un resultado adecuado?*
- *¿Comprendes ahora el concepto de **DevOps**?*
- *¿Entiendes que estos principios se aplican en local y en cualquier proveedor de nube que uses?*

< Ir al Índice

Elementos y Conceptos Comunes en una Infraestructura de Servidores

Conceptos generales que debemos conocer

Fuente: Bing Chat AI

Balanceo de carga >

Redundancia >

Caché >

Reverse Proxy >

Colas de peticiones >

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿QUÉ VAMOS A VER EN ESTE APARTADO?

José Manuel
Redondo López

- **Este apartado te va a explicar una serie de conceptos básicos a la hora de diseñar una infraestructura**
  - El **balanceo de carga** como forma de repartir las peticiones entre varias entidades que las atiendan
    - Las distintas formas existentes de conseguir balanceo de carga en función del presupuesto y necesidades de la aplicación
  - El concepto de **redundancia** y su importancia de cara a la disponibilidad a la hora de diseñar sistemas
  - El concepto de **caché** y su importancia de cara al rendimiento a la hora de diseñar sistemas
    - Las políticas más habituales a la hora de construir y gestionar cachés
  - El concepto de **reverse proxy** y su importancia en el diseño de infraestructuras para fomentar el aislamiento
  - El concepto de **cola de peticiones** y su importancia en el diseño de infraestructura de cara a aprovechar sus capacidades

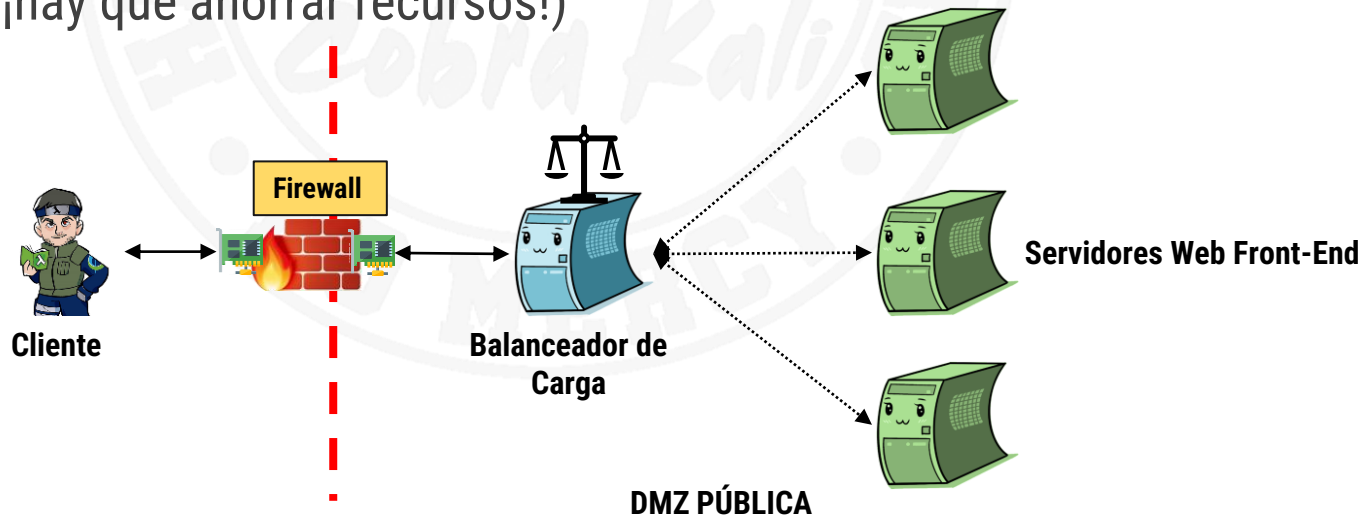
Balanceo de Carga

Lidiando con las peticiones de los clientes

¿QUÉ ES EL BALANCEO DE CARGA?

José Manuel
Redondo López

- **Es muy común que un solo sistema no pueda con el volumen de peticiones que le llegue en un momento dado**
  - Necesita la "ayuda" de otros idénticos
    - Puede ser algo **continuo:** el sistema se diseña con N servidores, **siempre** "balancea"
    - O **puntual**: el sistema "crece" en respuesta a "**picos de carga**" previstos previamente
  - También se puede usar un sistema de **balanceo elástico de carga** (como Kubernetes)
    - Si hay un "pico de carga" el sistema **escalará automáticamente** para lanzar más unidades de servicio que atiendan esas peticiones
    - Si el sistema se mantiene con baja carga un periodo determinado, **se destruirán automáticamente** esas "unidades de servicio" (¡hay que ahorrar recursos!)



Escalabilidad horizontal vs vertical

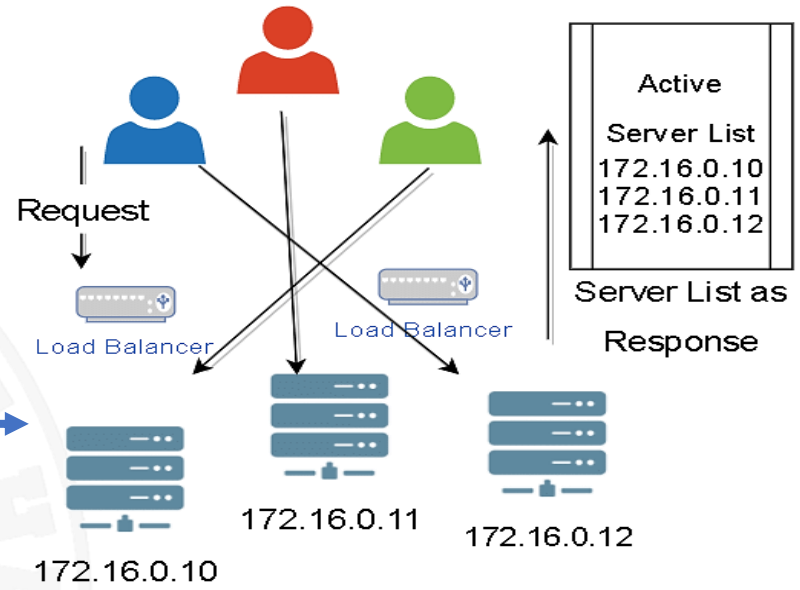
José Manuel
Redondo López

- **Hay dos formas de conseguir escalabilidad**
  - **Horizontal**: Añadir más nodos a un sistema, probablemente réplicas de los que ya existen
    - Los contenedores que veremos son un elemento ideal para ello, dado que facilitan mucho su replicación
  - **Vertical**: Añadir más capacidad de proceso a cada uno de los nodos del sistema
    - Puede ser CPU, RAM, Disco…
  - Más unidades de servicio VS más potencia a las unidades de servicio existentes
- **La escalabilidad horizontal es más conveniente**
  - Disminuye mucho la probabilidad de tener un déficit de recursos
  - Añadir más recursos a los nodos podría requerir interrumpir parcial o totalmente el servicio
  - La escalabilidad vertical **suele ser bastante más cara** que la horizontal

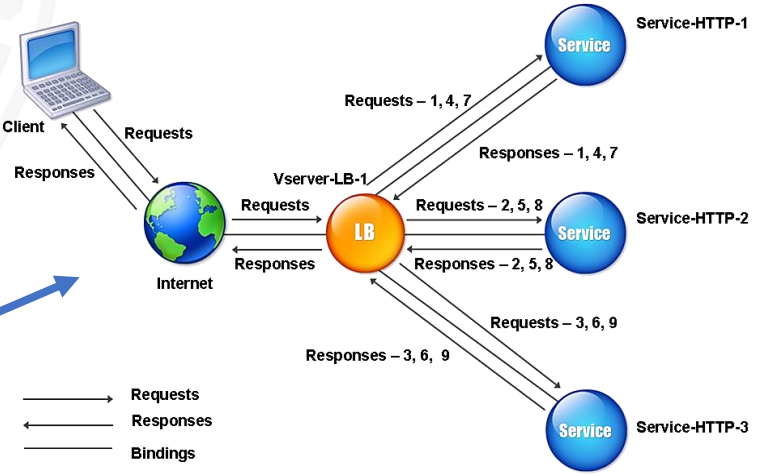
¿Quién decide a qué sistema va cada petición?

José Manuel
Redondo López

- **Es muy importante distribuir las peticiones entre los sistemas disponibles**
  - Evita la sobresaturación de un servicio
- **Los algoritmos más comunes son muy simples**
  - **Aleatorio**: Va a una unidad del servicio cualquiera
    - Con la esperanza de que haya una distribución más o menos equitativa
    - Se asume que todas las unidades de servicio se ejecutan en sistemas de más o menos las mismas capacidades
  - **Aleatorio con pesos**: Igual que la anterior, pero se tiene en cuenta **las capacidades** de cada unidad del servicio
    - Ocurre cuando el hardware de esas unidades del servicio es diferente en cada uno de los sistemas que las ejecutan
    - El hardware más potente atiende comparativamente más peticiones
  - **Round-Robin**: El típico algoritmo de reparto circular



https://www.researchgate.net/figure/Hybrid-load-balancing-strategy_fig10_329333997

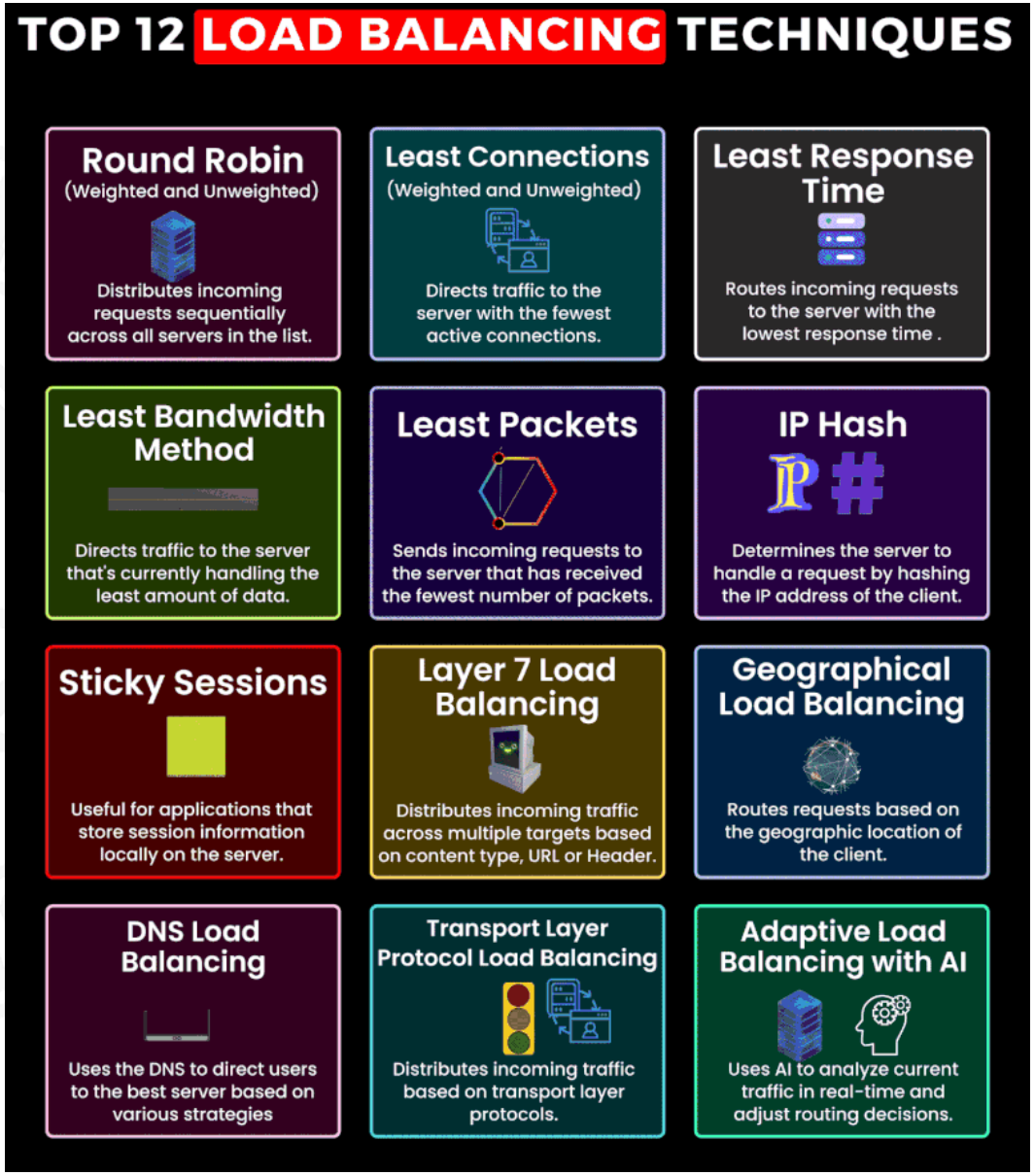


Fuente: https://docs.citrix.com

¿Quién decide a qué sistema va cada petición?

José Manuel
Redondo López

- El propio balanceador de carga es el que se configura para adoptar cualquiera de las políticas de balanceo que soporte
- Los servidores web con módulos para balanceo tienen todos sus modos documentados
- Usar uno u otro dependerá de la aplicación en sí, nuestras prioridades, características de cada máquina...

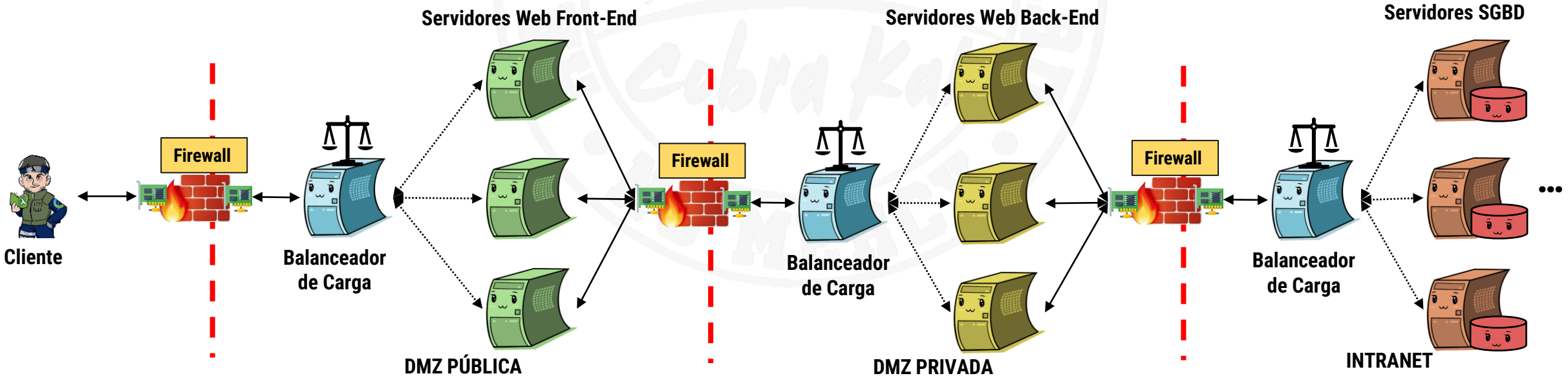


Fuente: Brij Kishore Pandey (LinkedIn)

¿CUÁNTOS BALANCEADORES DE CARGA NECESITO?

José Manuel
Redondo López

- Como hemos dicho, **depende del presupuesto y necesidades**
- Si quieres (y puedes pagar) escalabilidad y redundancia completa, necesitas 3
- O 4 si se usa una intranet segura para datos en SGBD críticos, de acuerdo con el esquema de red que vimos antes



¿QUÉ RECOMIENDAS PARA BALANCEAR LA CARGA?

José Manuel
Redondo López

- **Usar un "Smart client", ya sea hardware o software, que te de estos servicios**
  - Detecta un pool de servicios y balancea la carga por el algoritmo que queramos
  - Detecta cuándo algunos servicios se han caído
    - Y trata de reiniciarlos o mitigar el problema de otras formas que le indiquemos (servicios de respaldo)
    - Se denomina **health check**
  - Facilita añadir nuevos servicios (integrándolos automáticamente en el pool)
- **¿Software o Hardware?**
  - Depende de su coste (presupuesto), y de la capacidad necesaria de atender peticiones
  - El hardware puede atender más peticiones, pero tiene mayor coste
- **Entonces, si mi software (SGBD, servidor web, de aplicaciones) tiene funcionalidades de LB (Ejs.: MySQL, Apache Tomcat...), ¿las uso?**
  - ¡Si! Salvo que tengas que atender muchas peticiones
  - En ese caso es mejor usar servidores independientes que sólo se encarguen de LB o uno hardware

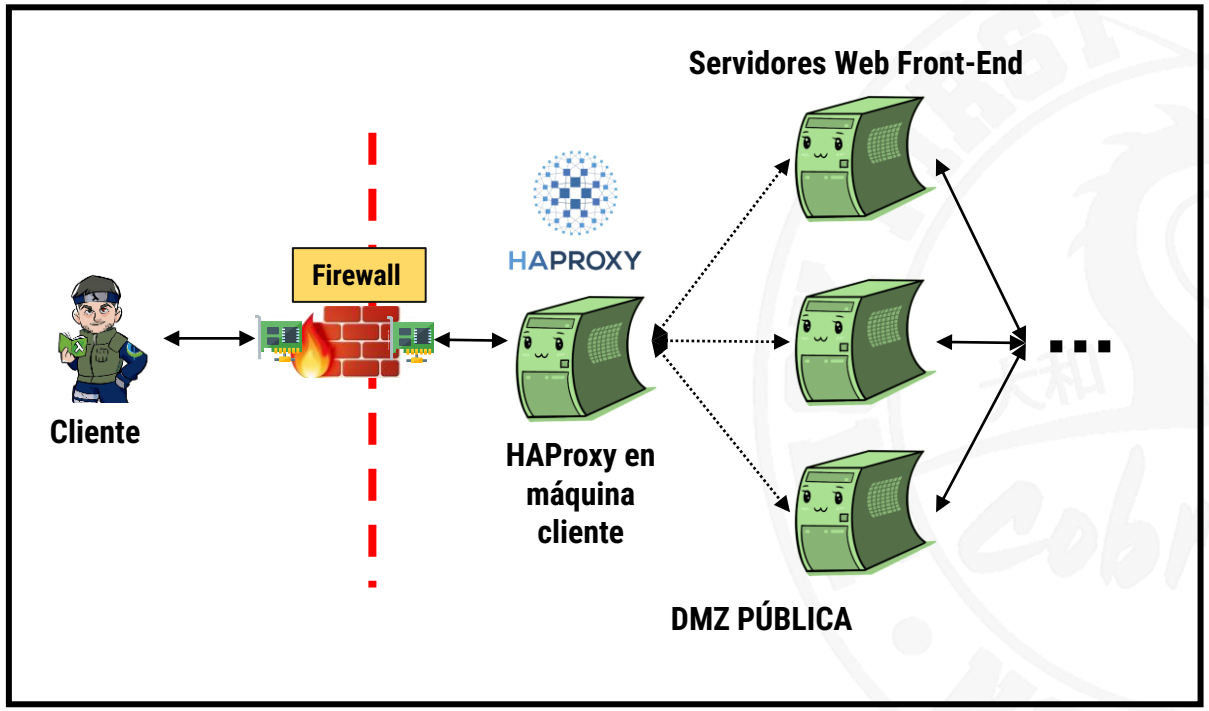
Tipos de Load Balancers

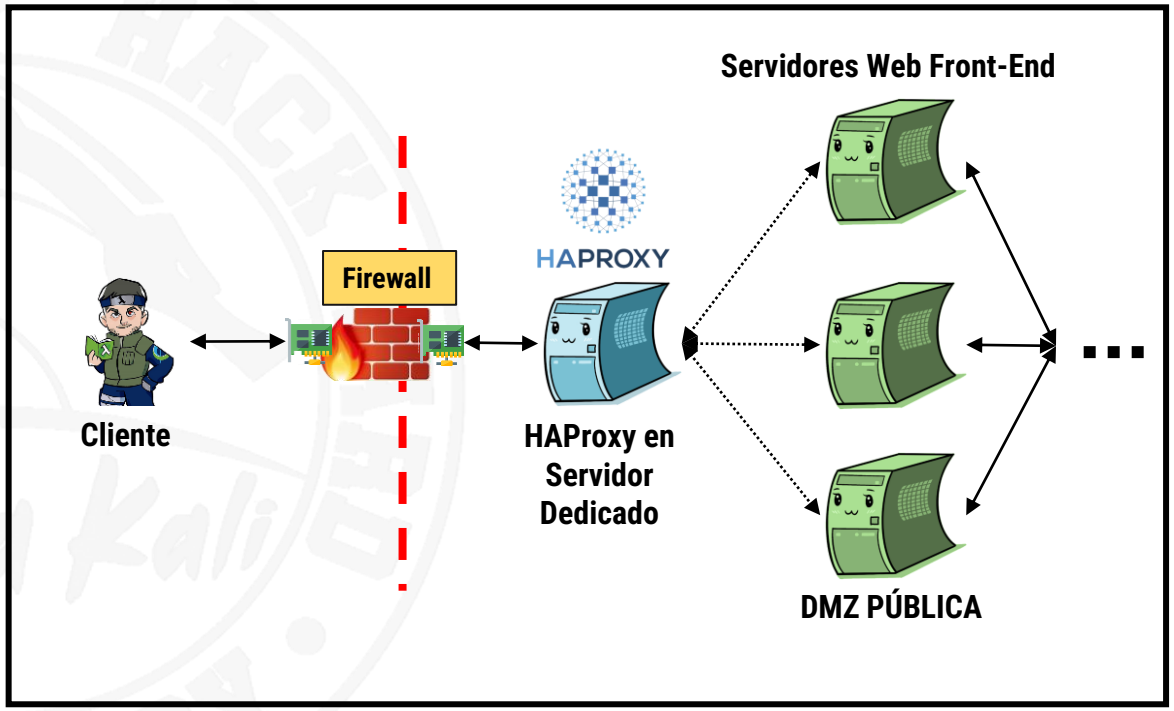
José Manuel
Redondo López

- **LB Hardware** (Ejs.: **Citrix ADC (Antiguo NetScaler), F5 Big-IP, ...**)
  - Caros, hardware dedicado que hay que comprar (https://www.citrix.com/es-es/products/citrix-adc/)
  - Mayor rendimiento que los de software
  - Su configuración puede ser compleja
- **LB Software** (Ejs.: **Round-Robin DNS**, **módulos del servidor web**)
  - Baratos (integrados con productos ya en uso, no se necesita hardware especial)
  - Más fáciles de configurar
  - **Los servidores web Apache 2 y Nginx** tienen módulos instalables que te dan ese soporte fácilmente
- **LB Híbridos**
  - Software de LB (ej.: HAProxy) ejecutándose en un sistema no dedicado (Ej.: PC estándar)
  - Se pueden ejecutar en la máquina del cliente (distribuye las peticiones salientes a las diferentes unidades de servicio dedicadas)
  - O en un servidor intermedio (en cuyo caso cumple las funciones de un Smart Client vistas)

Load Balancers Híbridos

José Manuel
Redondo López

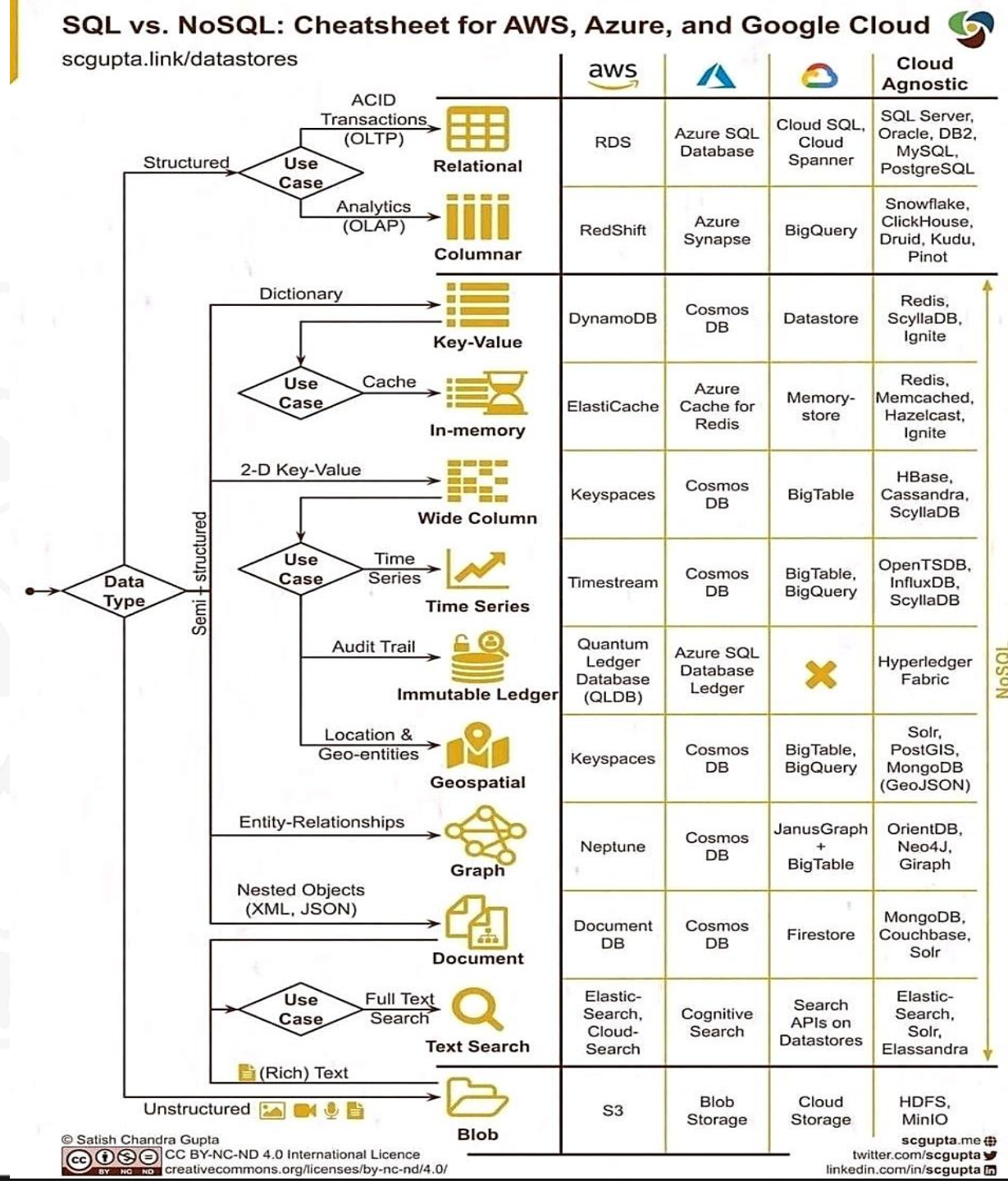




Aspectos a la hora de seleccionar SGBD

José Manuel
Redondo López

- **Tipo de BD** (SQL/NoSQL…)
- **Uso de sharding**: partir la BBDD o sus tablas entre varias máquinas
  - Incrementa el rendimiento, la disponibilidad, y el balanceo de carga
  - Al ser más fácil introducir escalabilidad horizontal en lugar de vertical
- **Creación de Índices**: mejora la velocidad de lectura de los datos
  - A costa de disminuir la de escritura (actualizar datos + índices)
- **Todo esto se debe tratar en asignaturas que lidien con SGBD**
  - Sistemas de Persistencia de Objetos



Redundancia

Evitando perder un servicio

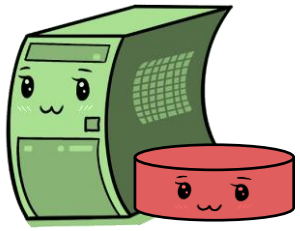
Redundancia y replicación

José Manuel
Redondo López

- La duplicación de datos y servicios críticos **incrementa la fiabilidad del sistema**
- Si algo es crítico, tenemos que asegurarnos de que hay varias copias en ejecución al mismo tiempo en distintos servidores / bases de datos
  - Es seguro si falla algún componente de nuestro sistema
  - Nos da sistemas de respaldo en caso de fallos críticos
- Dos estrategias generales
  - **Respaldos activos**: Múltiples servidores en ejecución al mismo tiempo sincronizados 100%. Si uno falla, los demás asumen la carga (failover)
  - **Espejos**: Servidores activos y servidores "espejo" no accesibles en primera instancia. Los datos se replican de los primeros a los segundos
    - Si alguno de los primarios falla, los secundarios entran en servicio a cubrir el problema

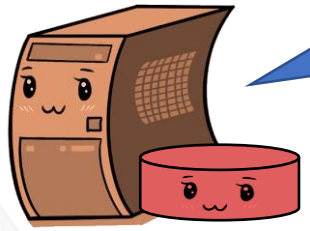
Respaldo activo vs Espejo

José Manuel
Redondo López



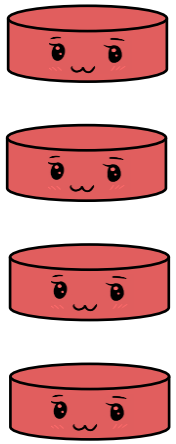
Servidor Primario

Failover



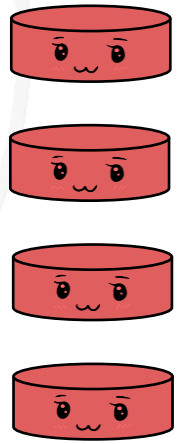
¡Yo te cubro!

Servidor Secundario



Datos Activos

Replicación de datos (Mirroring)



¡Somos clones!

Datos replicados (Mirrored Data)

Objetivos de redundancia

José Manuel
Redondo López

- **A qué hay que aspirar por lo general**
  - A que todos los nodos sean **lo más independientes posible** (o totalmente)
  - A que **no haya un sistema centralizado** que maneje el estado de los servicios
    - El sistema es más escalable
    - Si lo perdemos, **también perdemos la capacidad de tener redundancia**
  - A que se puedan **añadir nuevos servidores** sin condiciones especiales
- **Con esto el sistema general será más resistente a fallos**
  - No hay SPOF (Single Point Of Failure)
- **La redundancia de servidores se consigue con clústeres**
  - **Grupos de servidores unidos** mediante una red de alta velocidad
  - El conjunto **es visto como un único servidor** (si uno del grupo se cae, el cliente no se entera)
  - Los sistemas operativos Linux y Windows tienen tecnologías propias de clustering
  - También algunos servidores de aplicaciones, como Tomcat
    - https://www.mulesoft.com/tcat/tomcat-clustering

Redundancia en almacenamiento de datos

José Manuel
Redondo López

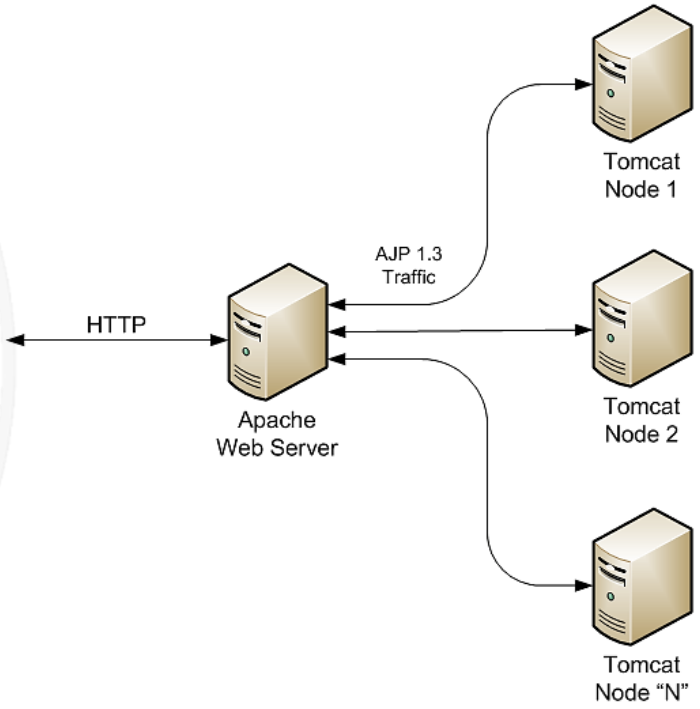
- **En infraestructuras grandes se suelen lograr con NAS (Network-Attached Storage)**
  - Hardware con software especializado que nos permite implementar la política de redundancia que más útil nos resulte
    - Primario/secundario, mirrors activos... (**políticas RAID**)
      - https://es.wikipedia.org/wiki/RAID
  - Se conectan a la red y se venden con distintas capacidades (bahías de discos)
    - Ej.: https://www.amazon.es/TS-673A-8G-Bay-2-2GHZ-4C-EXT/dp/B08XD4F57G/ref=sr_1_10

  - También se pueden usar para **guardar logs de forma centralizada** (ver red de administración del SNI)
- **Si no tenemos presupuesto o una necesidad tan grande, has soluciones software**
  - Puedes instalarlas **sobre cualquier PC/MV que tengas** con capacidad suficiente
    - Mismos servicios, pero mayor riesgo de fallos...
  - TrueNAS: https://www.redeszone.net/tutoriales/servidores/truenas-core-guia-instalacion-configuracion-nas/
  - FreeNAS: https://www.jorgedelacruz.es/2019/08/05/freenas-instalacion-y-configuracion-inicial-de-freenas-11-x-como-vm-dentro-de-vsphere/

CLUSTERING WEB PARA REDUNDANCIA Y BALANCEO DE CARGA

José Manuel
Redondo López

- **Una forma de lograr ambas propiedades en contexto web es combinando servicios que te den ambos**
  - **Kubernetes + contenedores**: Visto al final de la asignatura
  - **Servidores web + servidores de aplicaciones**: El 1º para balanceo y el 2º para redundancia
    - Ej.: En vuestra futura asignatura "Arquitecturas y Diseño de Sitios Web", usareis **clústeres de aplicaciones con Tomcat**
    - Estos se pueden combinar con el servidor web Apache 2
      - Apache 2 no tiene ninguna aplicación web , solo distribuye las peticiones
      - Los Tomcat se encargan de alojar réplicas de la aplicación para servir las peticiones que les lleguen
    - Ejemplos de instalación
      - https://tomcat.apache.org/tomcat-9.0-doc/cluster-howto.html
      - https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_and_managing_high_availability_clusters/assembly_configuring-active-passive-http-server-in-a-cluster-configuring-and-managing-high-availability-clusters
      - https://www.linuxtechi.com/high-availability-apache-cluster-on-rhel/



Balanceo de carga y redundancia web con Apache 2 + Tomcat. Fuente: https://vcfvct.wordpress.com/2012/12/18/tomcat-clustering-setup-using-mod_proxy/

El problema del clustering y las sesiones

José Manuel
Redondo López

- **Este esquema de trabajo suena muy bien, pero tiene un problema en web**
  - Tanto en entornos de nube (Ej.: Kubernetes) como on-premises
  - El balanceo de carga redistribuye las peticiones con criterios **que no tienen en cuenta la aplicación**
  - Si la web abre una sesión para el usuario, **lo hará en el nodo Tomcat que le toque** en ese momento
  - ¿Qué pasa entonces si en las siguientes peticiones le "toca" otro nodo?
    - Efectivamente, **el usuario no tendrá sesión en él** y perderá todo lo que haya hecho
    - Esto es inaceptable, y debemos enviar a toda costa que nos ocurra
- **El precio a pagar por la redundancia y el balanceo de carga es una mayor complejidad en el manejo de sesiones de nuestras aplicaciones**
  - Si no las hemos diseñado para tener esto en cuenta, tendremos problemas
    - Habría que cambiar el diseño y el código de la aplicación
  - Las aplicaciones web debemos diseñarlas pensando ya en si van a ser escalables
    - Usando una solución para la gestión de contextos y sesiones como las de la siguiente transparencia

El problema del clustering y las sesiones: Soluciones

José Manuel
Redondo López

- **Sesiones "sticky"**: A cada cliente lo atiende siempre el mismo nodo que lo atendió la primera vez, por lo que desaparecen los problemas vistos ☺
  - Riesgo de sobrecarga de un nodo, pero implementación más sencilla posible
- **Sistema de replicación automática de sesiones integrado** de Tomcat
  - Tomcat tiene un sistema que se encarga de replicar de manera activa las sesiones entre todos sus nodos activos en tiempo real
  - Rápido, funcional y transparente (una vez activo no hay que hacer nada), **pero podría saturar la red**
  - Puede hacerse con el software integrado en Tomcat para monitorizar los nodos "vivos" o con software de 3ºs como Pacemaker. Ejs.:
    - Que aporta detección de fallos, de integridad de los nodos, etc.
    - https://clusterlabs.org/pacemaker/doc/2.1/Clusters_from_Scratch/html/
    - https://vsgump.medium.com/app-deployment-setup-on-ubuntu-tomcat-cluster-apache2-and-mod-jk-load-balance-with-session-c8cc656e3db6
    - https://www.server-world.info/en/note?os=Ubuntu_22.04&p=pacemaker&f=5
- **Guardar la sesión en una caché distribuida centralizada**: Siguiente sección

Caché

Mejorando la capacidad de respuesta de nuestros sistemas

¿Para qué sirve la caché?

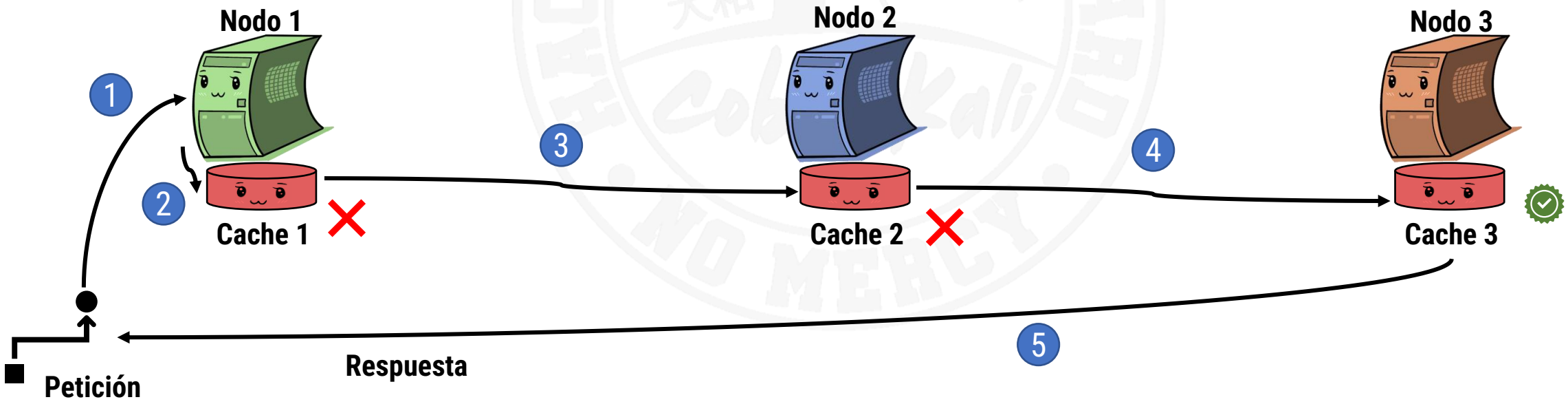
José Manuel
Redondo López

- **La caché se usa en casi cualquier capa de la computación**
  - En este caso podemos encontrarla en el servidor de aplicaciones
  - Se pone una caché antes de recibir las peticiones: si la petición está en la caché, se sirve de allí, sino se acude al servidor
- **Puede situarse en**
  - **La RAM**: muy rápida
  - **El disco local del nodo**: más rápido que ir a datos situados en nodos externos (ej. BBDD)
- **Problema: Necesitamos balanceo de carga para escalar horizontalmente un servicio**
  - Pero si el LB distribuye las peticiones aleatoriamente, habrá más fallos de caché
  - Las mismas peticiones se distribuyen entre más nodos ¡que pueden no tenerlas cacheadas! La caché pierde efectividad
- **¿Solución? Cachés distribuidas y globales**

Cachés distribuidas

José Manuel
Redondo López

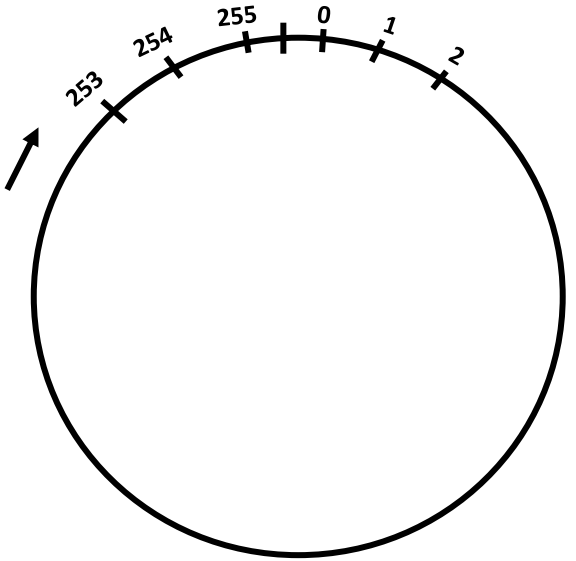
- **Se usa una función hash consistente (consistent hashing) para distribuir los elementos de la caché entre los diferentes nodos**
- **El problema es cuando uno de esos nodos falla**
  - Se puede resolver guardando varias copias de los datos en distintos nodos, pero es más complejo
  - Siempre se puede ir a leerlo al origen, en cualquier caso
  - Es muy fácil aumentar el espacio de la caché añadiendo nuevos nodos

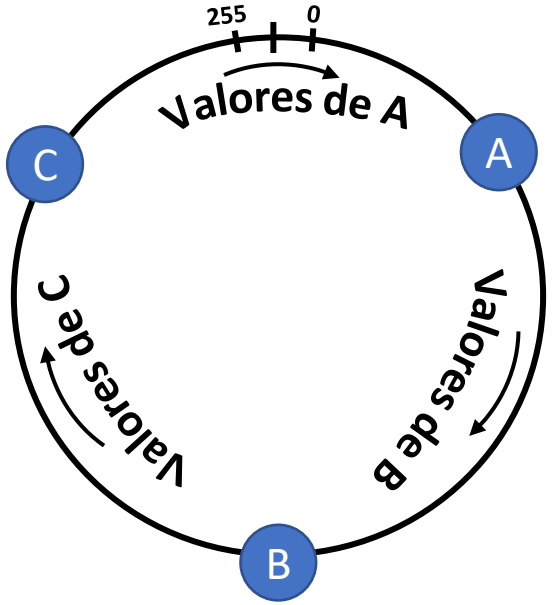


Consistent Hashing

José Manuel
Redondo López

- Es una tabla hash típica pero distribuida y especial para cachés distribuidas
- Minimizan la reorganización de la caché si el sistema escala hacia arriba / hacia abajo
- Solo se necesitan remapear k / n claves
  - **K**: número total de claves
  - **N**: número total de servidores
- Funcionan con un "**algoritmo del reloj**"
  - La función hash saca un rango de valores. Ej.: de 0 a 256
  - Se ponen en un círculo, subdividiéndolo a partes iguales
  - Se parte luego el círculo en trozos, uno por cada servidor disponible (En el ejemplo son 3: A, B y C)
    - No tienen por qué ser trozos iguales (diferentes capacidades de caché)

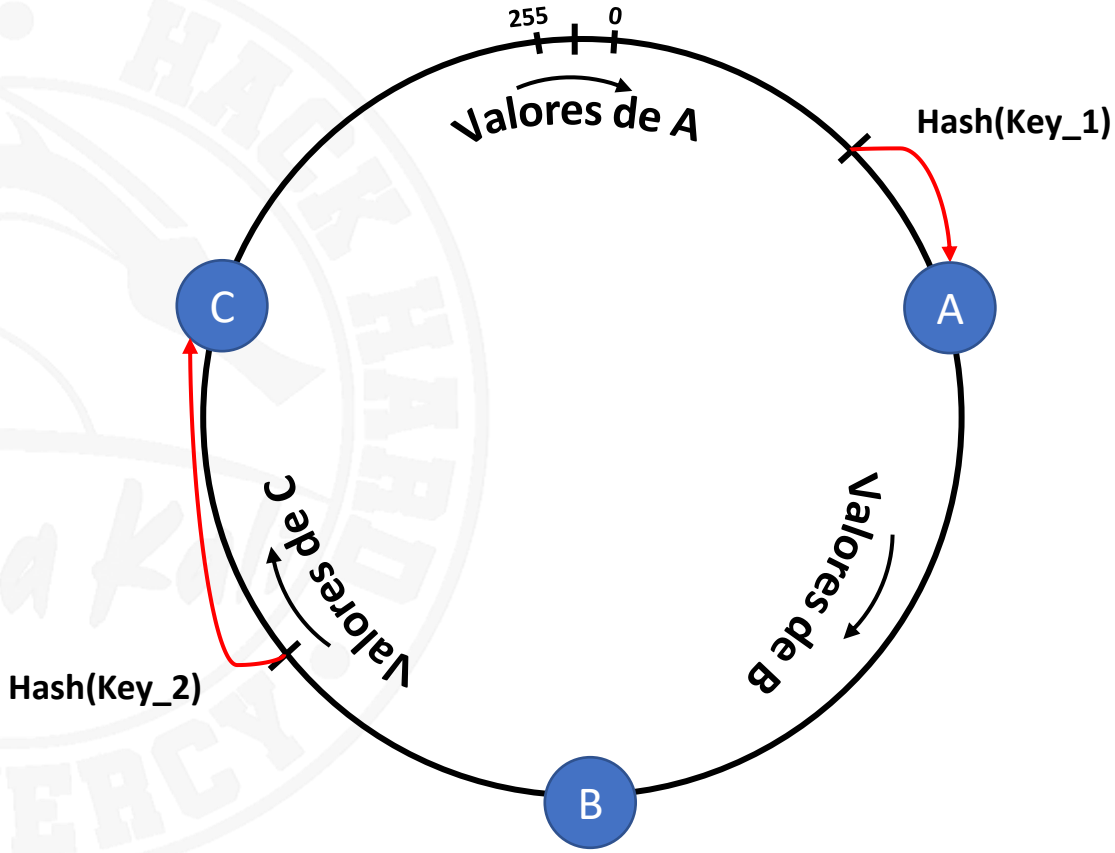




Consistent Hashing: Cómo funciona

José Manuel
Redondo López

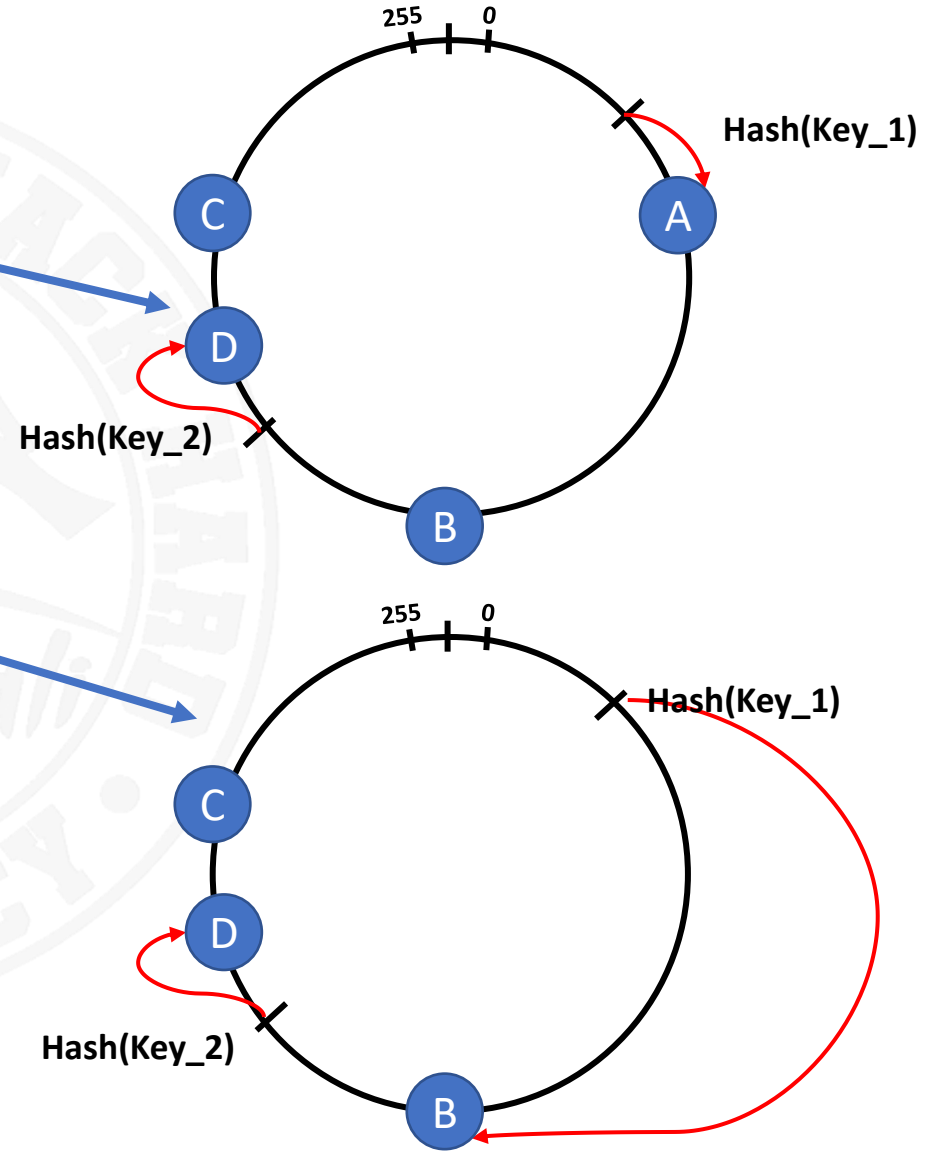
- **Ahora para saber en qué servidor se guarda un valor**
  - Se hashea a un valor (en el rango)
    - En el ejemplo `key_1` y `key_2`
  - Se avanza en el circulo en el sentido de las agujas del reloj hasta encontrar el servidor que se encarga de esos rangos en el círculo
  - Se mapea el valor a ese servidor
    - `key_1` se guarda en el servidor A
    - `key_2` en el servidor C



Consistent Hashing: cómo funciona

José Manuel
Redondo López

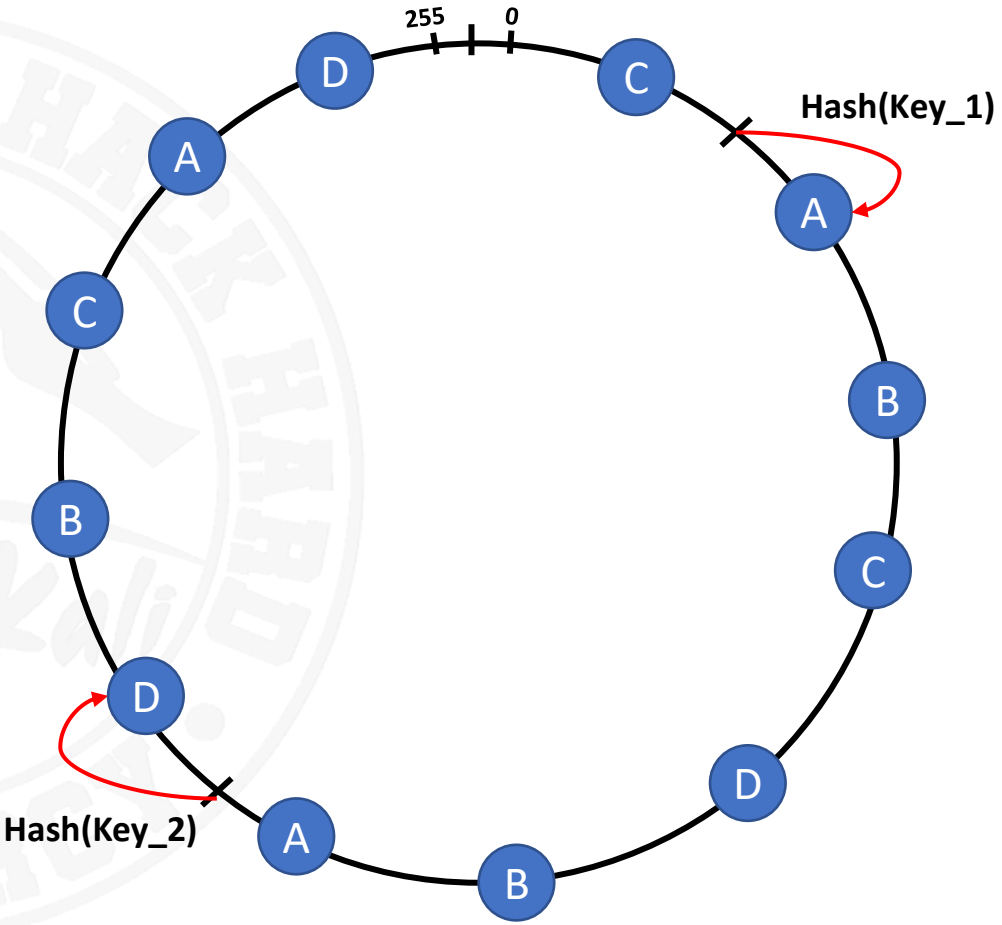
- **Añadir un servidor nuevo (D) hace que** `key_2` **se mueva a D**
  - El espacio de claves de C queda muy reducido, pero puede ser una decisión de diseño
- **Quitar un servidor (A) hace que** `key_1` **se mueva a B**
  - B ahora contendría muchas más claves...
- **¡Como puede verse, este algoritmo de caching permite escalar los servidores sin demasiada dificultad!**



Consistent Hashing: implementación real

José Manuel
Redondo López

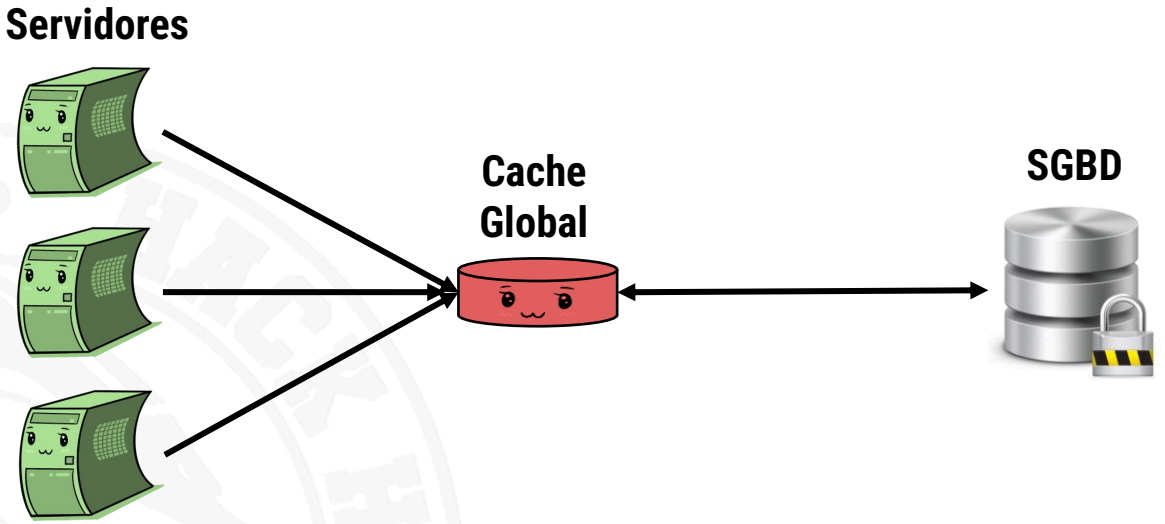
- En los casos de uso reales, no se mapean los nodos a un solo punto de este circulo
- Se mapean varios puntos
- Mayor nº de réplicas implica una distribución más igualitaria y un mejor balanceo de carga

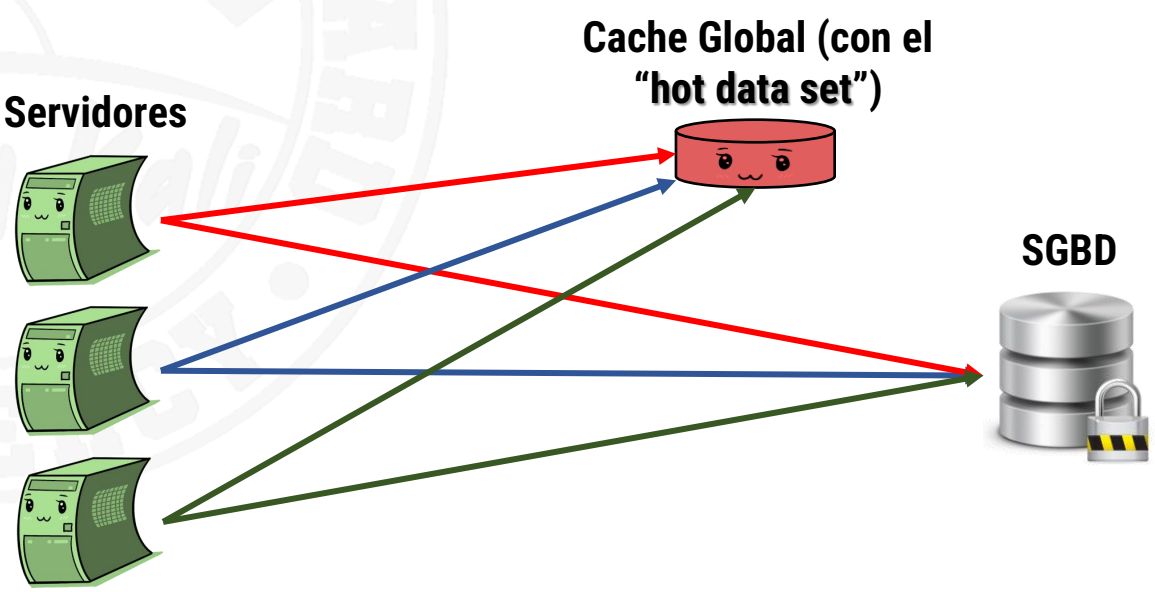


Cachés globales

José Manuel
Redondo López

- **Una caché para todos los nodos**
  - En un servidor de caché o de archivos (que sea más rápido que donde se guarda la información original)
  - **La caché sabe cómo extraer datos del SGBD**
- **Difícil de manejar si aumenta el nº de clientes / peticiones**
- **Efectivo si**
  - El conjunto de datos a cachear es conocido de antemano / fijo
  - Se usa hardware especial con I/O muy rápida
- **Dos formas:**
  - **Hot data set**: La caché guarda los datos más usados y va a por los fallos a la BBDD
  - **Application logic cache**: Cuando la aplicación entiende mejor los datos más usados que la caché





Caché eviction

José Manuel
Redondo López

- La cache tiene un tamaño finito, por lo que habrá que liberar parte de los datos cacheados para dejar espacio a los nuevos
- Para ello se suelen seguir políticas sencillas, entre otras:
  - **First In First Out (FIFO)**: Se descartan los datos más antiguos de la caché
  - **Last In First Out (LIFO)**: Se descartan los datos que han sido los últimos en cachearse
  - **Least Recently Used (LRU)**: Se descartan los datos menos usados recientemente
  - **Most Recently Used (MRU)**: Se descartan los datos que se han usado más recientemente
    - Aunque pueda parecer contradictorio, funciona muy bien con patrones de accesos aleatorios
  - **Least Frequently Used (LFU)**: Se descartan los datos usados menos frecuentemente
  - **Random**: Se descartan datos aleatoriamente

Caché invalidation

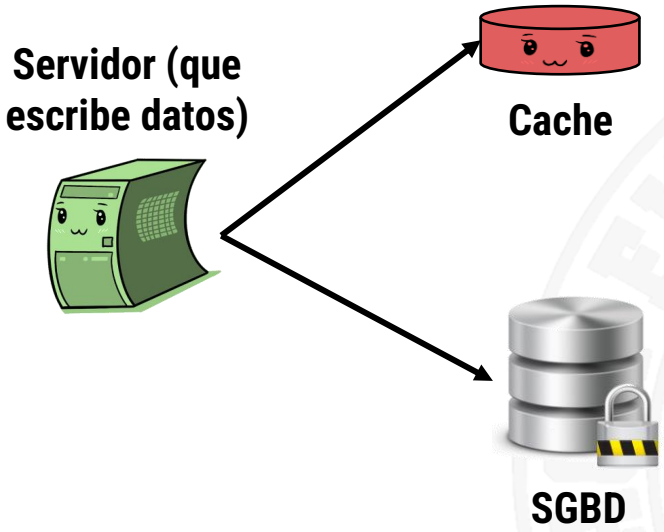
José Manuel
Redondo López

- **Los datos en la caché deben ser coherentes con los de la BBDD**
- **Si se modifican, el contenido correspondiente en la caché debe eliminarse según una de estas estrategias**
  - **Write-through cache**: Los datos se escriben a la vez en la caché y en la BD, logrando consistencia máxima y tolerancia a fallos
    - Se paga en latencia, **disminuyendo el rendimiento** (2 escrituras)
  - **Write around cache**: Las peticiones de lectura de un dato escrito nuevo se consideran fallos de caché automáticos
    - No se satura la caché con escrituras, pero hay **más latencia de lecturas**
  - **Write back cache**: Los datos se leen y se escriben en la caché
    - Tras un tiempo allí, o bajo ciertas condiciones, la caché escribe los datos a la BBDD
    - Baja latencia y mucho **througthput** de escritura si la aplicación hace muchas escrituras
    - Posibilidad de pérdida de datos (solo hay una copia en la caché, si se pierde sin guardar hay problemas)

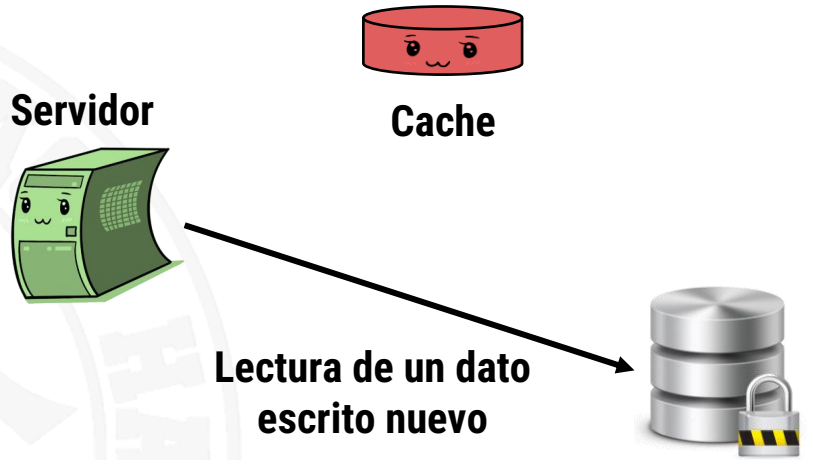
CACHE INVALIDATION

José Manuel
Redondo López

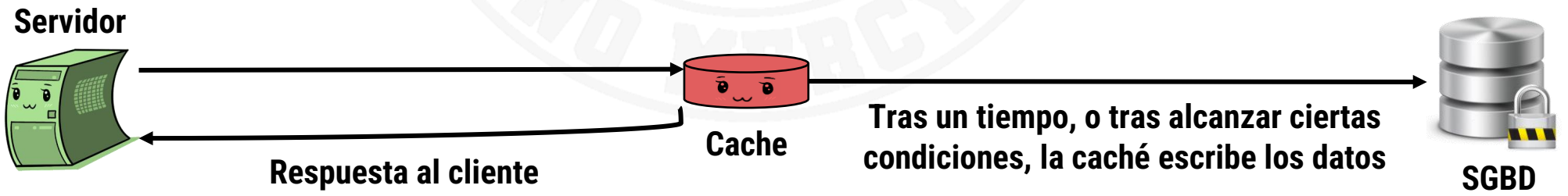
Write-through cache



Write-around cache



Write-back cache



Ejemplo de cachés locales: Memcached

https://memcached.org/

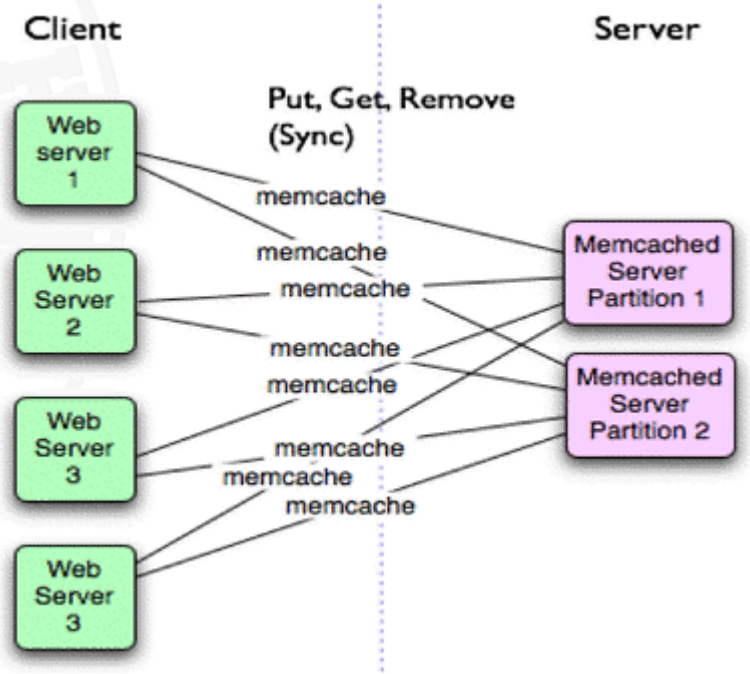
José Manuel Redondo López

- **Es un sistema de caché distribuida en RAM muy popular para acelerar aplicaciones**
  - Usado por Netflix, por ejemplo ☺
  - Tiene una velocidad de acceso alta y maneja grandes cantidades de datos
  - Se suele usar para mejorar la velocidad de acceso a BBDD
  - También puede usarse para guardar sesiones en un lugar centralizado
    - En caso de tener un **clúster de servidores** por temas de redundancia / alta disponibilidad
  - Tiene **librerías** para trabajar con ella en muchos lenguajes
- **Más información e instalación**
  - https://www.dragonflydb.io/guides/memcached
  - https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-memcached-on-ubuntu-22-04

Uso de Memcached en Python

```
import memcache

# Connect to Memcached
Server mc = memcache.Client(['localhost:11211'])
# Store a Value in the Cache
mc.set('my_key', 'my_value')
# Retrieve a Value From the Cache
result = mc.get('my_key')
print(result)
# Output: 'my_value'
```



Fuente: https://pressroom.hostalia.com/white-papers/memcached/

Content Delivery Networks (CDNs)

José Manuel
Redondo López

- **Es una caché especialmente diseñada para webs que sirven gran cantidad de contenido estático de gran tamaño**
  - Por ejemplo, multimedia como videos, ISOs, etc.
- **Se puede**
  - Contratar como servicio: Cloudflare, TransparentEdge…
  - Tener tu propia CDN (presupuesto…)
  - Usar workarounds si no se tiene el presupuesto para las otras dos
    - Servir los ficheros estáticos en un subdominio DNS separado (`static.tusitio.com`)
    - Usando un servidor web “aligerado” (con lo mínimo para dar ese servicio)
    - Es más fácil mover este DNS separado a una CDN luego si hace falta

Ejemplo de CDN: CloudFlare

José Manuel
Redondo López

- CDN en nube que además proporciona una capa de seguridad adicional a toda la infraestructura
  - Las solicitudes al firewall perimetral externo se filtran primero por la red CDN de CloudFlare
- Nos da características que requieren un presupuesto alto si las queremos implementar nosotros
  - **Protección anti DDoS completa y altamente probada**
  - **Garantiza conexiones https adecuadas**
  - **Firewall de aplicaciones web**: `mod_security`
    - Con un conjunto de reglas personalizado que da protección avanzada contra XSS, inyección SQL, XSRF/CSRF...
    - Envenenamiento de parámetros, filtrado de archivos privados, fugas de datos debido a una configuración incorrecta,... (ver "***F-103 Blas de Lezo***")
  - **Técnicas de "inteligencia colectiva" para prevenir nuevas amenazas**
  - **Protección/bloqueo basado en la reputación contra conexiones de cliente potencialmente malintencionadas**
  - **Protección contra el spam de comentarios (comment span)**
  - **Protección contra el hotlinking o el robo de contenido**
  - ...

Reverse Proxies

Máquinas que “dan la cara” por las demás

Reverse Proxies

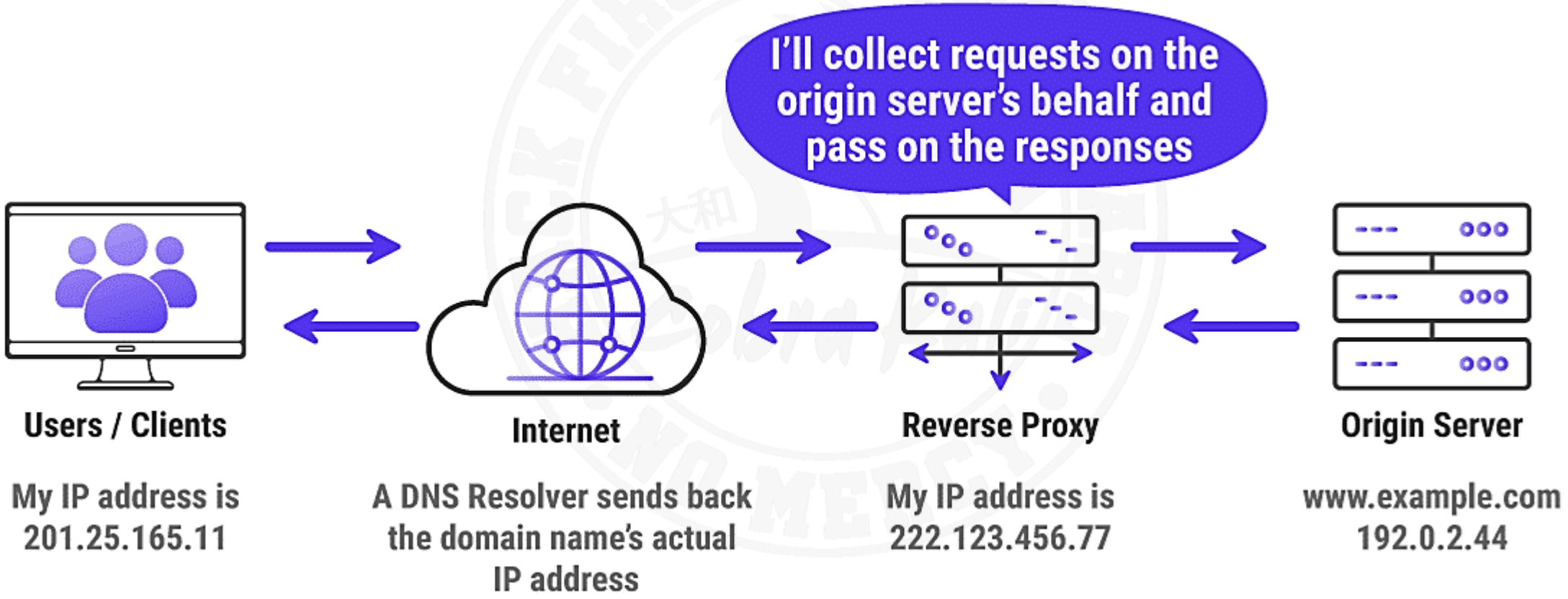
José Manuel
Redondo López

- Son máquinas que se ponen "en medio" entre los clientes y la web con varios propósitos
  - **Filtrado y registro (log) de peticiones**
  - **Transformación de peticiones**: añadir / modificar cabeceras HTTP, cifrado / descifrado, compresión…
  - **Coordinación**: a qué servidor se redirigen para ser atendidas
    - Ocultación de los detalles de la infraestructura subyacente a los clientes externos
  - **Caché de peticiones**
  - **Optimización de tráfico** minimizando las lecturas vía
    - Collapsed forwarding: juntar peticiones de acceso a los mismos datos en una sola
    - Spatial locality: juntar peticiones que hagan acceso a datos que estén espacialmente cerca
  - **Implementar mecanismos de seguridad** para toda la infraestructura ("F-103 *Blas de Lezo*")

Reverse Proxies

José Manuel
Redondo López

- Con uno de estos “dando la cara” **ningún cliente sabrá realmente qué o quién atiende sus peticiones**



https://kinsta.com/es/blog/proxy-inverso/

Colas de Peticiones (aka "Message brokers")

Optimizando el servicio de peticiones

Colas (Queues) (aka "Message brokers")

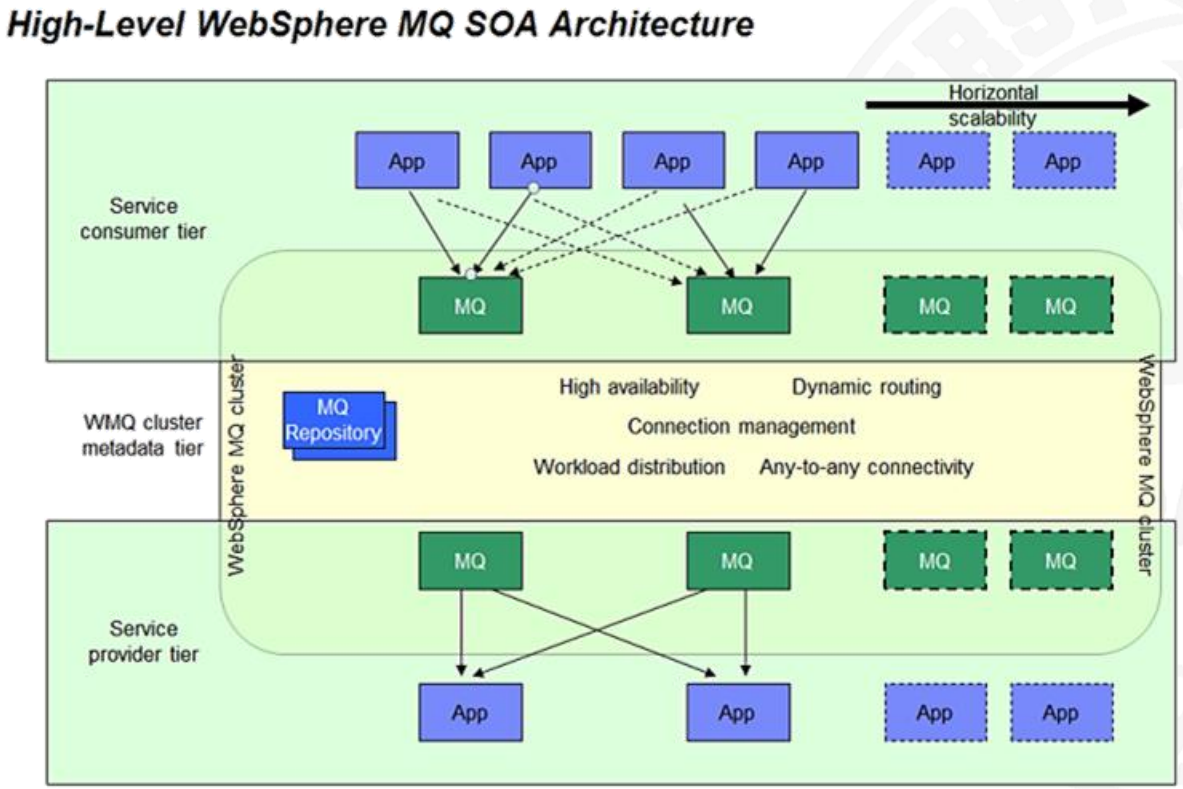
José Manuel
Redondo López

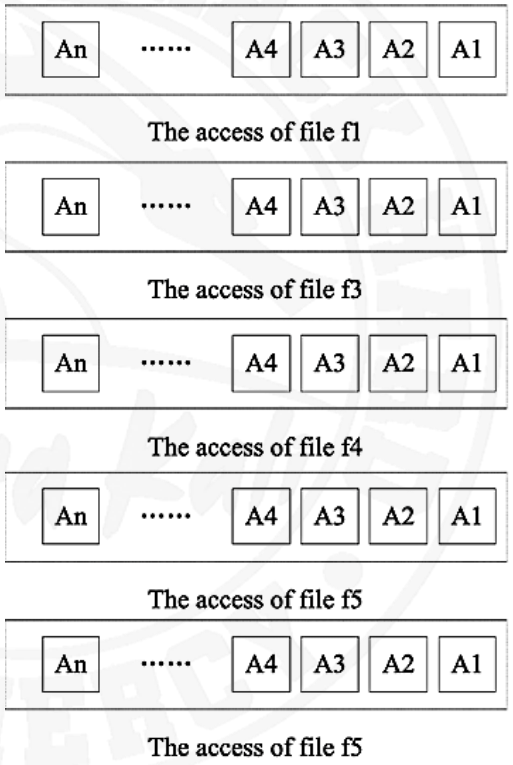
- **Mejoran el manejo de peticiones en sistemas distribuidos grandes**
- **Para un mejor rendimiento/disponibilidad los sistemas deben ser asíncronos**
- **Las colas asíncronas + balanceo de carga pueden conseguir que lo sean**
  - El cliente pide una tarea
  - La cola le da el ACK (recibo) con una "**promesa**" de hacer la tarea en el futuro
  - El cliente puede continuar haciendo cosas, hasta que necesite el resultado obligatoriamente
  - Entonces tendrá que esperar a que se complete...o no porque ya se ha completado ☺
  - Hay límites de tamaño de las peticiones y el número de peticiones en la cola
- **Las colas también proporcionan tolerancia a fallos**
  - Protegen al cliente de fallos en los servicios
  - Las tareas encoladas **se pueden reintentar en otros servidores** si el servidor original falla
  - Introducen QoS (Quality of Service), sin exponer a los clientes a fallos
- **Ejs.: RabbitMQ, ZeroMQ, ActiveMQ, BeanstalkD, Apache Kafka…**

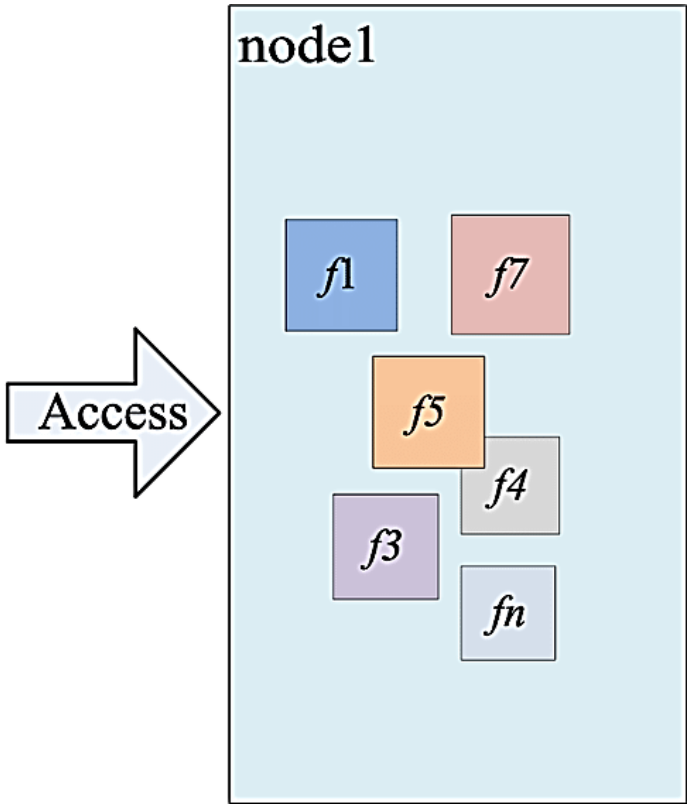
Colas (Queues) (aka "Message brokers")

José Manuel
Redondo López

- Las colas se pueden aplicar en el acceso a ficheros y a servicios







http://www.dreamsdesign.com.ua/b85a5/bowzae.php?37326/1138940.html

https://www.researchgate.net/figure/The-queue-model-used-in-the-storage-cluster-system_fig3_328317330

Colas (Queues): RabbitMQ

https://www.rabbitmq.com/

José Manuel Redondo López

- Es el sistema de colas open source más usado hoy en día
- Permite implementar fácilmente en tus aplicaciones web todas las propiedades vistas relacionadas con la asincronía
- Tiene implementaciones para usarlas desde muchos lenguajes
  - Si tienes un servidor instalado con su software, puedes usarlas directamente en tus aplicaciones
  - https://www.rabbitmq.com/getstarted.html
- Instalación
  - https://www.rabbitmq.com/install-debian.html

Un productor en Python envía un mensaje con el identificador "hello"

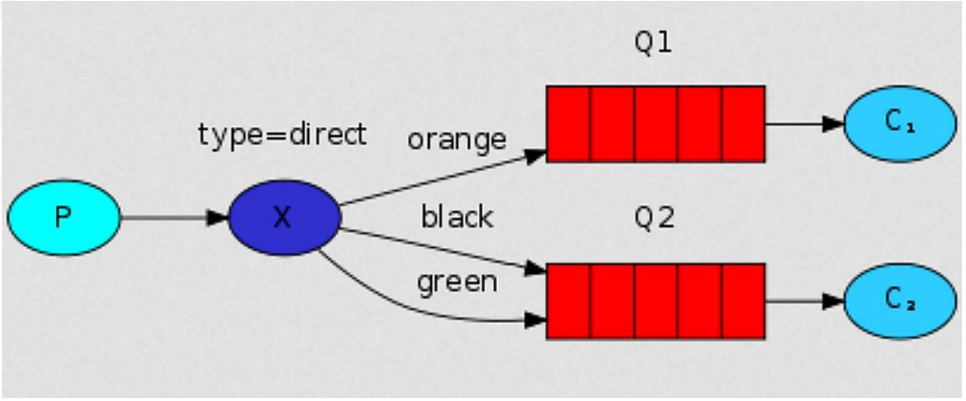
```
channel.basic_publish(exchange='',
routing_key='hello', body='Hello World!')
```

Un consumidor en Python recibe el mensaje con el indicador "hello" (la función callback asegura la asincronía, solo se invoca cuando realmente llega algo)

```
def callback(ch, method, properties, body):
    print(f" [x] Received {body}")

channel.basic_consume(queue='hello',
auto_ack=True, on_message_callback=callback)
```

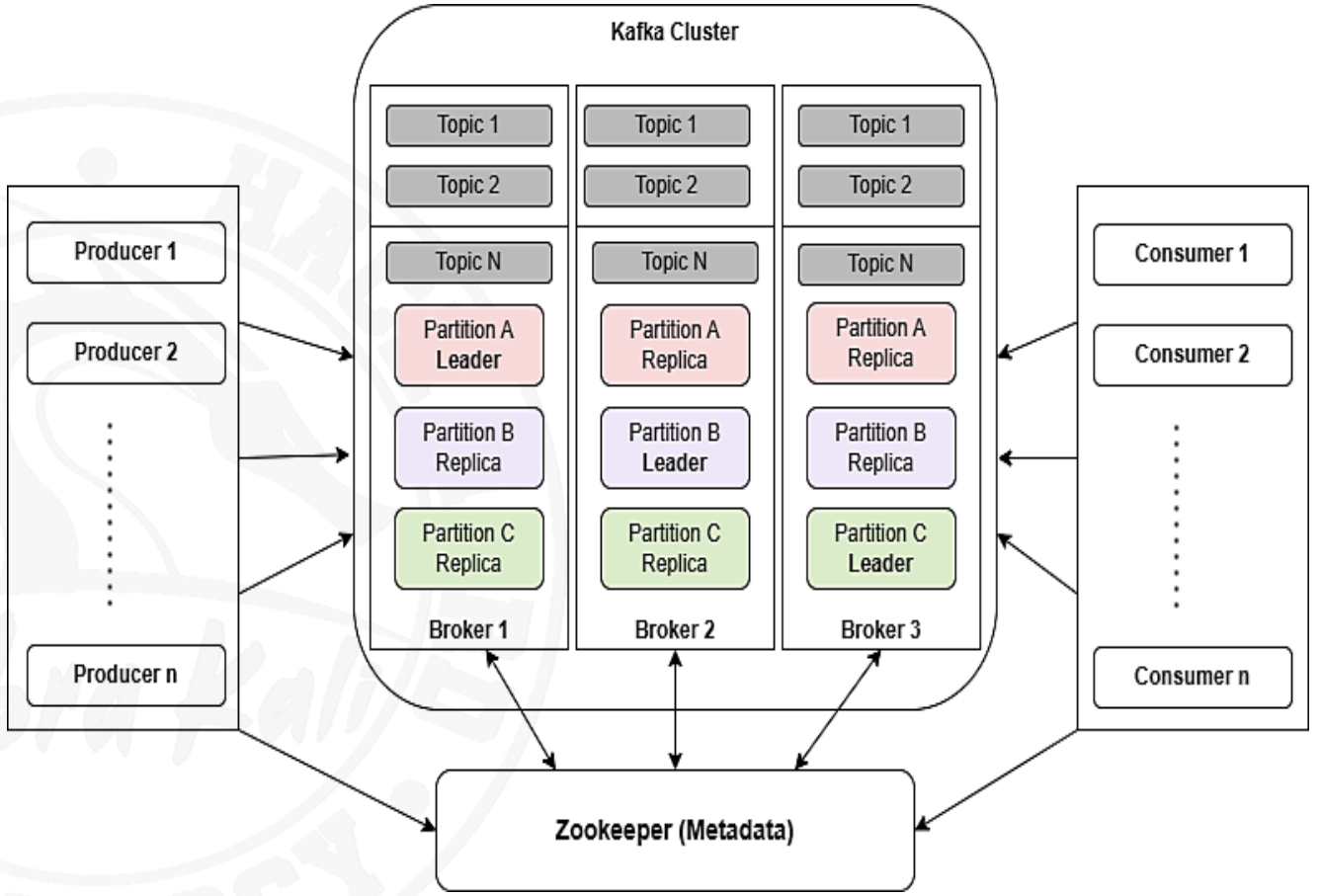
Un productor hace envío selectivo de mensajes a dos consumidores que leed de sus propias colas



Más allá: El event streaming

José Manuel
Redondo López

- **Son colas de peticiones enfocadas en el alto rendimiento**
  - Con baja latencia
  - Y una capacidad de escalado **MASIVA**
- **Su objetivo es manipulación en tiempo real de datos**
- **Sigue el patrón publicación – suscripción**
  - Como los eventos de programación, de ahí el nombre ☺
- **El más popular a nivel empresarial es Apache Kafka**
  - https://kafka.apache.org/
  - https://kafka.apache.org/quickstart



Estructura típica de un clúster de Apache Kafka. Fíjate en que hay muchos proveedores y consumidores, múltiples colas (Topics) y cada una tiene particiones lideres y réplicas para alta disponibilidad. Fuente: https://howtodoinjava.com/kafka/apache-kafka-tutorial/

¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

?

Balanceo de carga

- *¿Entiendes la importancia del balanceo de carga para atender más peticiones, maximizando el uso de tu infraestructura?*
- *¿Sabes distinguir entre balanceo de carga horizontal y vertical y cuál conviene más?*
- *¿Puedes enumerar algoritmos típicos para decidir a qué servidor va cada petición?*
- *¿Eres consciente de los distintos parámetros que pueden afectar el gasto en balanceo de carga (tipo de balanceador, cuántos son necesarios...)?*

Redundancia

- *¿Entiendes la importancia de la redundancia a la hora de crear sistemas fiables y que estén disponibles el mayor tiempo posible?*
- *¿Puedes enumerar los criterios generales de diseño de un sistema redundante y lo que hay que evitar?*

¿Crees que lo has entendido todo? ¡Autoevalúate!

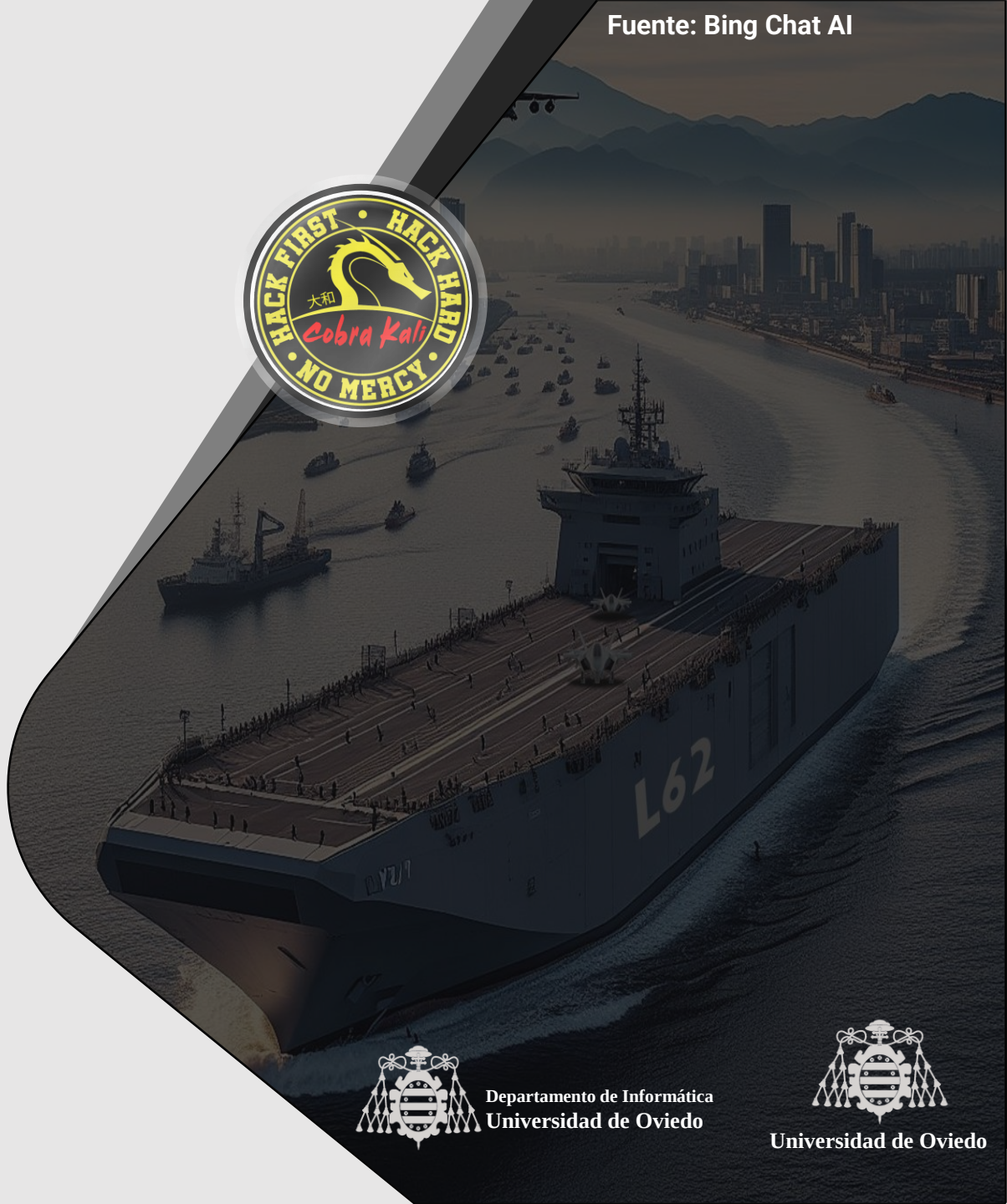
José Manuel
Redondo López

- **Caché**
  - *¿Entiendes el impacto en el rendimiento tan significativo que tiene una buena caché?*
  - *¿Comprendes los problemas que da usar a la vez balanceo de carga y una caché?*
  - *¿Comprendes los principios de una caché distribuida y del consistent hashing?*
  - *¿Entiendes los modos de caché global que existen?*
  - *¿Comprendes las distintas políticas a la hora de borrar o invalidar datos de una caché y la razón de que existan ambos tipos de políticas?*
  - *¿Entiendes la importancia de una CDN como un "servicio de caché externalizado"?*
- **Reverse proxy**
  - *¿Entiendes la importancia y los servicios que da un reverse proxy?*
  - *¿Comprendes como un reverse proxy le da aislamiento a tu infraestructura?*
- **Colas de peticiones**
  - *¿Entiendes cómo una cola de peticiones optimiza el uso de tu infraestructura y por qué proporcionan tolerancia a fallos?*
  - *¿Has entendido correctamente que el modo correcto de funcionamiento de colas es asíncrono?*

< Ir al Índice

Tecnologías de Comunicación con el Servidor

Más allá de la simple petición HTTP



Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿QUÉ VAMOS A VER EN ESTE APARTADO?

José Manuel
Redondo López

- **Cuando creas una aplicación web puedes elegir distintas tecnologías para comunicarte con servidores, y este tema te explica las más usadas**
  - Veremos que la web actual ya no solo se basa en peticiones / respuestas simples como inicialmente
  - Explicaremos la técnica de comunicación con el servidor AJAX Polling
  - Explicaremos la técnica de comunicación con el servidor HTTP Long Polling
  - Explicaremos la técnica de comunicación con el servidor Web Sockets
  - Explicaremos la técnica de comunicación con el servidor Server Side Events

TECNOLOGÍAS PARA COMUNICARNOS CON UN BACKEND

José Manuel
Redondo López

- **Una forma de a optimizar el uso de una infraestructura es decidir qué estrategia de comunicación con el servidor necesitamos**
  - Adaptada a nuestros requisitos / contexto
- **Como ingenieros del software debemos decidir cuál de las estrategias disponibles es más adecuada**
  - Nº de conexiones abiertas soportadas por los servidores
  - Direccionalidad (quién envía a quién) y frecuencia de los datos (ocasionales, frecuentes, periódicos)
- **Estos factores pueden aconsejar alguna de las tecnologías siguientes**
  - Que van mucho más allá que **la petición/respuesta HTTP típica**

https://medium.com/geekculture/ajax-polling-vs-long-polling-vs-websockets-vs-server-sent-events-e0d65033c9ba

AJAX Polling vs HTTP Long-Polling vs WebSockets vs Server Sent Events

José Manuel
Redondo López

- **AJAX polling**: Los clientes están **continuamente** haciendo **polling** al servidor para que le suministre datos
  - Es como el protocolo HTTP, pero las peticiones se hacen a intervalos regulares (Ej.: 0,5s.)
  - Tiene sentido si el servidor está constantemente actualizándose con nueva información
  - **Problema**: el cliente hace muchas peticiones y muchas tienen respuestas vacías: **overhead**
- **HTTP Long Polling (Hanging GET)**: El servidor nunca envía respuestas vacías
  - **Solo envía respuestas cuando hay datos nuevos disponibles**
  - El procedimiento es
    - El cliente hace una petición HTTP y espera por la respuesta
      - Debe programarse para que haga otras cosas mientas espera, y no quedarse "colgado"
    - El servidor no responde hasta que haya algún dato actualizable (u ocurra un **timeout**)
    - Cuando haya una actualización de la información, el servidor manda la respuesta completa al cliente
    - El cliente recibe la respuesta a su petición, que por fin puede procesar
    - El cliente manda otra nueva petición **long poll** nada más recibir la respuesta
      - O tras una pausa predeterminada
  - Sí hay un **timeout** **el cliente debe conectarse de nuevo**

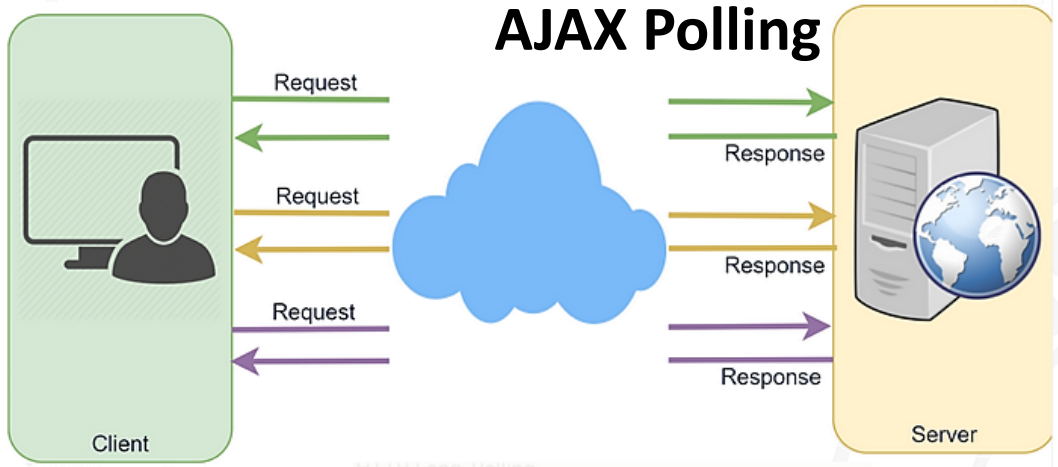
AJAX Polling vs HTTP Long-Polling vs WebSockets vs Server Sent Events

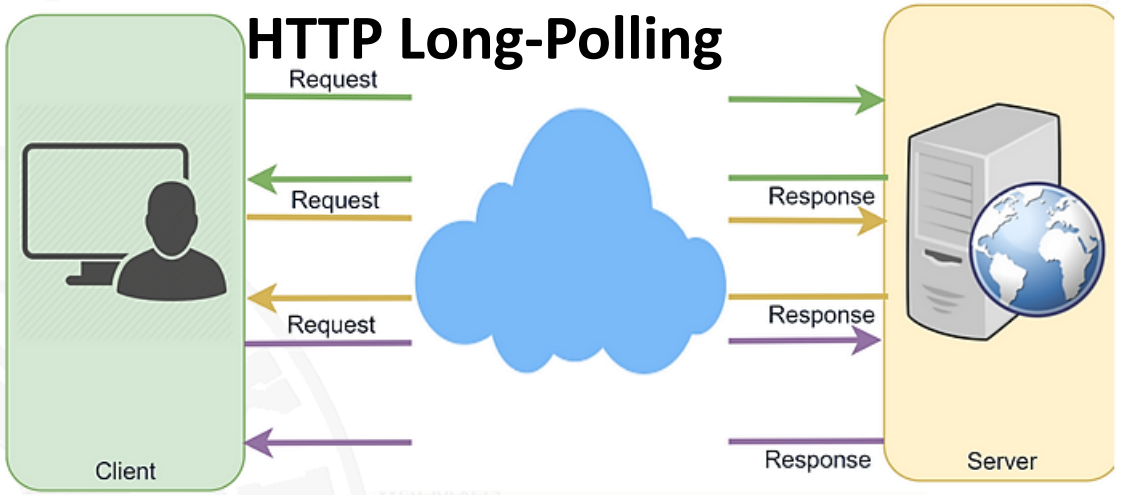
José Manuel
Redondo López

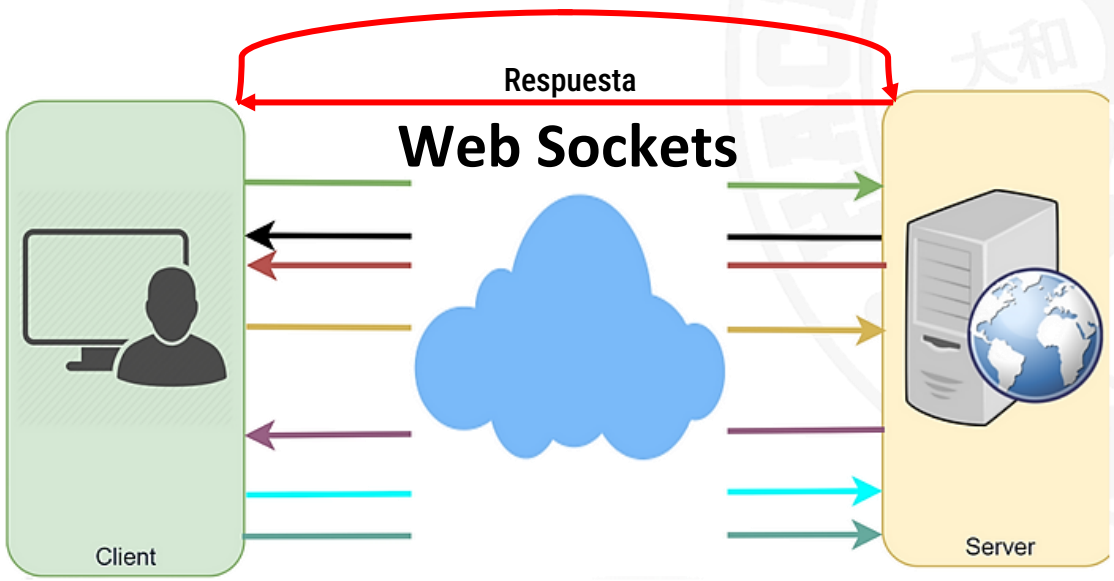
- **Web Sockets**: Canal de comunicación **full-duplex** a través de una conexión TCP única (menor uso de conexiones)
  - Proporciona una "**comunicación persistente**"
    - Cliente y servidor pueden enviar/recibir datos en cualquier momento
  - Es una comunicación que está siempre abierta para carga y transferencia de datos en tiempo real
  - No es adecuada si no hay una necesidad de comunicación bidireccional entre cliente y servidor
- **Server-Sent Events (SSE)**: El cliente establece una conexión con el servidor a largo plazo y persistente
  - El servidor usa esta conexión para enviar datos al cliente cuando haya nuevos disponibles
  - Si el cliente quiere enviar datos al servidor debe usar otra tecnología o protocolo, **la comunicación es unidireccional**
    - Esto obliga a usar alguna de las otras técnicas si el cliente quiere enviarle algo al servidor
  - Es como un manejador de eventos en un lenguaje de programación al uso
  - La mejor estrategia para **enviar datos en tiempo real** de los servidores al cliente
    - O cuando el servidor envía datos periódicamente (en bucle) durante mucho tiempo

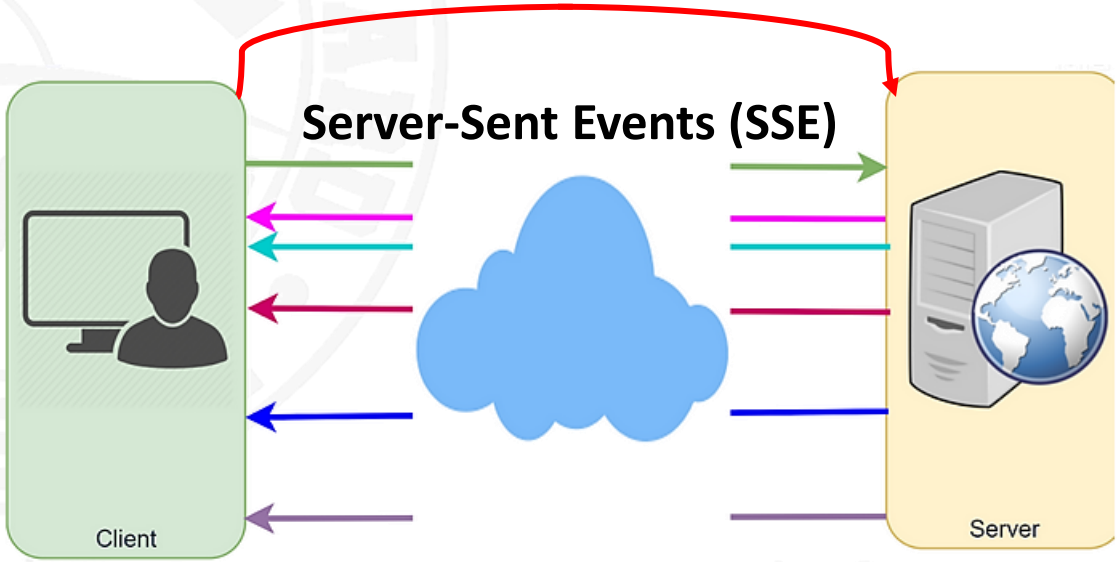
AJAX Polling vs HTTP Long-Polling vs WebSockets vs Server Sent Events

José Manuel
Redondo López







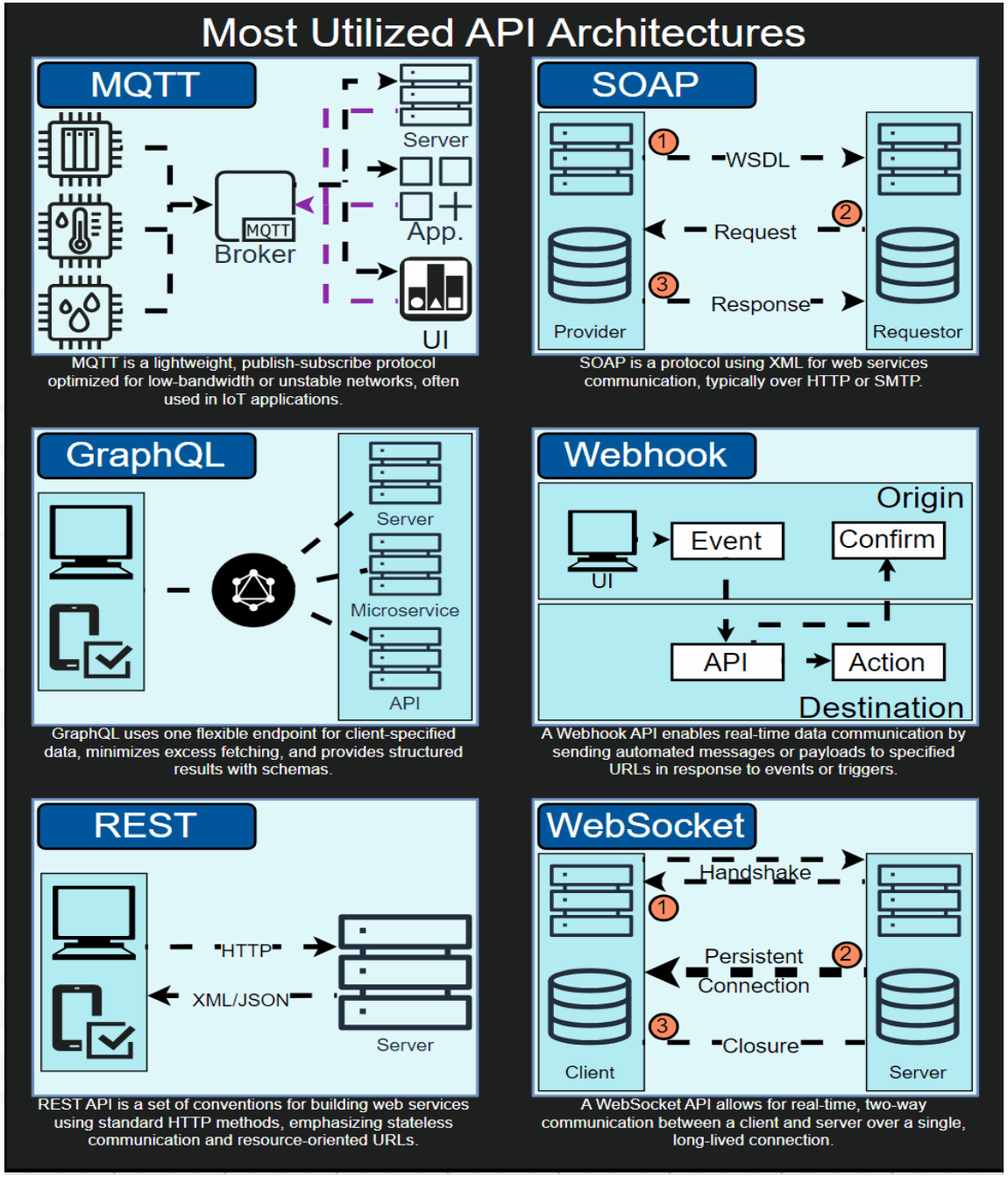


https://medium.com/geekculture/ajax-polling-vs-long-polling-vs-websockets-vs-server-sent-events-e0d65033c9ba

¿Y PARA APIs?

José Manuel
Redondo López

- No solo es importante elegir la tecnología más adecuada para comunicarte con un servidor web
- Si usas una API / servicio web, debes elegir también la que más se adapta a tus requisitos
- Esto lo veréis en la asignatura del máster “Servicios web”



Fuente: Vladimir Romanov (LinkedIn)

¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

- *¿Crees que serías capaz de decidir que técnica de comunicación con el servidor necesita una de tus aplicaciones porque sabes las ventajas y desventajas de cada una?*
- *¿Cuál es el principal inconveniente de AJAX Polling? ¿En qué escenario lo usarías en una de tus aplicaciones?*
- *¿Cuál es el principal inconveniente de HTTP Long Polling? ¿En qué escenario lo usarías en una de tus aplicaciones?*
- *¿Cuál es el principal inconveniente de Web Sockets? ¿En qué escenario lo usarías en una de tus aplicaciones?*
- *¿Cuál es el principal inconveniente de Server Side Events? ¿En qué escenario lo usarías en una de tus aplicaciones?*
- *¿Has entendido las distintas formas de comunicarte con APIs comunes?*

< Ir al Índice

Fuente: Bing Chat AI

Con hipervisores >

Con contenedores >

Tecnologías de virtualización

Hay más tecnologías de virtualización de las que imaginas

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿QUÉ VAMOS A VER EN ESTE APARTADO?

José Manuel
Redondo López

- Entender porque **la virtualización es imprescindible** hoy en día para construir infraestructuras
- Los **tipos de virtualización** con hipervisores existentes
- Las distintas **formas de virtualizar el hardware**
- Las ventajas y desventajas de la **virtualización con hipervisores**
- En qué se basan las tecnologías de **virtualización con contenedores**
- Las ventajas y desventajas de la virtualización con contenedores
- Las **similitudes y diferencias** entre contenedores de SO y de aplicaciones
- Los **casos de uso más adecuados** para los dos tipos de contenedores vistos

¿VIRTUALIZACIÓN?

José Manuel
Redondo López

- **Hace ya mucho tiempo que cada componente de las infraestructuras que hemos mencionado ya no es una máquina física aislada**
  - No tiene sentido (costes de comprar muchas máquinas físicas)
  - Las máquinas tienen tanta potencia que **quedarían infrautilizadas** y desaprovechadas
- **Por tanto, se necesitan mecanismos para ejecutar varios componentes de estas infraestructuras en una misma máquina física**
- **Y eso requiere conocer las diferentes tecnologías de virtualización, sus ventajas y desventajas**
  - Sea en local (on-premises) o en la nube
- **Tenemos que ser capaces de elegir la mejor para hacer lo que queremos**

Tecnologías de virtualización con hipervisores

Nuestras “clásicas” máquinas virtuales

VIRTUALIZACIÓN CON HIPERVISORES

José Manuel
Redondo López

- **Un hipervisor o Virtual Machine Monitor (VMM) es una capa de software crea y gestiona hardware virtual (emulado)**
  - Con ella podemos usar diferentes SO "guest" (**máquinas virtuales**) sobre uno "host"
  - Los SO "guest" pueden ejecutarse sin modificar o modificados (**paravirtualización**)
- **Existen dos tipos**
  - **Tipo 1 (Nativo o bare metal):** el hipervisor se ejecuta directamente sobre el hardware
    - Las máquinas virtuales se ejecutan sobre el hipervisor, que controla los accesos directos al hardware
    - **Ejemplos**: Proxmox VE, Xen (usado en nuestro departamento), Kernel-based Virtual Machine (KVM), Microsoft Hyper-V (Windows), VMware ESXi, Oracle VM Server
  - **Tipo 2 (hosted)**: Un SO "host" se ejecuta sobre el hardware
    - **El hipervisor es una aplicación estándar del mismo**
    - **Ejemplos**: VirtualBox, VMware Workstation, Parallels Desktop, VMware Player, QEMU…
    - ¿Sabéis por qué Hyper-V entra en conflicto con VirtualBox por las instrucciones de la CPU que aceleran la virtualización? Porque son hipervisores **de distintos tipos**

Virtualización con hipervisores: Formas de virtualizar el hardware

José Manuel
Redondo López

- **Completa o nativa.** El hipervisor simula un **hardware** que permite ejecutar aisladamente un SO **sin modificar**
  - Pueden usar **instrucciones para acelerar la virtualización** de las CPUs (VT-x de Intel o AMD-V de AMD) o bien obligar a las CPU a contar con ellas
  - Hoy en día es un requisito que **haya soporte hardware** (en la CPU) para la virtualización
  - **Ejemplos**: VMware Workstation, ESXi y vSphere, VirtualBox, Kernel-based Virtual Machine (KVM), Microsoft Hyper-V, Proxmox VE y Xen
- **Virtualización parcial o paravirtualización.** El hipervisor ofrece un interfaz especial para acceder al **hardware** subyacente
  - El **SO de la máquina virtual tiene que adaptarse (o modificarse)** usando llamadas especiales (hypercalls)
  - **Ejemplos**: Xen también puede virtualizar usando paravirtualización (además de virtualización completa)

Virtualización con hipervisores: Formas de virtualizar el hardware

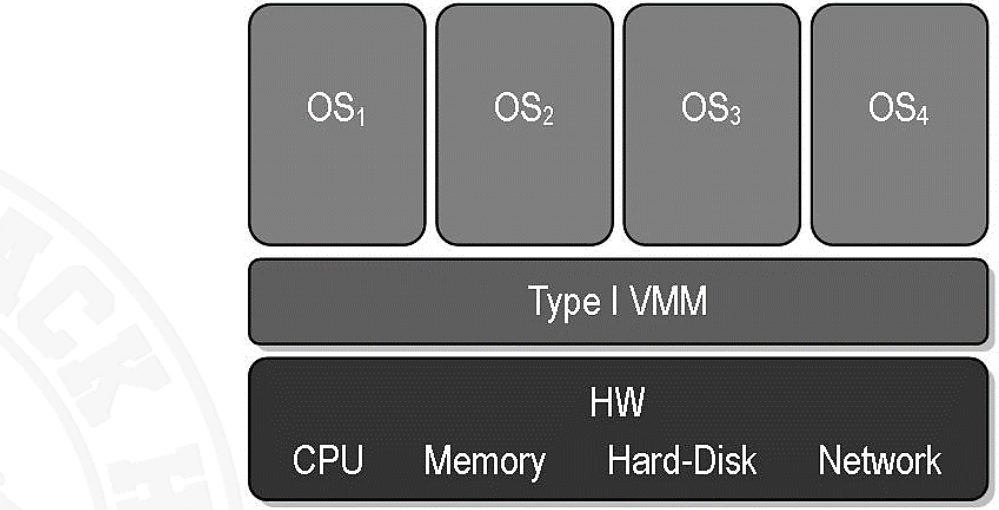
José Manuel
Redondo López

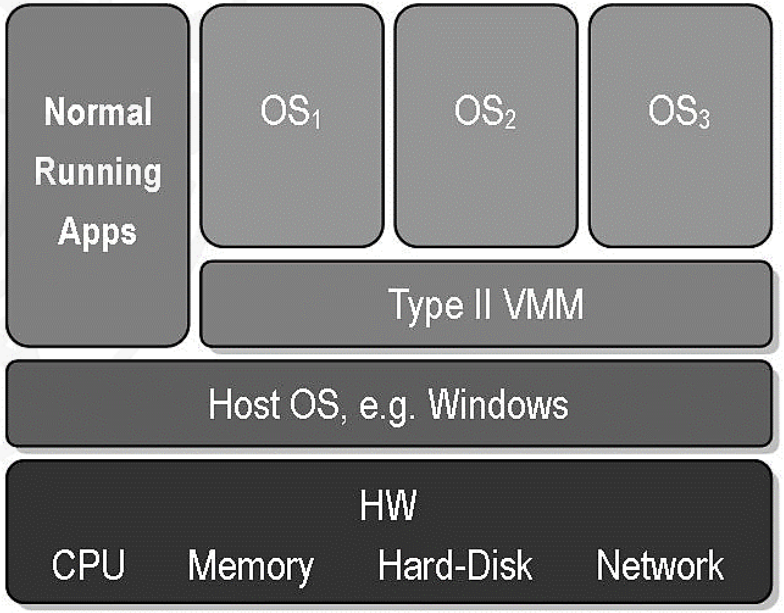
- **Emulación**. El hipervisor imita o suplanta vía **software** una arquitectura al completo
  - Procesador, memoria, conjunto de instrucciones, comunicaciones...
  - Hace **creer** a programas y SO diseñados para una arquitectura concreta que se ejecutan sobre ella, pudiendo así crearlos y probarlos sin falta de disponer de dicha arquitectura
    - Suele ofrecer un rendimiento bastante bajo, al realizar un proceso completo de traducción
    - En otras palabras, **no requiere un soporte hardware específico** para poder hacerse
  - **Ejemplos**
    - Bochs: Emula CPUs, dispositivos de E/S y BIOS
    - MAME: Emula máquinas recreativas
    - QEMU: Emula un SO dentro de otro
    - Wine: Emula partes de un SO
      - El código nativo se ejecuta directamente en la CPU

VIRTUALIZACIÓN CON HIPERVISORES

José Manuel
Redondo López

- **La principal ventaja de esta tecnología es su gran nivel de aislamiento y seguridad entre hosts y guests**
  - La comunicación entre ambos elementos se hace a través del hipervisor
  - "Escapar de la máquina virtual" **es muy difícil**
- **La principal desventaja es que el rendimiento de estas soluciones es muy bajo en comparación con otras alternativas**
  - Emular el hardware tiene mucho coste





https://es.wikipedia.org/wiki/Hipervisor

Tecnologías de virtualización con contenedores

Una forma “ligera” de obtener los mismos resultados que con las MVs en muchos escenarios

VIRTUALIZACIÓN CON CONTENEDORES

José Manuel
Redondo López

- **Funciona gracias a características del kernel que permite tener instancias aisladas en el espacio de usuario**
  - El kernel maneja los recursos de cada contenedor para que cada uno consuma recursos de forma razonable (no dejando sin recursos a otros)
  - Las características usadas son Linux namespaces y cgroups
  - Cada instancia agrupa una serie de procesos y recursos (RAM, CPU, disco…)
  - También los aísla, tanto del host como de otros contenedores
- **Las aplicaciones ven los contenedores como máquinas reales**
  - **Ven hardware**: dispositivos conectados, ficheros, carpetas (locales, compartidas), cores, etc.
  - Pero **ningún proceso dentro de un contenedor puede acceder a procesos o recursos hardware fuera del mismo**, solo ven lo asignado a su contenedor
- **Distinguimos dos tipos: de sistema operativo y de aplicaciones**

Virtualización con contenedores

José Manuel
Redondo López

- Ventajas
  - **Coste de recursos más bajo que la virtualización con hipervisores**: todos los contenedores comparten el mismo kernel del host
  - **La pérdida de rendimiento es casi nula**, especialmente si la comparamos con los hipervisores: usan mejor los recursos de la máquina
- Desventajas
  - **Ya no es posible instalar cualquier SO encima de otro**, solo aquellos que puedan trabajar con el kernel del host
    - Es imposible instalar un contenedor de Windows en Linux, o viceversa sin recurrir a "trucos" cómo tener una máquina virtual "por debajo"
  - Al compartir el kernel, **el aislamiento entre el host y los contenedores no es tan fuerte** como con los hipervisores
    - En el pasado han existido vulnerabilidades que han permitido "escapar" del contenedor, y pueden volver a producirse

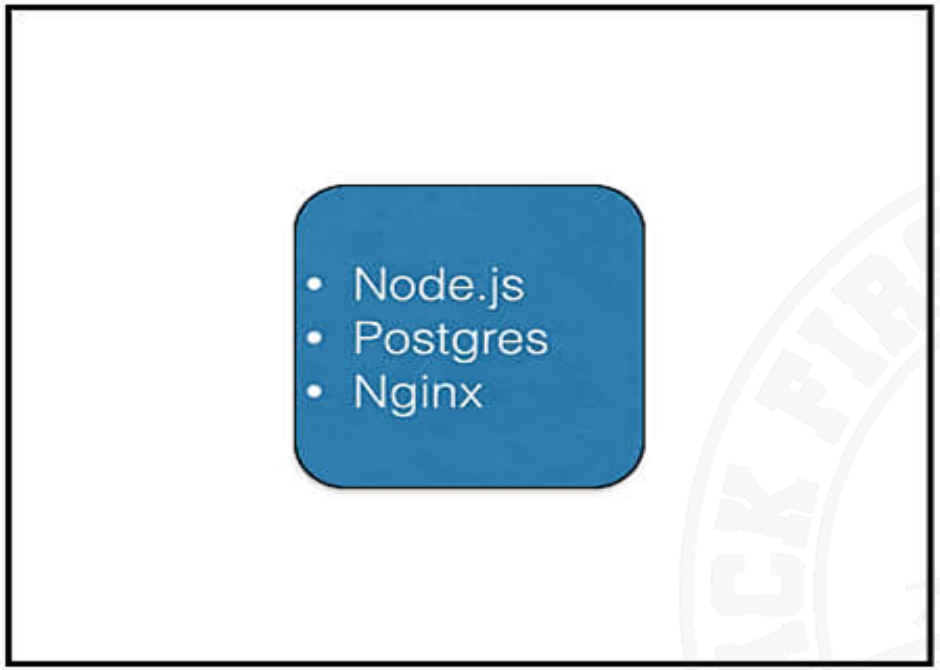
CONTENEDORES DE SISTEMA OPERATIVO

José Manuel
Redondo López

- **A efectos de uso son muy similares a máquinas virtuales**
  - Se pueden instalar programas y librerías y ejecutarlas de forma aislada
  - Solo ven los recursos de su contenedor asociado
- **Son adecuados para crear un gran nº de sistemas compatibles a nivel de kernel**
  - Funcionan preferentemente con sistemas Linux o similares
- **Se crean con plantillas/imágenes que tienen su estructura y contenidos**
  - Permiten replicar configuraciones con un conjunto de programas y versiones instalados muchas veces
- **Tecnologías que crean este tipo de contenedores**
  - LXC: https://linuxcontainers.org/ (la que veremos en esta asignatura)
  - OpenVZ: https://openvz.org/
  - Linux VServer: http://www.linux-vserver.org/Welcome_to_Linux-VServer.org
  - BSD Jails: https://www.freebsd.org/doc/handbook/jails.html
  - Solaris zones: https://docs.oracle.com/cd/E18440_01/doc.111/e18415/chapter_zones.htm#OPCUG426

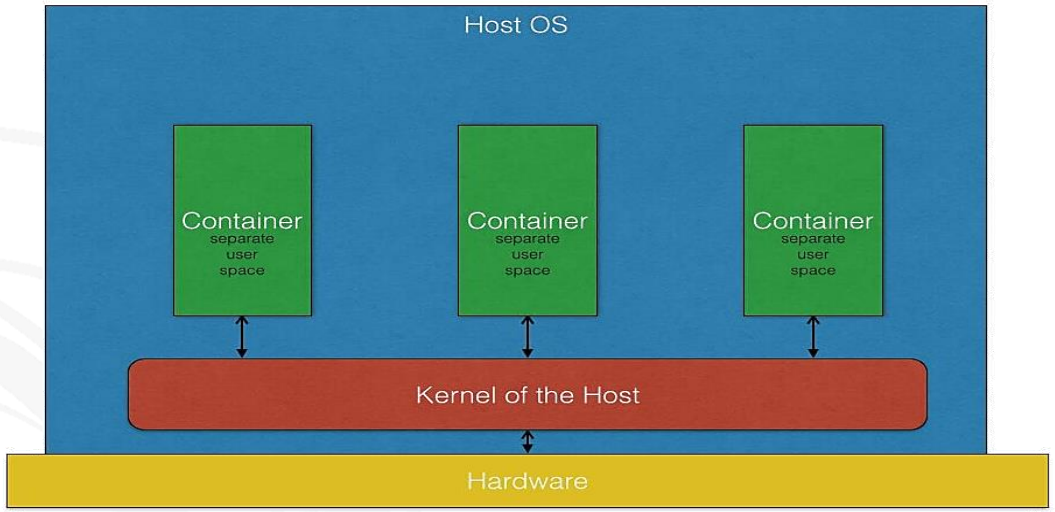
CONTENEDORES DE SISTEMA OPERATIVO

José Manuel
Redondo López

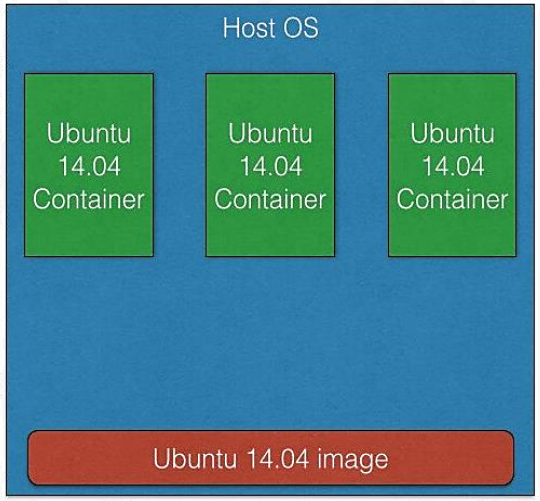


OS containers

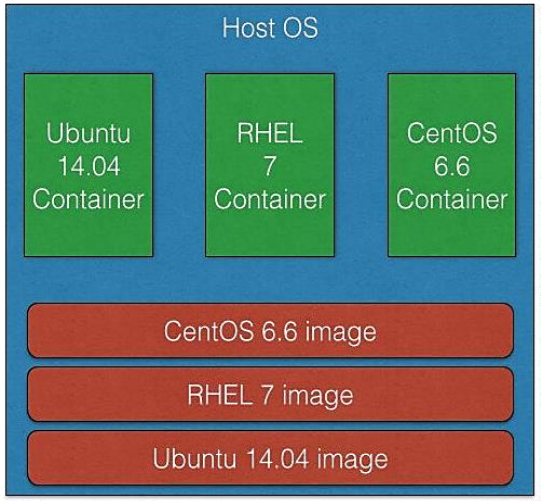
- Meant to used as an OS - run multiple services
- No layered filesystems by default
- Built on cgroups, namespaces, native process resource isolation
- Examples - LXC, OpenVZ, Linux VServer, BSD Jails, Solaris Zones



Operating System/Container Virtualization



Identical OS containers



Different flavoured OS containers

Fuente: https://blog.risingstack.com/operating-system-containers-vs-application-containers/

Contenedores de aplicaciones

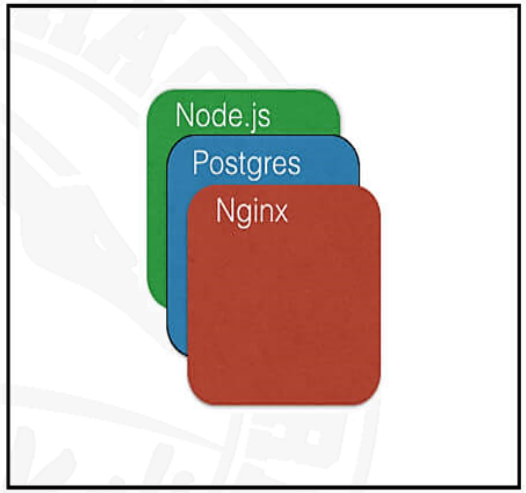
José Manuel
Redondo López

- Los contenedores de SO se diseñaron para ejecutar **múltiples** procesos y servicios
- Los contenedores de aplicaciones están diseñados para ejecutar **sólo uno**
  - No quiere decir que no puedan hacerlo, solo que no es el propósito para el que se idearon
- Ambos tipos de contenedores permiten empaquetar todas las dependencias para la ejecución de un **software**
  - **Independientemente del SO host que lo ejecute**
  - El contenedor se asegura de que todo lo necesario esté presente en todo momento
- Pero un contenedor de aplicaciones no es en principio tan similar a una máquina virtual cómo los anteriores
  - Cuando un contenedor de aplicaciones arranca **ejecuta un único proceso**, la aplicación
  - Normalmente se ejecuta un contenedor por aplicación
- Ejemplos de estas tecnologías son el popular **Docker** (visto en esta asignatura)

CONTENEDORES DE APLICACIONES

José Manuel
Redondo López

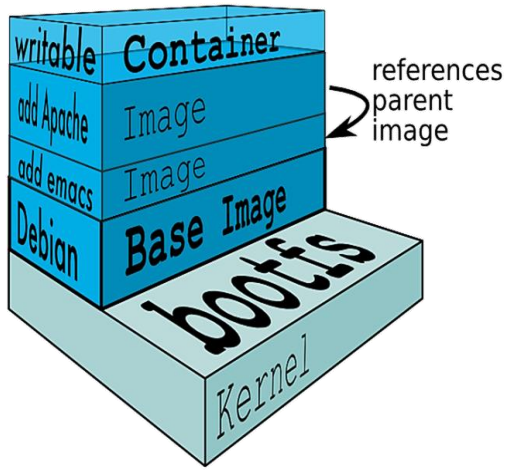
- **Los contenedores de aplicaciones se dividen en capas (layers)**
  - Cada vez que se ejecutan ciertos comandos **se crea una nueva capa** en el contenedor
  - Cada capa contiene los cambios que han hecho dichos comandos
- **Estas capas están pensadas para ser reutilizadas al máximo**
  - Un contenedor de aplicaciones se crea combinando una serie de **capas**
  - **Varios contenedores diferentes pueden usar las mismas capas para construir entornos** de ejecución para las aplicaciones distintos
  - ¡Los contenedores de aplicaciones **ahorran recursos**!
  - También permite deshacer cambios eliminando las nuevas capas y quedándose con las antiguas



App containers

- Meant to run for a single service
- Layered filesystems
- Built on top of OS container technologies
- Examples - Docker, Rocket

Fuente:
https://blog.risingstack.com/operating-system-containers-vs-application-containers/



https://dev4devs.com/2019/10/23/docker-using-build-arg-to-use-safely-confidential-information-to-build-docker-images/

CONTENEDORES DE APLICACIONES

José Manuel
Redondo López

- Se crea un contenedor distinto para cada componente de la aplicación
- Funciona especialmente bien si se quiere distribuir la aplicación como un sistema multicomponente usando **microservicios**
- De esta manera el equipo de desarrollo puede empaquetar cada parte de la aplicación en un contenedor independiente desplegable
  - Por ejemplo, un contenedor MySQL, otro Node.js y otro Nginx
  - En la misma o en distintas máquinas, local o en nube, independiente del SO
  - Gracias a tecnologías como Terraform, a lo largo de infraestructuras enormes
  - Y gracias a tecnologías como Kubernetes, hacer un escalado horizontal y/o vertical manual o bajo demanda
- El sistema es **un conjunto de aplicaciones y servicios**
  - **Cada uno ejecutándose en su contenedor**
  - Y comunicándose entre ellas usando las APIs y protocolos que soporten
  - ¡Así se crean las aplicaciones web modernas no monolíticas!

CONTENEDORES DE APLICACIONES

José Manuel
Redondo López

- **El uso de contenedores de aplicaciones requiere también lidiar con sus limitaciones**
  - Si un contenedor muere, **se pierden** todos los datos que contiene; son **efímeros**
    - En esta asignatura aprenderemos formas de evitar esto
  - Un contenedor de aplicaciones **NO es una máquina virtual**
    - El acceso a determinadas partes del SO no es posible o está muy limitado
    - El principal componente problemático es el firewall: estamos limitados a la hora de hacer modificaciones en la configuración de red de nuestros contenedores
  - Si en el proceso de construcción automática se hace alguna pregunta al usuario, **éste fallará** y no podrá completarse
    - **Todo el proceso DEBE ser no interactivo**
    - Existen formas de lograrlo, pero debemos recordar usarlas
  - Es necesario implementar **controles de seguridad** para evitar que contenedores maliciosos se usen para atacar al host

¿QUÉ TIPO CONTENEDOR ES MEJOR?

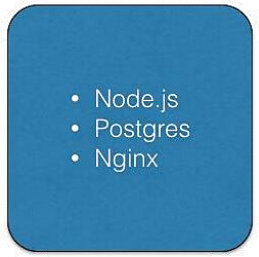
José Manuel
Redondo López

- **Si necesitas es algo similar a una máquina virtual, pero de muy bajo coste: contenedores de SO**
  - Aunque pierdes aislamiento a la hora de ejecutar programas, librerías, compiladores, etc.
- **Para desplegar aplicaciones, especialmente si tienen componentes en la nube: contenedores de aplicaciones, que tienen las siguientes ventajas**
  - **Son portables**: se define un formato para que una aplicación se empaquete con todas sus dependencias en un solo objeto
    - Puede transferirse a cualquier sistema de contenedores compatible
    - Esto garantizará que la aplicación se ejecutará sin ningún problema en cualquiera de los sistemas a las que se lleve, ya que garantiza que el entorno de ejecución es siempre el mismo
    - **Los contenedores de SO no garantizan esto**: si tienen algún tipo de configuración personalizada podrían no ejecutarse igual en diferentes sitios
      - La aplicación podría estar ligada a la configuración específica de cada contenedor: red, almacenamiento, log...
      - Los contenedores de aplicaciones proveen una abstracción sobre estos elementos para eliminar este problema
    - Permiten la **construcción automática** mediante código de un DSL
    - Dentro de ellos se pueden usar cualquier software de gestión de dependencias o de construcción
      - Ej. Maven

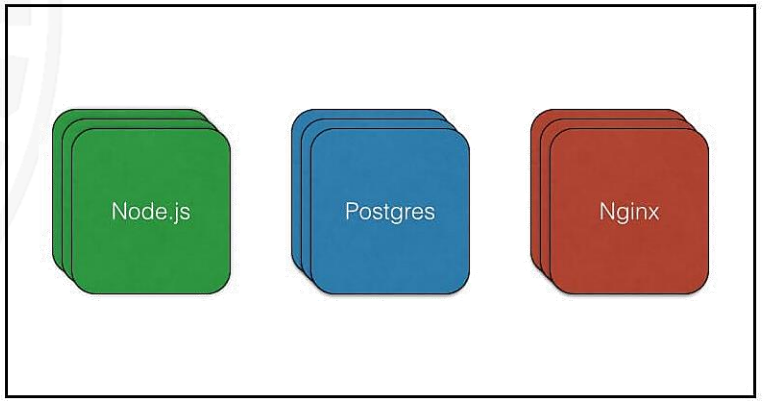
¿QUÉ TIPO CONTENEDOR ES MEJOR?

José Manuel
Redondo López

- **Versiones**: integran un control de versiones similar al de `git`, (commits con versiones nuevas, roolbacks, etc.)
  - Permiten tener constancia de quién ha sido el usuario que los construyó (trazabilidad completa de construcción)
- **Reutilización**: el sistema de capas ahorra recursos
- **Facilidad para compartir**: existen registros públicos y privados de imágenes de contenedores
  - Permite a cualquiera reutilizar cualquiera de ellos para sus proyectos ahorrando trabajo al construir aplicaciones
  - **Ejemplo**: si necesitamos servidores Apache y BBDD MySQL, existen contenedores con ellos ya instaladas que podemos reutilizar para construir nuestra aplicación
- **Ecosistema de herramientas**: gran número de herramientas que automatizan su creación y despliegue
  - Orquestación de despliegue, GUIs de gestión, automatización de lo instalado dentro de ellos, habilitar integración continua...



Typical 3-tier architecture in the simplest sense



Typical 3-tier architecture using Docker containers

Fuente:
https://blog.risingstack.com/operating-system-containers-vs-application-containers/

¿CREES QUE LO HAS ENTENDIDO TODO? ¡AUTOEVALÚATE!

José Manuel
Redondo López

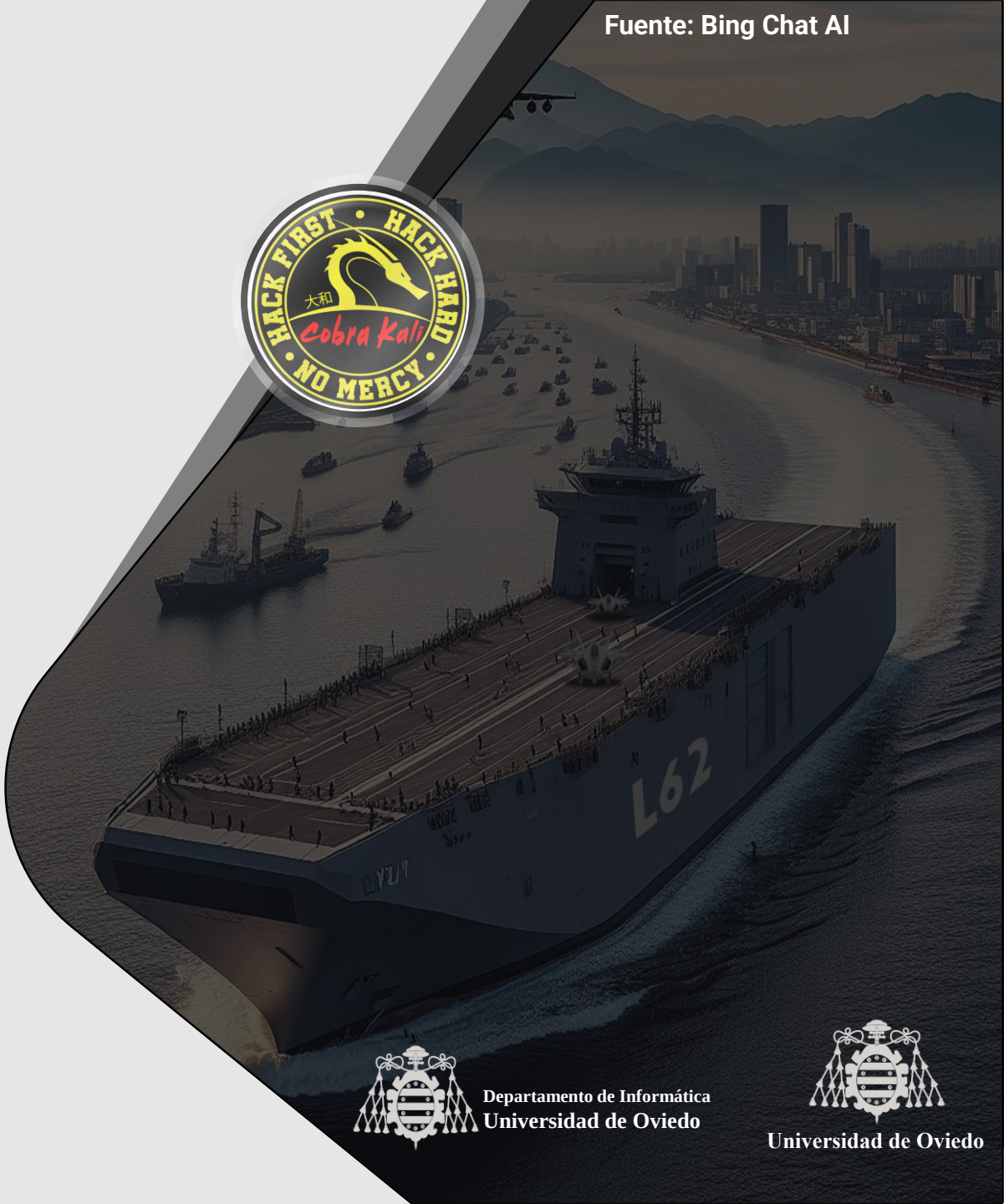
?

- ***¿Entiendes porque hoy en día no se entienden las infraestructuras sin que se use alguna tecnología de virtualización?***
- **Virtualización con hipervisores**
  - *¿Sabes las diferencias entre un hipervisor tipo 1 y uno tipo 2?*
  - *¿Sabes las diferencias entre una virtualización completa y una paravirtualización?*
  - *¿Sabes las diferencias entre una virtualización completa y la emulación?*
  - *¿Entiendes por qué la virtualización con hipervisores proporciona un grado de aislamiento superior?*
  - *¿Comprendes que para ello tiene un coste en rendimiento muy elevado y por qué?*
- **Virtualización con contenedores**
  - *¿Comprendes en qué características del SO se basa la virtualización con contenedores?*
  - *¿Puedes recordar las principales ventajas y desventajas de este tipo de virtualización frente a la otra?*
  - *¿Comprendes la distinta filosofía de uso que hay entre contenedores de SO y de aplicaciones?*
  - *¿Entiendes que un contenedor de aplicaciones es un conjunto de capas y las ventajas que tiene usarlas?*
  - *¿Puedes explicar la filosofía de uso y las precauciones a la hora de usar un contenedor de aplicaciones?*
  - *¿Eres capaz de recordar los criterios que debes usar para elegir entre un tipo de contenedor u otro en función de lo que necesites?*

< Ir al Índice

Tecnologías para la creación automatizada de infraestructuras

Diferentes formas de usar Infraestructure as Code (IaC)



Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿QUÉ VAMOS A VER EN ESTE APARTADO?

José Manuel
Redondo López

- El concepto de Infrastructure as Code o IaC
- Vagrant como forma de IaC para máquinas virtuales
- LXD como forma de IaC para contenedores de SO
- Docker como forma de IaC para contenedores de aplicaciones
- Terraform como forma de IaC para la creación de grandes infraestructuras que se puedan versionar
- Kubernetes como forma de IaC para desplegar aplicaciones basadas en contenedores de aplicaciones que se puedan orquestar y escalar
- El concepto de provisioner

Infraestructure as Code (IaC)

José Manuel
Redondo López

- ¡Virtualizar no implica construir “a mano” las infraestructuras que vamos a usar!
- IaC son un conjunto de tecnologías que permiten crear infraestructuras **a partir de código fuente**
  - Por infraestructuras entendemos elementos que pueden albergar servicios (¡**no tienen por qué ser solo MV**!)
  - Y, por supuesto, sus **relaciones** (redes entre ellas)
  - Incluyendo además **restricciones** a la hora de conectarse entre ellos (aislamiento)
  - Y todo el **software** que ejecutan
  - Y la **configuración de SO**, usuarios, controles de seguridad…**¡TODO!**
- Es decir, la SNI que hemos visto **puede construirse automáticamente a partir de un programa**
- **¿No es eso lo más adecuado para los ingenieros del software?** ☺

INFRAESTRUCTURE AS CODE CON HIPERVISORES

José Manuel
Redondo López

- **Vagrant** (https://www.vagrantup.com/) es referencia para crear y configurar automatizadamente entornos virtuales **basados en hipervisores**
- Permite gestionar MVs (**boxes**) que se ejecutan potencialmente sobre cualquier **proveedor de virtualización** o de contenedores
  - VirtualBox, VMware, Hyper-V, KVM, Docker, Linux Containers (LXC)…
- Es decir, **se logra cierta independencia del proveedor de virtualización**
  - Se crea una especificación de máquina virtual (código fuente), pero no se dice con qué se virtualiza
- Permite trabajar con MV de forma simple, reproducible y portable
- Además, permite usar herramientas de automatización de configuraciones / instalación de software (**provisioners**)
  - Ansible, Puppet, Chef, Salt,…o simplemente scripts de bash

INFRAESTRUCTURE AS CODE CON CONTENEDORES

José Manuel
Redondo López

Contenedores de Sistemas Operativos

- Los Linux Containers (https://linuxcontainers.org/) es la tecnología de referencia para ellos
  - Nosotros usaremos su implementación LXD
- Usan mecanismos nativos del Linux anfitrión para ofrecer la misma experiencia de usuario que una MV

Contenedores de Aplicaciones

- El software de referencia para ello es Docker (https://www.docker.com/)
- Cada contenedor tiene un sistema de ficheros
  - Que se crea introduciendo todo lo que unas las aplicaciones que ejecuta necesitan para funcionar
  - Código, librerías, herramientas,…
- El objetivo es que las aplicaciones de un contenedor **siempre se ejecuten de la misma forma**
  - Sin importar el entorno en el que lo hagan
  - La aplicación no se mueve de un sistema a otro, **se mueve su contenedor**
- **También tiene un coste de rendimiento muy bajo**
  - **Pero algunas aplicaciones podrían no funcionar dentro de ellos** (acceso a partes privilegiadas del SO)

INFRAESTRUCTURE AS CODE PARA EL DESPLIEGUE DE INFRAESTRUCTURAS COMPLEJAS

José Manuel
Redondo López

- **Terraform** (https://www.terraform.io/)
  - **Permite construir, cambiar y hacer versiones de infraestructuras** de máquinas virtuales o contenedores de manera eficiente a partir de código fuente
  - **Tanto en despliegues locales como en nube**, incluso trabajando con varios proveedores de nube en una misma infraestructura (entornos multi-cloud)
  - Además, **permite ver lo que se va a hacer antes** de hacerlo: las operaciones son más seguras
  - Incluye además otros elementos de las infraestructuras
    - **De bajo nivel**: instancias de computación, almacenamiento, redes, etc.
    - **De alto nivel**: DNS, Software as a Service, etc.
- **Kubernetes** (K8s, https://kubernetes.io/) **+ Helm** (https://helm.sh/)
  - Sistema para **automatizar el despliegue de aplicaciones en contenedores**
    - Permite el escalado automático de las aplicaciones desplegadas
    - Establece unas primitivas para construir sistemas de forma flexible y adaptados a distintas necesidades
    - También dispone de API de programación
  - Helm es el gestor de paquetes para K8s
    - Permite desplegar aplicaciones en clústeres K8s mediante código

¿Y TODO ESTO POR QUÉ ES?

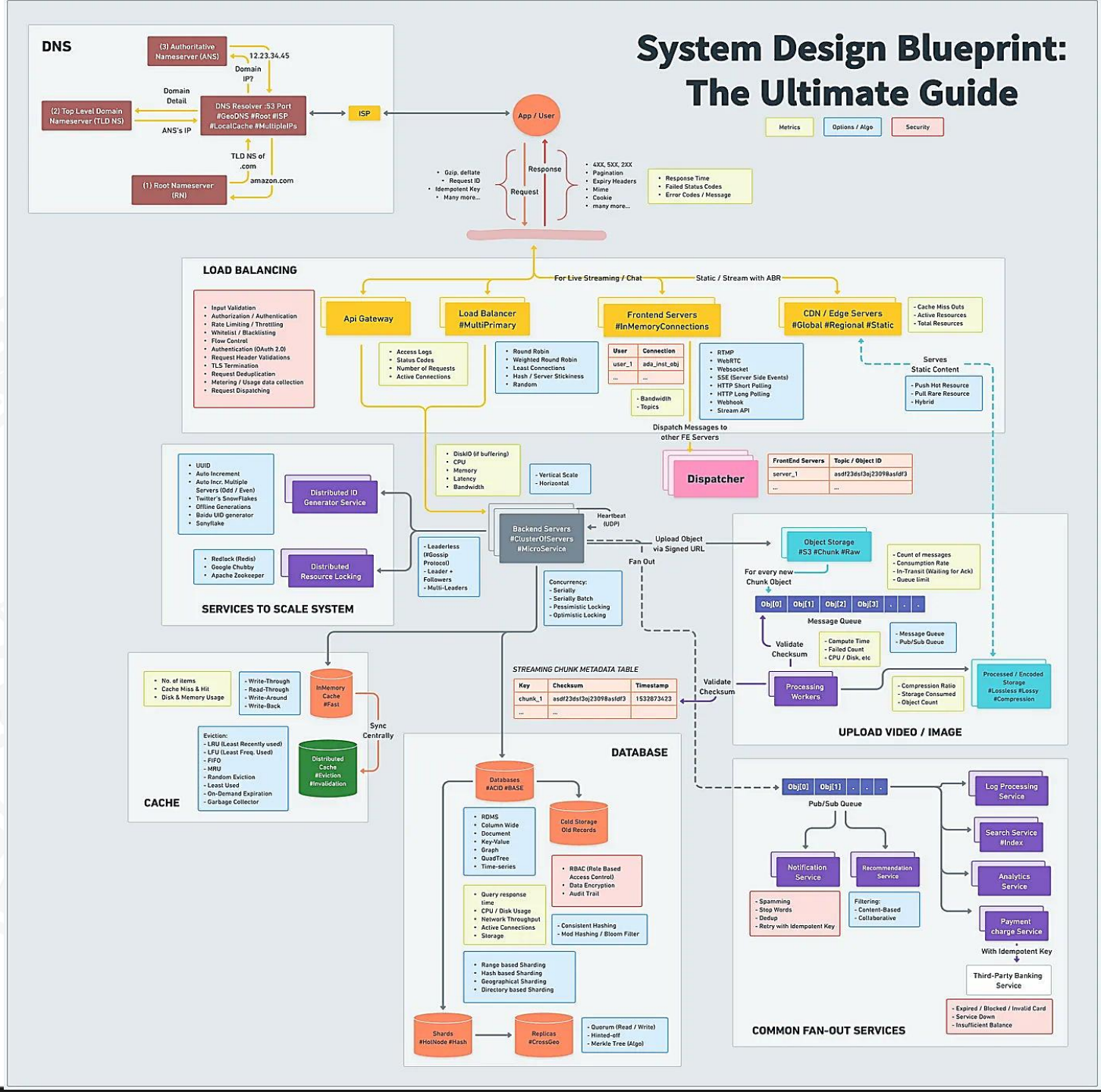
José Manuel
Redondo López

- **Porque es lo que vemos a hacer en el resto de la asignatura: usar código para todo**
  - Vagrant para máquinas virtuales con hipervisores
  - LXD para contenedores de sistemas operativos
  - Docker para contenedores de aplicaciones
  - Kubernetes para despliegue de grandes infraestructuras de contenedores
  - Terraform para ver cómo se construyen grandes infraestructuras
  - ¡Y todo ello con la posibilidad de ser **provisionado** con Ansible!
- **Al ser código, las puedes subir, actualizar, versionar, etc. en un repositorio, como el resto del código de tu aplicación**
  - Por ejemplo, puedes subir a GitHub las especificaciones completas de la infraestructura sobre la que tu aplicación web va a funcionar (¡Dev+Ops! ☺)
  - ¡Así cualquiera **puede usar ese código para crear dicha infraestructura** tal y como tú la quieres!
- **Tecnologías de vanguardia, adaptadas a lo que se usa en el mercado laboral**
  - ¡Hasta el infinito, y más allá! ☺

¿Y TODO ESTO POR QUÉ ES?

José Manuel
Redondo López

- Las tecnologías IaC son las que permiten realmente diseñar infraestructuras masivas
- Sin la automatización es imposible desplegar algo que cumpla con los requisitos de las aplicaciones web modernas
- Este curso no te permitirá crear algo como lo de la imagen, pero te enseñará las bases para empezar a hacerlo
  - **Fuente**: https://medium.com/tag/system-design-concepts



Poniéndolo todo junto...¿Como monto una infraestructura "pro" que atienda un gran volumen de peticiones?

José Manuel
Redondo López

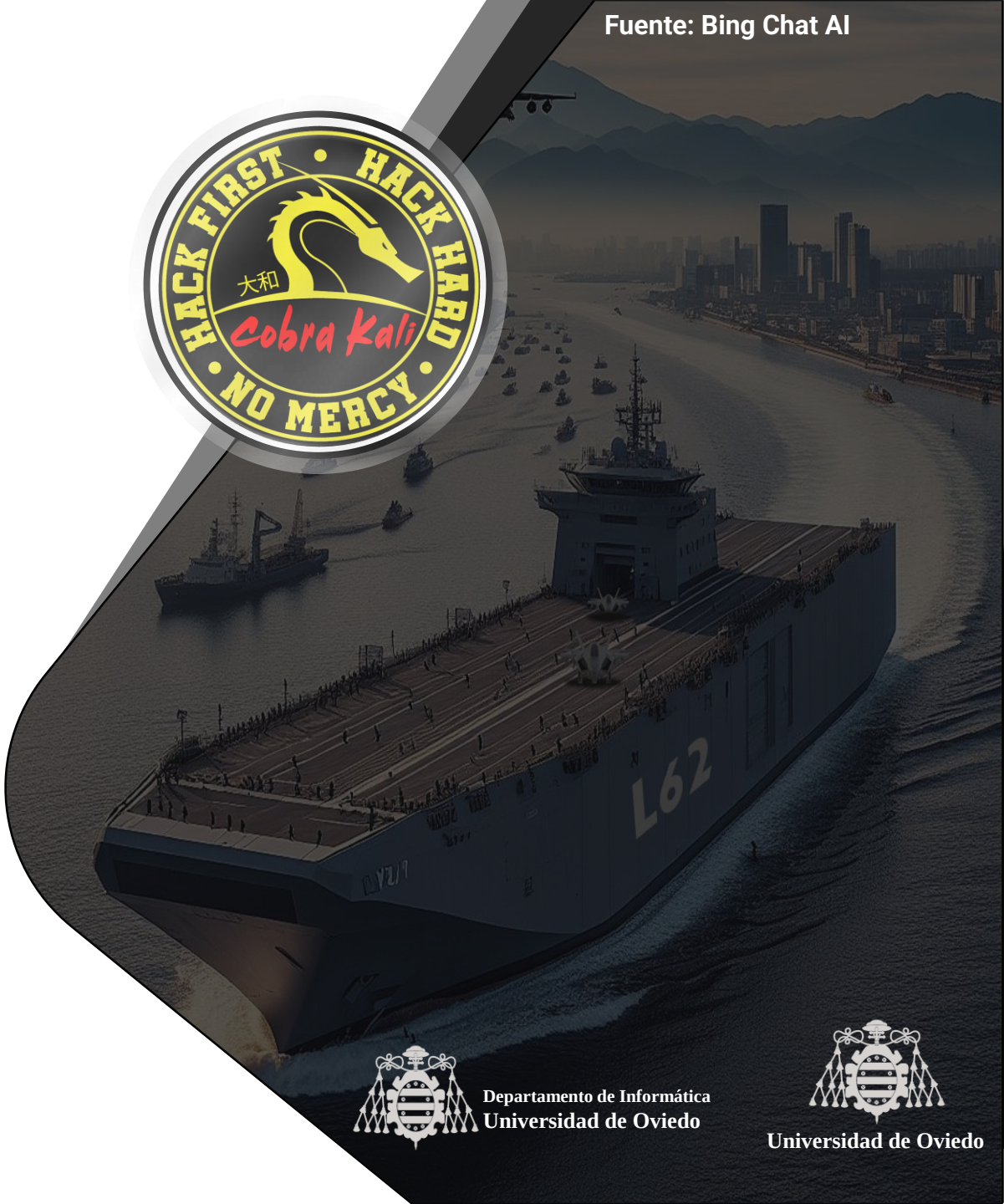
- DMZ Pública
  - **Balanceo de carga**, para **distribuir las peticiones** entre los distintos servidores web que tengo
  - **Reverse proxies** (con balanceo también), para ocultar los detalles de nuestros servidores web reales
    - Y en ellos, **medidas de seguridad contra peticiones maliciosas** ("**F-103 Blas de Lezo**")
- DMZ Privada
  - Un **clúster de servidores de aplicaciones** con **replicación de sesiones** (Ej.: Tomcat)
    - Probablemente con una **caché distribuida centralizada** (Ej.: memcached) para gestionarla
- Intranet e Intranet privada
  - Un **sistema de colas** (Ej.: RabbitMQ) y **otra caché distribuida** para mejorar el acceso a la BBDD
- Y en general... ¡Solo nos quedaría "**pegar las piezas**"! ☺
  - Usando la **tecnología de comunicación** con los servidores/servicios de la siguiente capa adecuada
  - Todos los endpoints virtualizados, monitorizados y securizados
    - Contenedores o MVs, en nube, on-premises o híbrido, "**F-103 Blas de Lezo**"
  - No se instala nada "desde cero": Se construyen unidades de servicio con código
    - **O a partir de otras oficiales** que tengan todo el software necesario instalado

¿CREES QUE LO HAS ENTENDIDO TODO? ¡AUTOEVALÚATE!

José Manuel
Redondo López

- *¿Entiendes por qué el uso de IaC es el futuro de la creación y trabajo con infraestructuras?*
- *¿Comprendes por qué el uso de código para crear infraestructuras es una pieza imprescindible en DevOps?*
- *¿Sabes elegir el producto IaC más adecuado en función de lo que quieras usar para desplegar aplicaciones (MVs, contenedores...)?*
- *¿Entiendes que hay tecnologías que te facilitan mucho desplegar aplicaciones escalables y/o que formen grandes infraestructuras?*
- *¿Entiendes que hay tecnologías que te facilitan el aprovisionamiento (instalación, configuración...) automatizado de todos los componentes de una infraestructura por código?*
- *¿Entiendes a nivel general como juntar todo eso?*

PRINCIPIOS DE DISEÑO DE INFRAESTRUCTURAS DE SERVIDORES



Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo