Aplicaciones y sistemas distribuidos

Introducción y modelos de comunicaciones

Rubén Usamentiaga
Área de Arquitectura y Tecnología de Computadores
Universidad de Oviedo

Aplicaciones y sistemas distribuidos

Introducción

Introducción

- Definiciones de sistema distribuido

Tanembaum

Conjunto de computadores independientes que ante los usuarios del sistema parecen un solo computador

Leslie Lamport

You know you have a distributed system when the crash of a computer you've never heard of stops you from getting any work done.

Introducción

- Características de un sistema distribuido
  - No hay un reloj común
  - No existe memoria compartida
  - Separación geográfica
  - Autonomía y heterogeneidad

Introducción

- Motivos
  - Compartir recursos
    - Recursos distribuidos geográficamente
  - Coste
    - Gran capacidad computacional por menos precio
  - Escalabilidad
    - Posibilidad de crecer
  - Rendimiento
    - Velocidad de procesamiento, aprovechar el paralelismo
  - Tolerancia a fallos
    - Permite la redundancia

Introducción

- Retos de los sistemas distribuidos
  - Hacer los recursos accesibles
    - Permitir el acceso a recursos remotos y compartirlos de forma controlada y eficiente
  - Transparencia
    - Ocultar que los recursos están distribuidos y proporcionar un acceso similar al local
  - Apertura
    - Proporcionar servicios con sintaxis y semántica de acuerdo a reglas conocidas que permitan interoperabilidad

Introducción

- Retos de los sistemas distribuidos
  - Escalabilidad
    - Capacidad para adaptarse al crecimiento
  - Concurrencia
    - Gestionar accesos simultáneos a la información
  - Calidad del servicio (QoS)
    - Asegurar cierto valor en métricas que definen aspectos de un servicio
  - Seguridad
    - Privacidad, integridad, autenticidad

Introducción

- Dificultades:
  - La red no es fiable
  - La latencia no es cero
  - El ancho de banda es limitado
  - La red no es segura
  - La topología cambia
  - Hay múltiples administradores
  - Coste del transporte
  - La red es heterogénea

Aplicaciones y sistemas distribuidos

Comunicaciones

Comunicaciones

- Modelos de comunicaciones
  - Es el modo en el que dos aplicaciones pueden interactuar
  - Dos modelos principales:
    - Comunicación síncrona
      - Emisor espera por una respuesta
    - Comunicación asíncrona
      - Emisor continúa sin esperar
    - Cada modelo se puede implementar de diversas formas

Comunicaciones

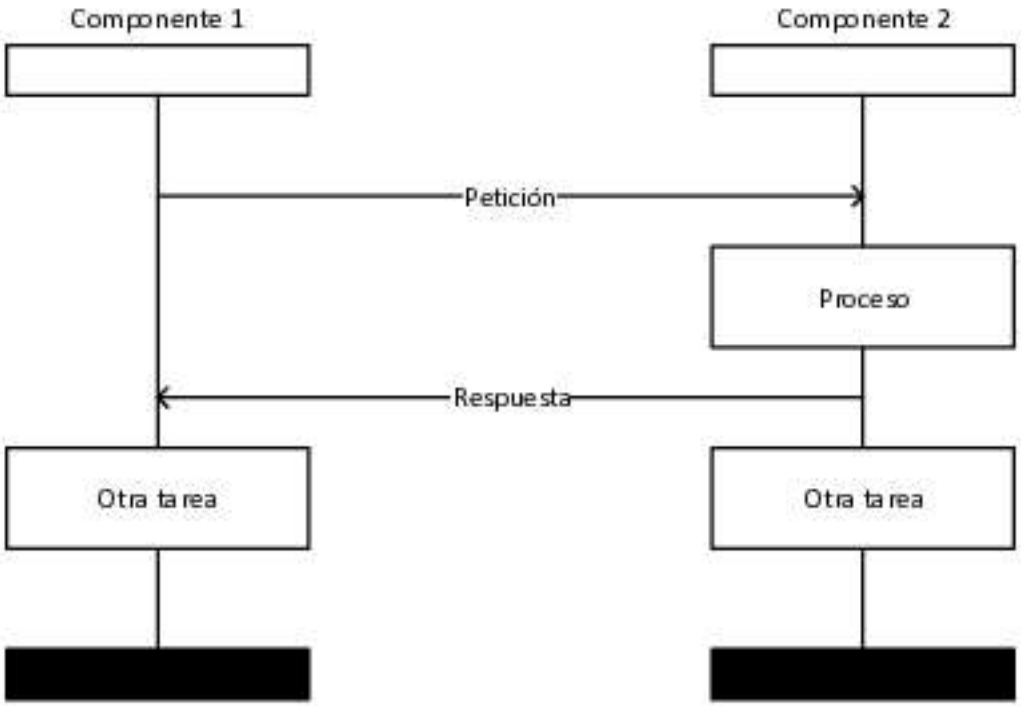
- Comunicación síncrona
  - Se realiza entre emisor y receptor de una manera coordinada
  - Emisor y receptor operan dependiendo del procesamiento de una petición
  - Escenarios de aplicación:
    - El emisor necesita un resultado para proseguir
    - Comunicación interactiva
  - Se requiere un canal de comunicación fiable
  - Variantes:
    - Comunicación síncrona de petición/respuesta
    - Comunicación síncrona de un sentido
    - Comunicación síncrona por sondeo

Comunicaciones

- Comunicación síncrona de petición/respuesta
  - Proceso de comunicación:
    - El emisor envía una petición y espera hasta obtener una respuesta para continuar
  - Aplicabilidad:
    - Cuando se necesita una respuesta para seguir procesando (problema de bloqueo si la respuesta no llega)
  - Modelo de comunicación pull
    - Adecuado cuando el productor es rápido, y el receptor no tiene que esperar mucho tiempo por la respuesta

Comunicaciones

- Comunicación síncrona de petición/respuesta

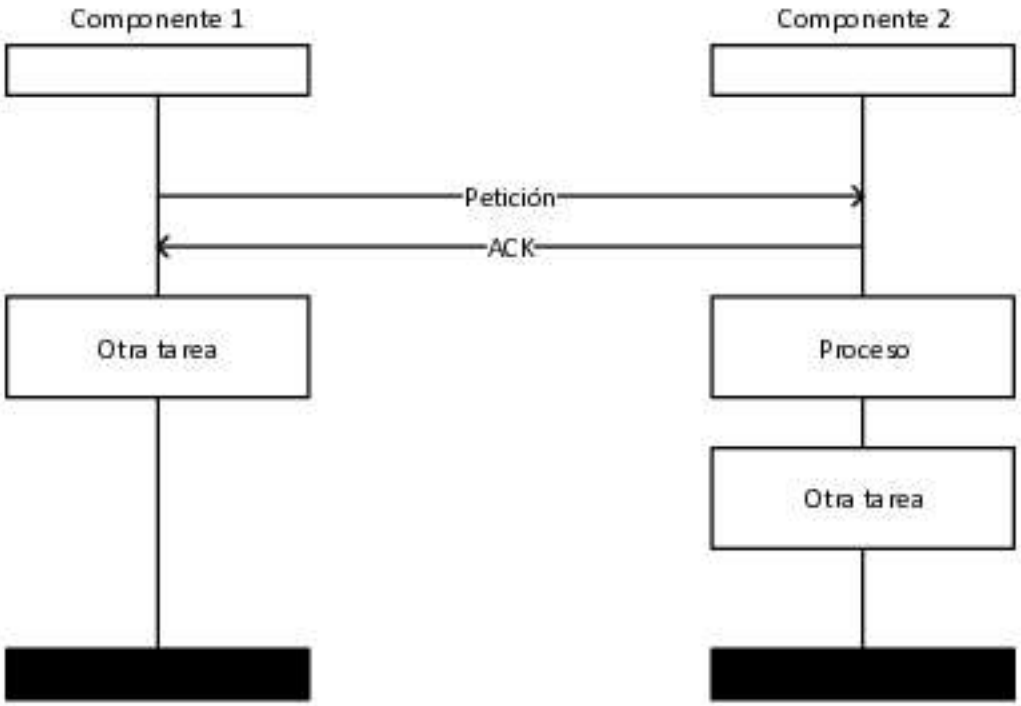


Comunicaciones

- Comunicación síncrona de un sentido
  - Proceso de comunicación:
    - El emisor envía una petición y espera hasta obtener un asentimiento para continuar
  - Aplicabilidad:
    - Cuando no es necesario recibir respuesta, solo confirmación de llegada (problema de bloqueo si el ACK no llega)

Comunicaciones

- Comunicación síncrona de un sentido

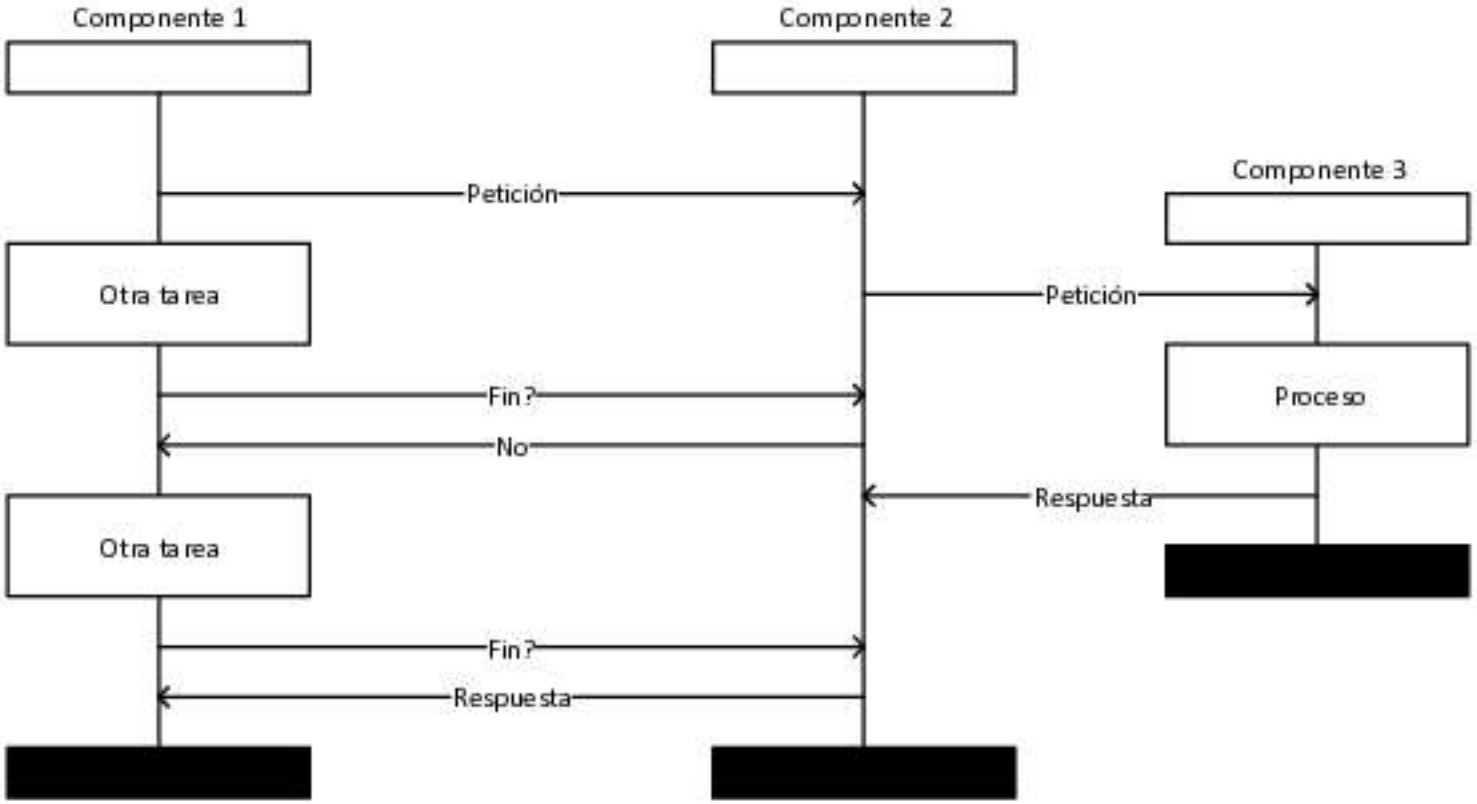


Comunicaciones

- Comunicación síncrona por sondeo
  - Proceso de comunicación:
    - Emisor envía una petición y continúa, comprobando la recepción de una respuesta a intervalos de tiempo
    - En la respuesta inicial se puede incluir el punto de acceso para obtener la información posterior
  - Aplicabilidad:
    - Se requiere una respuesta no urgente

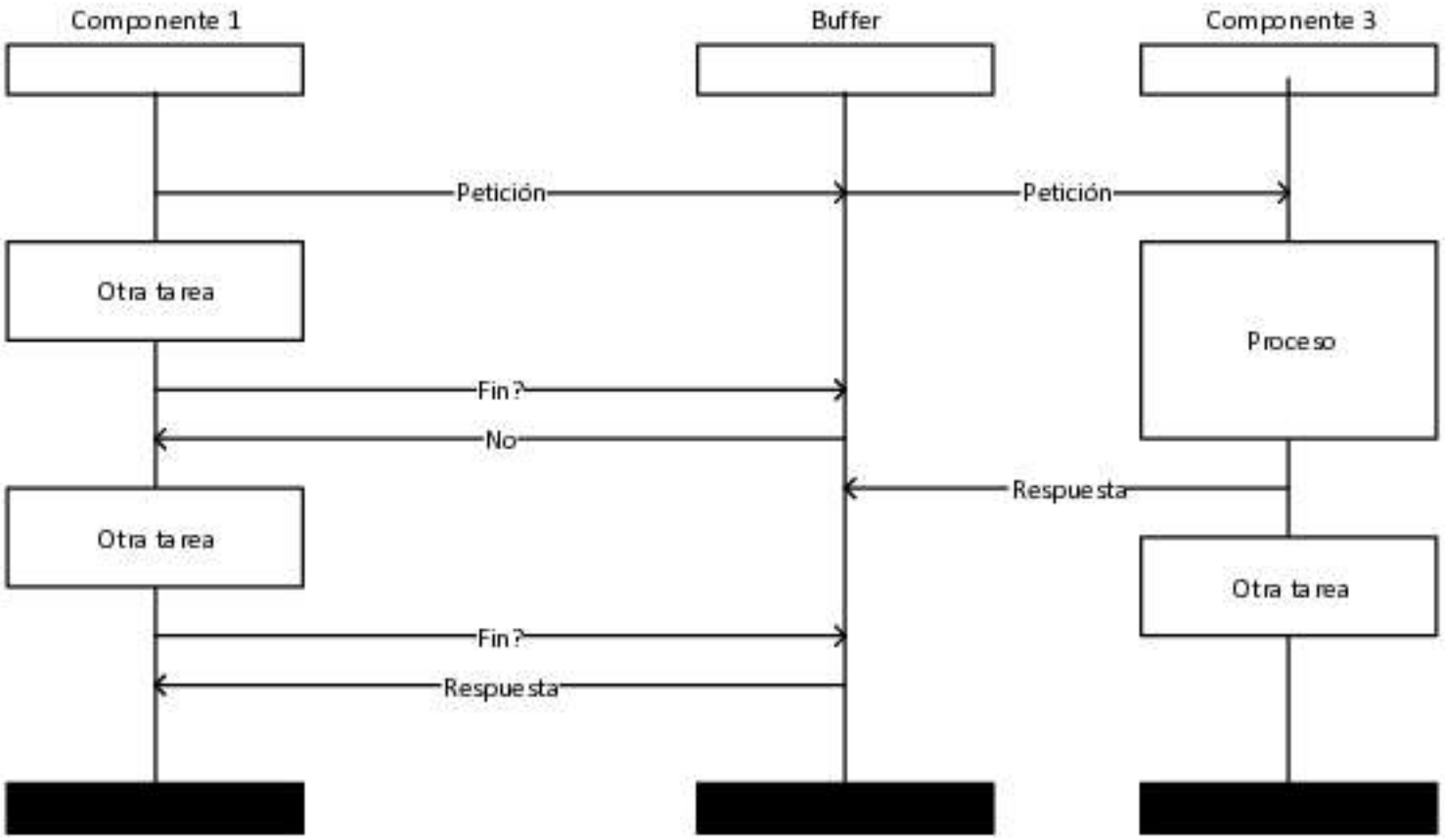
Comunicaciones

- Comunicación síncrona por sondeo



Comunicaciones

- Comunicación síncrona por sondeo



Comunicaciones

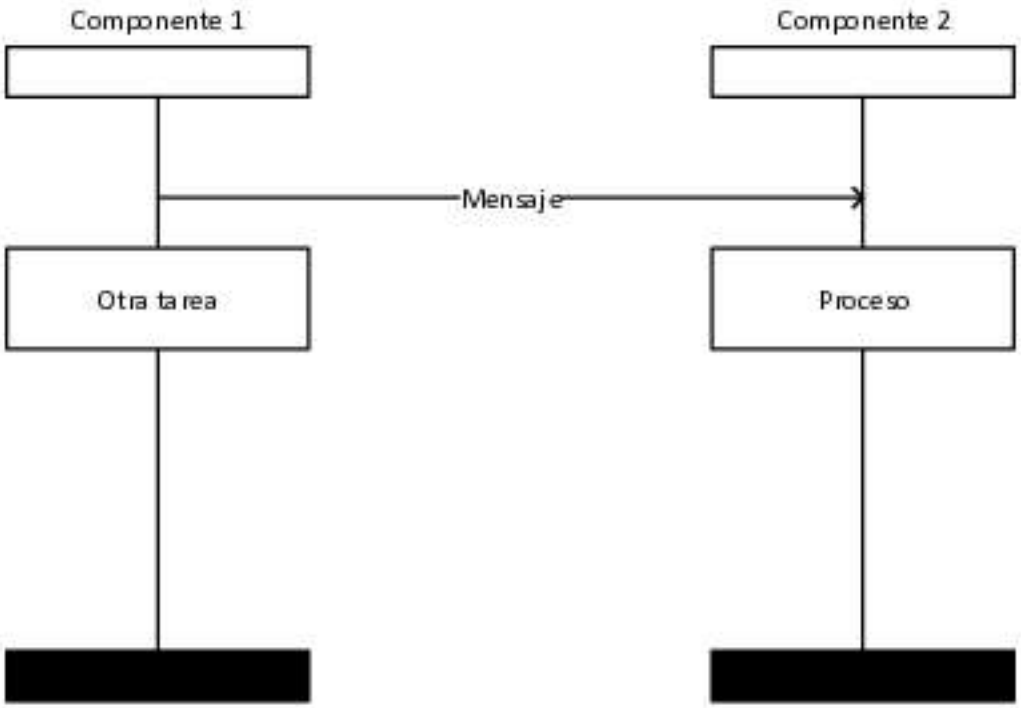
- Comunicación asíncrona
  - La comunicación entre emisor y receptor se lleva a cabo de manera que ambos puedan operar de forma independiente
  - Emisor y receptor no necesitan estar coordinados
  - Escenarios de aplicación:
    - El propósito final de la comunicación es la transferencia de información
    - Entornos de comunicación no fiables
  - Variantes:
    - Comunicación asíncrona de paso de mensajes
    - Comunicación asíncrona de publicación/suscripción
    - Comunicación asíncrona por difusión

Comunicaciones

- Comunicación asíncrona de paso de mensajes
  - Proceso de comunicación:
    - Emisor envía un mensaje y continúa, sin esperar la recepción de una respuesta
  - Aplicabilidad:
    - No se requiere respuesta
      - Asume un mecanismo de transmisión fiable o implementa un mecanismo de reintento

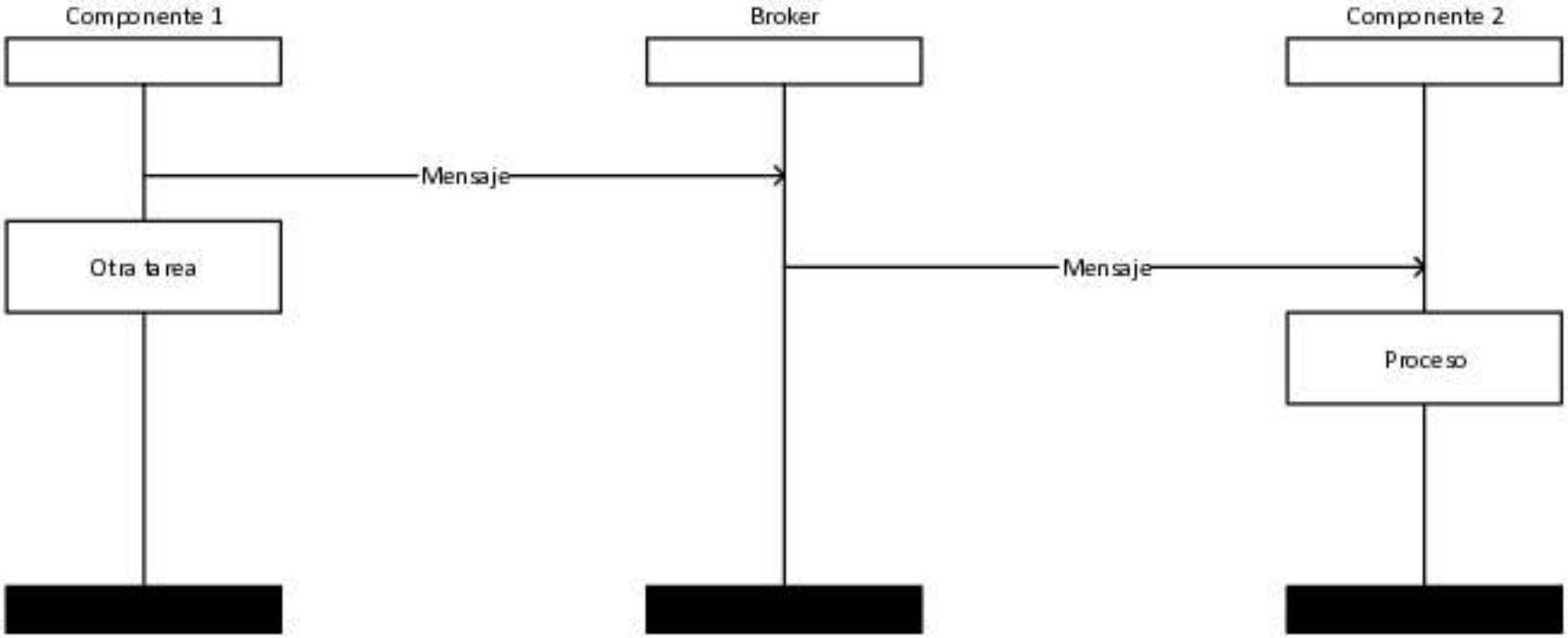
Comunicaciones

- Comunicación asíncrona de paso de mensajes



Comunicaciones

- Comunicación asíncrona de paso de mensajes

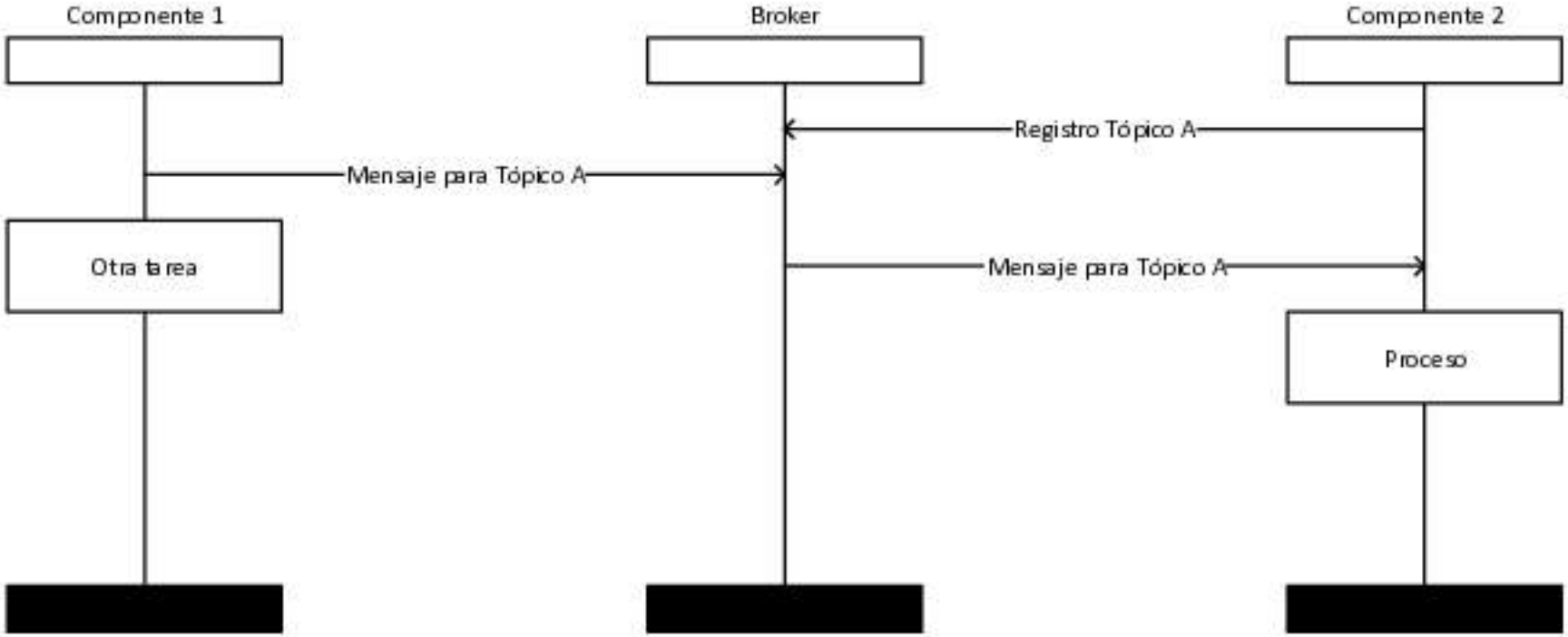


Comunicaciones

- Comunicación asíncrona de publicación/suscripción
  - Proceso de comunicación:
    - El emisor envía un mensaje y el receptor es determinado mediante una declaración previa de intereses
  - Aplicabilidad:
    - No se requiere respuesta y hay que filtrar el destinatario de la información
  - Modelo de comunicación push
    - Adecuado cuando el productor es más lento que el consumidor

Comunicaciones

- Comunicación asíncrona de publicación/suscripción

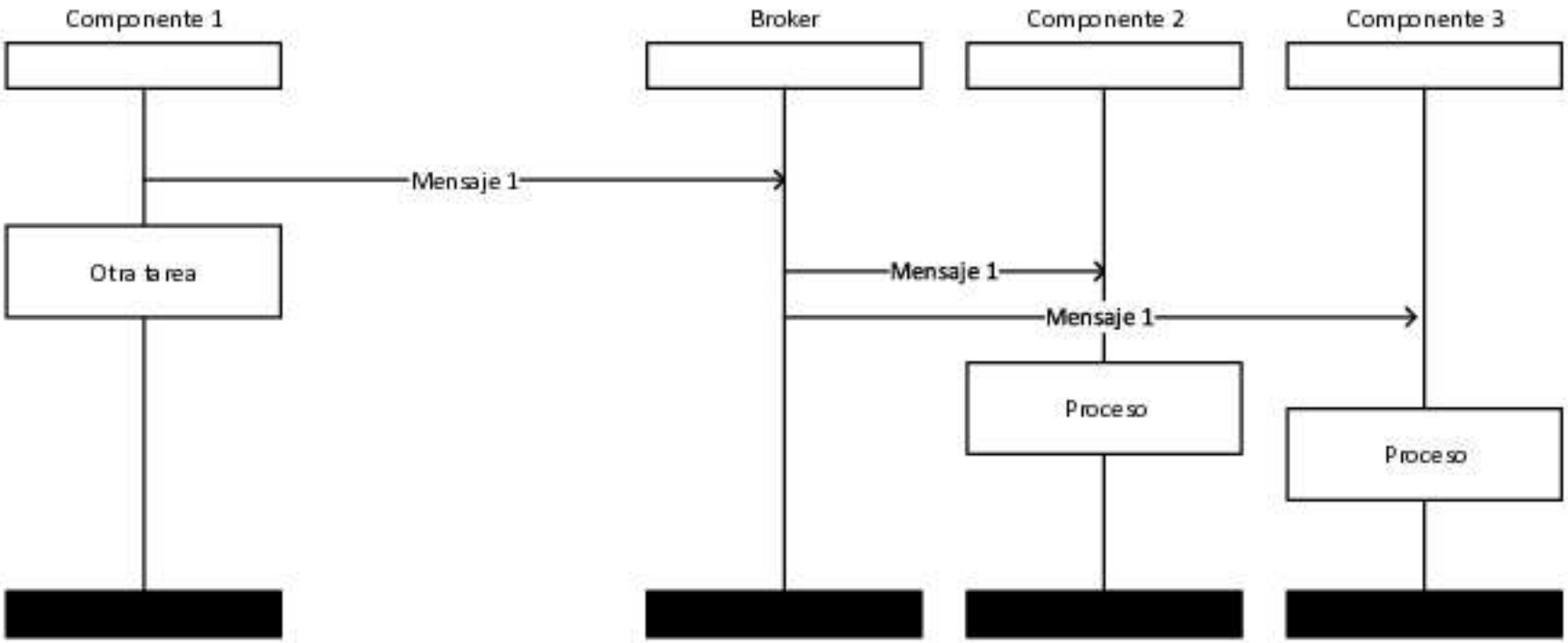


Comunicaciones

- Comunicación asíncrona por difusión:
  - Emisor envía un mensaje a todos los receptores en una red
- Aplicabilidad:
  - No se requiere respuesta y hay que enviar la misma información a muchos destinatarios

Comunicaciones

- Comunicación asíncrona por difusión:



Comunicaciones

- Mecanismos de intercambio de información
  - Mecanismo para diseñar las peticiones que se intercambian emisor y receptor
  - Dos grandes tipos:
    - Mensajes
    - Definición de interfaces

Comunicaciones

- Comunicación mediante mensajes
  - El emisor construye un mensaje que envía al receptor.
  - El mensaje contiene información sobre las acciones y datos
    - Ejemplo: Descargar el fichero 'ff.txt'
  - El emisor:
    - Crea el mensaje en el formato apropiado
    - Envía el mensaje a través del sistema de comunicaciones
  - El receptor:
    - Recibe el mensaje del sistema de comunicaciones
    - Extrae la información relevante
      - Qué hacer y sobre qué datos

Comunicaciones

- Comunicación mediante mensajes
  - Ventajas e inconvenientes:
    - Bajo acoplamiento: todo va en el mensaje
    - Fácil de implementar
    - Muy flexible: nuevos mensajes/datos
    - Se requiere cierta estandarización: formato mensajes y datos
    - En ocasiones no se sabe qué componentes aceptan qué mensajes

Comunicaciones

- Comunicación mediante interfaces
  - El emisor se comunica mediante una interfaz que define qué acciones pueden ser invocadas por una aplicación
    - Ejemplo: DescargarFichero('ff.txt')
  - La interfaz define las operaciones que se pueden realizar
    - Generalmente también se define la forma de intercambiar los parámetros y el resultado
  - El emisor:
    - Crea una invocación o llamada al componente
    - Ejecuta la llamada
  - El receptor:
    - Recibe la invocación remota
    - Ejecuta la acción basada en la interfaz particular que se usa

Comunicaciones

- Comunicación mediante interfaces
  - Ventajas e inconvenientes:
    - Muy similar a una llamada a un procedimiento convencional
    - Más fácil de reutilizar y mantener
    - Mayor acoplamiento
      - Cambios afectan a otros módulos

Aplicaciones y sistemas distribuidos

Arquitecturas

Rubén Usamentiaga
Área de Arquitectura y Tecnología de Computadores
Universidad de Oviedo

Aplicaciones y sistemas distribuidos

Introducción

Introducción

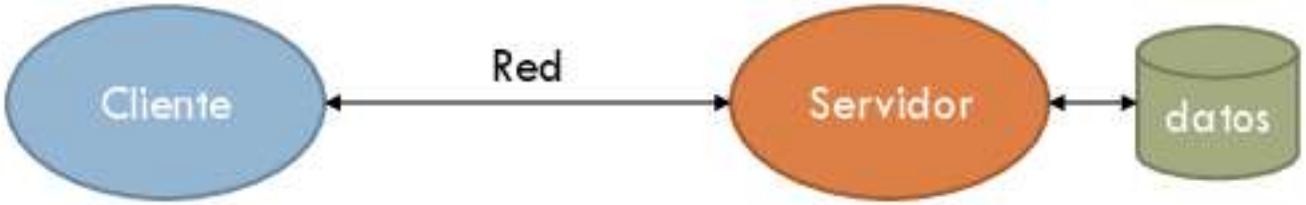
- Arquitectura
  - Solución estructurada a un sistema, describiendo sus componentes y relaciones
- Principios fundamentales
  - Separación de responsabilidades
  - Abstracción
  - Diseño para el cambio

Aplicaciones y sistemas distribuidos

Arquitectura cliente/servidor

Arquitecturas

- Arquitectura cliente/servidor
  - Dos componentes principales:
    - Cliente
      - Realiza peticiones a un servidor
    - Servidor
      - Recibe las peticiones, las procesa y produce una respuesta



Arquitecturas

- Arquitectura cliente/servidor
  - Tipos de cliente:
    - Ligero
      - Toda la lógica de la aplicación la lleva a cabo el servidor
      - El cliente se encarga únicamente de mostrar la información
    - Grueso
      - El servidor se encarga de gestionar los datos
      - El cliente implementa cierta lógica e interacciones del sistema

Arquitecturas

- Arquitectura cliente/servidor
  - Tendencias:
    - Cliente (frontend)
      - Plataforma tecnológica: Web, móvil, escritorio
      - Web rica: HTML, CSS, Javascript
    - Servidor (backend)
      - API REST accesible por HTTP
      - Proporciona los servicios de procesamiento pesados
      - Proporciona persistencia de información
        - Ubiquidad

Arquitecturas

- Framework del backend web
  - Enrutado
    - Mapean peticiones HTTP a una función de un controlador
    - Usan expresiones regulares
    - Pueden incluir parámetros
  - Decoradores
    - Añaden lógica adicional a las funciones del controlador
    - Ejemplo: necesita_autorización
  - Controlador
    - Contiene las funciones que responden a las peticiones

Arquitecturas

- Framework del backend web
  - ORM (Object-relational mapping)
    - Capa de abstracción con la base de datos
    - Mapea objetos y registros de forma trasparente
  - Plantillas
    - Combina un fichero base (HTML) con los datos dinámicos
    - No mezclar contenido con presentación
  - Serializador
    - Convierte objetos al formato requerido por la petición de forma trasparente (JSON, XML)

Arquitecturas

- Framework del backend web
  - Ejemplo: Python Bottle

```
@route('/')
@route('/hello/<name>')
def greet(name='Stranger'):
    return template('Hello {{name}}, how are you?', name=name)


@app.route('/book', method="GET")
def getgbook():
    result = decode_list(r.lrange("book", 0, -1))
    return {"result": result}
```

Arquitecturas

- Framework del backend web
  - Ejemplo: .NET Core

```
public class ProgramQuery
{
   public string source;
}

public class ProgramResponse
{
   public string result;
   public string stats;
}

[Route("api/[controller]")]
public class ProgramController : Controller
{
   [HttpPost]
   public IActionResult RunProgram([FromBody]ProgramQuery program)
   {
      ProgramResponse results = lua.RunProgram(program.source);
      return Ok(results);
   }
}
```

Aplicaciones y sistemas distribuidos

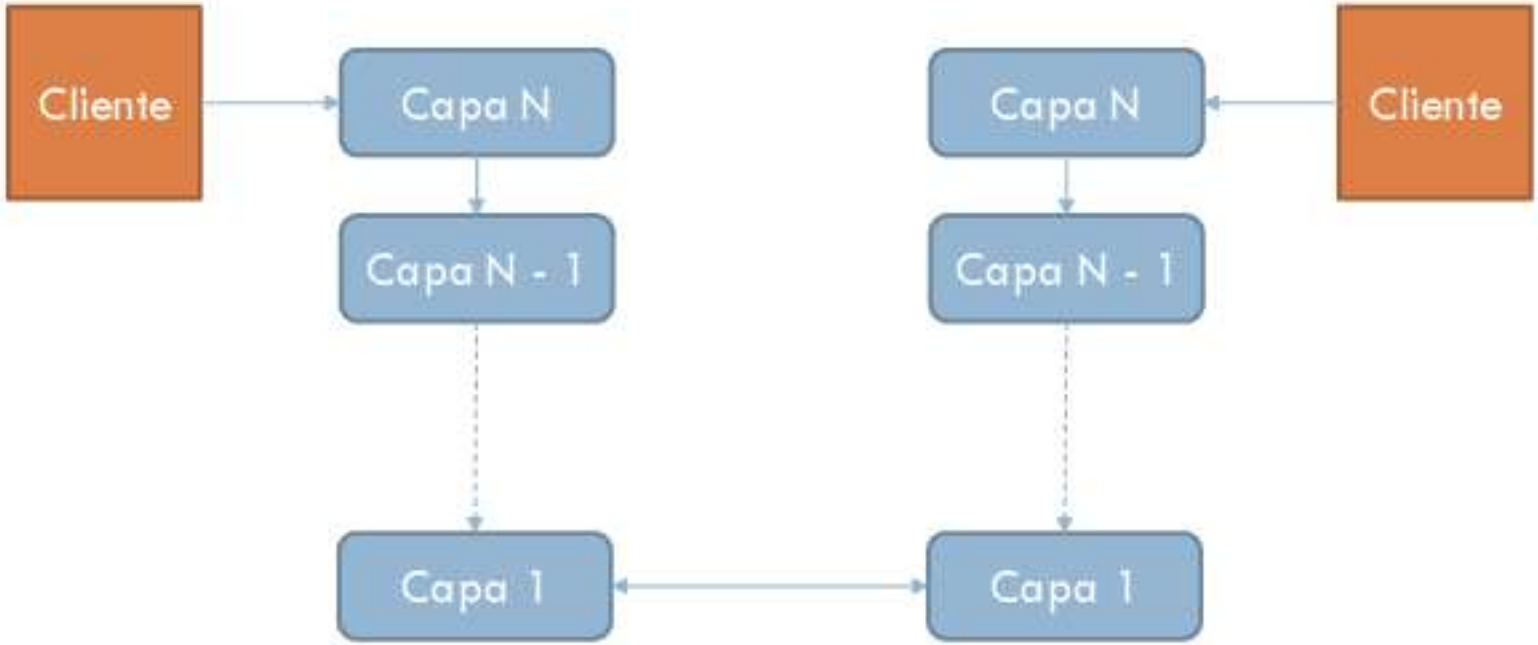
Arquitecturas en capas

Arquitecturas

- Arquitectura en capas (*Layers*)
  - Permite estructurar una aplicación que puede ser descompuesta en grupos de tareas con un nivel particular de abstracción
  - Múltiples componentes de una capa usan el mismo componente de la capa inferior
    - Ejemplo: HTTP, FTP, SMTP con TCP
  - Aplicabilidad:
    - Ejemplo clásico: protocolos de comunicaciones
    - Para resolver problemas con distintos niveles de abstracción

Arquitecturas

- Arquitectura en capas



Arquitecturas

- Arquitectura en capas
  - Ejemplo para aplicación genérica

Arquitecturas

- Arquitectura en capas
  - Pasos
    - Definir criterios de abstracción para el agrupamiento de tareas en capas
    - Definir el número de niveles de abstracción y dar nombre
    - Especificar servicios de cada capa y asignar tareas
    - Especificar interfaces entre capas
    - Establecer mecanismo de comunicación entre capas
    - Desacoplar capas adyacentes

Arquitecturas

- Arquitectura en capas
  - Ventajas
    - Alta cohesión
    - Bajo acoplamiento
    - Alta abstracción
    - Alta reusabilidad
    - Flexibilidad
    - Se anticipa la obsolescencia
    - Se facilita la portabilidad

Aplicaciones y sistemas distribuidos

Arquitecturas en tuberías y filtros

Arquitecturas

- Arquitectura de tuberías y filtros (*Pipes and filters*)
  - Proporciona una estructura para un proceso que se aplica de forma secuencial a un flujo de datos
  - Los datos se transfieren a través de *pipes*
    - Mecanismos de comunicación
  - Los filtros son bloques de procesamiento
    - Reciben datos por una entrada
    - Producen los resultados (transformación de la entrada) por una salida
  - Aplicabilidad:
    - Para procesar flujos de datos continuos

Arquitecturas

- Arquitectura de tuberías y filtros

Arquitecturas

- Arquitectura de tuberías y filtros
  - Ejemplo de procesamiento de texto
    - Línea de comandos

```
ls -al | grep Aug | grep -v '200[456]' | more
```

Arquitecturas

- Arquitectura de tuberías y filtros
  - Pasos
    - Dividir las tareas del sistema en un conjunto de etapas de procesamiento
    - Definir el formato de datos de entrada y salida para cada etapa
    - Definir el mecanismo de comunicación entre etapas

Aplicaciones y sistemas distribuidos

Arquitecturas de paso de mensajes

Arquitecturas

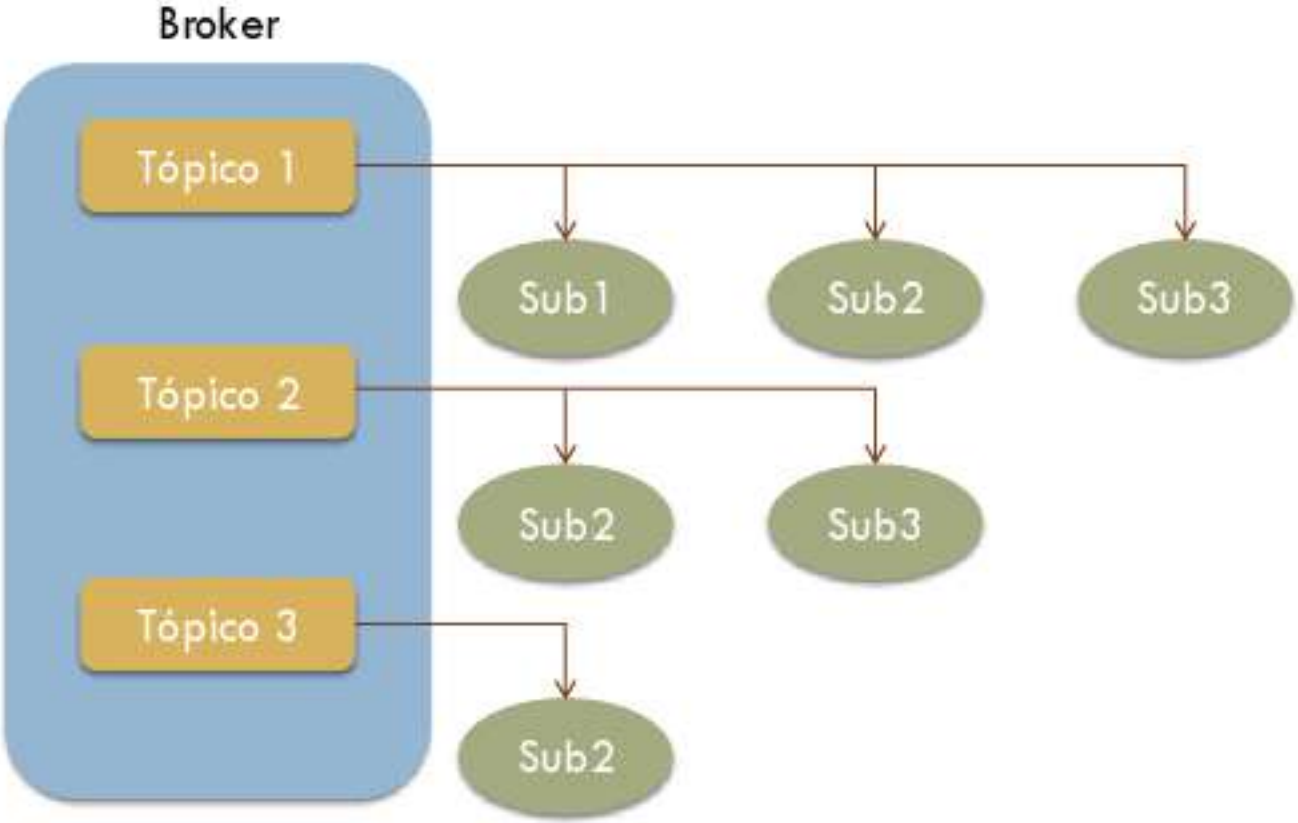
- Arquitectura de paso de mensajes
  - Proporciona un desacoplamiento entre componentes y los servicios utilizados
    - El bróker es responsable de coordinar las comunicaciones
    - Enviar las peticiones
    - Transmitir los resultados y las excepciones
  - Aplicabilidad:
    - Para invocar servicios locales o remotos de forma transparente

Arquitecturas

- Arquitectura de paso de mensajes

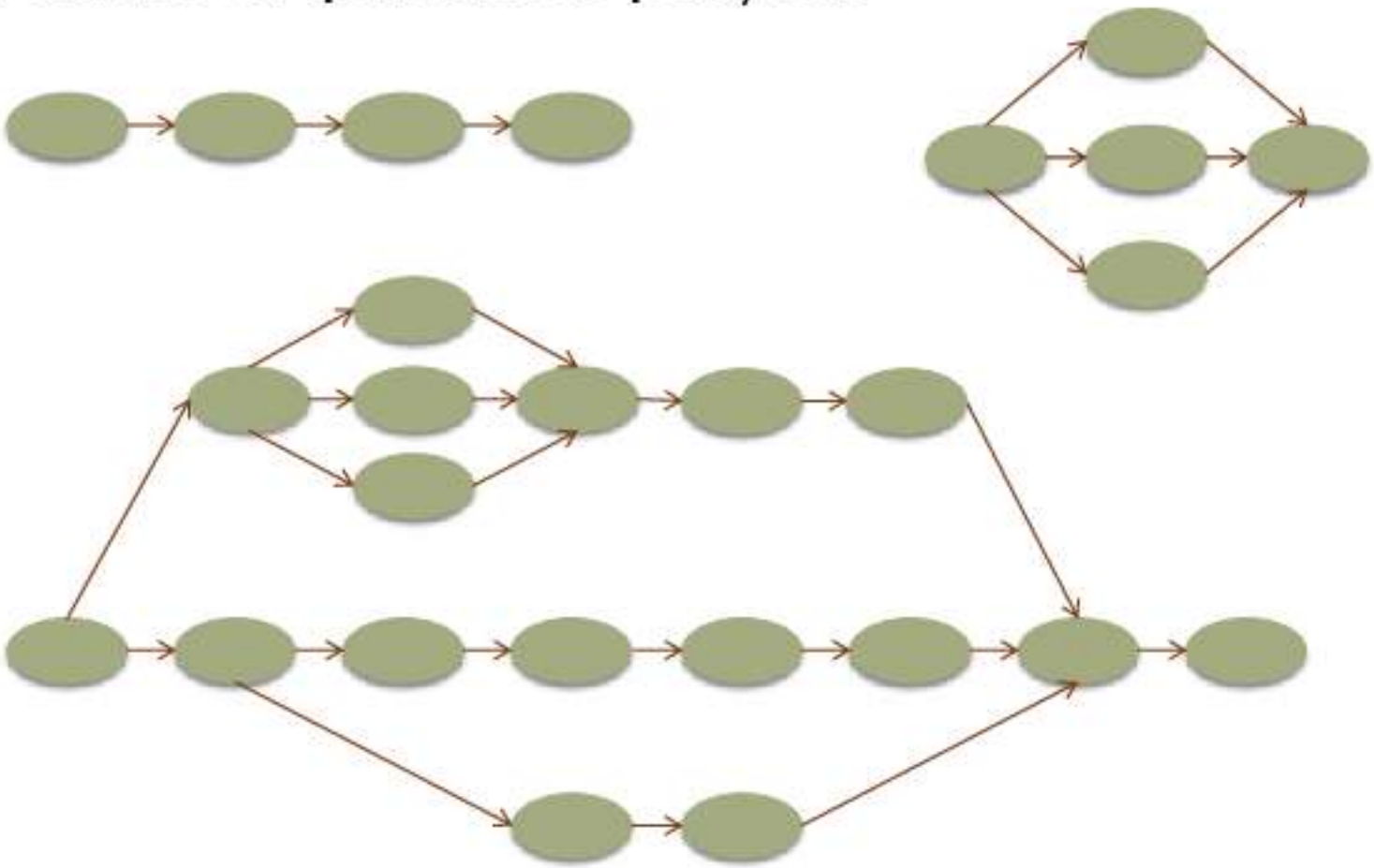
Arquitecturas

- Arquitectura de paso de mensajes
  - Gestión de un sistema *publish/subscribe*



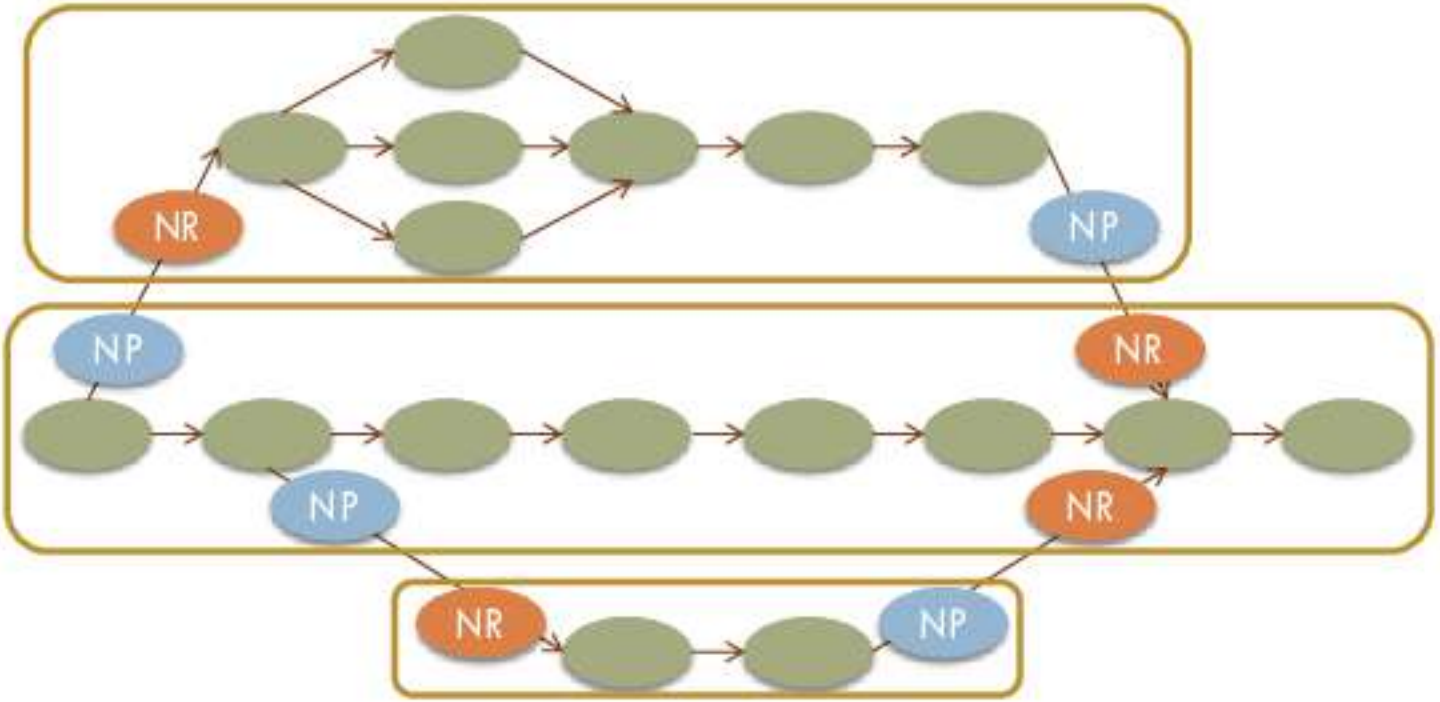
Arquitecturas

- Posibles arquitecturas pub/sub



Arquitecturas

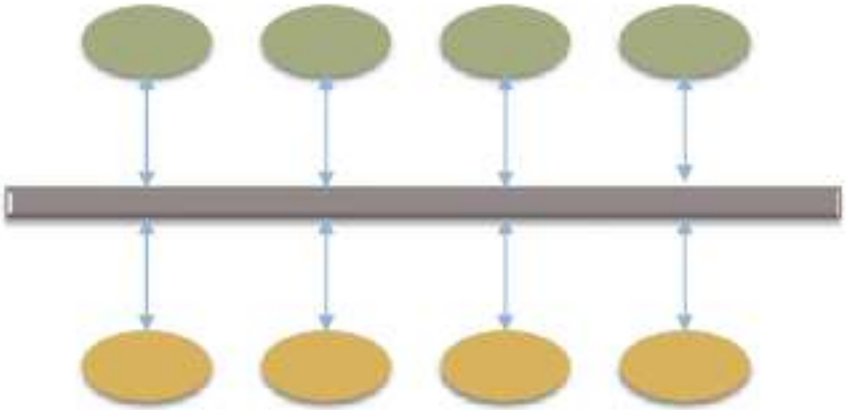
- Flexibilidad de la arquitectura pub/sub



NP: NetPublisher
NR: NetReceiver

Arquitecturas

- Enterprise Service Bus ESB
  - Bus que desacopla los proveedores de servicios de los consumidores



Aplicaciones y sistemas distribuidos

Arquitecturas basadas en componentes

Servicios web

- Arquitectura basada en componentes
  - Descomposición de un sistema en un conjunto de componentes individuales
  - Abstracción de mayor nivel siguiendo los principios de la orientación a objetos
    - Componente ≈ Objeto en lenguaje de programación
  - Cada componente expone una interfaz con métodos, eventos y propiedades
    - Modularidad → Reusabilidad →Abstracción, etc
  - Ejemplos: DCOM, Enterprise JavaBeans

Aplicaciones y sistemas distribuidos

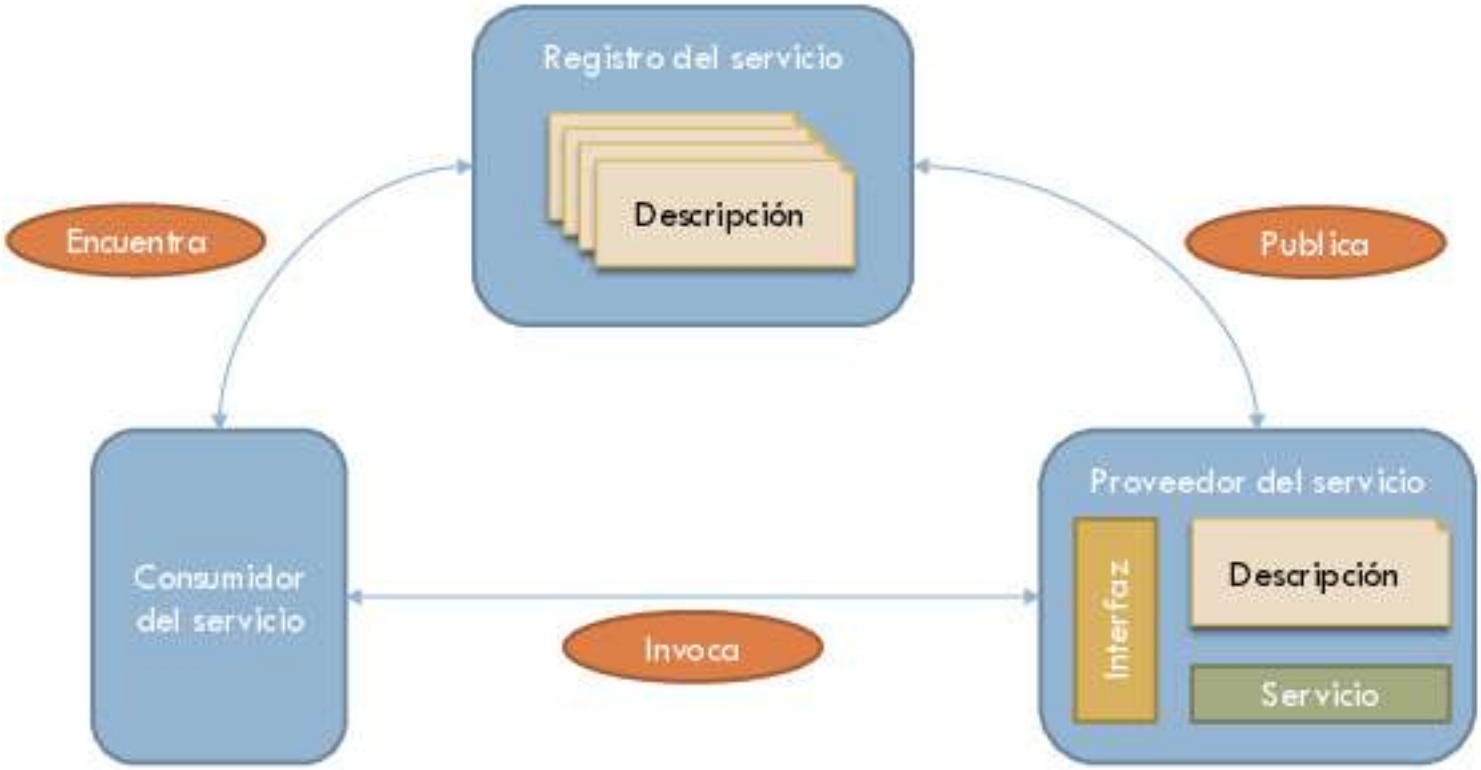
Arquitecturas orientadas a servicios

Arquitecturas

- Arquitecturas orientadas a servicios, SOA (Service-Oriented Architecture)
  - Mecanismo para diseñar, desplegar y gestionar aplicaciones
  - El software se organiza en servicios que son accesibles a través de la red
  - Las interfaces de los servicios usan estándares públicos para facilitar la interoperabilidad
  - Idea clave: servicio como componente reusable y de bajo acoplamiento

Arquitecturas

- Arquitectura SOA (Service-Oriented Architecture)



Arquitecturas

- Arquitecturas orientadas a servicios
  - Se desacopla el servicio de la aplicación que lo usa
  - Cada servicio es independiente de los demás
  - Los cambios en el servicio no afectan a la aplicación que los usa si se mantiene la interfaz
  - La plataforma tecnológica de cada servicio y de las aplicaciones que los usan son independientes
  - Favorece la reusabilidad de los servicios
  - Favorece la escalabilidad y la detección de cuellos de botella
  - Favorece la abstracción de los servicios

Arquitecturas

- Arquitecturas orientadas a servicios
  - Características que se persiguen:
    - Desacoplamiento
    - Flexibilidad
    - Interoperabilidad
    - Reusabilidad

Arquitecturas

- Arquitecturas orientadas a servicios
  - Arquitectura ligada a los servicios web pesados
    - XML como mecanismo de intercambio de información
    - Definición de estándares (WS-*) para los protocolos, formatos de datos (SOAP) e interoperabilidad (WSDL)
    - Persigue la interoperabilidad entre fabricantes
      - Evitar los problemas del pasado: RPC, DCOM, CORBA, RMI
      - Evitar lock-in de los fabricantes
    - Aplicaciones de integración empresarial

Aplicaciones y sistemas distribuidos

Arquitecturas de microservicios

Arquitecturas

- Arquitectura de microservicios
  - Arquitectura para solucionar algunos de los problemas que plantean las aplicaciones monolíticas
    - Código base enorme
      - Un pequeño cambio requiere conocer toda la arquitectura
    - Necesitan mucho tiempo para compilar y testing
    - Limitaciones para escalar (múltiples copias tras un balanceador de carga)
    - Dificultades para desplegar y actualizar
    - Dificultades para integrar distintas pilas tecnológicas
    - Dificultades para gestionar múltiples equipos de trabajo

http://www.martinfowler.com/articles/microservices.html

Arquitecturas

- Arquitectura de microservicios
  - Micro vs Macro
    - Microkernels vs monolithic kernels
    - RISC vs CISC

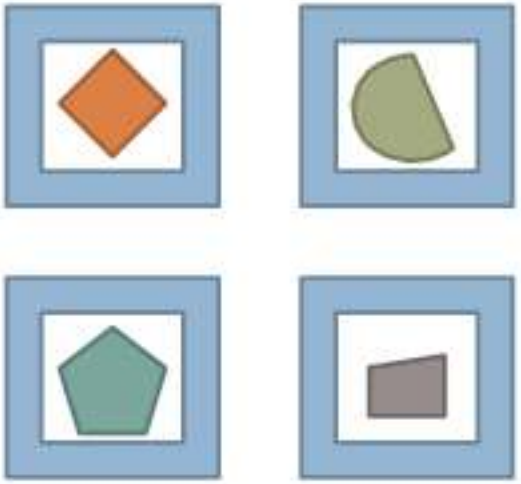
Arquitecturas

- Arquitectura de microservicios
  - Descomposición funcional de un sistema en un conjunto de pequeños servicios independientes y manejables
    - Surgen de la observación de la forma de trabajo de empresas con éxito en el sector
      - Amazon, Netflix
    - Permiten un crecimiento ágil
    - Permiten una división del trabajo eficiente
      - Pequeños grupos de 5-9 personas
    - Permiten una evolución y desarrollo independiente

Arquitecturas

- Arquitectura de microservicios

Aplicación monolítica:
Toda la funcionalidad en un solo proceso



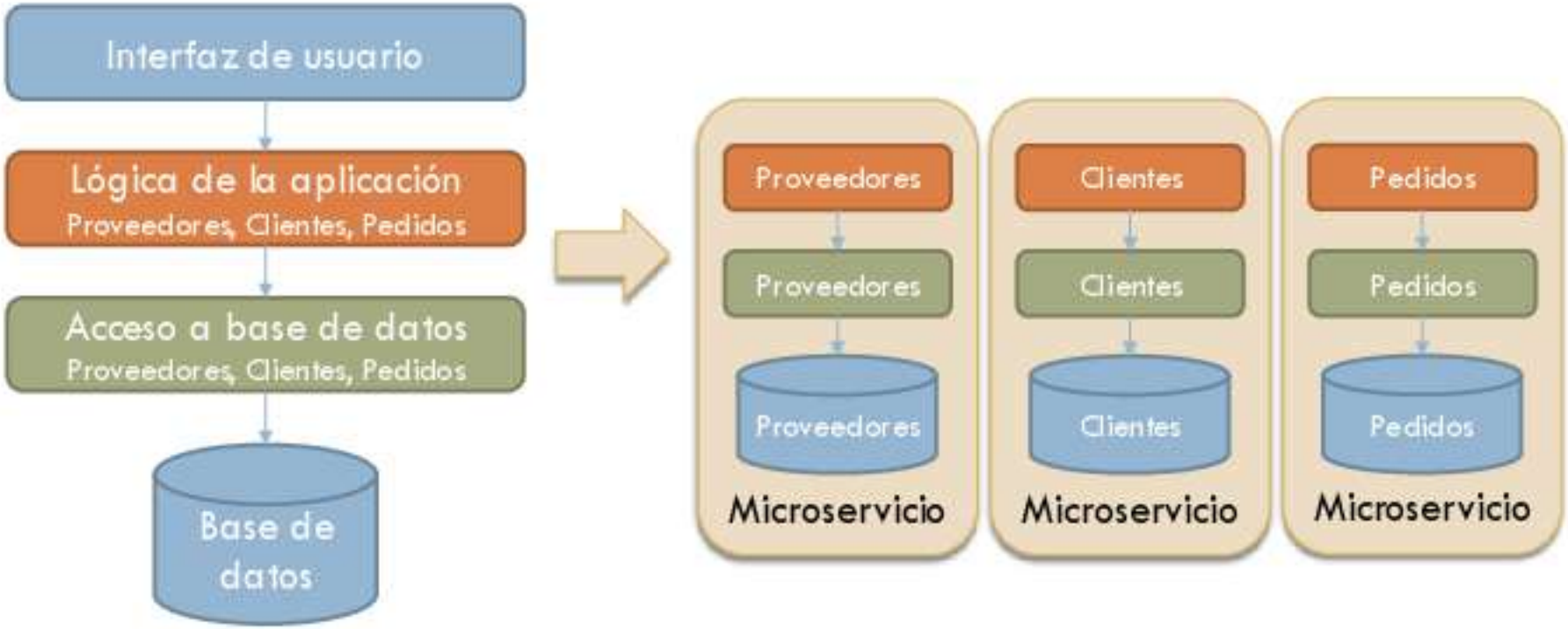
Arquitectura de microservicios:
La funcionalidad se divide en servicios ejecutándose en procesos independientes

Arquitecturas

- Arquitectura de microservicios
  - Microservicio
    - Gestionable por un solo equipo de trabajo
    - Surge de una descomposición vertical por funcionalidad
      - La descomposición horizontal sería la arquitectura de capas
    - Independientes y desacoplados
      - Desplegables y escalado de forma independiente (continua)
      - Pila de tecnologías específica
      - No tienen estado compartido con otros servicios
      - Tienen su propio repositorio
    - Funcionalmente completos
      - Gestión de recursos y datos
    - Ligeros que usan tecnologías ligeras

Arquitecturas

- Arquitectura de microservicios



Arquitecturas

- Arquitectura de microservicios
  - Muchas de las ventajas están relacionadas con el desacoplamiento
    - El desacoplamiento conlleva facilidad en la compresión, en el mantenimiento y en la introducción de cambios
    - Los cambios en una parte no afectan a todo lo demás
    - Facilitan la modularidad y el intercambio de módulos
  - Similitudes con el diseño de un programa
    - Modularidad, separación de interfaz e implementación, y todas sus ventajas

Arquitecturas

- Arquitectura de microservicios
  - Robustez de la arquitectura
    - Los fallos están contenidos dentro del microservicio y no afectan a todo el sistema
    - Más fácil de gestionar, detectar y arreglar
    - Más fácil gestionar la monitorización sobre una infraestructura
      - Ejemplo: Chaos monkey

Arquitecturas

- Arquitectura de microservicios
  - Comunicación
    - Síncrona
      - HTTP
        - Protocolo de comunicación
      - REST
        - Forma de diseñar servicios web
      - JSON
        - Formato de los datos
    - Asíncrona
      - Paso de mensajes
        - Advanced Message Queuing Protocol

Arquitecturas

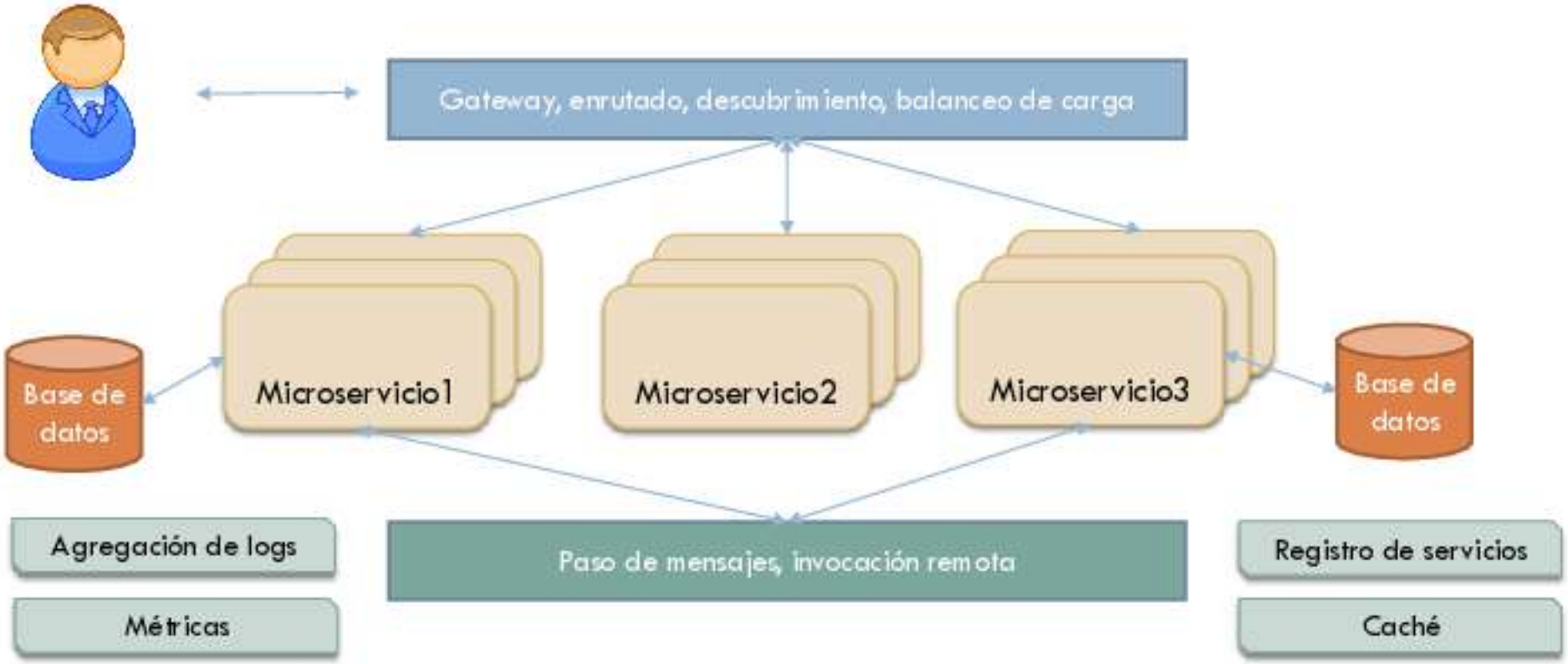
- Arquitectura de microservicios
  - Problemáticas
    - La comunicación es más costosa que en una aplicación monolítica
    - Un sistema distribuido es más complejo de gestionar
    - Necesitan múltiples bases de datos
      - En ocasiones con información redundante
      - Es necesario mantener consistencia
    - Dificultades en la reasignación de responsabilidades
    - Posibilidades de duplicación de esfuerzos

Arquitecturas

- Arquitectura de microservicios
  - Comparación con SOA
    - Se persiguen muchos de los mismos objetivos
    - Muchos lo ven como una versión ligera de SOA
      - Fine-grained SOA
    - Los microservicios deben ser completamente independientes
      - No comparten datos y su despliegue es independiente
    - Las tecnologías son completamente diferentes
      - Menos estándar ad-hoc y más agilidad

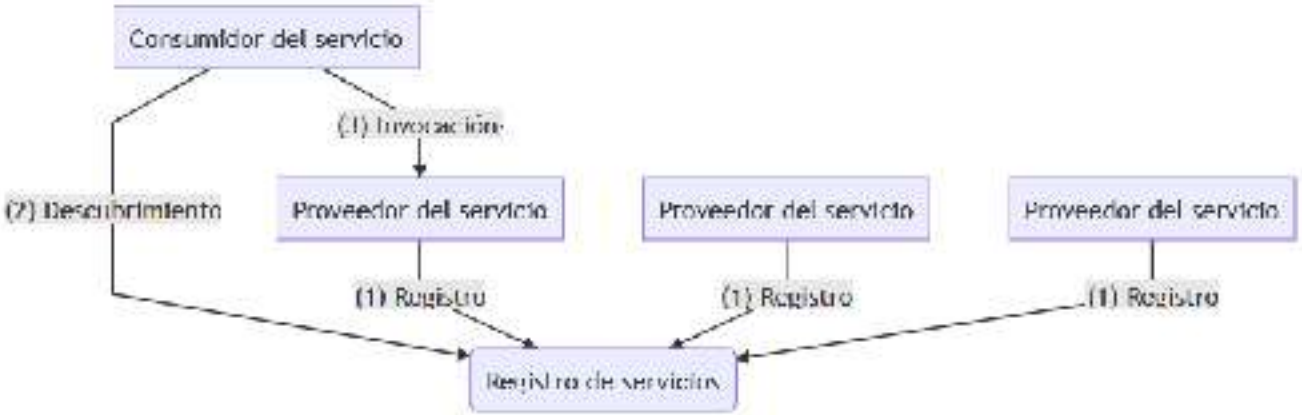
Arquitecturas

- Arquitectura de microservicios



Arquitecturas

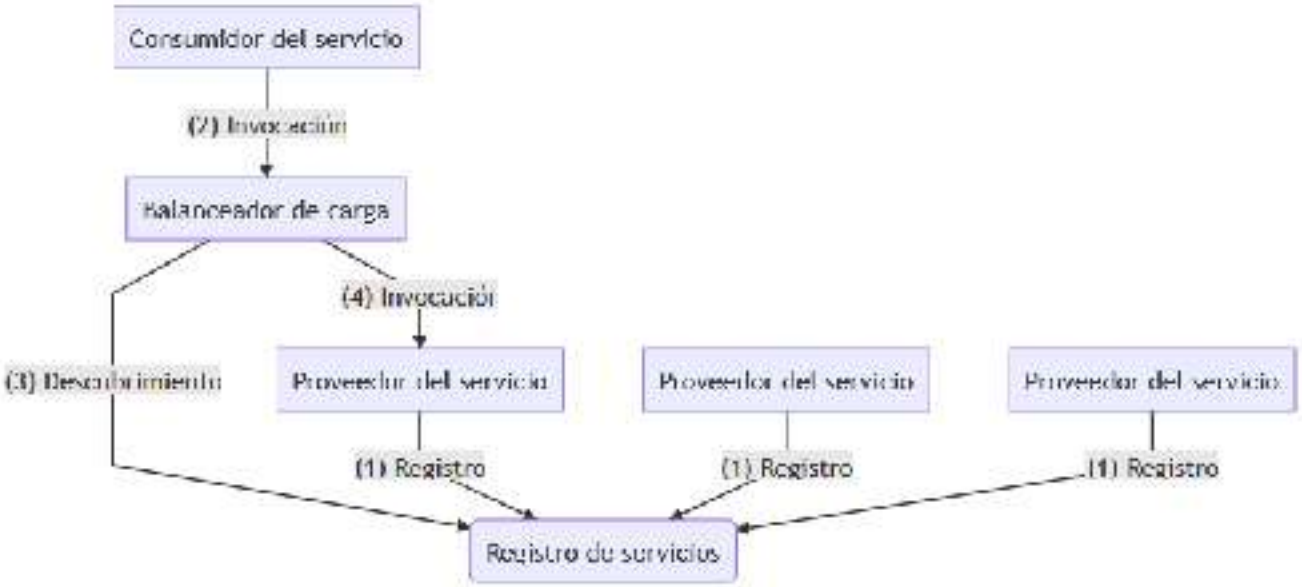
- Arquitectura de microservicios
  - Client-side service discovery



Self-Registration
3rd registration

Arquitecturas

- Arquitectura de microservicios
  - Server-side service discovery



Self-Registration
3rd registration

Aplicaciones y sistemas distribuidos

Metodología twelve-factor app

Arquitecturas

- The twelve-factor app
  - Metodología para construir aplicaciones
    - Diseñadas para escalar en un sistema distribuido
    - Siguiendo un modelo de software como servicio
    - Diseñadas para ser desplegadas en la nube
      - Sin preocuparse de los servidores ni de la administración de sistemas
    - Utilizando formatos declarativos para automatizar el desarrollo

https://12factor.net/

Arquitecturas

- The twelve-factor app
  1. Código base
     - Uno código base bajo control de versiones
  2. Dependencias
     - Declaración explicita de dependencias con aislamiento
  3. Configuración
     - Almacenar la configuración en el entorno
  4. *Backing services*
     - Tratar los Backing services como recursos conectables
  5. Construir, desplegar, ejecutar
     - Separar las fases de compilación, lanzamiento y ejecución
  6. Procesos
     - Ejecutar las aplicaciones en uno o más procesos sin estado

Arquitecturas

- The twelve-factor app
  7. Asignación de puertos
     - Publicar servicios exponiendo puertos
  8. Concurrencia
     - Escalar mediante el modelo de proceso
  9. Desechabilidad
     - Maximizar robustez con arranques rápidos y terminación segura
  10. Paridad desarrollo/producción
     - Similitud entre las fases de desarrollo y producción
  11. Logs
     - Tratar los logs como flujos de eventos
  12. Procesos de administración
     - Tareas de administración como procesos independientes

Arquitecturas

- The twelve-factor app (actualización)
  13. Primero la API (API First)
      - Todo es un servicio. Se asume que el código será consumido por un diente front-end, gateway u otro servicio
  14. Telemetría
      - El diseño debe incluir la recopilación de datos de monitorización, específicos del dominio y de salud/sistema.
  15. Autenticación/Autorización
      - Implementación de gestión de identidad desde el principio

Aplicaciones y sistemas distribuidos

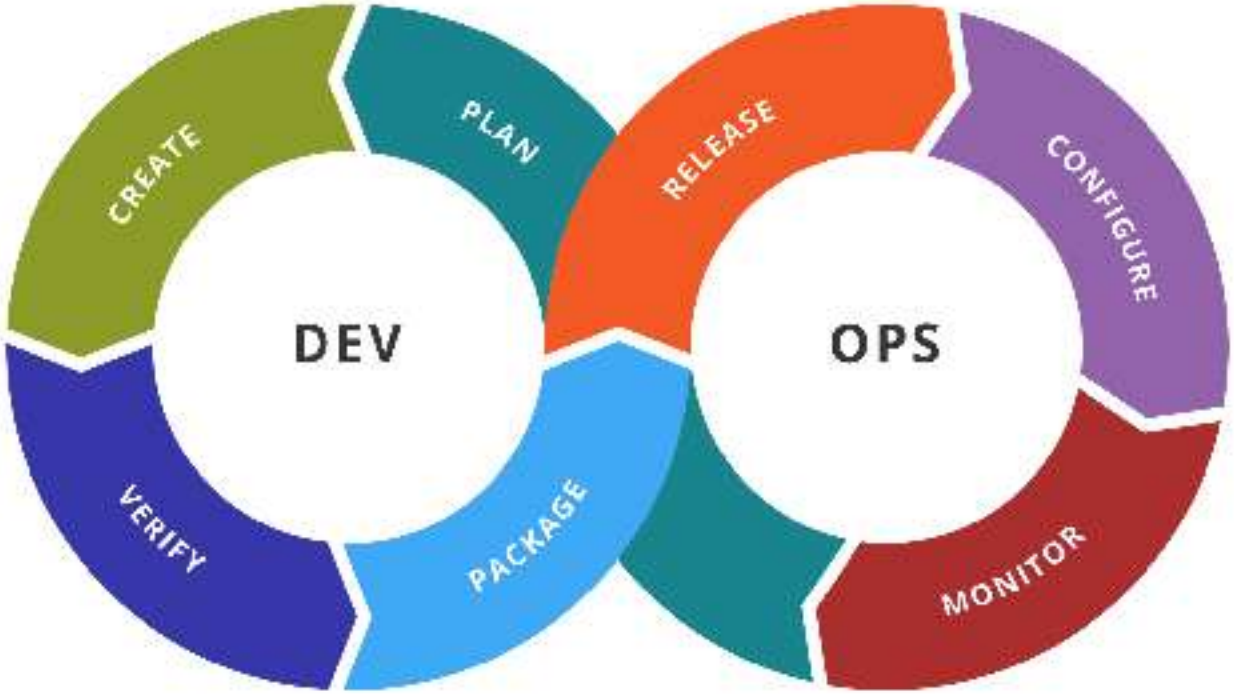
Cultura DevOps

Arquitecturas

- DevOps
  - Acrónimo inglés de development y operations
  - Objetivo: unificar el desarrollo de software (Dev) y la operación del software (Ops)
  - Objetivo: automatizar
    - Construcción del software
    - Integración
    - Pruebas
    - Infraestructura
  - Fomenta ciclos de desarrollo más cortos y con mayor frecuencia

Arquitecturas

- DevOps



https://upload.wikimedia.org/wikipedia/commons/thumb/0/05/Devops-toolchain.svg/2000px-Devops-toolchain.svg.png

Arquitecturas

- Aspectos clave de DevOps
  - Integración continua
    - Automatizar integraciones del software de forma frecuente, incluyendo la construcción y pruebas para así poder detectar fallos cuanto antes
  - Entrega continua
    - Automatizar la generación de nuevas versiones del software de producción cuando sea necesario
  - Control de versiones
    - Automatizar la gestión de los cambios en el código fuente

https://www.martinfowler.com/

Arquitecturas

- Aspectos clave de DevOps
  - Planificación ágil
    - Metodología de desarrollo de software basada en el desarrollo iterativo e incremental
  - Monitorización y logs
    - Utilización de sistemas de monitorización y logs para obtener feedback del sistema en ejecución
  - Computación en la nube
    - Uso de la nube como plataforma de ejecución de aplicaciones

Arquitecturas

- Aspectos clave de DevOps
  - Microservicios
    - Desacoplar casos de uso de negocio como pequeños servicios reusables que se comunican mediante contratos
  - Contenedores
    - Capa de virtualización ligera para la ejecución de aplicaciones
  - Infraestructura como código
    - Automatizar la creación de entornos de ejecución usando código para describir la infraestructura

Aplicaciones y sistemas distribuidos

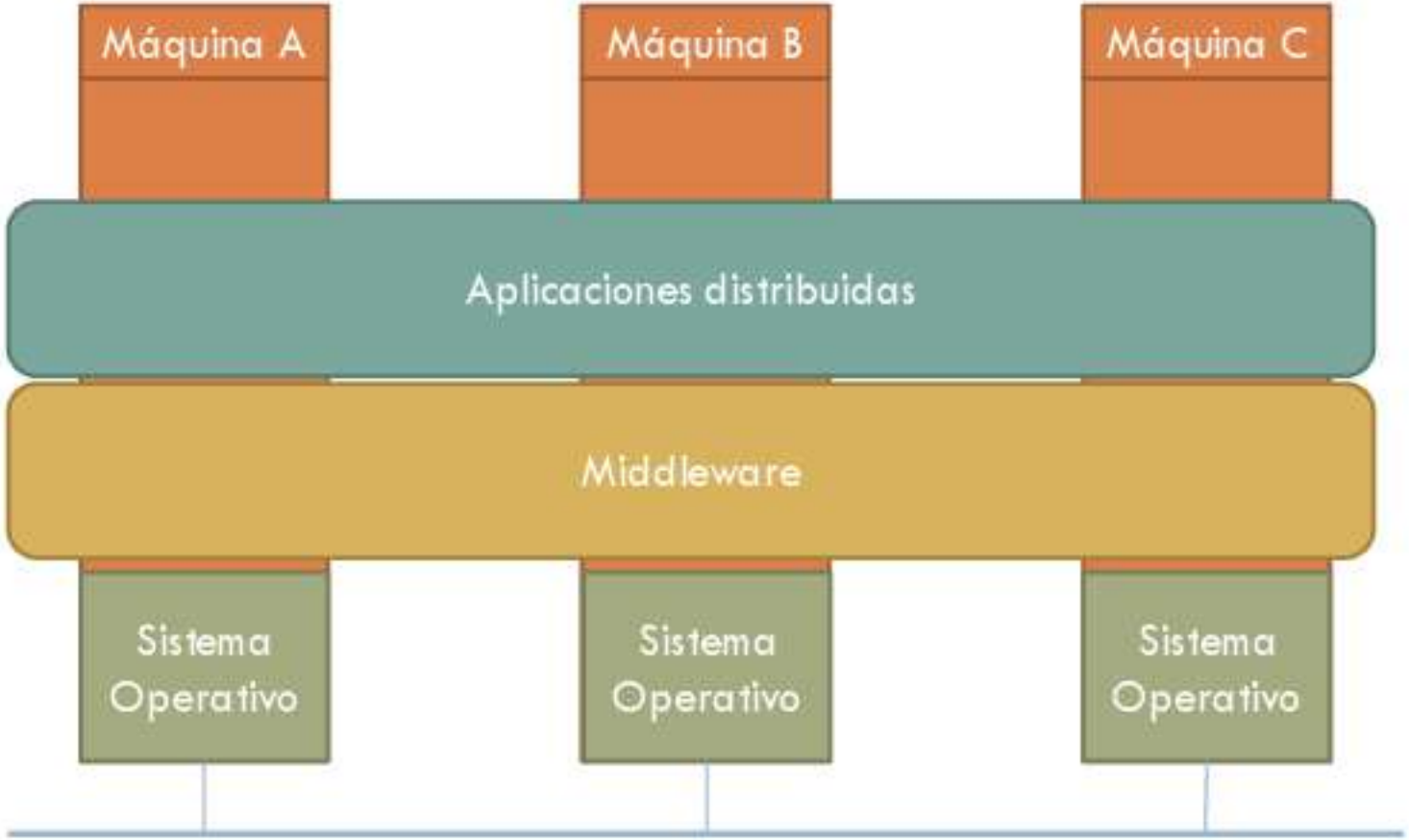
Integración y tecnologías

Rubén Usamentiaga
Área de Arquitectura y Tecnología de Computadores
Universidad de Oviedo

Aplicaciones y sistemas distribuidos

Integración y tecnologías

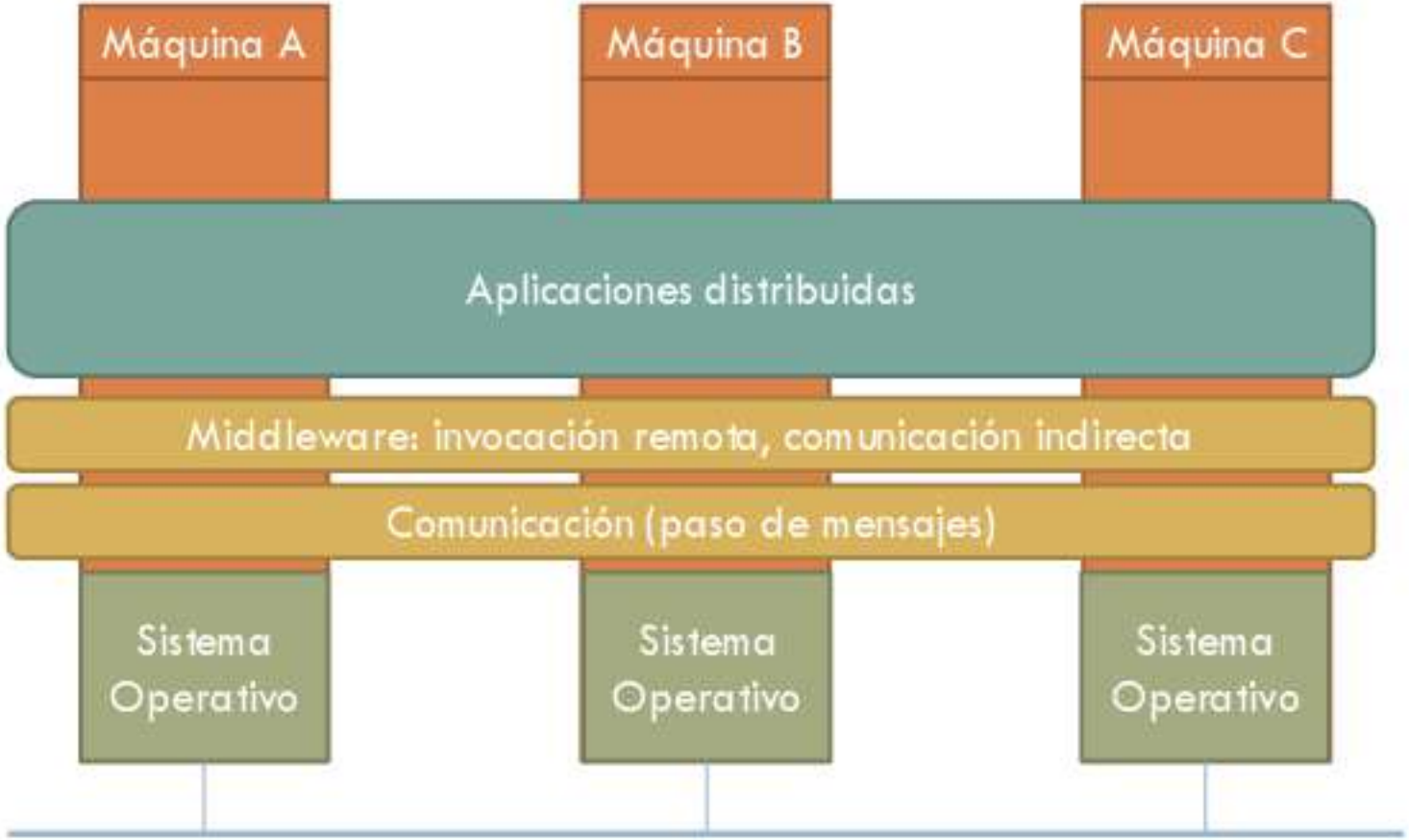
Integración



Integración

- Middleware:
  - Capa entre las aplicaciones y el sistema operativo
    - Oculta la heterogeneidad
    - Oculta la localización
    - Provee servicios genéricos
    - Permite integrar aplicaciones
    - Permite interoperabilidad
    - Incremental el nivel de abstracción
    - Independiente de plataforma y de la red

Integración



Integración

- Bajo el middleware:
  - Paso de mensajes
    - Primitivas
      - Enviar
      - Recibir
    - Modelos
      - Con conexión
      - Sin conexión
    - Ejemplo: sockets
      - Punto a punto
      - Multicast
      - Broadcast

Integración

- Modelos de middleware:
  - Invocación remota
    - RPC, Corba, RMI, Servicios web
  - Paso de mensajes
  - Agentes móviles
  - Peer to peer

Aplicaciones y sistemas distribuidos

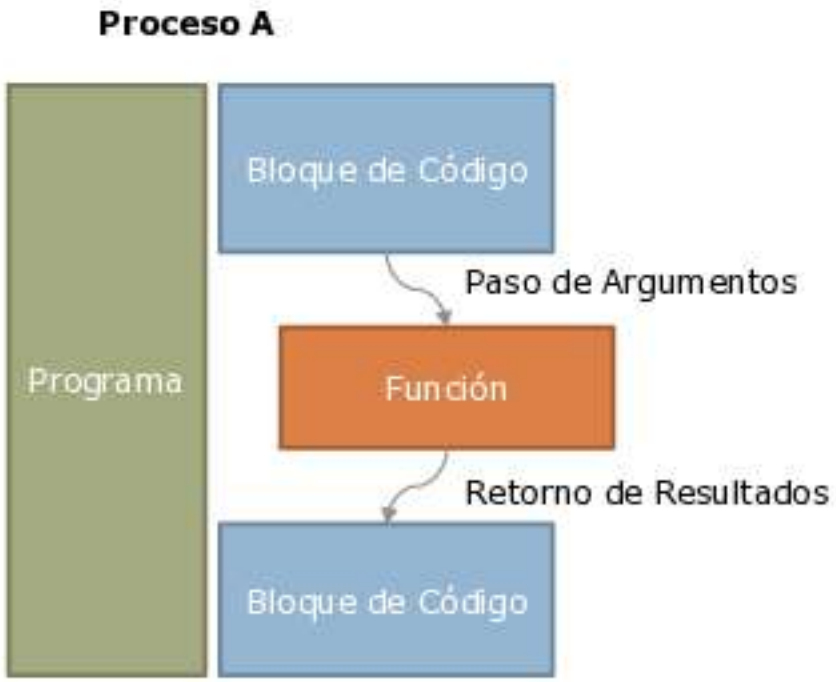
Integración mediante invocación remota

Invocación remota

- Invocación remota
  - Un mecanismo de abstracción básico es el módulo, función o procedimiento
    - Un subproblema se resuelve de forma independiente
    - Recibe unas entradas y produce un resultado
  - Objetivo de la invocación remota
    - Que la función se ejecute de remotamente pero de forma transparente para el que la invoca
    - Solucionando el problema del intercambio de la información
      - Proporciona una solución a la codificación de los parámetros de la función y del resultado

Invocación remota

- Pasos en la invocación local



Invocación remota

- Pasos en la invocación remota
  - El proceso invocador envía los parámetros en un mensaje por la red
    - Típicamente usando sockets
  - El proceso invocador se bloquea esperando el mensaje de respuesta
    - Ejemplo de comunicación síncrona
  - El proceso llamado está esperando conexiones
  - Recibe una conexión y lee el mensaje con los parámetros
  - Decodifica los parámetros y ejecuta el código de la función solicitada
  - Envía el resultado en un mensaje de respuesta
  - El proceso llamador recibe el mensaje de respuesta y se desbloquea

Invocación remota

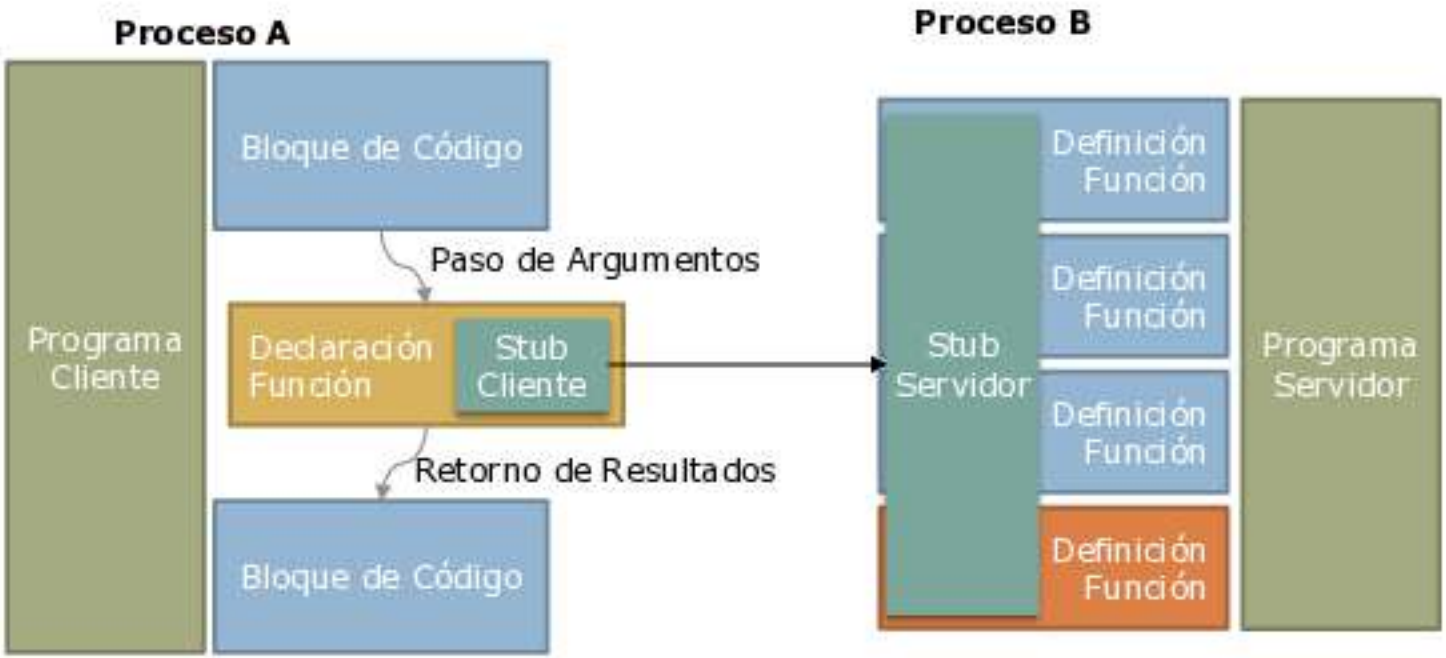
- Pasos en la invocación remota
  - Tareas
    - Localizar el servidor
      - Siempre de forma similar
    - Enviar y recibir información
      - Siempre de forma similar
    - Codificar y decodificar parámetros
      - Depende del tipo de los parámetros
    - Invocar a la función
      - Depende de la función

Invocación remota

- Soluciones a invocación remota
  - Localizar el servidor y enviar y recibir información
    - Se proporciona una biblioteca que implementa estas funciones
  - Codificar y decodificar parámetros, e invocar a la función
    - Se genera el código necesario de manera automática a partir de la definición de los tipos y de la función que se invoca
    - Se genera código para el cliente y para el servidor
      - Se denomina stub

Invocación remota

- Pasos en la invocación remota



Invocación remota

- Transparencia en la invocación remota
  - El cliente invoca a la función como si fuera local
  - Usa código que se genera de forma automática
    - Aparece el concepto de descripción de la interfaz
      - Define el contrato entre el invocador y el implementador
      - Define los tipos que se intercambian
  - Se solucionan los problemas que existen cuando se usan sockets directamente en un programa
  - Proporciona una abstracción que permite centrar el problema en otro tipo de aspectos

Invocación remota

- Middleware de invocación remota
  - ONC RPC
  - DCE RPC
  - Microsoft RPC (DCOM)
  - CORBA
  - Java RMI
  - .NET Remoting
  - Web Services (SOAP)
  - gRPC

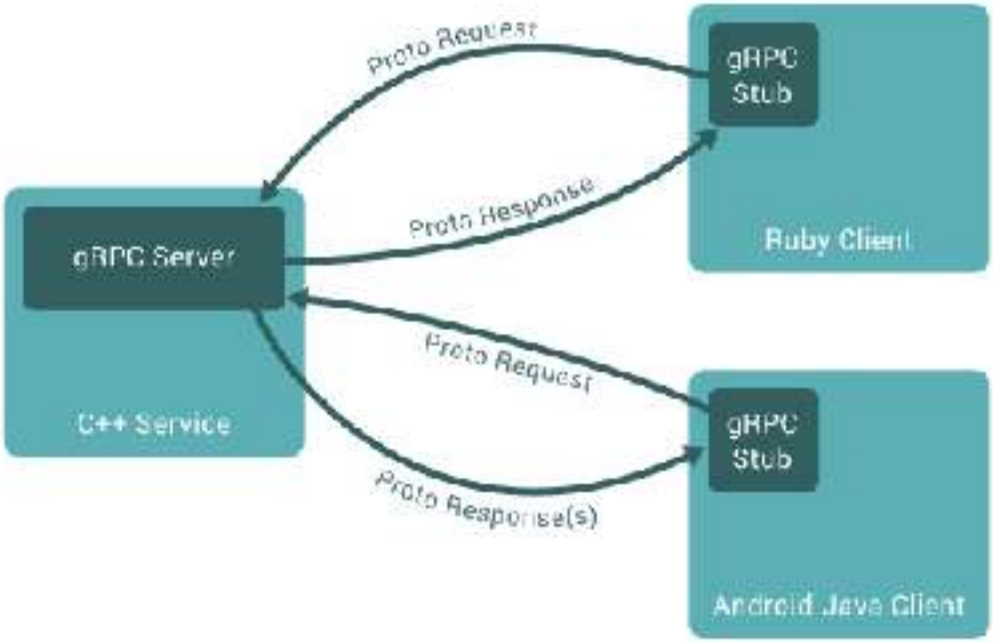
Invocación remota

- gRPC
  - RPC de Google
  - Usa HTTP para el transporte
  - Usa Protocol Buffers para la descripción de la interfaz y la serialización de los mensajes
  - Soporte para lenguajes populares: C++, Java, Python, Go, Ruby, Node.js, C#, Objective-C, PHP
  - Compilador de interfaz: protoc

https://grpc.io/

Invocación remota

- gRPC



https://grpc.io/

Invocación remota

- gRPC
  - Definición del servicio y mensajes

```
// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

https://grpc.io/

Invocación remota

- gRPC
  - Servidor

```
class GreeterImpl : Greeter.GreeterBase
{
    // Server side handler of the SayHello RPC
    public override Task<HelloReply> SayHello(HelloRequest request, ServerCallContext context)
    {
        return Task.FromResult(new HelloReply { Message = "Hello " + request.Name });
    }
}
...
class Program
{
    const int Port = 50051;

    public static void Main(string[] args)
    {
        Server server = new Server
        {
            Services = { Greeter.BindService(new GreeterImpl()) },
            Ports = { new ServerPort("localhost", Port, ServerCredentials.Insecure) }
        };
        server.Start();
        ...
```

Invocación remota

- gRPC
  - Cliente C#

```
public static void Main(string[] args)
{
    Channel channel = new Channel("127.0.0.1:50051", ChannelCredentials.Insecure);

    var client = new Greeter.GreeterClient(channel);
    String user = "you";

    var reply = client.SayHello(new HelloRequest { Name = user });
    Console.WriteLine("Greeting: " + reply.Message);

    var secondReply = client.SayHelloAgain(new HelloRequest { Name = user });
    Console.WriteLine("Greeting: " + secondReply.Message);

    channel.ShutdownAsync().Wait();
    Console.WriteLine("Press any key to exit...");
    Console.ReadKey();
}
```

https://grpc.io/

Invocación remota

- gRPC
  - Cliente JS

```
function main() {
  var client = new hello_proto.Greeter('localhost:50051',
                                       grpc.credentials.createInsecure());
  client.sayHello({name: 'you'}, function(err, response) {
    console.log('Greeting:', response.message);
  });
  client.sayHelloAgain({name: 'you'}, function(err, response) {
    console.log('Greeting:', response.message);
  });
}
```

https://grpc.io/

Sistemas Distribuidos

Servicios web

Servicios web

- La Web
  - Nace como repositorio de documentos accesibles a través de internet
  - Páginas estáticas en lenguaje HTML a través de HTTP
    - HTML: HyperText Markup Language
    - Desarrolla la idea del hipertexto como idea de enlazar la información de forma global
      - El origen se considera en un libro de Jorge Luis Borges de 1941
  - La idea clave fue combinar hipertexto e internet
  - Humanos como destinatarios de la información

Servicios web

- La Web

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Título</title>
  </head>
  <body>
    <p>Hola</p>
  </body>
</html>
```

Servicios web

- La Web dinámica
  - Pronto surge la necesidad de generar la información de forma dinámica
  - La petición HTTP no hace referencia a una página web, sino a un programa
  - El programa se ejecuta en el servidor y genera el HTML
    - Por ejemplo, con información extraída de una base de datos
  - Los humanos siguen siendo los destinatarios de la información

Servicios web

- La Web dinámica

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Título</title>
  </head>
  <body>
    <p>Sat, 14 Feb 2015 14:38:41 -0500</p>
  </body>
</html>
```

PHP

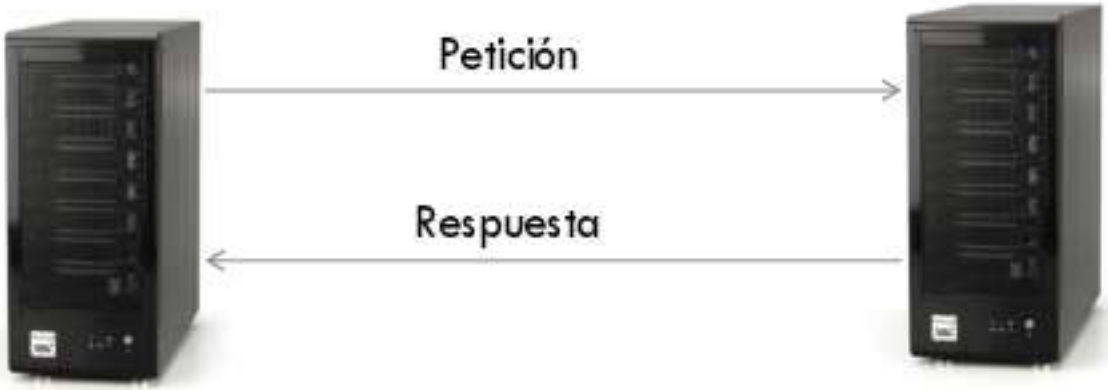
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Título</title>
  </head>
  <body>
    <p><?php echo date(DATE_RFC2822); ?></p>
  </body>
</html>
```

Servicios web

- Los servicios web
  - Idea similar a la web dinámica
    - Se realiza una petición a un programa en el servidor que genera un resultado de forma dinámica
    - Se usa HTTP para realizar una petición a un recurso
      - Se codifica la información adicional en la URL o en el cuerpo de la petición
    - La diferencia es que el resultado está diseño para ser interpretado por otros programas, no por humanos

Servicios web

- Los servicios web




```
[
{"ID":1,"name":"Rufo","longi":"0","l
ati":"0"},
{"ID":2,"name":"Jose","longi":"14","
lati":"0"},
{"ID":3,"name":"Rubén","longi":"123.
0","lati":"123.0"},
{"ID":4,"name":"Manuel","longi":"0",
"lati":"25"}
]
```

PHP

```
<?php

require("amigo_mapper.php");

$Mapper = new AmigoMapper;
$res = $Mapper->FindAll();

header('Content-type: application/json');
echo json_encode($res);
?>
```

Sistemas Distribuidos

Servicios web: XML-RPC

Servicios web

- XML-RPC
  - Mecanismo de invocación remota
    - Usa XML para los datos
    - Usa HTTP para el transporte
    - Simple e interoperable
      - Posibilita trabajar sin stub
  - Los parámetros y la respuesta se codifican en XML
    - Los parámetros se codifican en el cuerpo del mensaje HTTP (Tipo POST)
    - La respuesta se codifica en el cuerpo

http://xmlrpc.scripting.com/spec.html

Servicios web

- XML-RPC
  - Ejemplo de petición

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
   <methodName>examples.getStateName</methodName>
   <params>
      <param>
         <value><i4>41</i4></value>
      </param>
   </params>
</methodCall>
```

Servicios web

- XML-RPC
  - Ejemplo de respuesta

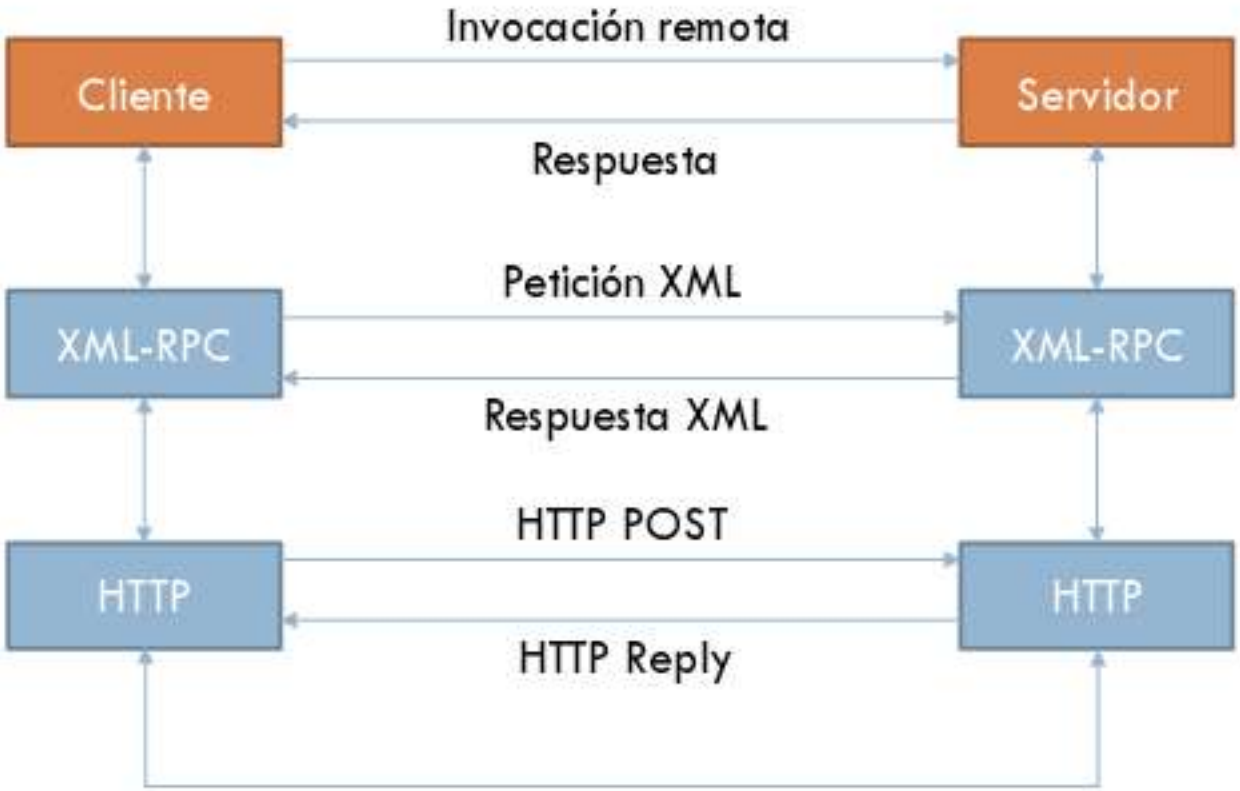
```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <value><string>South Dakota</string></value>
      </param>
   </params>
</methodResponse>
```

- XML-RPC
  - Ejemplo de respuesta con fallo

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 426
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:02 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?> <methodResponse> <fault>
   <value>
      <struct>
         <member>
            <name>faultCode</name>
            <value><int>4</int></value>
         </member>
         <member>
            <name>faultString</name>
            <value><string>Too many parameters.</string></value>
            </member> </struct> </value> </fault> </methodResponse>
```

Servicios web

- XML-RPC



Servicios web

- XML-RPC
  - Tipos

```
<int>123</int>
<double>-12.345</double>
<string>hello soap</string>
<boolean>1</boolean>
<dateTime.iso8601>2004-05-24T09:30:59
</dateTime.iso8601>
<base64>cualquier cosa</base64>

<array>
   <data>
      <value><i4>12</i4></value>
      <value><boolean>0</boolean></value>
      <value><i4>-31</i4></value>
   </data>
</array>
```

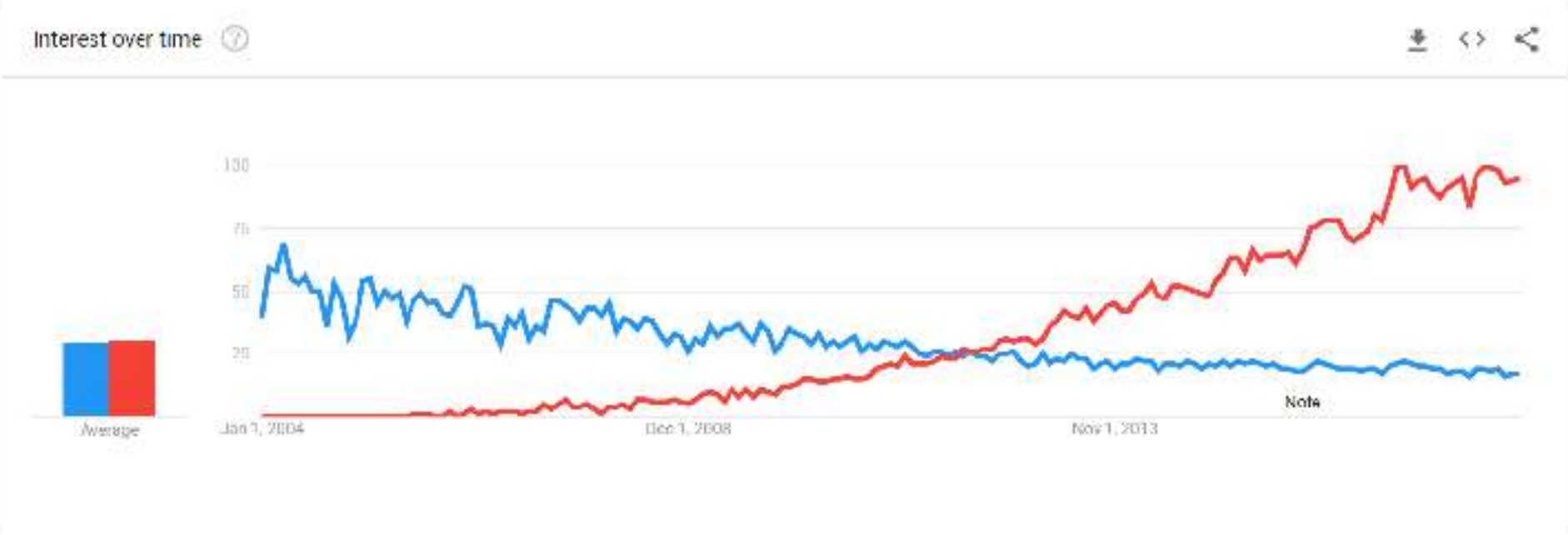
```
<struct>
   <member>
      <name>lowerBound</name>
      <value><i4>18</i4></value>
      </member>
   <member>
      <name>upperBound</name>
      <value><i4>139</i4></value>
   </member>
</struct>
```

Servicios web

- XML-RPC
  - Diseñado con el objetivo de ser extremadamente simple
    - La especificación tiene 6 páginas
    - "*understand what it's doing, and be able to modify it and have it work on the first or second try*" (Especificación)
  - Variante: JSON-RPC
    - Ejemplo de uso: Kodi
  - Muchas cuestiones no están resueltas
    - No busca solucionar todos los problemas
  - Sin el apoyo de las grandes empresas del sector

Servicios web

- XML-API vs JSON-API?



Sistemas Distribuidos

Servicios web: SOAP

Servicios web

- Servicios web pesados, grandes, SOAP
  - Principios similares a XML-RPC
    - Usa XML para los datos
    - Usa HTTP para el transporte (pueden ser otros)
  - Más extenso que XML-RPC
    - Proporciona más opciones y soluciones
  - Con el apoyo de grandes empresas (2000)
    - IBM y Microsoft

Servicios web

- Arquitectura de los servicios web (SOAP)
  - Descubrimiento:
    - Mecanismo para localizar y descubrir los servicios web
    - Solución más extendida: UDDI
  - Descripción:
    - Mecanismo para describir el servicio web (~IDL)
    - Solución más extendida: WSDL
  - Empaquetamiento:
    - Formato de los datos
    - Solución más extendida: SOAP
  - Transporte:
    - Mecanismo para realizar la comunicación
    - Solución más extendida: HTTP

Servicios web

- Arquitectura de los servicios web (SOAP)
  - Arquitectura SOA (Service-Oriented Architecture)

Servicios web

- SOAP
  - *Simple Object Access Protocol*
    - Estándar W3C basado en XML
  - Protocolo de comunicación basado en mensajes que permite el intercambio de datos estructurados entre aplicaciones de un entorno distribuido
  - Funciones:
    - Un mecanismo para describir qué hay en un mensaje y cómo se debe procesar
    - Un conjunto de reglas para incluir información específica de cada aplicación
    - Una convención para invocar llamadas a procedimientos remotos (RPC) y procesar sus respuestas

Servicios web

- SOAP
  - Estructura de un mensaje
    - Envoltorio (Envelope)
      - Define la raíz del documento XML
    - Cabecera (Header)
      - Información de control opcional
    - Cuerpo (Body)
      - El contenido del mensaje (payload)

Servicios web

- SOAP
  - Ejemplo de petición

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <Action s:mustUnderstand="1"
xmlns="http://schemas.microsoft.com/ws/2005/05/addressing/none">http://tem
puri.org/IMath/Prime</Action>
  </s:Header>
  <s:Body>
    <Prime xmlns="http://tempuri.org/">
      <value>50</value>
    </Prime>
  </s:Body>
</s:Envelope>
```

Servicios web

- SOAP
  - Ejemplo de respuesta

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header />
  <s:Body>
    <PrimeResponse xmlns="http://tempuri.org/">
      <PrimeResult>false</PrimeResult>
    </PrimeResponse>
  </s:Body>
</s:Envelope>
```

Servicios web

- SOAP
  - Algunos conceptos sobre XML
    - Diseñado para describir datos
      - Muy utilizado para compartir información
    - Espacios de nombres
      - Se usan para evitar conflictos con nombres iguales
        - Ej: xmlns="http://tempuri.org/"
    - Schema
      - Se usan para describir la estructura de un documento
      - Permite validar la información

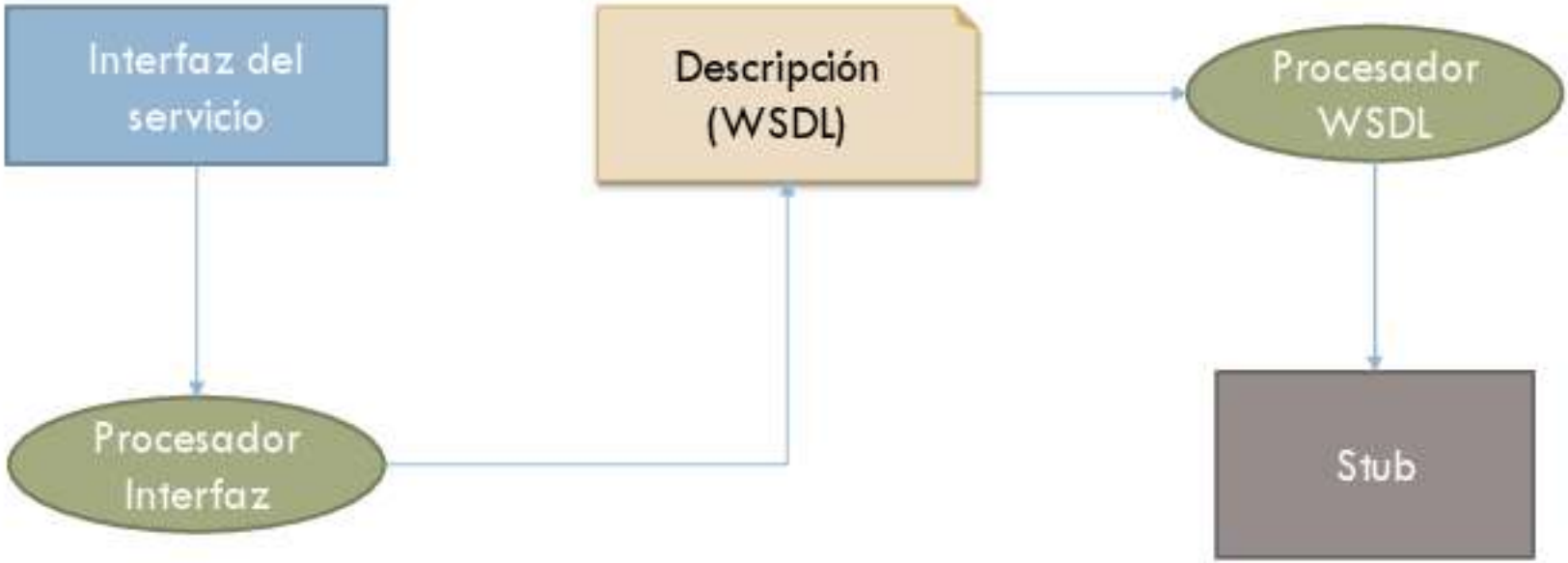
Servicios web

- WSDL
  - *Web Service Description Language*
    - Estándar W3C basado en XML
  - Mecanismo formal para definir el servicio
    - El tipo de mensaje que se debe enviar
    - Datos de entrada
    - Resultados que devuelve
    - Cómo realizar la invocación
  - Define la información que se incluirá dentro de un envoltorio SOAP
  - Define el protocolo mediante el cuál enviar el mensaje SOAP

```
<wsdl:portType name="IMath">
   <wsdl:operation name="Prime">
      <wsdl:input wsaw:Action="http://tempuri.org/IMath/Prime"
         message="tns:IMath_Prime_InputMessage"/>
      <wsdl:output wsaw:Action="http://tempuri.org/IMath/PrimeResponse"
         message="tns:IMath_Prime_OutputMessage"/>
   </wsdl:operation>

...

<wsdl:message name="IMath_Prime_InputMessage">
           <wsdl:part name="parameters" element="tns:Prime"/>
</wsdl:message>
<wsdl:message name="IMath_Prime_OutputMessage">
           <wsdl:part name="parameters" element="tns:PrimeResponse"/>
</wsdl:message>

...

<xs:element name="Prime">
   <xs:complexType>
      <xs:sequence>
                <xs:element minOccurs="0" name="value" type="xs:int"/>
     </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="PrimeResponse">
   <xs:complexType>
      <xs:sequence>
         <xs:element minOccurs="0" name="PrimeResult" type="xs:boolean"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Servicios web

- WSDL



Servicios web

- Ejemplo: WCF
  - Descripción del servicio

```
using System.Runtime.Serialization;
using System.ServiceModel;

namespace MathService
{
    [ServiceContract]
    public interface IMath
    {
        [OperationContract]
        bool Prime(int value);

        [OperationContract]
        Tuple Sum(Tuple tuple);
    }
...
```

```
    [DataContract]
    public class Tuple
    {
        double[] _data;
        string _name;

        [DataMember]
        public double[] Data
        {
            get { return _data; }
            set { _data = value; }
        }

        [DataMember]
        public string Name
        {
            get { return _name; }
            set { _name = value; }
        }
    }
}
```

Servicios web

- Ejemplo: WCF
  - Implementación del servicio

```
namespace MathService
{
    public class Math: IMath
    {
        public bool Prime(int value)
        {
...

        public Tuple Sum(Tuple tuple)
        {
...

            return tupleRes;
        }
    }
}
```

Servicios web

- Ejemplo: WCF
  - Hosting del servicio

```
namespace MathServiceHost
{
    class Program
    {
        static void Main(string[] args)
        {
            using (ServiceHost host = new ServiceHost(typeof(MathService.Math)))
            {
                host.Open();

                Console.WriteLine("Servicio ejecutandose. Pulsa <enter> para terminar");
                Console.ReadLine();

                host.Close();
            }
        }
    }
}
```

Servicios web

- Ejemplo: WCF
  - Invocación del servicio

```
namespace MathClientConsole
{
    class Program
    {
        static void Main(string[] args)
        {
            //MathClient client = new MathClient("NetTcpBinding_IMath");
            MathClient client = new MathClient("BasicHttpBinding_IMath");

            int x = 23;

            if (client.Prime(23))
                Console.WriteLine("{0} es primo", x);
            else
                Console.WriteLine("{0} no es primo", x);
        }
    }
}
```

Servicios web

- UDDI
  - Universal Description and Discovery Integration
  - Idea
    - Proporcionar un mecanismo centralizado de registro y descubrimiento de servicios
    - Los proveedores proporcionan descripciones en WSDL
    - Los clientes solicitantes las descripciones del servicio
  - Sin aplicación práctica

Servicios web

- WS-*
  - Conjunto de especificaciones que se han publicado en torno a los servicios Web
    - WS-I
    - WS-Addressing
    - WS-Security
    - WS-Transaction

Servicios web

- Servicios web pesados, grandes, SOAP
  - Muy estructurados
    - Muy complejos
  - Se consideran muy *verbose*
  - No encajan con la filosofía de la web
    - El registro es el DNS
  - Muy dependientes de herramientas adicionales
  - Muchas especificaciones
    - WS-*
  - XML está siendo sustituido por JSON
    - Nativo para Javascript

Sistemas Distribuidos

Servicios web: REST

Servicios web

- Servicios REST
  - Idea acuñada por Roy Fielding (tesis doctoral)
    - REST: Representational State Transfer
  - REST es un estilo de arquitectura para diseñar servicios web
    - Los servicios web que lo implementan se denominan RESTful
  - Su objetivo es aprovechar al máximo las características de HTTP
    - En XML-RPC y SOAP se usa una sola URL y un solo verbo
  - Aprovechar infraestructura Web
    - HTTP cache, proxy

the REST bible: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Servicios web

- Conceptos fundamentales sobre REST
  - Recursos
  - Representaciones
  - Operaciones
  - Hipertexto
  - *Statelessness*

Servicios web

- REST
  - Recursos
    - Cada recurso lleva asociado un identificador
    - El identificador usado en la web es la URL
      - La URL identifica dónde encontrar el recurso y cómo acceder a él
    - El identificador será
      - Legible
      - Fácil de depurar
      - Fácil de construir por los clientes
      - Corto
      - Con sentido
      - Predecible

Servicios web

- REST
  - Ejemplos de recursos
    - El pedido con código 1234
      - http://www.ejemplo.com/pedidos/1234
    - Todos los pedidos
      - http://www.ejemplo.com/pedidos
    - El producto código ABC del pedido 1234
      - http://www.ejemplo.com/pedidos/1234/producto/ABS
      - Se sigue una estructura jerárquica para que el identificador sea más fácil de entender

Servicios web

- REST
  - Representaciones
    - Usando HTTP los sistemas RESTful le dicen al servidor el formato en el que quieren recibir los datos
    - Se utiliza la cabecera '*Accept*' para indicar el tipo de datos que se quiere en la respuesta
    - En un servicio REST, el cliente típicamente solicita un recurso representado en XML o JSON
      - Accept: application/xml

Servicios web

- REST
  - Representaciones
    - Sin REST
      - http://www.ejemplo.com/pedidos/1234/data.txt
      - http://www.ejemplo.com/pedidos/1234/data.xls
      - http://www.ejemplo.com/pedidos/1234/data.htm
    - Con REST

```
GET /pedidos/1234 HTTP/1.1
Host: www.ejemplo.com
Accept: application/vnd.ms-excel
```

Servicios web

- REST
  - Representaciones
    - Con REST los recursos son independientes de su representación
      - No mezclar contenido con presentación
    - Si el servidor no soporta el tipo MIME solicitado
      - Retorna error de forma estándar (HTTP 406)
    - Ventajas
      - Se desacopla la información de su representación
      - Se permite negociar el contenido sin un protocolo ad hoc
      - El sistema es flexible y le permite adaptarse a los cambios
        - Añadir un nuevo mecanismo de codificación no modifica lo que ya existe (principio abierto pero cerrado)

Servicios web

- REST
  - Operaciones
    - Se definen cuatro operaciones básicas
      - GET
        - Recupera la representación de un recurso
      - DELETE
        - Borra un recurso
      - PUT
        - Modifica un recurso
      - POST
        - Crea un recurso
    - Las operaciones son verbos HTTP
      - CRUD: Create Read Update Delete

Servicios web

- REST
  - Ejemplos de operaciones
    - Obtiene la información sobre un producto
      - GET /productos/1234
    - Elimina un producto
      - DELETE /productos/1234
    - Crea un producto
      - POST /productos
      - La información sobre el producto va en el cuerpo del mensaje
    - Modifica un producto
      - PUT /productos/1234
      - La información sobre el producto va en el cuerpo del mensaje

Servicios web

- REST
  - Características de las operaciones
    - Segura
      - No modifica los recursos
    - Idempotente
      - No tiene efectos adicionales si es invocada más de una vez con los mismos parámetros
    - El cliente y el servidor deben asumir ciertas características para cada operación

Servicios web

- REST
  - Características de las operaciones
    - GET
      - Segura e idempotente
      - No modifica nada y se puede invocar múltiples veces sin efectos
    - DELETE
      - No segura pero si idempotente
      - Elimina y se puede invocar múltiples veces sin efectos
        - Las invocaciones segunda y sucesivas se ignorarán
    - PUT
      - No segura pero si idempotente
      - Modifica y se puede invocar múltiples veces
    - POST
      - No segura y NO idempotente
      - Crea un recurso cada vez que se invoca

Servicios web

- REST
  - POST idempotente
    - Requerido en diversas guidelines (ej. Microsoft)
    - La idempotencia es importante para construir una API tolerante a fallos
    - Ejecución de operación solamente si:
      - Repeatability-First-Sent (cabecera) está dentro de la ventana del servidor (diferencia con hora actual de más de 5 minutos)
      - La petición no ha sido vista previamente (Repeatability-Request-ID con valor único)
    - Otra opción: using a unique business constraint
      - Recurso con valor único en campo (ej: nombre de usuario)

Servicios web

- REST
  - Hipertexto
    - La información de retorno puede contener enlaces a recursos donde obtener información adicional
    - Seguir un enlace permite obtener más información

```
GET /api/tasks/1234 HTTP/1.1
HTTP/1.1 200 OK

Content-Type: application/xml
<Task Id="1234" Status="Active" DateCreated="2011-08-15" Owner="Sally" Category="Projects" >
    <link rel="self" href="/api/tasks/1234" method="GET" />
    <link rel="users" href="/api/tasks/1234/users" method="GET" />
    <link rel="complete" href="/api/tasks/1234" method="DELETE" />
    <link rel="update" href="/api/tasks/1234" method="PUT" />
</Task>
```

Servicios web

- REST
  - *Statelessness*
    - El cliente es responsable del estado de la aplicación
      - El servidor es responsable del estado del recurso
    - El servidor no mantiene información de estado de la comunicación con el cliente
      - En REST no hay estado
      - La petición debe incluir toda la información necesaria
    - Facilita la escalabilidad
      - Las peticiones REST se pueden dirigir a diversos servidores
      - Se puede balancear la carga fácilmente
    - Facilita la robustez
      - Si un servidor se cae se pueda usar otro de forma transparente

Servicios web

- REST
  - Gestión de errores
    - Se utilizan los código de respuesta HTTP
      - Categorías 2xx, 4xx
    - Se puede incluir información adicional en el cuerpo de la respuesta
      - Codificada en XML/JSON
    - Se mantiene la idea de maximizar el uso de HTTP
      - Se usa un estándar ya conocido
      - No es necesario especificar los tipos de errores bajo un mecanismo ad hoc

Servicios web

- REST
  - Ejemplos de errores
    - 200 OK The resource was read, updated, or deleted
    - 201 Created The resource was created
    - 400 Bad Request The data sent in the request was bad
    - 403 Not Authorized The Principal named in the request was not authorized to perform this action
    - 404 Not Found The resource does not exist
    - 409 Conflict A duplicate resource could not be created
    - 500 Internal Server Error A service error occurred

Servicios web

- REST
  - Nivel de madurez
    - Nivel 0
      - Una URL un verbo HTTP (XML-RPC y SOAP)
    - Nivel 1
      - Muchas URLs (recursos) y un solo verbo HTTP
    - Nivel 2
      - Muchas URLs (recursos) y muchos verbos HTTP
    - Nivel 3
      - Muchas URLs (recursos), muchos verbos HTTP e hipertexto
      - HATEOAS: Hypertext As The Engine Of Application State

Servicios web

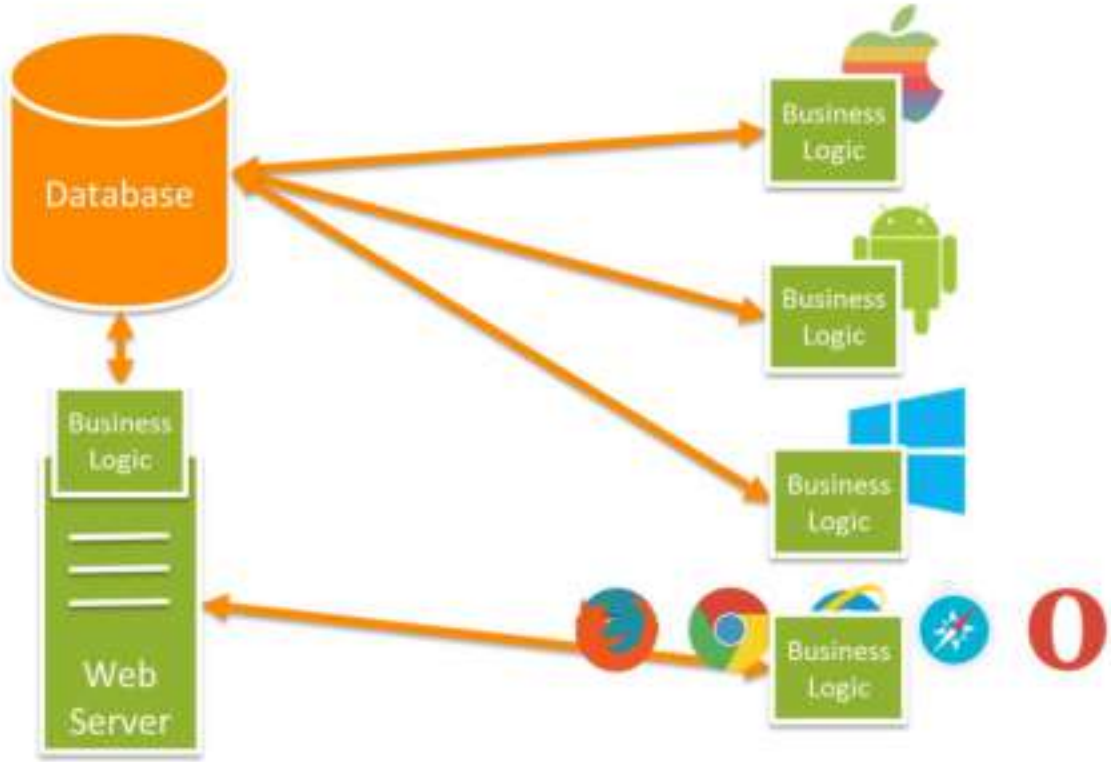
- REST
  - Comparación con otras formas de invocación remota
    - Invocación de operación no CRUD
      - http://servidor.com/add/3/4
      - http://servidor.com/add?a=3&b=4
    - En REST no existe definición formal del contrato
      - Se podría usar XML schema
      - En muchas ocasiones el proveedor del servicio proporciona código para invocar al servicio como una capa sobre REST
      - Generalmente se autogenera información sobre el API en HTML junto con la implementación del servicio
      - Más flexible
      - Se pueden utilizar sin herramientas adicionales que generen stub

Servicios web

- REST
  - Modelo de comunicación
    - Fundamentalmente síncrono con petición/respuesta
    - Se puede también utilizar comunicación síncrona por sondeo
      - Se invoca una operación y se retorna HTTP 202
        - Aceptada pero aún no terminada
        - También se retorna una URL donde obtener información sobre la operación
      - El cliente usa la URL obtenida en la respuesta para conocer cuando se acaba la operación
        - Hay que usar polling

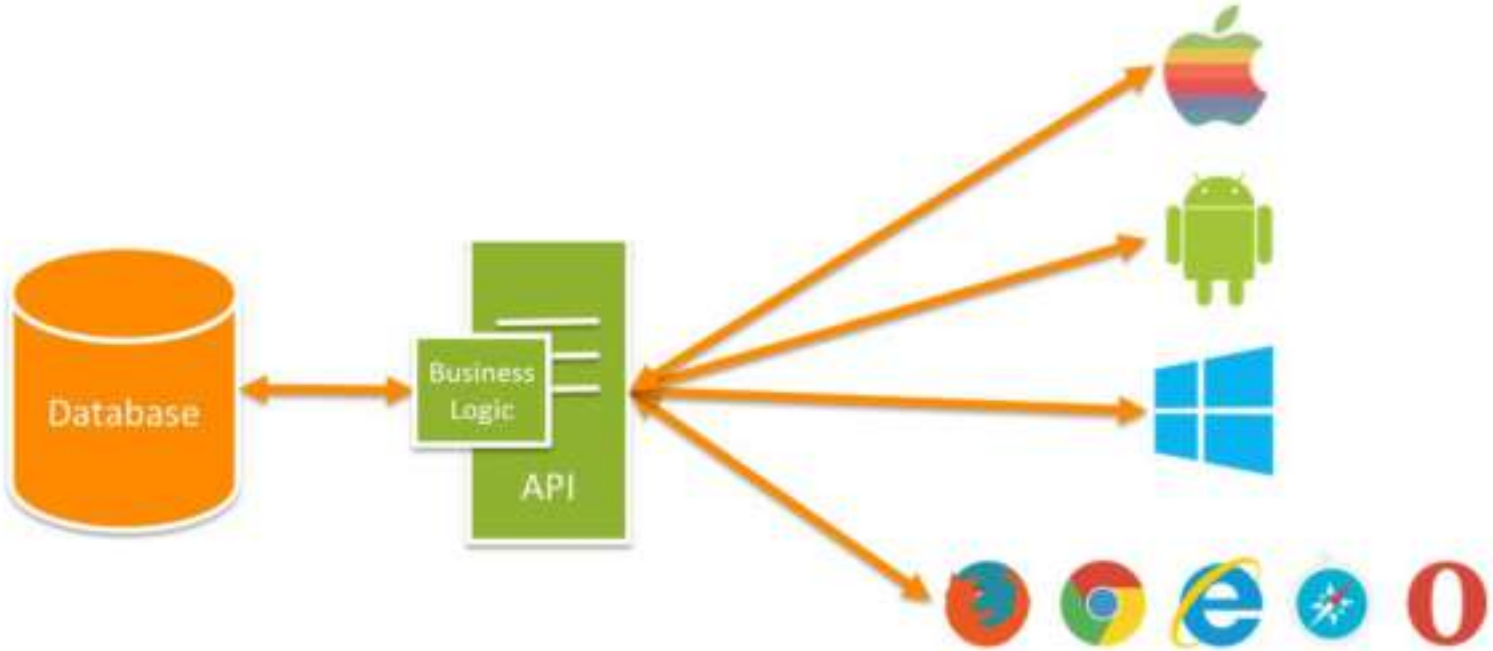
Servicios web

- REST
  - Acceso a bases de datos sin REST



Servicios web

- REST
  - Acceso a bases de datos con REST



http://blogs.msdn.com/b/martinkearn/archive/2015/01/05/introduction-to-rest-and-net-web-api.aspx

Sistemas Distribuidos

Servicios web: seguridad en servicios REST

Seguridad en REST

- Definiciones
  - Codificar y decodificar (encoding and decoding)
    - Transformar la información para su almacenamiento o transmisión
  - Cifrar y descifrar (encrypt and decrypt)
    - Transformar la información para evitar que otros accedan a ella y protegerla
  - Resumen criptográfico (hashing)
    - Transformar información de tamaño variable a un tamaño fijo. Además, el hashing criptográfico debe ser imposible de invertir y distintas entradas deben producir distintas salidas
  - Autenticar
    - Comprobar la identidad. Determinar si alguien es quien dice ser
  - Autorizar
    - Comprobar los permisos para realizar una acción. Determinar si alguien puede hacer algo
  - Integridad
    - Verificar el contenido de un mensaje no ha sido modificado por alguien distinto al emisor

Autenticación en REST

- Basic HTTP Authentication
  - Se incluye usuario y contraseña en cada petición
    - Se codifican en base64: base64(user + ":" + password)
  - Obliga a utilizar HTTPS

Autenticación en REST

- Digest HTTP Authentication
  - Se incluye usuario y hash con la contraseña en cada petición
    - Ejemplo: MD5(nonce, username, realm, URI, password)
  - La contraseña no se envía pero debe ser conocida por ambas partes
    - Permitiría intercambiar información con HTTP
  - Se evitan repeticiones
    - nonce: one-time use number
    - Valor generado por el servidor para evitar aceptar una petición más de una vez

Autenticación en REST

- Digest HTTP Authentication

Autenticación en REST

- JSON Web Token (JWT)
  - Se basa en la utilización de tokens que incluyen firma digital
    - Firma mediante hash
      - El que firma usa un secreto
      - El que verifica necesita conocer el secreto
      - Escenario: donde el que firma y el que verifica son el mismo
      - Ejemplo de algoritmo: HMAC-SHA256
    - Firma mediante cifrado asimétrico
      - El que firma usa la clave privada
      - El que verifica necesita conocer la clave pública
      - Escenario: donde el que firma y el que verifica son distintos
      - Ejemplo de algoritmo: RSA-256

Autenticación en REST

- JSON Web Token (JWT)
  - Un token JWT tiene tres campos:
    - Cabecera (header)
    - Cuerpo (payload)
    - Firma (signature)
  - El token se construye combinando los tres campos:
    -
```
Base64(header) + "." + Base64(payload) + "." +
    HMAC(
        (Base64(header) + "." + Base64(payload)), secreto
    )
```
    -
```
Base64(header) + "." + Base64(payload) + "." +
    RSA(
        (Base64(header) + "." + Base64(payload)), clave_privada
    )
```

Autenticación en REST

- JSON Web Token (JWT)
  - Cabecera (declara el tipo y el algoritmo de firma)
    - alg (algorithm)
    - typ (type)

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Autenticación en REST

- JSON Web Token (JWT)
  - Cuerpo (declara los *claim*)
    - iss (issuer)
    - exp (expiration time)
    - sub (subject)
    - aud (audience)
    - [otros]

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Autenticación en REST

- JSON Web Token (JWT)
  - Token resultante
    - Cabecera.Cuerpo.Firma
    - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

https://jwt.io/

Autenticación en REST

- JSON Web Token (JWT) con hash HMAC

Autenticación en REST

- JSON Web Token (JWT) con RSA

Autenticación en REST

- Aspectos de JWT
  - Usar HMAC para generar tokens que son verificados por el propio sistema
    - Si el usuario encaja con la contraseña el servidor genera un token
    - No es necesario volver a enviar usuario y contraseña hasta que el token expire o el servidor lo invalide (cambie secreto o claves)
  - Usar RSA para generar tokens que son verificados por otro sistema
    - Necesita distribuir claves públicas

Autenticación en REST

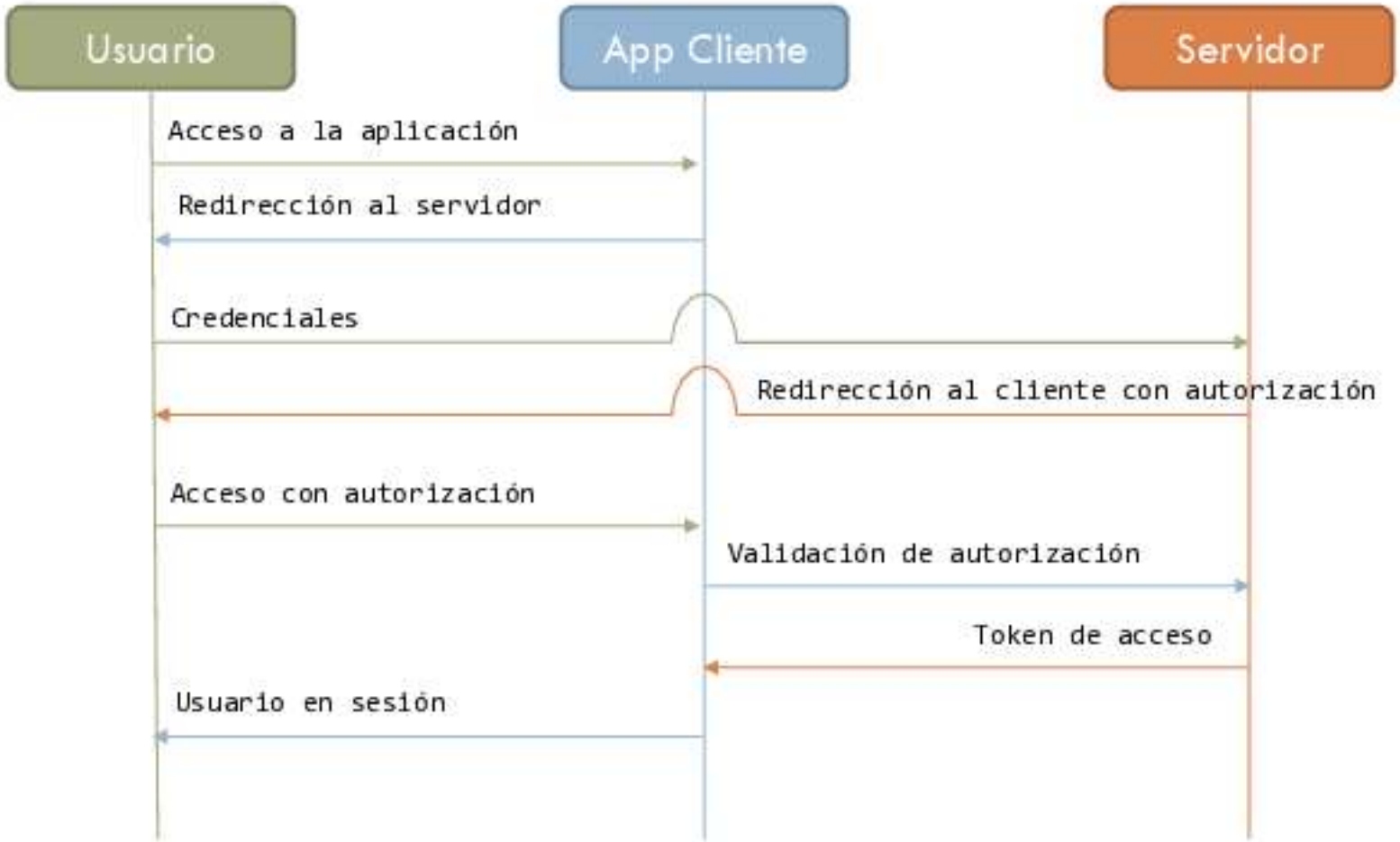
- Aspectos de JWT
  - JWT define cómo generar el token
  - JWT define cómo firmar el token
  - JWT define como verificar el token
  - El resto queda abierto

Autenticación en REST

- OAuth 2.0
  - Protocolo de autorización para permitir que una aplicación accede a los datos de otra
    - Ejemplo: mi aplicación de calendario (Outlook) accede a mi calendario de Gmail
  - Evita tener que dar mis credenciales a la aplicación que quiere usar mis datos almacenados en otra
    - Outlook puede acceder a los eventos del calendario de Gmail pero no necesita saber mi contraseña
  - Proporciona múltiples escenarios de autorización con roles específicos
    - Resource Owner, Client, Resource Server, Authorization Server

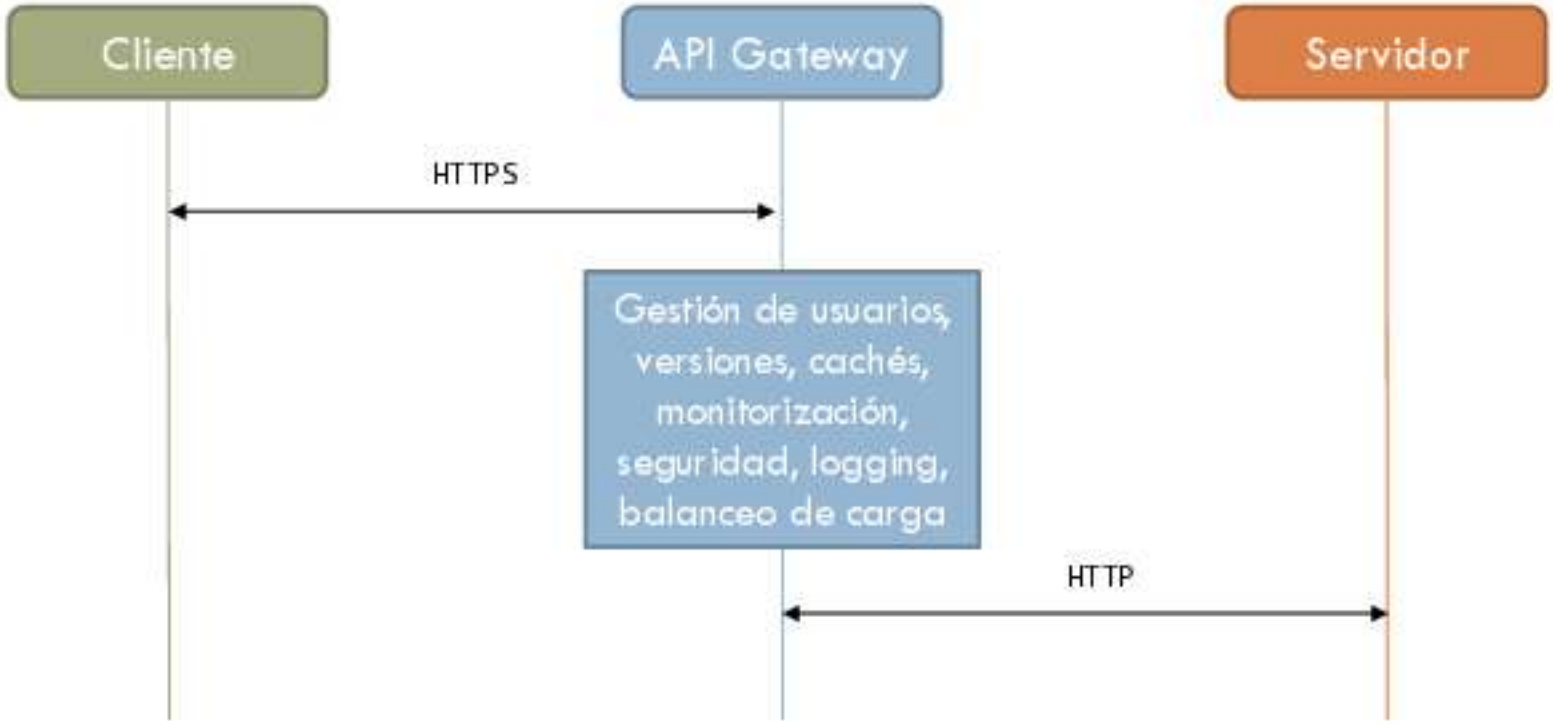
Autenticación en REST

- OAUTH 2



Autenticación en REST

- API gateway
  - Módulo que se sitúa entre cliente y servidor y proporciona servicios de seguridad entre otros



Sistemas Distribuidos

Arquitecturas orientadas a mensajes

Mensajes

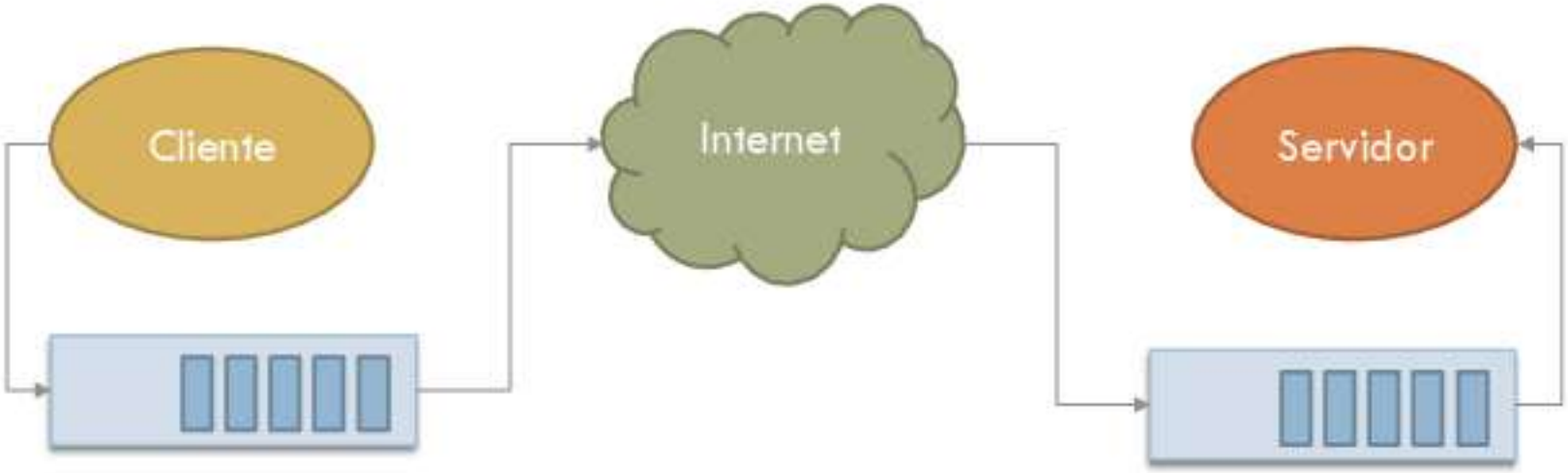
- Problemas de la invocación remota
  - Típicamente el cliente se bloque hasta que recibe la respuesta de la petición
  - Requiere un conexión directa entre cliente y servidor
  - Requiere que tanto el cliente como el servidor estén en funcionamiento durante todo el intercambio de información
  - Fundamentalmente se adapta a un modelo de comunicación síncrono

Mensajes

- Problemas de la invocación remota

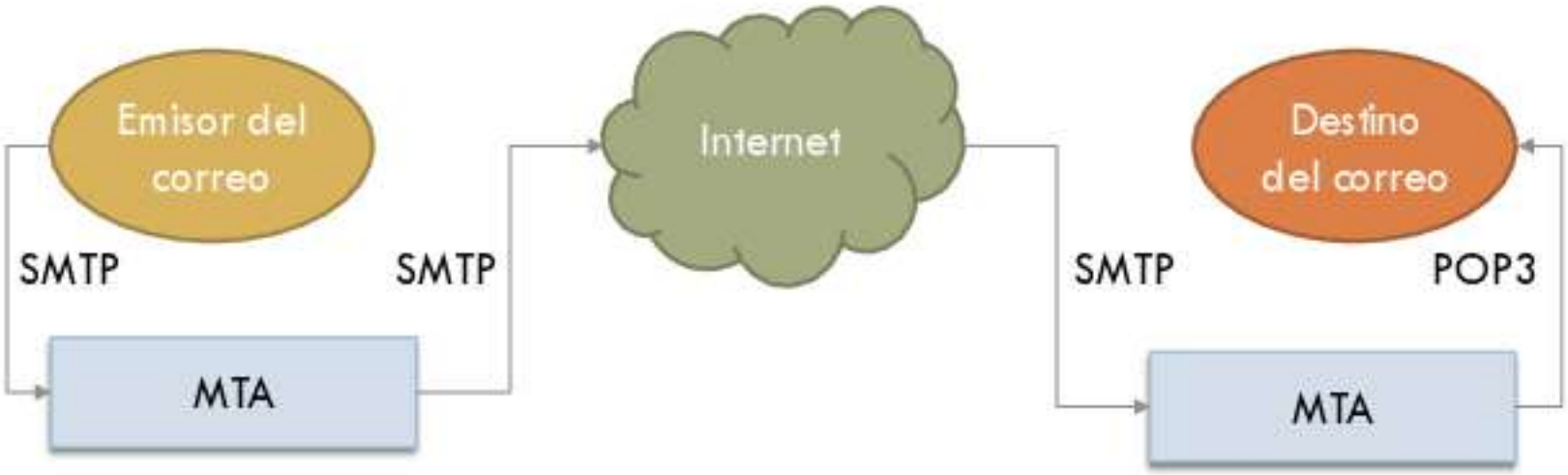
Mensajes

- Solución: colas de mensajes
  - Desacoplar cliente y servidor



Mensajes

- Correo electrónico



Mensajes

- Ventajas de las arquitecturas orientadas a mensajes
  - El cliente y el servidor están desacoplados
    - No tienen que estar activos al mismo tiempo
    - El emisor no tiene que conocer la localización del servidor
    - Comunicación fiable (*store & forward*)
    - Se realiza la sincronización cuando hay conexión
  - Facilita la comunicación asíncrona
    - Se escribe el mensaje en la cola y se continúa
    - Sin bloqueo (*send & forget*)
  - Facilita la transformación de la información
  - Un emisor rápido no satura a un receptor lento
    - La cola amortigua las diferencias de rapidez

Mensajes

- Conceptos
  - Canal (cola)
    - Tubería virtual que conecta al emisor y al receptor
  - Mensaje
    - Un paquete de información transmitido a través del canal
  - Enrutado
    - Mecanismo para determinar el destinatario
  - Transformación
    - Mecanismo para transformar el mensaje antes de que llegue al destino

Mensajes

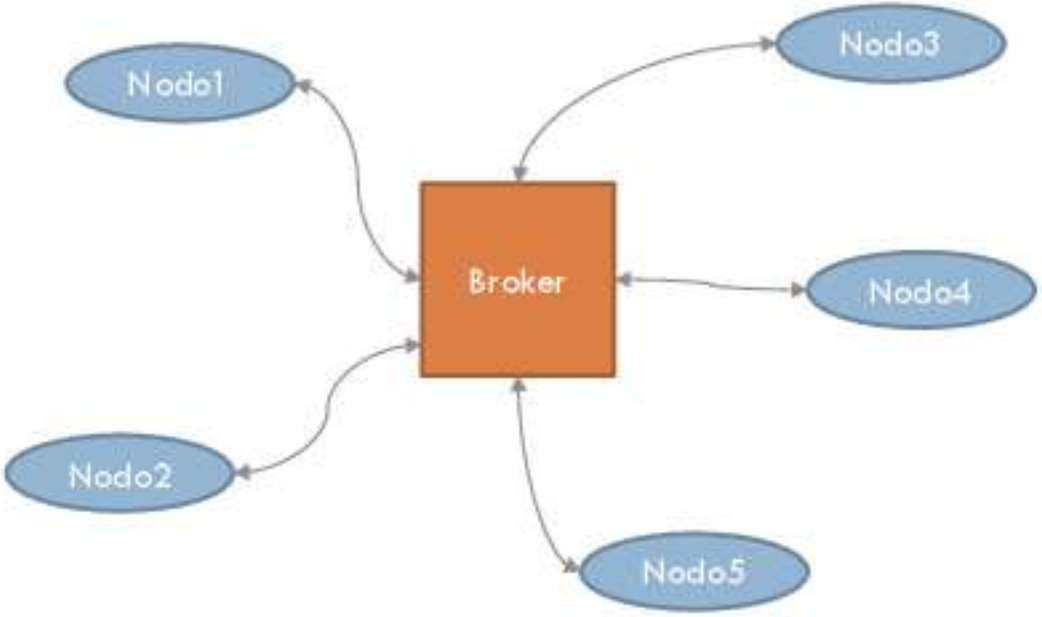
- Mensaje
  - Información que puede contener
    - Datos de negocio (Body)
    - Información de control (Header)
    - Dirección de retorno
    - Número de secuencia
    - Expiración del mensaje
    - Indicador de formato (por si aparecen nuevas versiones)
  - Un mensaje puede utilizarse:
    - Para invocar a una función remota
    - Para enviar un conjunto de datos
    - Para notificar un evento

Mensajes

- Arquitecturas de mensajes
  - Centralizadas
    - Se utiliza un bróker para gestionar el intercambio de mensajes
      - También implementa el resto de la funcionalidad
      - Problema: *single point of failure*
        - Solución: redundancia
  - Descentralizadas
    - El equivalente al bróker se enlaza en cada uno de los nodos
      - Pierde algunas ventajas (fiabilidad)
      - Gana otras (sencillez)

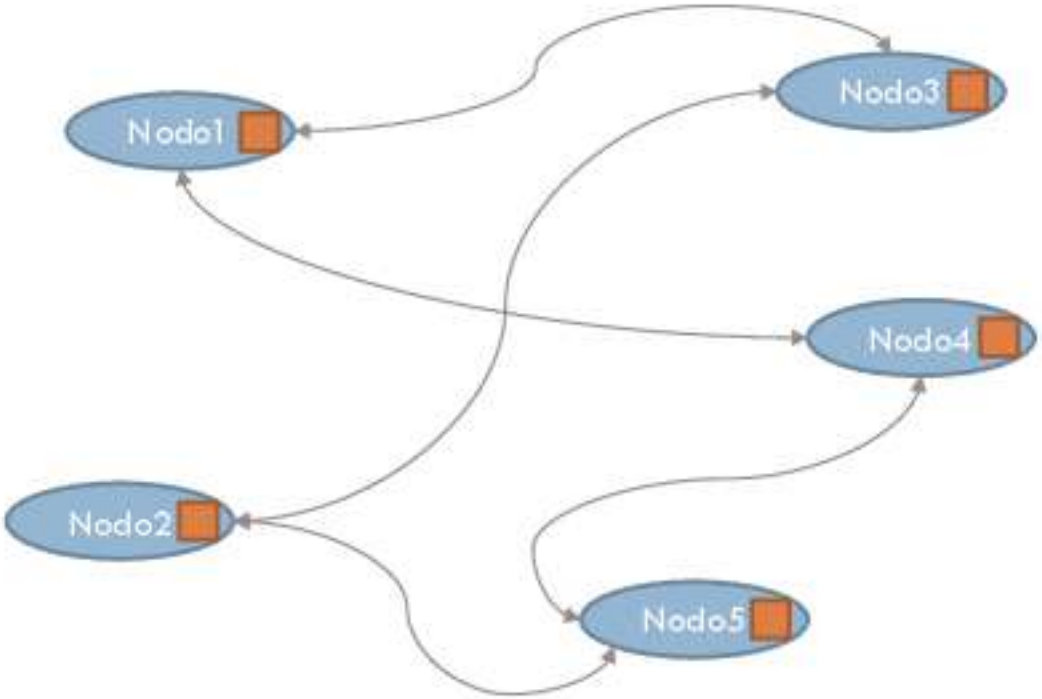
Mensajes

- Arquitecturas de mensajes
  - Centralizadas



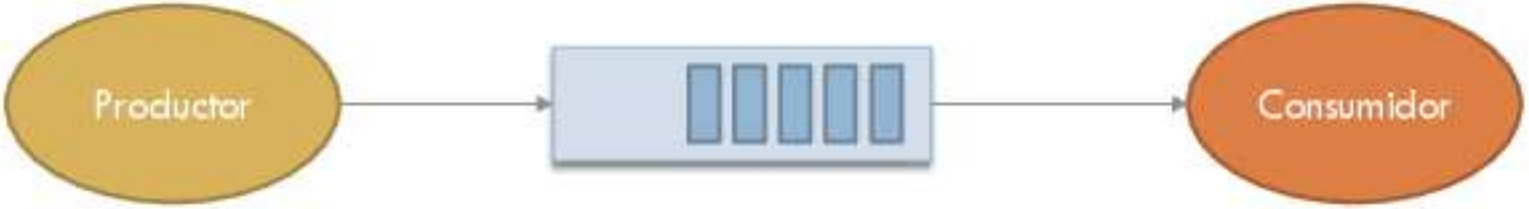
Mensajes

- Arquitecturas de mensajes
  - Descentralizadas



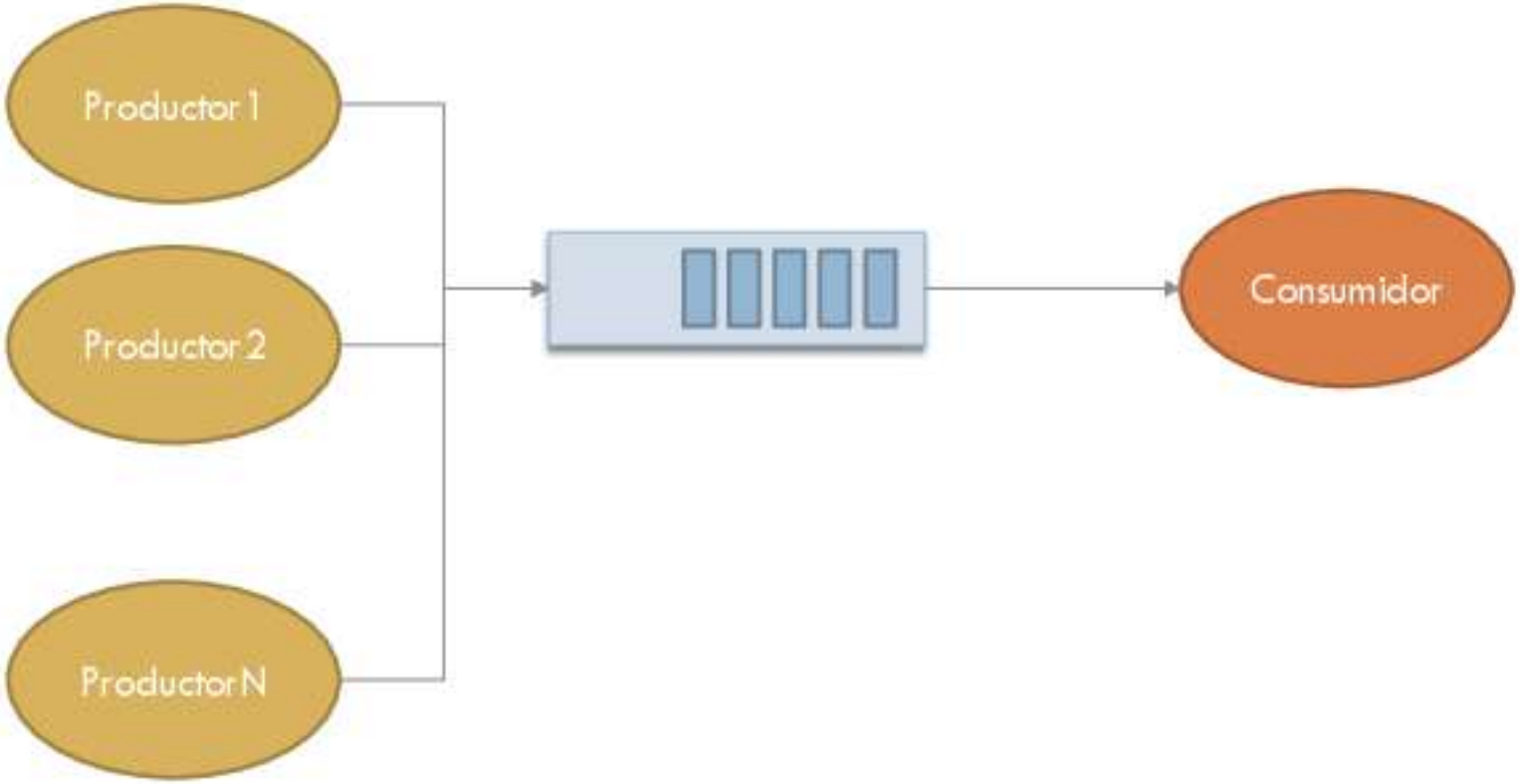
Mensajes

- Modelos de mensajería
  - Punto a punto
    - Una cola por consumidor



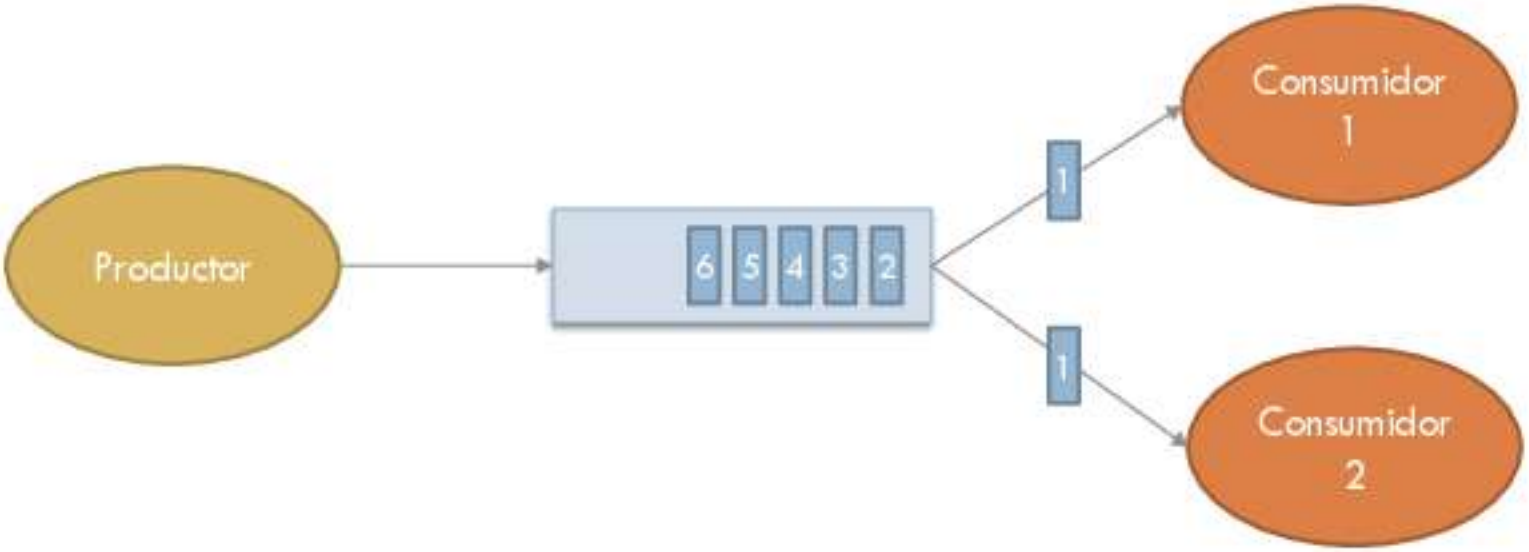
Mensajes

- Modelos de mensajería
  - Punto a punto (Una cola por consumidor)



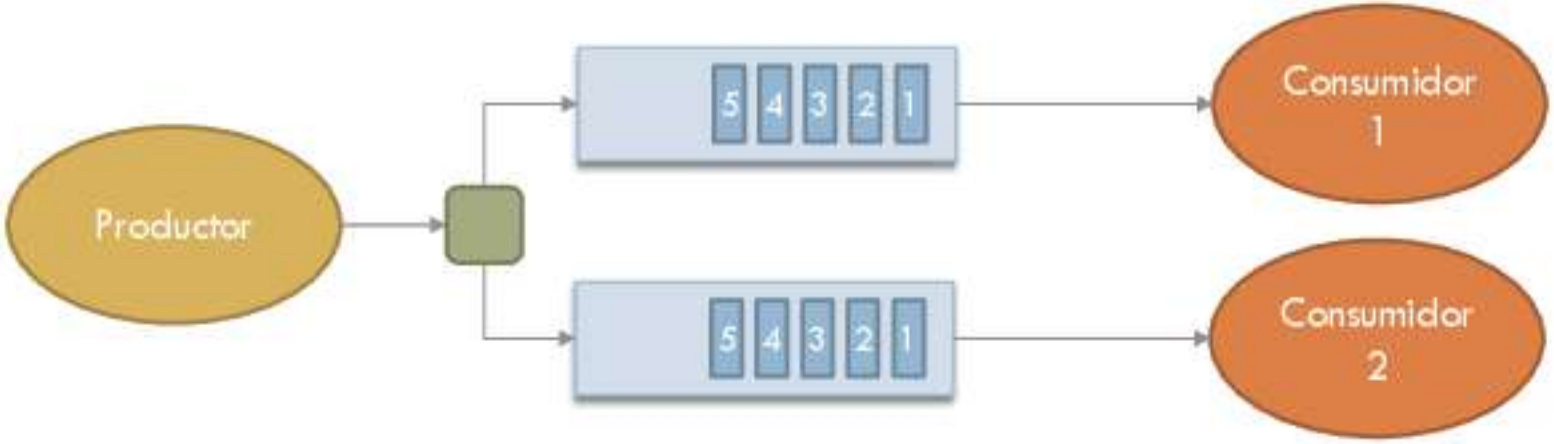
Mensajes

- Modelos de mensajería
  - Difusión
    - El mismo mensaje llega a varios destinatarios



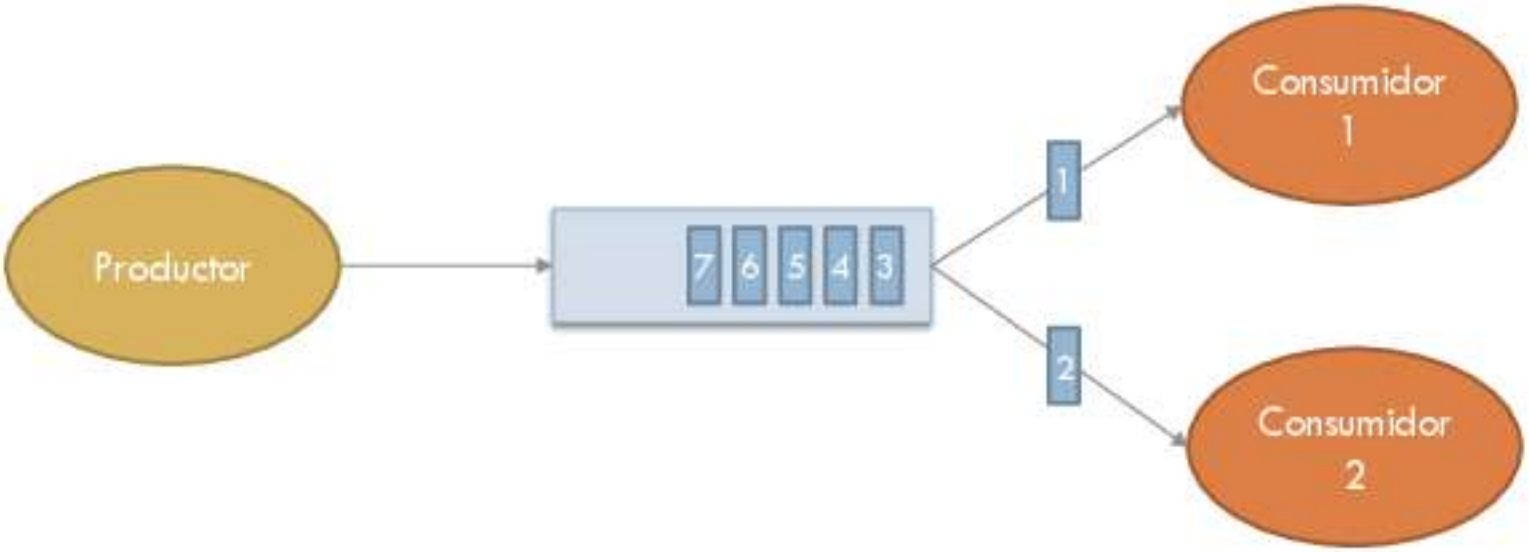
Mensajes

- Modelos de mensajería
  - Difusión
    - El mismo mensaje llega a varios destinatarios
      - Se copia a la cola de cada consumidor



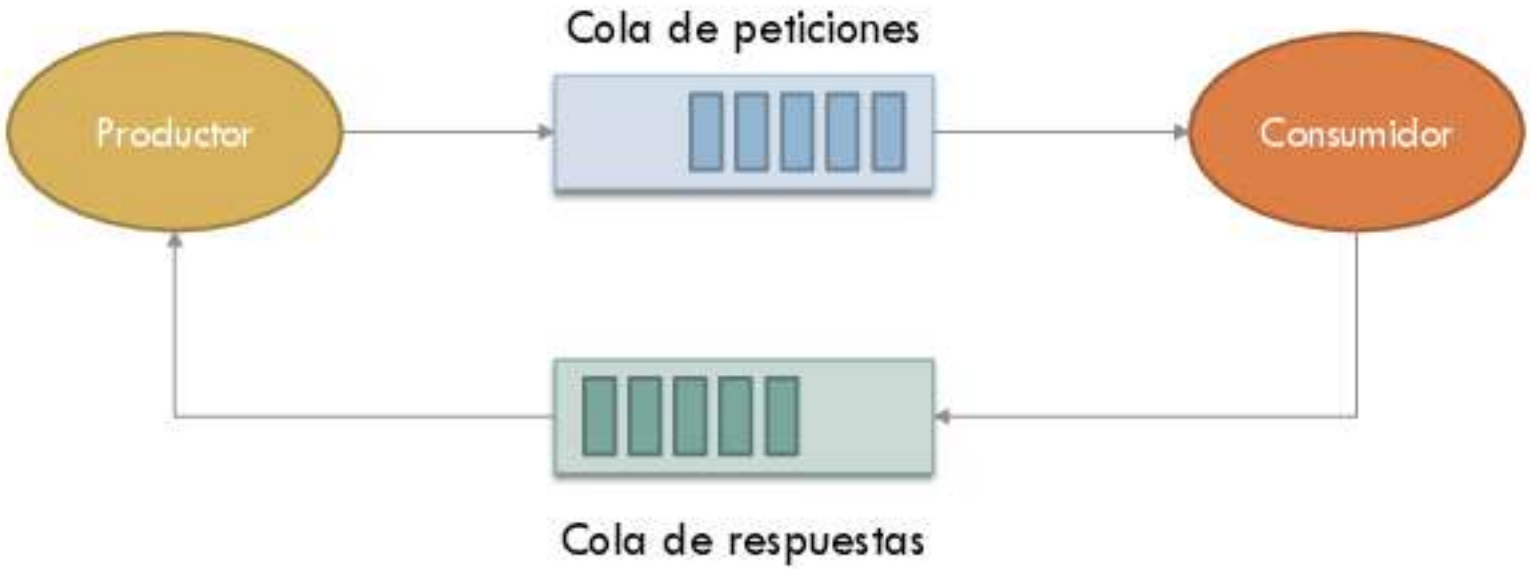
Mensajes

- Modelos de mensajería
  - Colas de trabajo
    - Cada mensaje a un consumidor



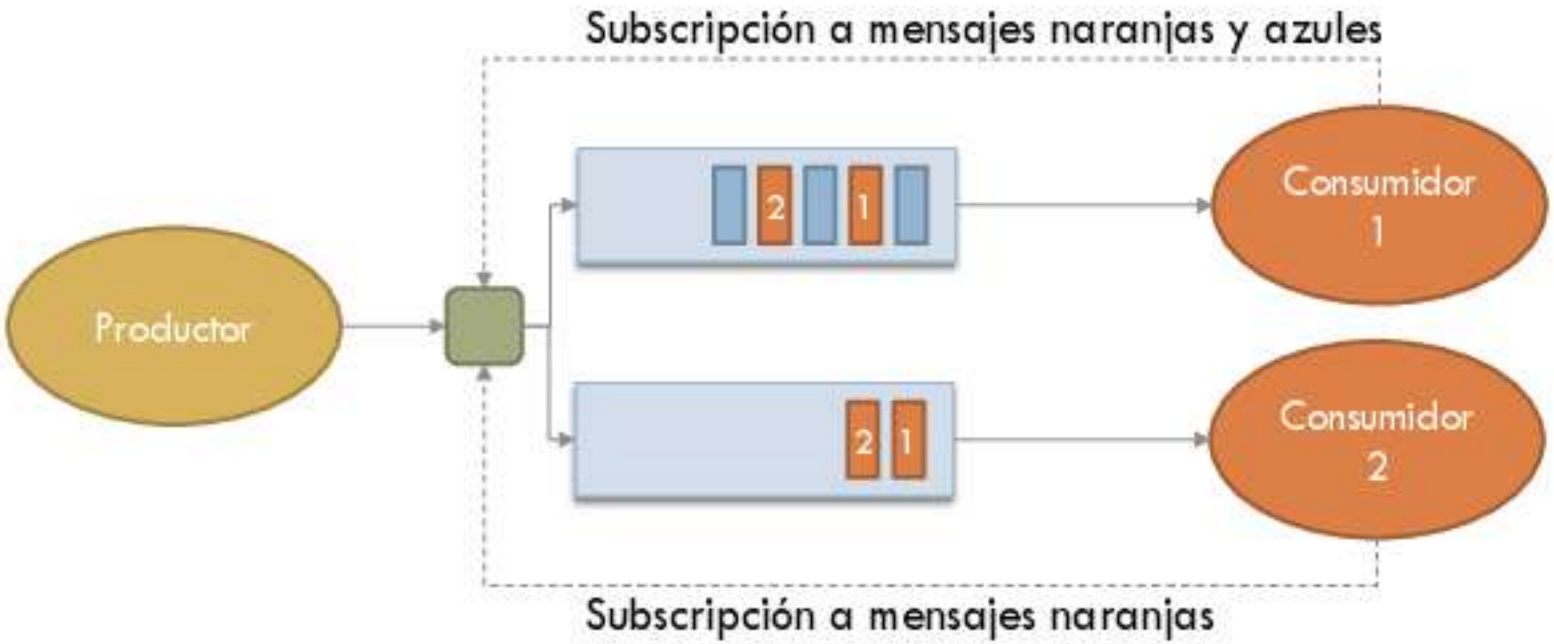
Mensajes

- Modelos de mensajería
  - Petición/respuesta
    - Simula una invocación remota



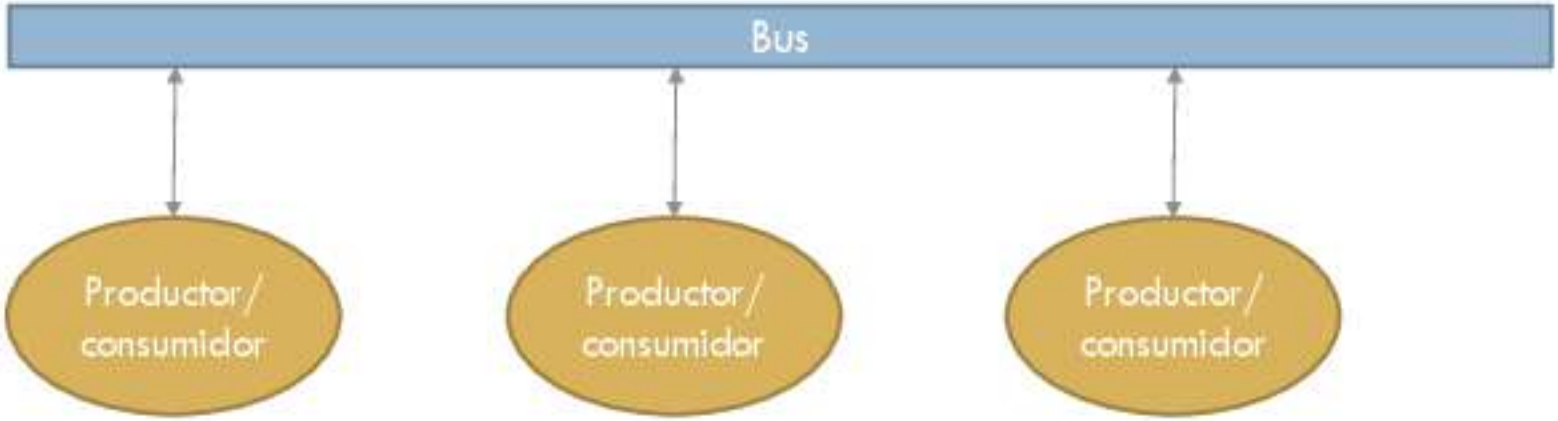
Mensajes

- Modelos de mensajería
  - Publicación/subscripción
    - El mensaje llega a los destinatarios interesados



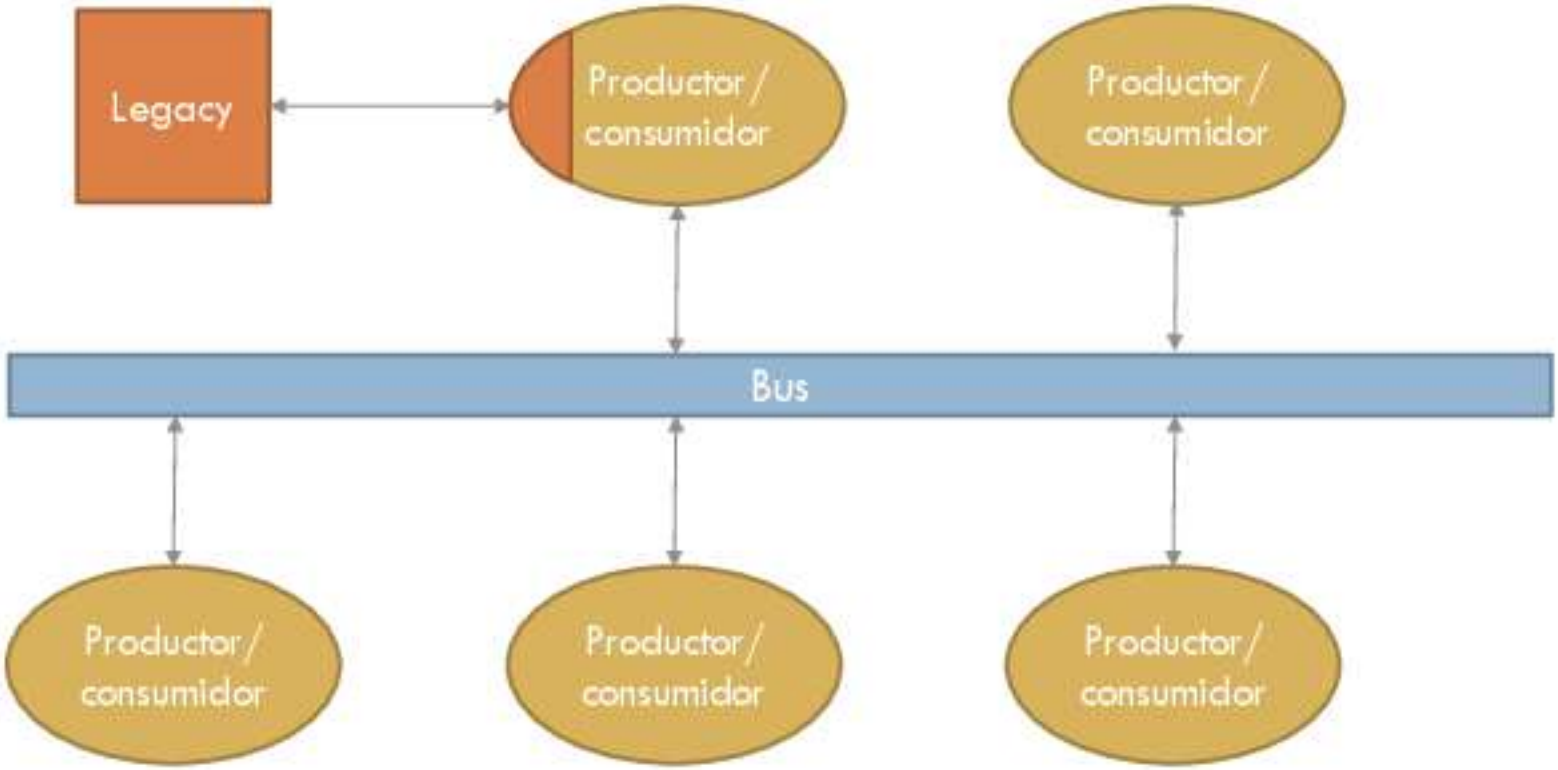
Mensajes

- Modelos de mensajería
  - Bus de mensajes
    - Sistema de compartición de datos a través de mensajería
    - Las aplicaciones de pueden añadir o eliminar de forma sencilla
    - Permite compartir datos de manera desacoplada



Mensajes

- Integración de aplicaciones con un bus
  - *Enterprise service bus* (implementación de SOA)



Mensajes

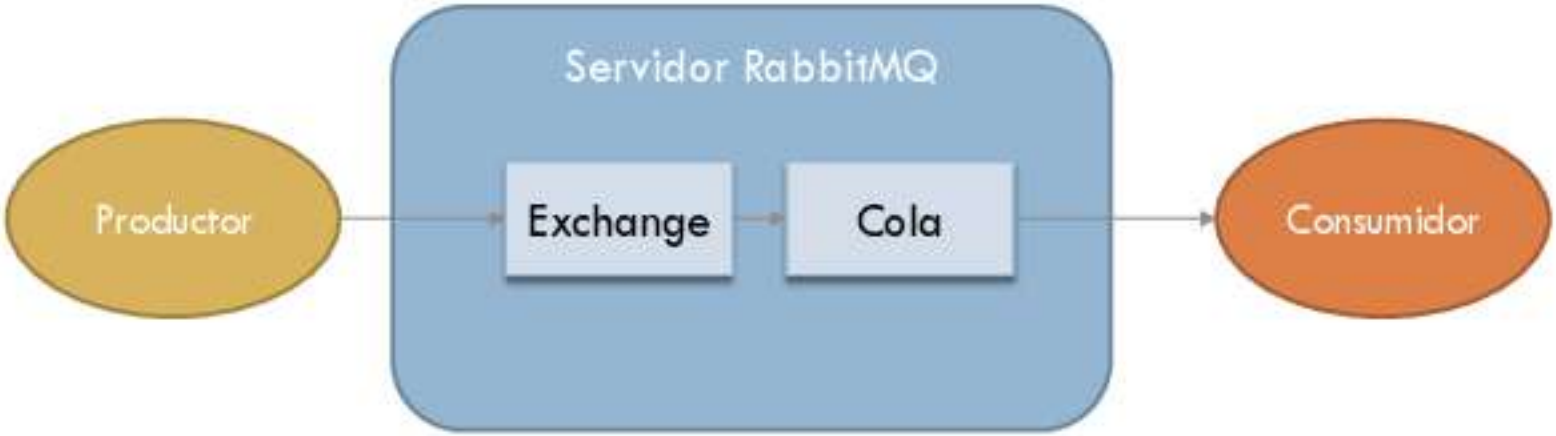
- Middleware para arquitecturas de mensajes
  - MOM: Message-oriented middleware
    - Arquitectura centralizada
      - RabbitMQ
      - IBM WebSphere MQ
      - Apache ActiveMQ
      - MSMQ
      - Jboss
      - Kafka (distributed event streaming platform)
      - MQTT Broker (Mosquitto, RabbitMQ, apache ActiveMQ)
    - Arquitectura descentralizada
      - ZeroMQ
      - NNG (nanomsg-next-generation)

Mensajes

- RabbitMQ
  - Broker de mensajes
  - Utiliza el protocolo AMQP (soporta MQTT)
    - Advanced Message Queuing Protocol
    - Estándar abierto para aplicaciones de mensajería
      - Interoperable
  - Soporta múltiples modelos de mensajería
  - Permite persistencia de mensajes
    - Fiabilidad si el consumidor no está disponible
    - Cuando el consumidor se active recibe los mensajes anteriores

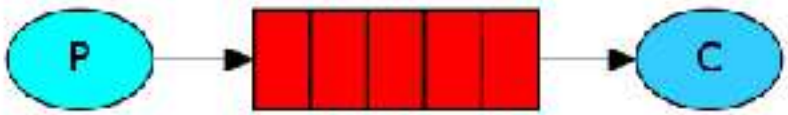
Mensajes

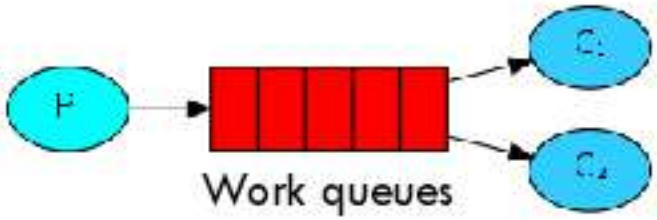
- RabbitMQ
  - Arquitectura

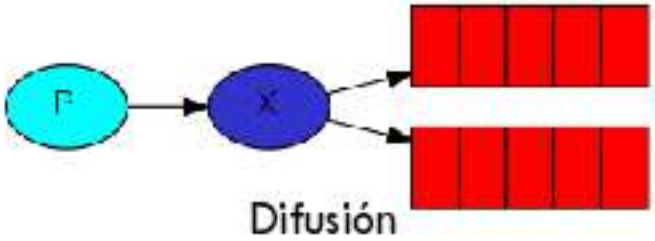


Mensajes

- RabbitMQ
  - Modelos de mensajería



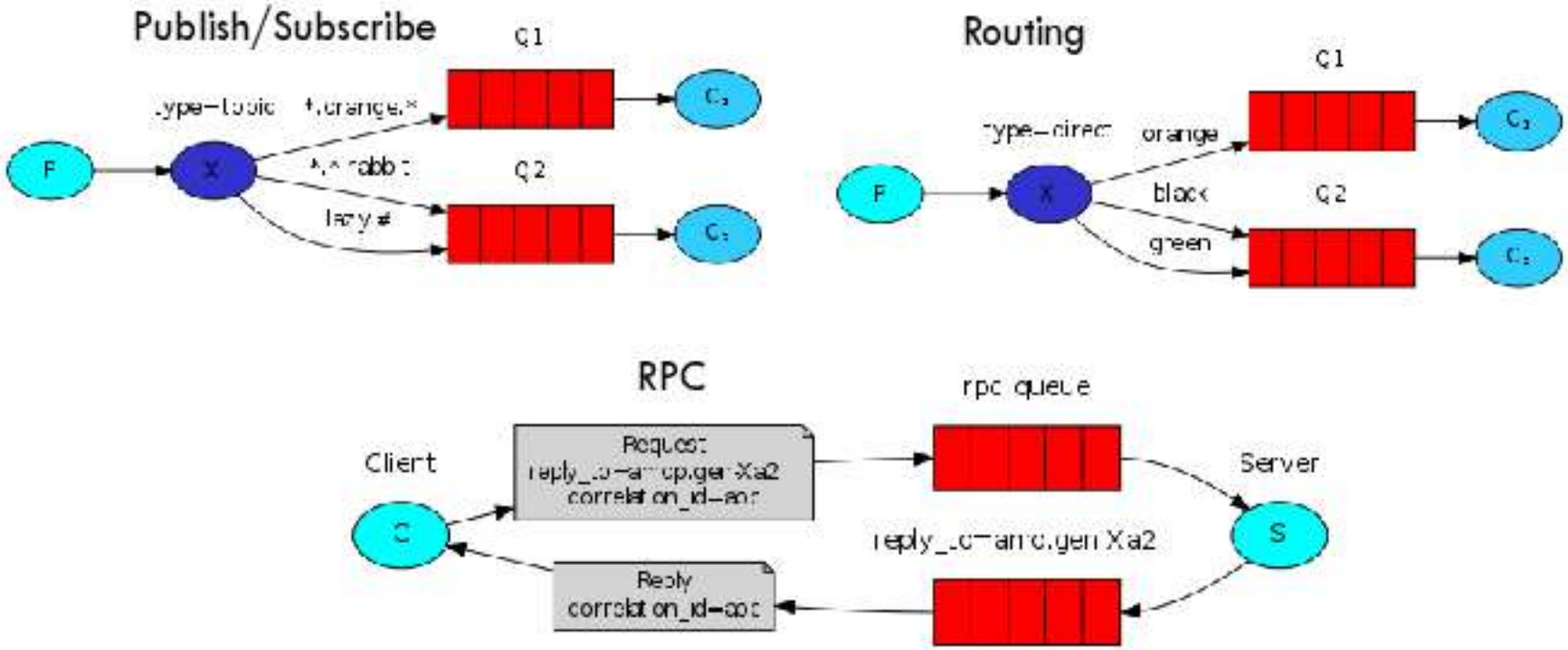




http://www.rabbitmq.com

Mensajes

- RabbitMQ
  - Modelos de mensajería



http://www.rabbitmq.com

```
namespace Publisher
{
    class Program
    {
        const string ROUTING_KEY = "alerta.critica";
        const string ROUTING_KEY2 = "alerta.aviso";
        const string EXCHANGE = "Alertas";
        static void Main(string[] args)
        {
            var factory = new ConnectionFactory() { HostName = "IP" };
            using (var connection = factory.CreateConnection())
            {
                using (var channel = connection.CreateModel())
                {
                    channel.ExchangeDeclare(EXCHANGE, "topic");

                    for (int i = 0; i < 10; i++)
                    {
                        string message = "Hola caracola";
                        var body = Encoding.UTF8.GetBytes(message);
                        channel.BasicPublish(EXCHANGE, ROUTING_KEY, null, body);

                        Console.WriteLine(" [x] Sent '{0}':'{1}'", ROUTING_KEY, message);
                    }
                    for (int i = 0; i < 10; i++)
                    {
                        string message = "Hola caracola";
                        var body = Encoding.UTF8.GetBytes(message);
                        channel.BasicPublish(EXCHANGE, ROUTING_KEY2, null, body);

                        Console.WriteLine(" [x] Sent '{0}':'{1}'", ROUTING_KEY2, message);
                    }
                }
            }
        }
    }
}
```

```
namespace Subscriber
{
    class Program
    {
        const string BINDING_KEY = "alerta.critica";
        const string EXCHANGE = "Alertas";
        static void Main(string[] args)
        {
            var factory = new ConnectionFactory() { HostName = "IP" };
            using (var connection = factory.CreateConnection())
            {
                using (var channel = connection.CreateModel())
                {
                    channel.ExchangeDeclare(EXCHANGE, "topic");

                    var queueName = channel.QueueDeclare();

                    channel.QueueBind(queueName, EXCHANGE, BINDING_KEY);

                    var consumer = new QueueingBasicConsumer(channel);
                    channel.BasicConsume(queueName, true, consumer);

                    while (true)
                    {
                        var ea = (BasicDeliverEventArgs)consumer.Queue.Dequeue();
                        var body = ea.Body;
                        var message = Encoding.UTF8.GetString(body);
                        var routingKey = ea.RoutingKey;
                        Console.WriteLine(" [x] Received '{0}':'{1}'", routingKey, message);
                    }
                }
            }
        }
    }
}
```

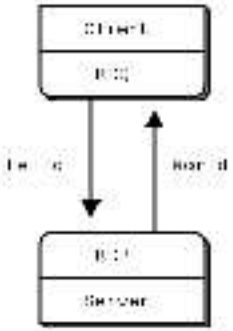
Mensajes

- ZeroMQ
  - ØMQ = "zero broker"
  - No hay Broker de mensajes
    - No hay administración (zero)
  - Se define por sus creadores como sockets con esteroides
  - Se distribuye como una biblioteca que se enlaza con los programas que la usan

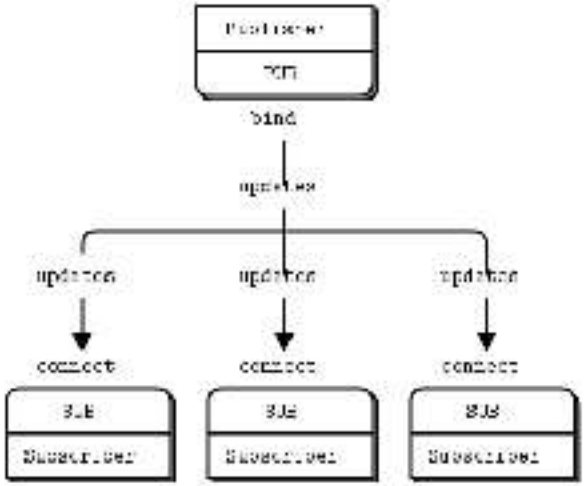
Mensajes

- ZeroMQ
  - Modelos de mensajería

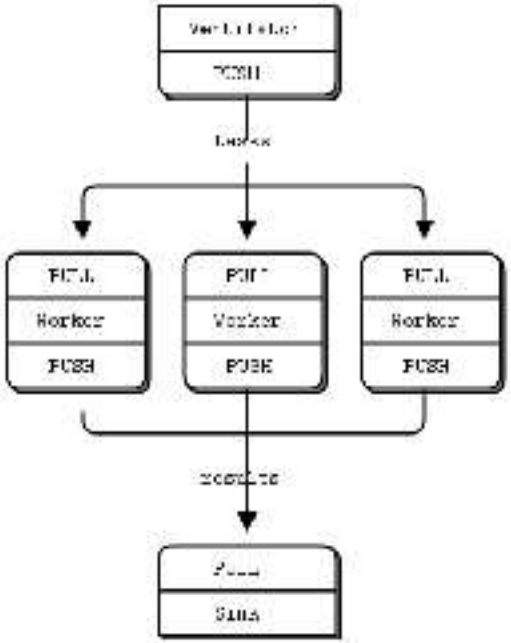
Request-Reply



Publish-Subscribe



Parallel Pipeline

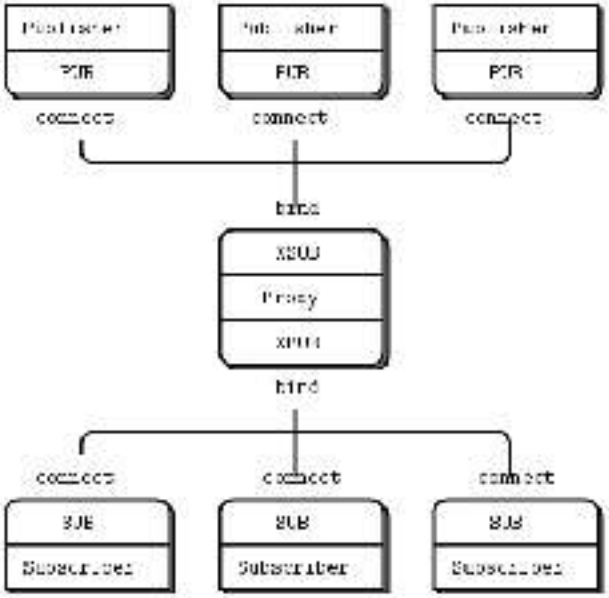


http://zguide.zeromq.org

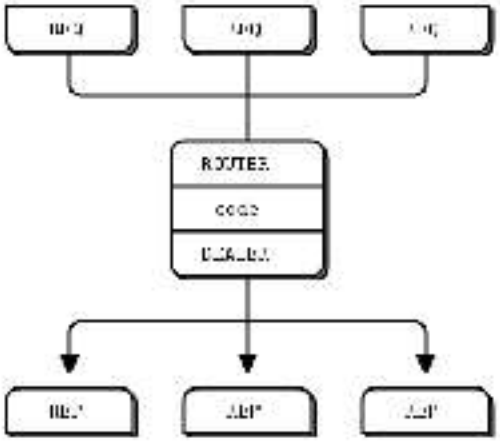
Mensajes

- ZeroMQ
  - Modelos de mensajería

Pub-Sub Network with a Proxy



Extended Request-Reply



http://zguide.zeromq.org

```
namespace ZeroMQ.Test
{
    static partial class Program
    {
        public static void WUServer(string[] args)
        {
            using (var context = ZContext.Create())
            using (var publisher = ZSocket.Create(context, ZSocketType.PUB))
            {
                string address = "tcp://*:5556";
                Console.WriteLine("I: Publisher.Binding on {0}", address);
                publisher.Bind(address);

                // Initialize random number generator
                var rnd = new Random();

                while (true)
                {
                    // Get values that will fool the boss
                    int zipcode = rnd.Next(99999);
                    int temperature = rnd.Next(-55, +45);

                    // Send message to all subscribers
                    var update = string.Format("{0:D5} {1}", zipcode, temperature);
                    using (var updateFrame = new ZFrame(update))
                    {
                        publisher.Send(updateFrame);
                    }
                }
            }
        }
    }
}
```

```python
# Cliente en python

import sys
import zmq

# Socket to talk to server
context = zmq.Context()
socket = context.socket(zmq.SUB)

print("Collecting updates from weather server…")
socket.connect("tcp://localhost:5556")

# Subscribe to zipcode, default is NYC, 10001
zip_filter = sys.argv[1] if len(sys.argv) > 1 else "10001"

# Python 2 - ascii bytes to unicode str
if isinstance(zip_filter, bytes):
    zip_filter = zip_filter.decode('ascii')
socket.setsockopt_string(zmq.SUBSCRIBE, zip_filter)

# Process 5 updates
total_temp = 0
for update_nbr in range(5):
    string = socket.recv_string()
    zipcode, temperature, relhumidity = string.split()
    total_temp += int(temperature)

print("Average temperature for zipcode '%s' was %dF" % (
    zip_filter, total_temp / update_nbr)
)
```

Aplicaciones y sistemas distribuidos

Consenso

Rubén Usamentiaga
Área de Arquitectura y Tecnología de Computadores
Universidad de Oviedo

Introducción

- Consenso
  - Objetivo
    - De forma distribuida poner de acuerdo a múltiples entidades
  - Problema fundamental para proporcionar tolerancia a fallos
    - La tolerancia se consigue replicando la información
    - Cada replica debe aplicar las mismas operaciones en el mismo orden
    - La consistencia de la información almacenada se consigue mediante el consenso
    - El consenso se debe llevar a cabo en un entorno distribuido con posibles fallos

Introducción

- Tipos de fallos en los sistemas distribuidos
  - Fallo de parada
    - Las entidades fallan y dejan de funcionar (crash)
  - Fallo bizantino
    - Las entidades fallan pero pueden seguir funcionando
    - Exhiben comportamiento errático y arbitrario
    - Su comportamiento puede ser malicioso

Introducción

- Problema de los generales bizantinos
  - Problema descrito por Leslie Lamport
  - Un grupo de generales comanda una parte del ejército bizantino que rodea una ciudad
  - Los generales deben decidir si atacar o retirarse de forma conjunta
  - Puede haber generales traidores
  - Los generales están distribuidos geográficamente y tienen que enviar sus votos a través de mensajeros

Resoluble bajo ciertas condiciones y considerando que 1/3 de los generales no son traidores

Consenso

Raft

RAFT

- RAFT
  - Algoritmo de consenso distribuido diseñado para ser fácil de entender e implementar
    - En contraposición a Paxos
  - Resuelve el problema de poner de acuerdo a múltiples entidades teniendo en cuenta fallos de paradas
    - No es tolerante a fallos bizantinos
  - El estado compartido consiste en una estructura de datos y un diario de operaciones replicado (log)
  - El sistema está operativo siempre que la mayoría de entidades se encuentren en funcionamiento

Ongaro, D., & Ousterhout, J. K. (2014, June). In search of an understandable consensus algorithm.

RAFT

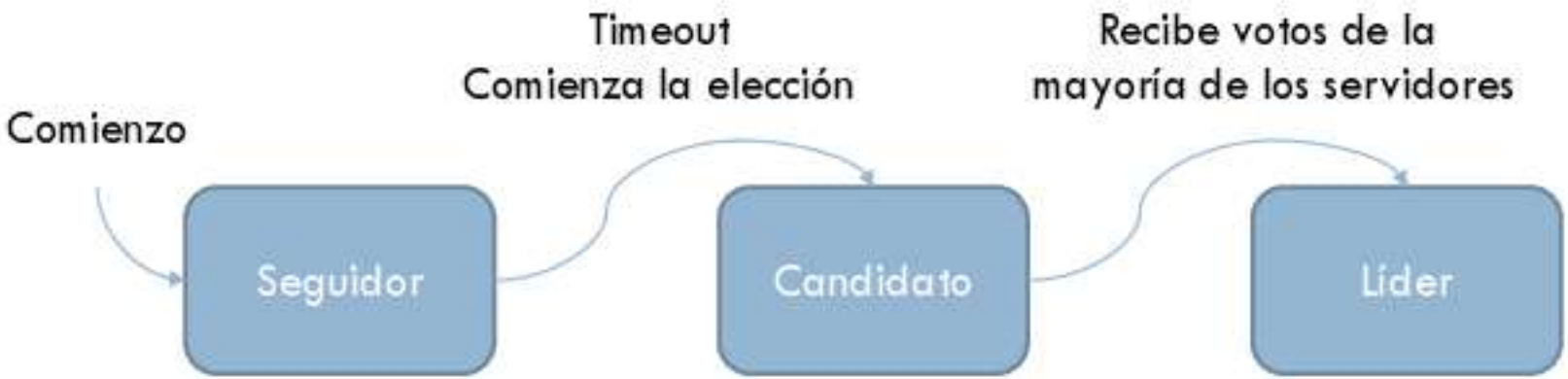
- Solución al consenso en RAFT
  - Un conjunto de servidores se organizan en un cluster
  - Los servidores se encargan de elegir un líder
  - El líder se encarga de
    - Aceptar las peticiones de los clientes
    - Gestionar las replicas del log
    - Informar al resto de servidores de los cambios

RAFT

- Descomposición del problema
  - Elección del líder
    - Selección de un servidor para actuar como líder
    - Detectar fallos y elegir nuevo líder
  - Replicación del log
    - El líder recibe peticiones de los clientes y las añade a su log
    - El líder repica su log a los otros servidores
  - Seguridad
    - Los cambios realizados sobreviven a los cambios de líder
    - Definir reglas para realizar cambios y repetir la elección de líder

RAFT

- Estados de un servidor
  - Líder, seguidor (replica pasiva), candidato (para la elección de líder)
  - Durante la operación normal hay un líder y todos los demás son seguidores



RAFT

- Periodos (terms)
  - El tiempo se divide en periodos de longitud arbitraria
  - Al principio de cada periodo se elige un líder
    - Que permanece como líder durante el resto del periodo
  - El número de periodo se incrementa en cada elección
  - Tras la elección, el líder envía mensajes de forma periódica al resto de servidores (hearbeat)
    - La elección de un nuevo líder comienza cuando un servidor no recibe el heartbeat

RAFT

- Elección del líder
  - Cuando un servidor no recibe el hearbeat del servidor se convierte en candidato
    - Incrementando el número de periodo
  - Vota por si mismo y pide votos a los demás
    - Si recibe la mayoría de los votos se convierte en nuevo líder
    - Si no hay mayoría no se elige nuevo líder en ese periodo

RAFT

- Replicación del log
  - El líder recibe una petición de un cliente
  - El líder replica la petición a todos los servidores
  - Si recibe una respuesta positiva de la mayoría de los servidores guarda la petición en el log

RAFT

- Visualización

http://thesecretlivesofdata.com/raft/

Consenso

Blockchain

Blockchain

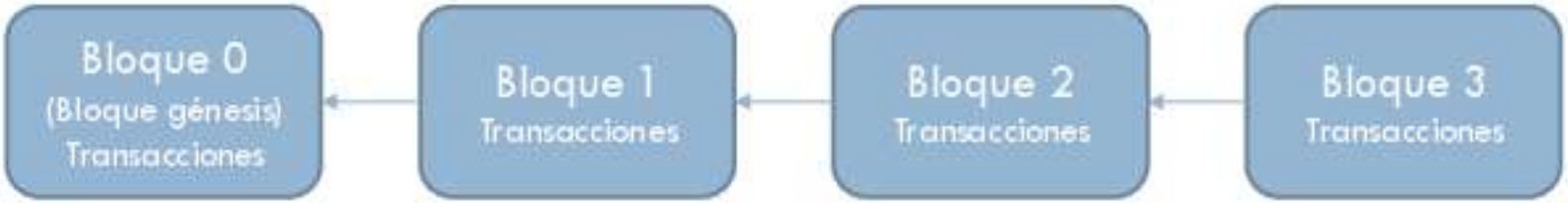
- Blockchain
  - Tecnología fundamental en las criptomonedas incluyendo el bitcoin
  - Permite realizar transacciones en una red p2p descentralizada y sin confianza mutua
  - Resuelve el problema del consenso considerando fallos bizantinos (Byzantine Fault Tolerance, BFT)
    - No se establecen restricciones sobre el comportamiento de los participantes
  - Resuelve el problema del doble gasto sin autoridad central

Blockchain

- Blockchain
  - Implementa de un registro contable (ledger) distribuido
  - Garantiza las operaciones incluso con actores malignos
  - No se basa en una autoridad centralizada
  - Utiliza un algoritmo probabilístico denominado Proof-of-Work para resolver el problema del consenso
  - Propuesto por el investigador con pseudónimo "Nakamoto" para su uso con el Bitcoin en 2008

Blockchain

- Cadena de bloques
  - Registro contable (estructura de datos)
  - Crece de forma constante
    - Lista de bloques enlazados
  - Guarda todas las transacciones
  - Permanente, inmutable y segura
  - Cronológica



Blockchain

- Bloques
  - Los usuarios crean transacciones y las envían a la red
    - Transacción: cantidad (número de bitcoins) y destinatario
  - Nodos de la red (miners) validan las transacciones y las agrupan creado un bloque
  - Cada bloque contiene una lista de transacciones válidas
    - Con múltiples entradas a múltiples salidas cada una identificada por su clave pública
  - Los nodos de la red compiten por añadir un nuevo bloque a la cadena de bloques

https://www.blockchain.com/explorer

Blockchain

- Proof of work
  - Los nodos buscan un número nonce tal que
    - hash(nonce + transacciones + hash bloque anterior) < N
  - N es un valor muy pequeño ajustado dinámicamente
    - Ejemplo: que empiece con 72 ceros
  - El primer nodo que encuentre el valor correcto de nonce añade el bloque a la cadena y lo comunica a los demás nodos
    - Además recibe una recompensa (12.5 btc)
  - Los nodos añaden el nuevo bloque y siguen compitiendo por añadir nuevos bloques a continuación

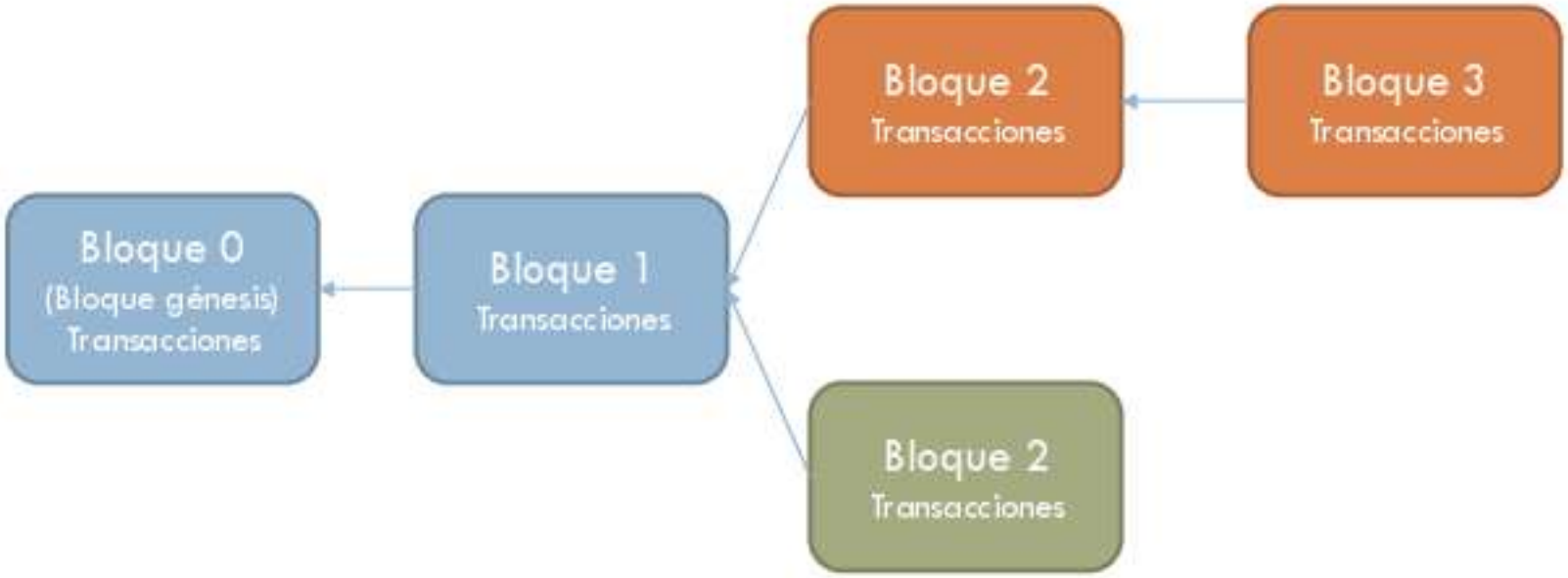
RAFT

- Demo

https://anders.com/blockchain/

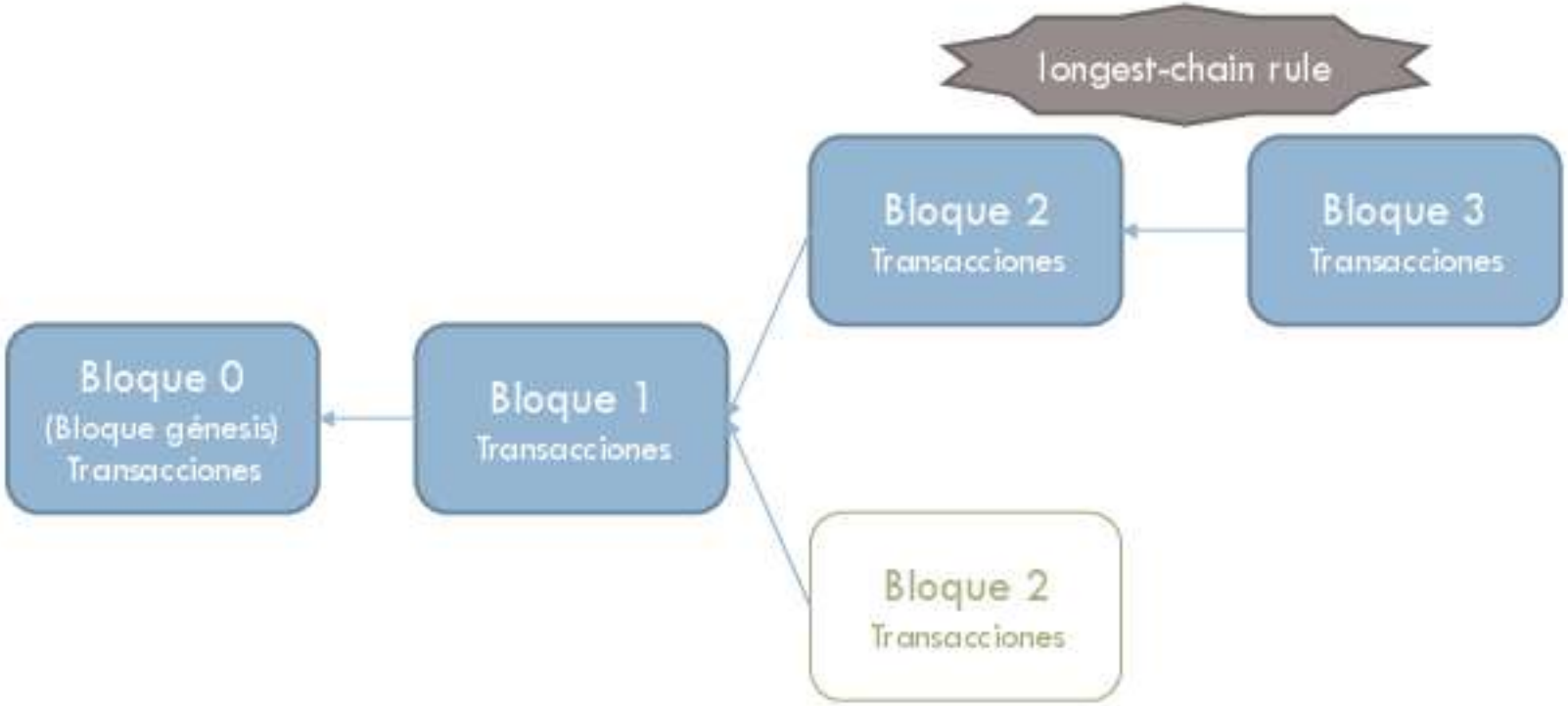
Blockchain

- Concurrencia en Proof of work
  - Dos nodos encuentran un bloque válido a la vez



Blockchain

- Concurrencia en Proof of work
  - Dos nodos encuentra un bloque válido a la vez



Blockchain

- Protección contra ataques
  - Toda la red tiene almacenada la cadena de bloques
    - 180 GB (2018)
  - Para añadir bloques maliciosos se necesitarían añadir los bloques maliciosos más rápido que la red
    - Solo sería posible si dispone de más de 51% de la capacidad de computo
  - Para modificar un nodo intermedio habría que modificar toda la cadena

Blockchain

- Limitaciones
  - El tamaño de los bloques está limitado a 1MB
  - Para mantener estable la red solo se añaden nuevos bloques cada 10 minutos
    - Se controla con la dificultad del hash
  - El tamaño de bloque y el número nuevos bloques equivale a 3.33 transacciones por segundo
  - Enorme consumo de energía para minar bloques

Blockchain

- Bitcoin forks
  - Bitcoin cash
- Otras criptomonedas
  - Ethereum
    - Extiende las transacciones a los Smart contracts
- Otras formas de consenso
  - Proof-of-Stake
    - Los nuevos bloques son añadidos por quienes tienen monedas

https://coinmarketcap.com/

Blockchain

- Proof of stake (Prueba de apuesta)
  - Objetivo: conseguir el consenso con menos consumo de energía
  - El proceso de propuesta y validación de nuevos bloques se realiza solo entre los que poseen tokens de blockchain (monedas)
  - En funcionamiento en Ethereum desde 2022

Blockchain

- Proof of stake (Prueba de apuesta)
  - Los validadores deben dejar un deposito (stake) de moneda para poder participar
    - La probabilidad de participar depende de la cantidad del deposito
  - Creación de nuevos bloques
    - Periódicamente, se elige un validador aleatoriamente para proponer un bloque y enviarlo a los otros nodos
    - Se elige un comité de validadores para verificar la validez del bloque. Cada validador vota su validez
  - Recompensa y castigo
    - Los validadores que muestran un comportamiento incorrecto pierden el deposito. El resto recibe una recompensa

Blockchain

- Proof of stake (Prueba de apuesta)
  - En PoW el periodo de creación de nuevos bloques depende de la dificultad del puzle criptográfico
  - En PoS el tiempo se divide en slots (12 segundos) y epochs (32 slots)
    - En cada epoch se seleccionan un conjunto de validadores: 1 forger y el resto attestors (127)
    - El forger propone un bloque en cada slot
    - Los attestors votan para determinar la validez del bloque propuesto

Aplicaciones y sistemas distribuidos

La nube

Rubén Usamentiaga
Área de Arquitectura y Tecnología de Computadores
Universidad de Oviedo

Cloud computing

- Cloud vs Grid Computing
  - Grid Computing
    - Paradigma que surge en la década anterior
      - Analogía con la red eléctrica (power grid)
      - Al conectarse a la red eléctrica se obtiene potencia eléctrica
      - Al conectarse a un grid se obtiene potencia de procesamiento y almacenamiento
    - Enfocado al procesamiento de datos científicos
    - Conexión de centros de super computación

Aplicaciones y sistemas distribuidos

Cloud computing

Cloud computing

- Definición

ACM Computer Communication Review

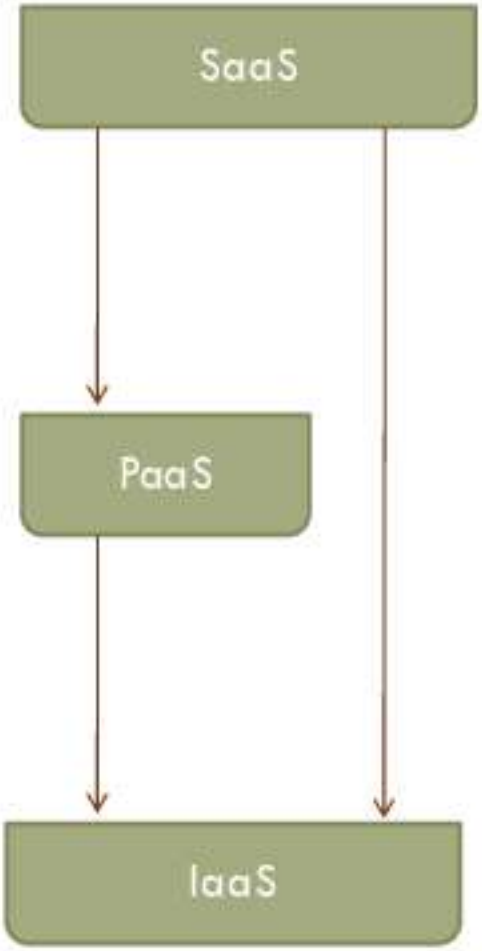
Large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs

Cloud computing

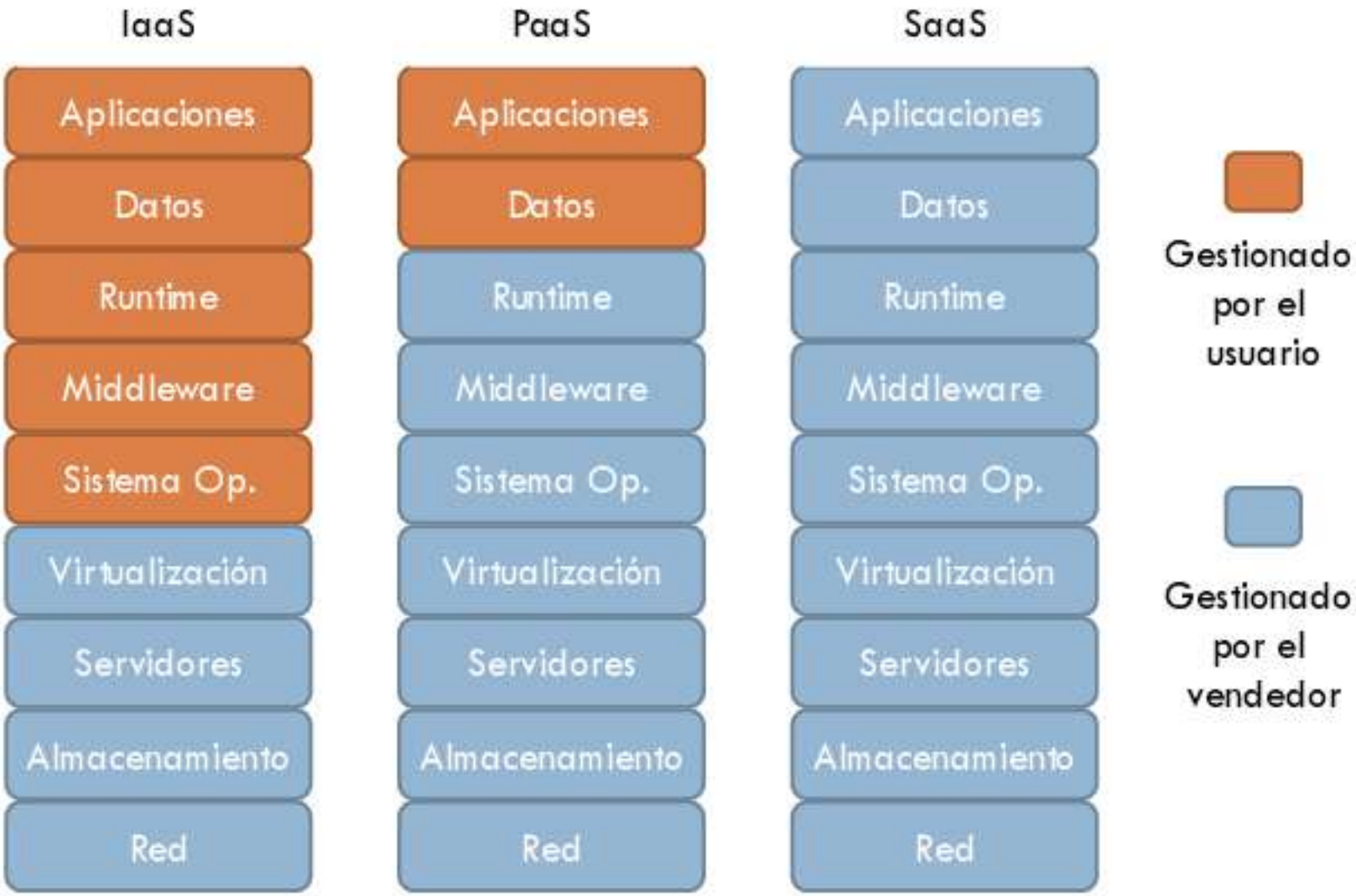
- Elementos habituales en todas las definiciones
  - Acceso a través de Internet (la "nube")
  - Virtualización
  - Escalabilidad
  - Coste por uso
- Conceptos relacionados
  - Utility Computing
  - Autonomic Computing
  - ¿Grid?

Cloud computing

- Los "aaS":
  - SaaS (Software as a Service)
    - Aplicaciones ya desarrolladas
    - Ej.: Salesforce, Google Apps
  - PaaS (Platform as a Service)
    - Plataforma de ejecución de nuevas aplicaciones
    - Ej.: Google App Engine
  - IaaS (Infrastructure as a Service)
    - Computación y almacenamiento
    - Ej.: Amazon EC2



Cloud computing



Cloud computing

- Algunos ejemplos:
  - Grandes empresas:
    - Amazon AWS
    - Google Cloud Platform
    - Microsoft Azure
    - Ejemplo de SaaS: Google Docs, Microsoft Office365, Dropbox, iCloud, Gmail, OnLive
  - Software libre:
    - OpenStack

Cloud computing

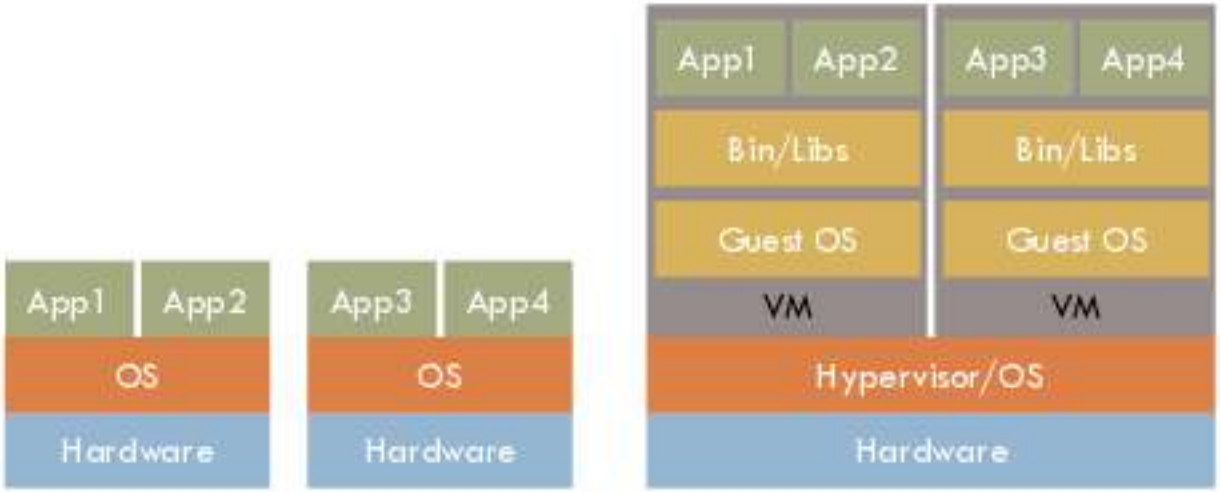
- Modelos de cloud:
  - Público:
    - Proporcionada por un vendedor externo
  - Privado:
    - Infraestructura propia (internet/intranet)
  - Hibrido:
    - Parte externa y parte interna
      - Permite ajustar costes y adaptarse a la demanda.
      - Seguridad de datos

Cloud computing

- Elasticidad
  - Capacidad de adaptarse a las necesidades
    - Pieza clave de la nube
  - Se fundamenta en la virtualización
    - Capa en la que se sustenta IaaS
    - Virtualizan hardware donde se instala un SO
      - Sobre el SO se ejecutan las aplicaciones y servicios
    - Proporcionan una solución al problema de la seguridad mediante el aislamiento
    - Requieren una imagen completa del SO
      - Incluyendo binarios adicionales y bibliotecas

Cloud computing

- Legacy vs Virtualización/IaaS



Cloud computing

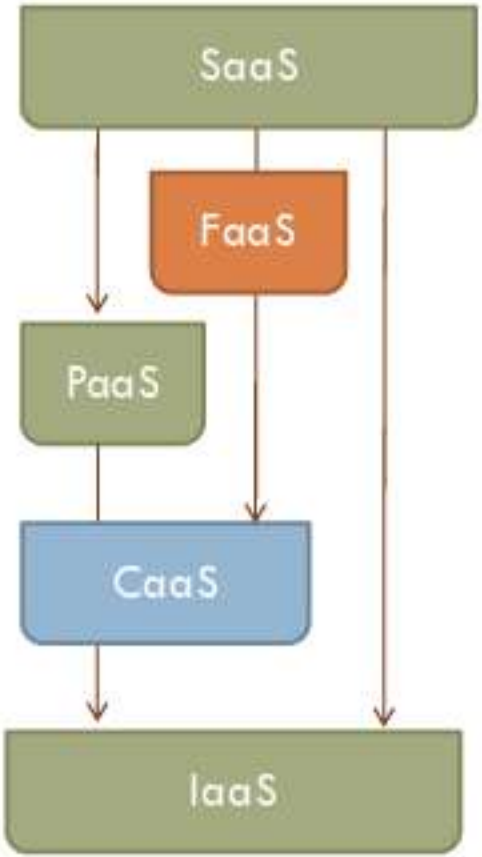
- Maquinas virtuales
  - Problemáticas a la hora de desplegar aplicaciones
    - Requieren una copia completa del SO + binarios y bibliotecas
    - Requieren mucha memoria y disco
    - Cierta sobrecarga en la ejecución de aplicaciones
    - Lentas en arrancar, migrar y parar
  - Solución: PaaS
    - Permite centrarse en la aplicación, no en la infraestructura
    - Pueden consistir en plataformas cerradas

Aplicaciones y sistemas distribuidos

PaaS, Contenedores y CaaS

La nube y los contenedores

- CaaS



Cloud computing

- Evolución del PaaS
  - Primera generación
    - Arquitecturas cerradas con API específico
    - Ejemplo: Google App Engine
  - Segunda generación
    - Arquitecturas abiertas (sobre contenedores o equivalentes)
    - Ejemplos: Cloud Foundry (diego), Open Shift (docker)
  - Tercera generación
    - Diseñadas sobre los contenedores
    - Despliegue sobre cloud publico o privado

Cloud computing

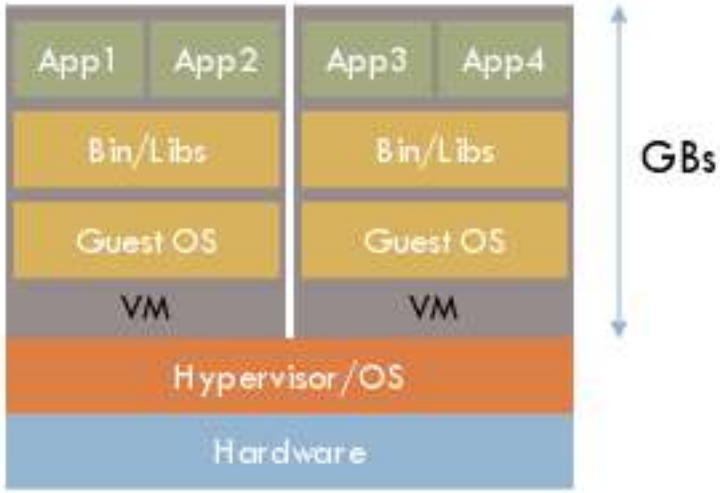
- Contenedor
  - Definición
    - Forma de virtualización ligera
      - Virtualizan los subsistemas del SO (un solo kernel)
    - Incluye una aplicación y sus dependencias
  - Ventajas
    - Permite la ejecución de manera aislada
    - Permite el despliegue rápido y consistente en diferentes entornos (desarrollo/producción)
    - Permite un uso más eficiente de los recursos
    - Facilita el desacoplamiento de las aplicaciones (microservicios)
    - Facilita la trazabilidad de las aplicaciones y sus dependencias

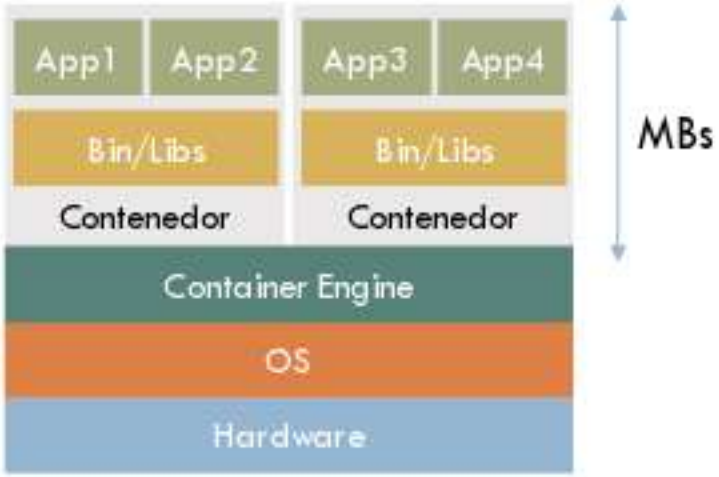
Cloud computing

- Hypervisor
  - Ejecutan distintas copias del kernel para cada máquina virtual
    - Permite distintos SO
  - Se comunica con el host mediante emulación
- Contenedor
  - Comparten el mismo kernel
    - Uso más eficiente de recursos
    - Permiten más densidad
  - Se comunican con el host mediante llamadas al sistema
  - Están tan aislados como una máquina virtual

Cloud computing

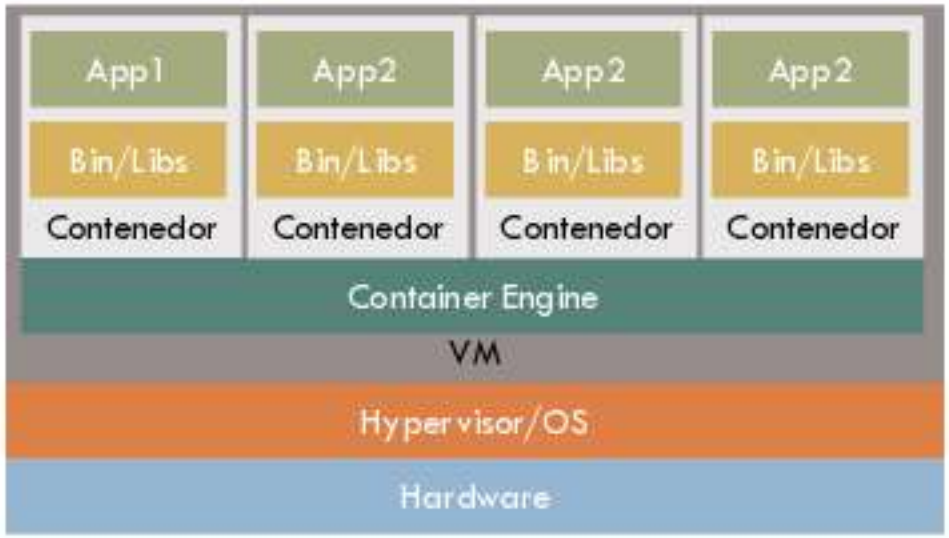
- Virtualización/IaaS vs Contenedores/CaaS





Cloud computing

- Virtualización/IaaS + Contenedores/CaaS



Cloud computing

- Granularidad de contenedores
  - Número de aplicaciones por contenedor
  - Una aplicación por contenedor
    - Permite componer nuevas pilas tecnológicas de forma sencilla
      - Ejemplo: cambiar o actualizar el servidor web
    - Los componentes de la aplicación se pueden sustituir o actualizar con facilidad
    - Sobrecarga de gestión y comunicación

Cloud computing

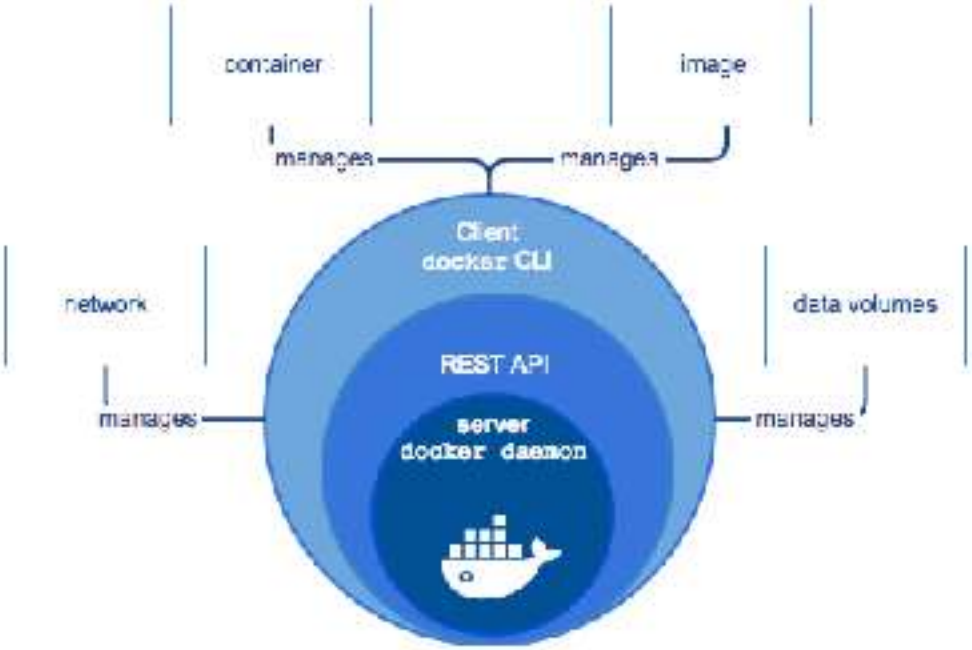
- Orquestación de contenedores
  - Gestión de los contenedores que conforman una aplicación
    - Desde una sola máquina hasta múltiples clusters de máquinas
  - Funciones
    - Configuración declarativa
      - Información sobre los contenedores necesarios
    - Reglas y restricciones
      - Dónde y como ejecutar cada contenedor
    - Descubrimiento
      - Saber como acceder a ellos en un entorno dinámico
    - Monitorización
      - Asegurarse de que el despliegue encaja con estado deseado

Cloud computing

- Ejemplos de herramientas para la gestión de contenedores
  - Docker
  - Rkt

Cloud computing

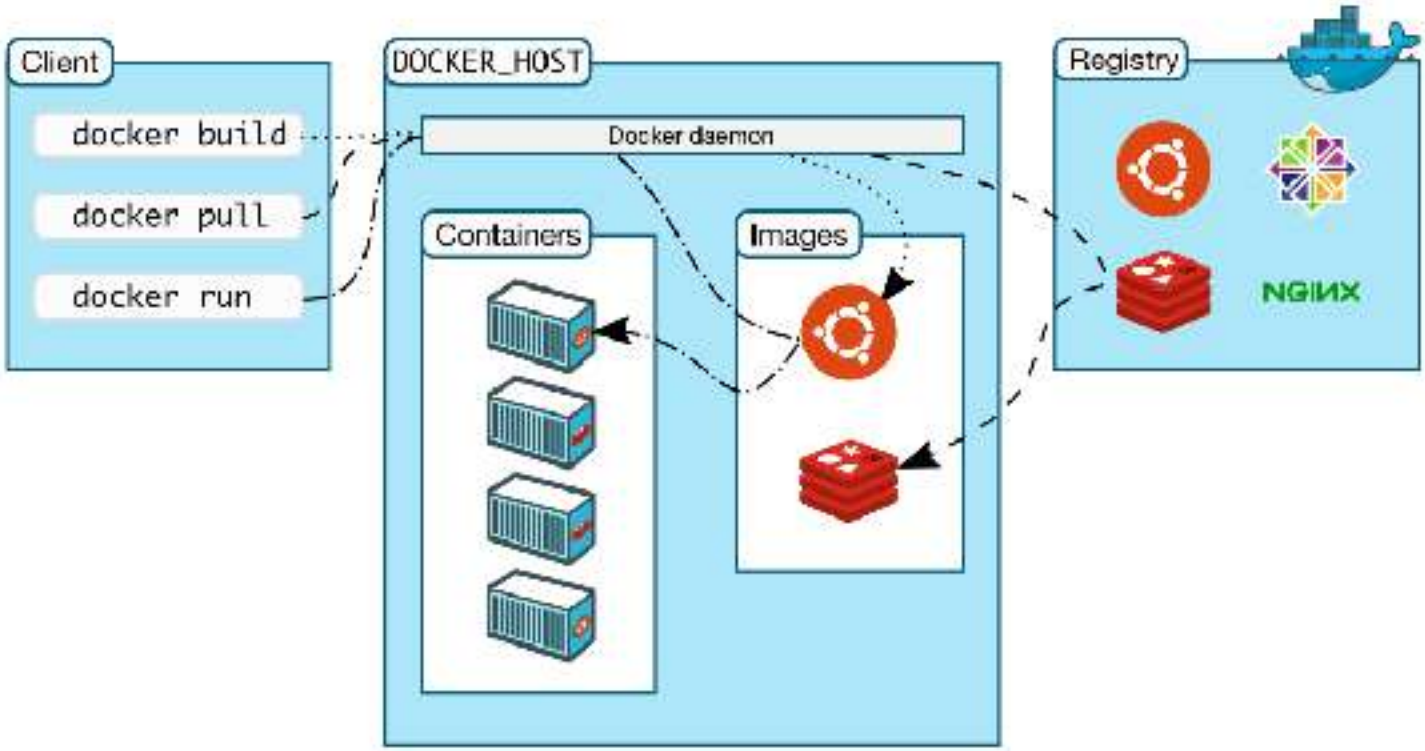
- Docker



https://docs.docker.com/engine/understanding-docker/

Cloud computing

- Docker



https://docs.docker.com/engine/understanding-docker/

Dockerfile

```
# MariaDB Dockerfile

# Pull base image.
FROM dockerfile/ubuntu

# Install MariaDB.
RUN \
  apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 0xcbcb082a1bb943db && \
  echo "deb http://mariadb.mirror.iweb.com/repo/10.0/ubuntu `lsb_release -cs` main" >
/etc/apt/sources.list.d/mariadb.list && \
  apt-get update && \
 ...

# Define mountable directories.
VOLUME ["/etc/mysql", "/var/lib/mysql"]

# Define working directory.
WORKDIR /data

# Define default command.
CMD ["mysqld_safe"]

# Expose ports.
EXPOSE 3306
```

- Dockerfile

```
FROM node:argon

# Create app directory
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# Install app dependencies
COPY package.json /usr/src/app/
RUN npm install

# Bundle app source
COPY . /usr/src/app

EXPOSE 8080
CMD [ "npm", "start" ]
```

Dockerfile multi-stage

```
# Se usa una imagen con el SDK para compilar
FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /app

# Se copia el csproj y se restaura para cachear los paquetes en capas
COPY *.csproj ./
RUN dotnet restore

# Se copia el resto y se compila
COPY . ./
RUN dotnet publish -c Release -o out

# Se usa la imagen con el runtime para ejecutar
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS runtime
WORKDIR /app
COPY --from=build /app/out .
ENTRYPOINT ["dotnet", "minapi.dll"]
```

Cloud computing

- Ejemplos de herramientas para la orquestación de contenedores
  - Docker swarm
  - Kubernetes

kubernetes
by Google

La nube y los contenedores

- Kubernetes
  - Un gestor de cluster para Docker
  - Originalmente desarrollado para Google Cloud Engine
  - Despliegue de contenedores sobre cluster de máquinas
  - Escalado y balanceo de carga
    - Mediante replicas
  - Monitorización y control del estado
  - Actualización de contenedores con zero downtime
    - Y vuelta a versiones anteriores
  - Un maestro del cluster gestiona varios workers (minions)

La nube y los contenedores

- Kubernetes
  - POD
    - Grupo de contenedores
    - Unidad básica de Kubernetes
    - Todos los contenedores forman parte de un pod se planifican juntos en el mismo worker
  - Controladores de replicación
    - Se aseguran de que un número específico de pods se encuentran en ejecución
    - Crean nuevos pods si es necesario
    - Se encargan de gestionar actualizaciones, creando versiones actualizadas de nuevos pods y terminando las antiguas

La nube y los contenedores

- Kubernetes
  - Servicios
    - Actúan de puntos de acceso y balanceadores de carga
    - Establecen un punto de acceso único y distribuyen la carga de trabajo en los pods replicados
    - La replicación de pods se registra en los servicios

Aplicaciones y sistemas distribuidos

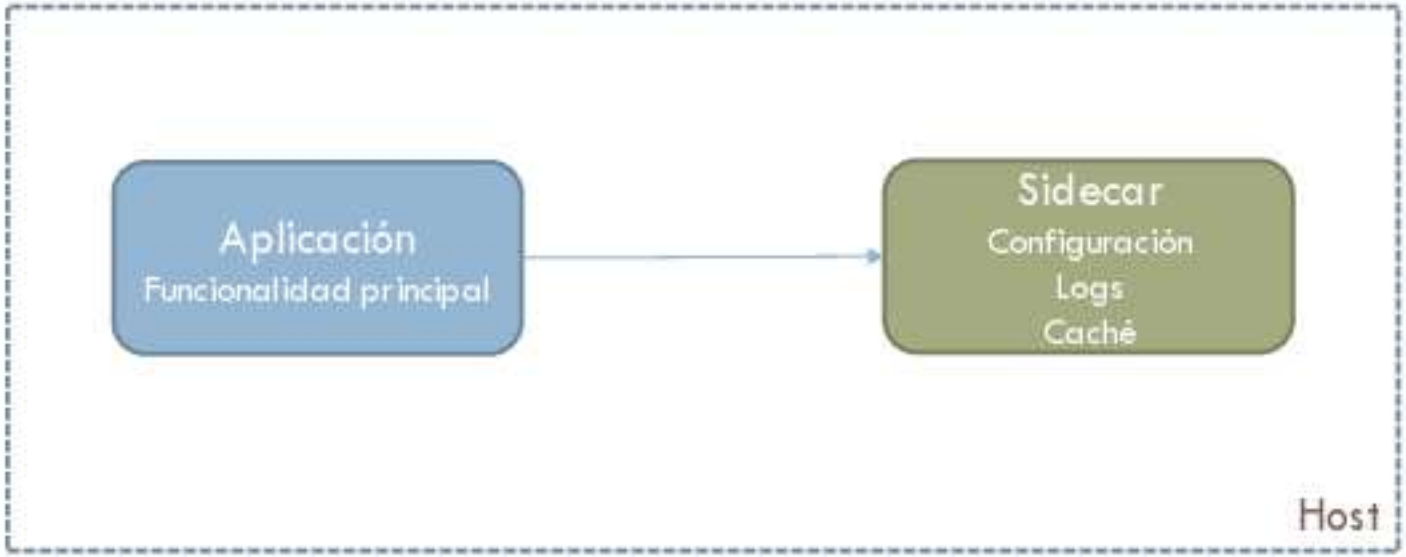
Estrategias de despliegue

Cloud computing

- Despliegue
  - Un servicio por host
    - Aislamiento
    - Poco eficiente
  - Múltiples servicios por host
    - Problema de conflictos en las dependencias
    - Poco eficiente
  - Un servicio por contenedor
    - Aislamiento
    - Eficiente

Cloud computing

- Despliegue con patrón sidecar
  - Desplegar los componentes de la aplicación en procesos o contenedores independientes para proporcionar aislamiento y encapsulación



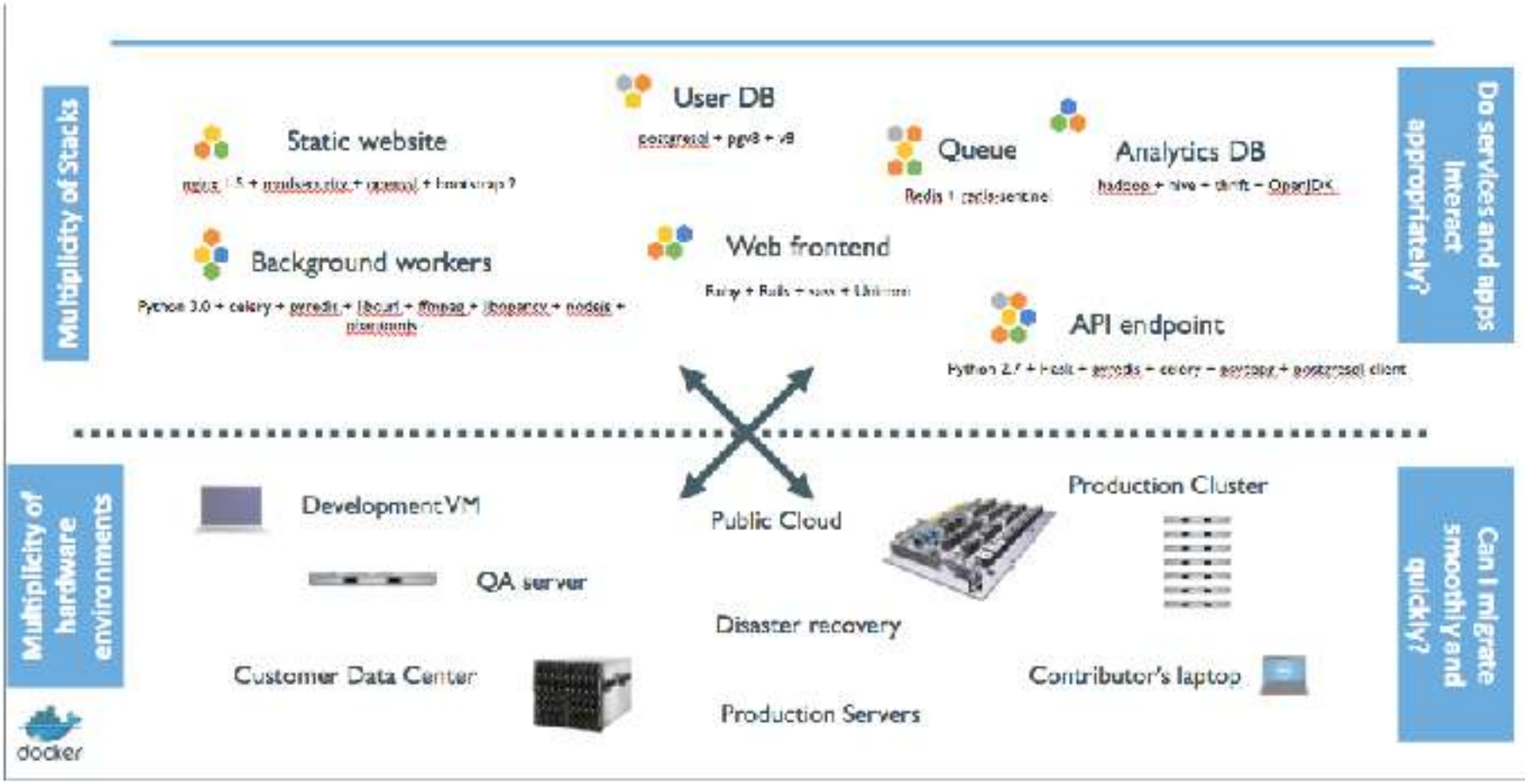
Docker

Docker

- Motor de los cambios recientes en el modelo de trabajo
  - Antes
    - Aplicaciones monolíticas
    - Ciclos de desarrollo largos
    - Entornos únicos
    - Dificultad para escalar
  - Ahora
    - Servicios desacoplados
    - Mejoras rápidas e iterativas
    - Muchos entornos completamente diferentes
    - Facilidad para escalar

Docker

- El problema del despliegue



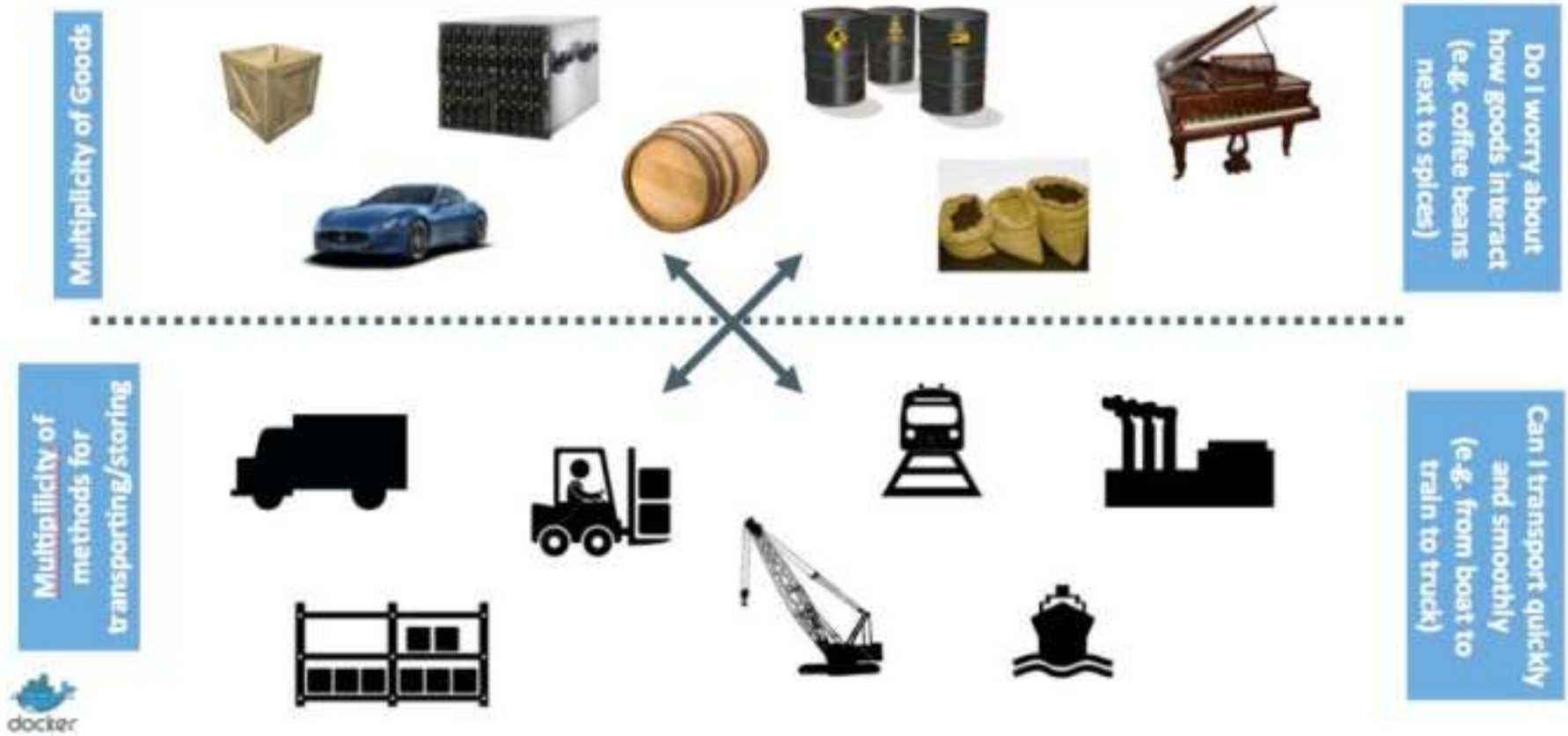
Docker

- Matriz de despliegue

	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

Docker

- Paralelismo con la industria del transporte



Docker

- Paralelismo con la industria del transporte: contenedor

Docker

- Evolución de la industria del transporte

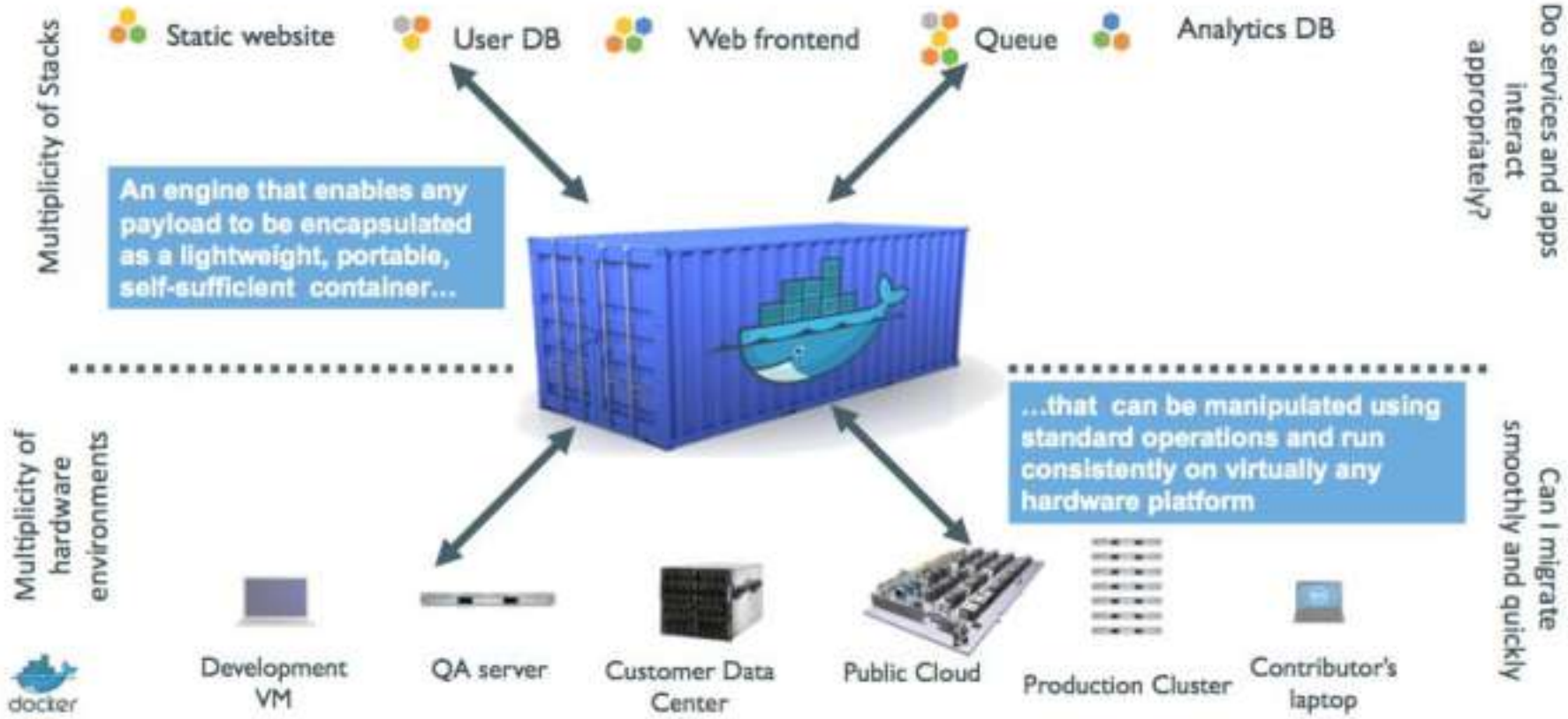
- 90% of all cargo now shipped in a standard container
- Order of magnitude reduction in cost and time to load and unload ships
- Massive reduction in losses due to theft or damage
- Huge reduction in freight cost as percent of final goods (from >25% to <3%)

→ massive globalization

- 5000 ships deliver 200M containers per year

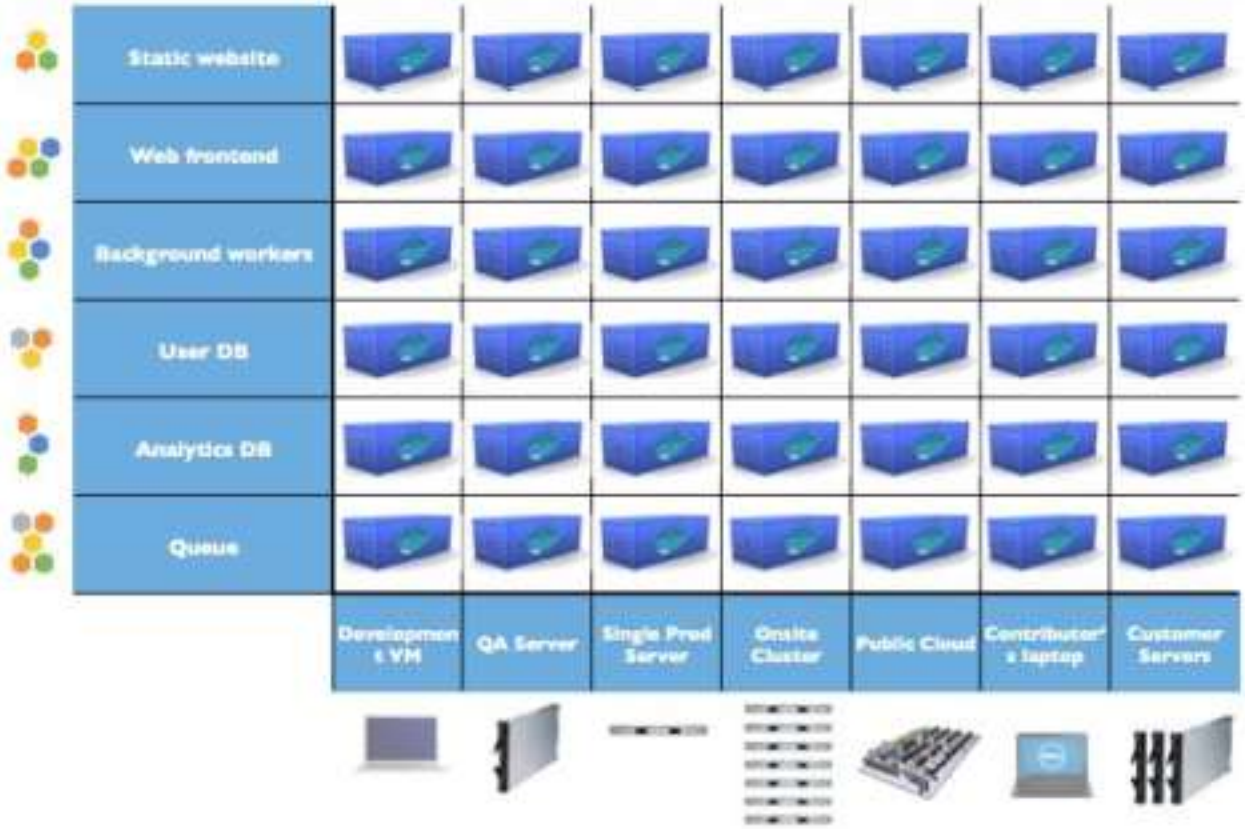
Docker

- Contenedores para aplicaciones



Docker

- Evolución de la matriz de despliegue



Docker

- Algunos resultados
  - Reducción del tiempo de desarrollo a producción de 9 meses a 15 minutos (ING)
  - Trabajos de integración continua reducidos más del 60 % (BBC)
  - Despliegues 100 veces al día (GILT)
  - Consolidación del 70% de infraestructura (MetLife)
  - Consolidación del 60% de infraestructura (Intesa Sanpaolo)

Docker

- Instalación y dependencias
  - Antes
    - Manual de instalación
  - Ahora
    - Se escribe un script que contiene los comandos de instalación
    - La instalación se incluye en un fichero de descripción de la imagen del contenedor
    - Si el contenedor funciona localmente, funcionará en cualquier otro entorno con independencia de la plataforma
    - Evita el "el mi ordenador funcionaba", "no es la misma versión", "falta una dependencia"

Docker

- Distribución de aplicaciones
  - Antes
    - Paquetes: rpm, deb, tgz
    - Problema de dependencias
  - Ahora
    - Las imágenes contienen todas las dependencias
    - Las imágenes ocupan más, pero se optimizan con capas
    - Cada capa se distribuye de forma independiente

Docker

- Forma de trabajo
  - Se escribe un Dockerfile para cada componente de la aplicación o microservicio
  - Se usan imágenes oficiales para interconectar el sistema: base de datos, caché, registro de servicios, logs, visualización
  - Se escribe un fichero de composición con la descripción de todos los servicios que se tienen que lanzar y sus opciones
  - Lanzar la aplicación implica solo dos comandos:
    - git clone …
    - docker-compose up

Docker

- Uso de contenedores para integración continua
  - Se construyen entornos de tests para las aplicaciones siguiendo del esquema anterior
  - Se pueden lanzar nuevos contenedores para cada test
  - Cada ejecución se puede realizar en un nuevo entorno
  - Cada test se ejecuta con independencia

Docker

- Uso de contenedores para compilar aplicaciones
  - Se construyen entornos para compilar las aplicaciones
  - Se almacenan los resultados en repositorios o localmente
  - Se pueden ejecutar para hacer pruebas o en producción con gran accesibilidad
  - Es fácil volver a versiones anteriores si algo falla

Docker

- Desacoplamiento de componentes
  - Un componente accede a los servicios a través de nombres: db, redis, api
  - El servicio de DNS resuelve el acceso de forma segura
  - Los contenedores que forman parte de una aplicación solo se ven entre ellos
  - Facilita la escalabilidad y replicación de componentes

Contenedores

Contenedores

- Contenedor (definición de docker)
  - Una unidad estandarizada de software
  - Contiene una aplicación y todas sus dependencias
  - Se ejecuta correctamente con independencia del entorno
  - Aseguran la ejecución de forma aislada
  - Comparte el sistema operativo del host por tanto son ligeros
  - Es un proceso normal del sistema, pero con restricciones de acceso al entorno

Contenedores

- Orígenes de los contenedores
  - Jails, chroot
    - Cambia el directorio raíz del sistema
    - Los procesos que hacen referencia a / en realidad acceden a un directorio de usuario
    - Permite exponer a los procesos a una parte del sistema de archivos
    - Con problemas de seguridad y sin aislamiento de red u otros procesos
  - Tecnologías
    - IBM VM/370 (1972)
    - Linux VServers (2001)
    - Solaris Containers (2004)
    - FreeBSD jails (1999-2000)

Contenedores

- Linux Control Group (cgroup)
  - Característica del kernel que permite limitar y gestionar los recursos asignados a procesos
  - Introducida en 2008 en el kernel de Linux por ingenieros de Google
  - Permite establecer unos criterios de uso de recursos a un grupo de procesos

Contenedores

- Linux Namespaces
  - Característica del kernel que permite crear vistas aisladas de recursos y asignarlos a procesos (llamada al sistema clone)
  - Permite que los procesos dentro del namespace tenga su propia instancia del recurso
  - Cada grupo de procesos puede tener su propia vista parcial del sistema
  - El desarrollo comienza en 2002 con el namespace mnt y llega a una versión completa en 2013 con el namespace user
  - Todavía hoy hay namespace en desarrollo

Contenedores

- Linux Namespaces
  - *mnt*
    - *Gestión de puntos de montaje y directorios*
  - *pid*
    - *Gestión de pids*
  - *net*
    - *Gestión de la red*
  - *Ipc*
    - *Gestión de comunicación entre procesos*
  - *uts*
    - *Gestión de nombres de dominio y de host*
  - *user*
    - *Gestión de usuarios y grupos*
  - *cgroup*
    - *Limitación de recursos*

Contenedores

- LXC (Linux Containers)
  - Tecnología introducida en 2008 para permitir la ejecución en entornos aislados de aplicaciones usando cgroup y namespaces
  - Permite que grupos de procesos se ejecuten de manera completamente aislada

Contenedores

- Docker
  - Tecnología para la gestión de contenedores
  - Inicialmente se ejecutaba sobre LXC
    - Posteriormente se introduce libcontainer
  - Introduce la posibilidad de que usuarios empaqueten sus aplicaciones en imágenes que luego se ejecuten dentro de un contenedor (standard software unit)
    - Las aplicaciones se pueden mover y copiar fácilmente entre máquinas
  - En la actualidad se ejecuta sobre runC, una especificación abierta del Open Container Initiative (OCI)

Contenedores

- Estandarización
  - Docker se convierte en estándar alrededor de la OCI (Open Containers Initiative)
  - Se desarrollan otros motores de contenedores compatibles
  - Se desarrolla una plataforma de herramientas: docker-compose, swarm, docker-machine
  - Se lanza una versión comercial

Contenedores

- Un contenedor comparte el kernel con el host
  - El contenedor debe realizar al llamadas al sistema
    - Las aplicaciones que se ejecutan dentro de un contenedor están diseñadas para un sistema operativo concreto
    - Una aplicación de Windows no se puede ejecutar en un contenedor que se ejecuta sobre Linux
- El contenedor comparte la arquitectura con el host
  - El contenedor ejecuta código sobre la CPU del host
    - Las aplicaciones que se ejecutan dentro de un contenedor están diseñadas para una arquitectura concreta (x86-64, arm32)
- Las imágenes son específicas para sistema operativo y arquitectura
  - docker run arm64v8/busybox
    - exec user process caused "exec format error"

Contenedores

- Ejemplo: .NET Core Runtime
  - Imagen base
    - mcr.microsoft.com/dotnet/core/runtime:2.1
  - Etiquetas:
    - Linux amd64 Tags: 2.2.7-stretch-slim, 2.2.7-alpine3.9, …
    - Linux arm64 Tags: 3.0.0-preview9-buster-slim-arm64v8, …
    - Windows Server amd64 Tags: 2.2.7-nanoserver-1903, …

Contenedores

- Windows y Mac pueden ejecutar contenedores de Linux
  - Cómo?
    - Los ejecutan sobre una máquina virtual de linux
  - Windows también soporta contenedores nativos de windows
  - En rápida evolución

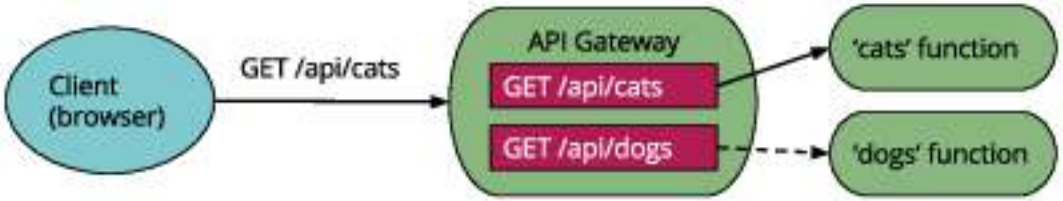
FaaS

FaaS

- Definición

Martin Fowler

Fundamentally, FaaS is about running backend code without managing your own server systems or your own long-lived server applications.



https://martinfowler.com/articles/serverless.html

FaaS

- Function as a service (FaaS)
  - Se realiza el despliegue de las aplicaciones como servicios
  - Cada servicio es una función
  - No se gestionan servidores ni infraestructura (serverless architecture)
  - Coste por invocación
  - El escalado es automático
  - Ejemplos: AWS Lambda, Google Cloud Functions, Microsoft Azure Functions

FaaS

- AWS Lambda

https://aws.amazon.com/lambda/

FaaS

Azure functions

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;

namespace LocalFunctionProj
{
    public static class HttpExample
    {
        [FunctionName("HttpExample")]
        public static async Task<IActionResult> Run(
            [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post", Route = null)] HttpRequest req,
            ILogger log)
        {
            log.LogInformation("C# HTTP trigger function processed a request.");

            string name = req.Query["name"];

            string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
            dynamic data = JsonConvert.DeserializeObject(requestBody);
            name = name ?? data?.name;

            return name != null
                ? (ActionResult)new OkObjectResult($"Hello, {name}")
                : new BadRequestObjectResult("Please pass a name on the query string");
        }
    }
}
```

FaaS

Azure functions

```python
import logging

import azure.functions as func


def main(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('Python HTTP trigger function processed a request.')

    name = req.params.get('name')
    if not name:
        try:
            req_body = req.get_json()
        except ValueError:
            pass
        else:
            name = req_body.get('name')

    if name:
        return func.HttpResponse(f"Hello, {name}. This HTTP triggered function executed successfully.")
    else:
        return func.HttpResponse(
             "This HTTP triggered function executed successfully. Pass a name in the query string.",
             status_code=200
        )
```

Aplicaciones y sistemas distribuidos

Procesamiento distribuido

Rubén Usamentiaga
Área de Arquitectura y Tecnología de Computadores
Universidad de Oviedo

Introducción

- Procesamiento distribuido
  - Objetivo
    - Incrementar el rendimiento a través del paralelismo en CPU, memoria, disco y red
  - Se necesitan mecanismos para almacenar la información
  - Se necesitan mecanismo para procesar la información de manera escalable

Procesamiento distribuido

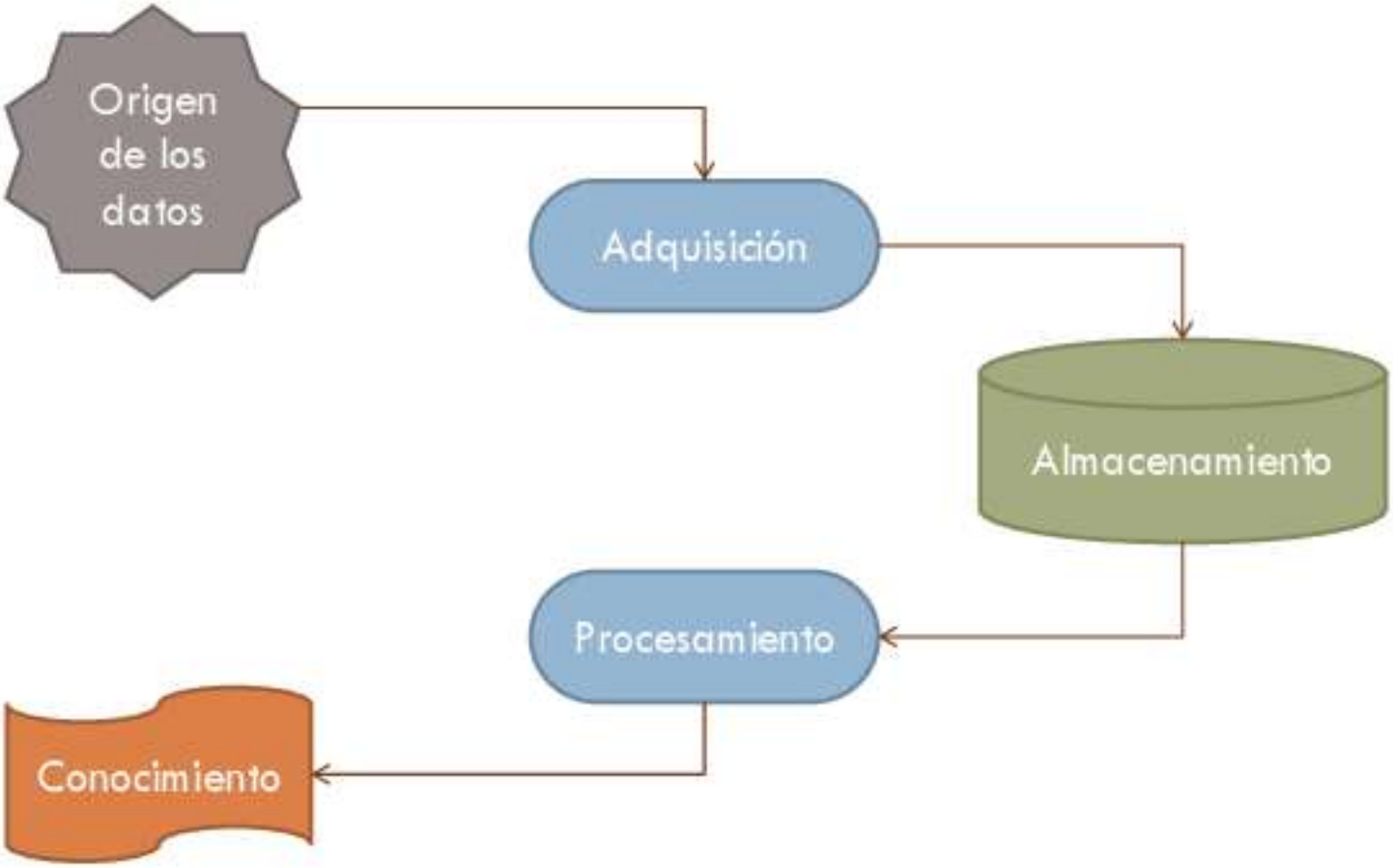
Datos

Datos

- Origen de los datos:
  - Intrumentación
    - Sensores
  - Simulación
    - Física, química, etc
  - Imágenes
    - Video, imágenes médicas, renderización
  - Otros
    - Documentos, …

Datos

- Escenario de procesamiento de datos típico:



Datos

- Un ejemplo: Large Hadron Collider (LHC)
  - Objetivo
    - Explorar los orígenes del universo
  - Funcionamiento
    - Acelera partículas (haces de hadrones, partículas subatómicas)
    - Las hace colisionar cuando alcanzan el 99% de la velocidad de la luz
    - La colisión produce altísimas energías que permiten simular algunos eventos ocurridos durante o inmediatamente después del big bang

Datos

- Datos en el LHC:
  - Varios detectores convierten la física en datos
  - Se generan aproximadamente 15 petabytes al año
    - 500 megabytes/segundo de media
    - 1000-1600 megabytes/segundo cuando el LHC está funcionando
  - Miles de científicos alrededor del mundo quieren acceder y analizar los datos

Datos

- Más ejemplo:
  - Distributed Aircraft Maintenance Environment (DAME)
    - Procesamiento de datos obtenidos de los aviones durante el vuelo
  - Earth Systems Grid (ESG)
    - Procesamiento de datos sobre el clima
  - Network for Earthquake Engineering Simulation (NEES)
    - Procesamiento de datos sobre terremotos
  - Grid for Ocean Diagnostics, Interactive Visualisation and Analyis (GODIVA)
    - Procesamiento de datos sobre el oceano

Datos

- Otro ejemplo:
  - En un mundo digitalizado cada acción deja una huella digital
    - Accesos a una web
    - Búsquedas
    - Compras
    - Conversaciones
    - Escuchar música
    - Fotos
  - Análisis de información
    - Genera conocimiento que ayuda en la toma de decisiones
    - Ventaja competitiva

Datos

- Cantidad de información

How much information 2000 - Berkeley

The world has produced about 1.5 billion gigabytes of information in 1999

How much information 2003 - Berkeley

The world has produced about 3 billion gigabytes of information in 2002

http://www2.sims.berkeley.edu/research/projects/how-much-info/
http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/

Datos

- Cantidad de información

Eric Schmidt, 2010

There were 5 Exabytes of information created between the dawn of civilization through 2003, but that much information is now created every 2 days

Datos

- Definición de big data

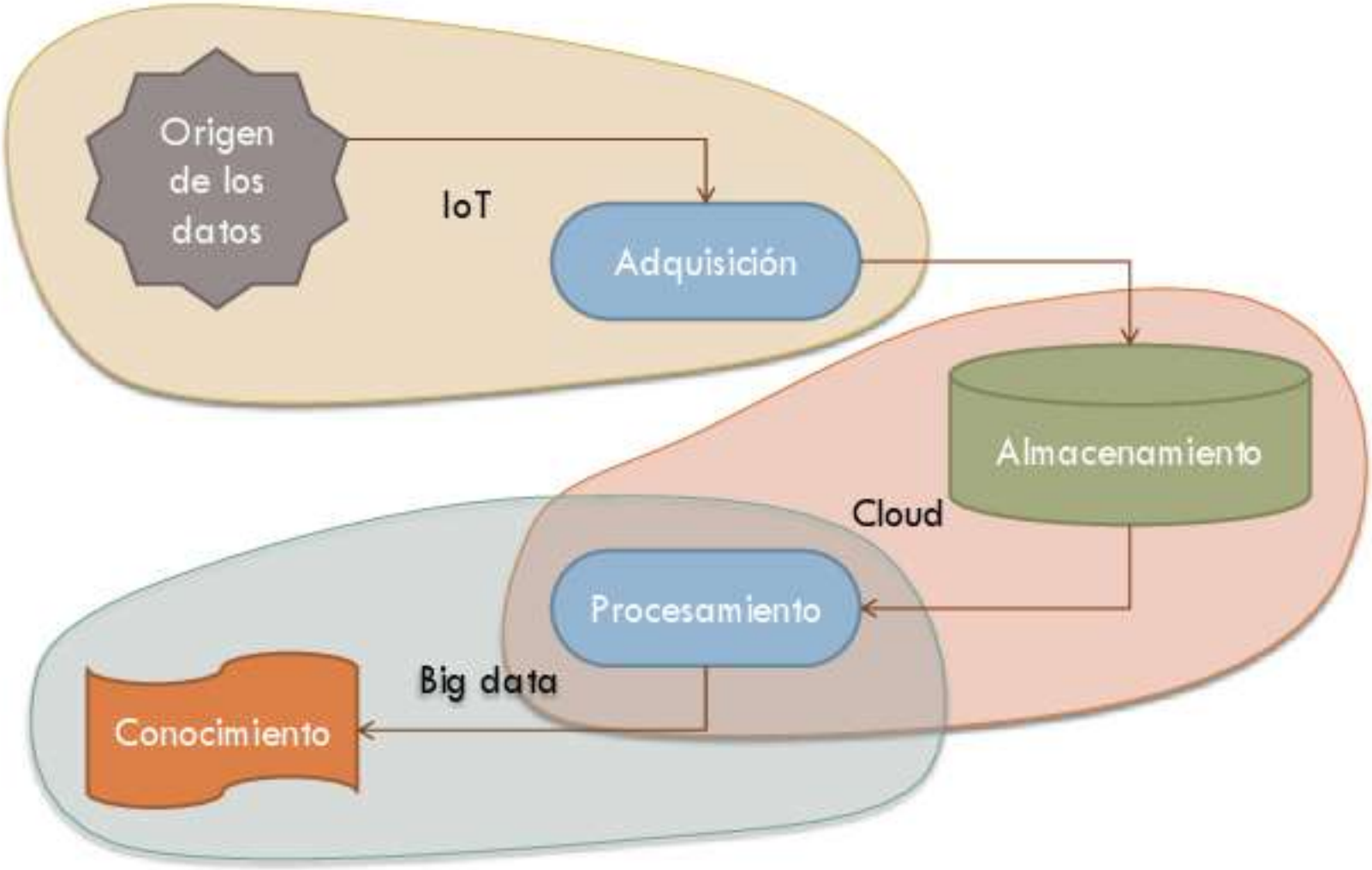
Oxford English Dictionary

Extremely large data sets that may be analysed computationally to reveal patterns, trends, and associations, especially relating to human behaviour and interactions

Datos

- Las Vs de Big data:
  - Volumen
    - Escala de los datos
  - Variedad
    - Diferentes formatos de los datos
  - Velocidad
    - Análisis de los flujos de datos
  - Veracidad
    - Incertidumbre de los datos
  - Visualización
  - Valor
  - Variabilidad

Datos


Origen de los datos
IoT
Adquisición
Almacenamiento
Cloud
Procesamiento
Big data
Conocimiento

Datos

- Almacenamiento
  - Sistema de ficheros distribuidos
    - HPC: Lustre
    - Big data: HDFS
      - Proyecto de Apache Inspirado en GFS (Google File System)
      - Desarrollado para dar soporte a tareas Map/Reduce
      - Pensado para para grandes ficheros cuyo acceso sigue un patrón de una escritura y muchas lecturas
      - Tolerante a fallos gracias a la replica de información en los nodos

Procesamiento distribuido

Modelos de procesamiento

Procesamiento

- Pasos para distribuir la ejecución de un problema
  - Descomponer el problema en tareas
    - Implica encontrar la concurrencia
      - Problema clave
    - Tipos
      - Descomposición de los datos
      - Descomposición de las funciones
  - Asignar las tareas a unidades de ejecución
    - Demasiadas tareas por unidad contra pocas tareas por unidad
    - Dependiente de las tareas

Procesamiento

- Modelos
  - Procesamiento en lotes (batch)
  - Procesamiento en tiempo real (stream)
  - Procesamiento híbrido

Procesamiento distribuido

Procesamiento con Condor

Condor

- Procesamiento en lotes en HPC
  - Se describe el trabajo junto con sus requisitos
  - Se envía a un cluster para que se planifique su ejecución
    - Segundos, minutos, días
  - Un gestor se encara buscar los recursos necesario para planificar el trabajo
    - Ejemplos: PBS, Torque, Condor

Condor

- Ejemplo Condor

Condor elige cuando y donde ejecutar el trabajo, monitoriza su progreso y, finalmente, informa al usuario de su terminación

Condor

- Pasos para ejecutar un trabajo en Condor:
  1. Elegir un universo
  2. Hacer el trabajo *batch-ready*
  3. Crear el fichero de descripción del trabajo
  4. Enviar el trabajo

Condor

1. Elección del universo:
   - El universo indica la forma en la que Condor debe gestionar el trabajo
   - Tipos:
     - *Vanilla: para casi cualquier tipo de trabajo secuencial*
     - *Standard: para trabajos con checkpoints y migración*
     - *Java: para trabajos programados en Java*
     - *VM: para máquinas virtuales*
     - *Grid: para trabajos que se ejecutan en un grid externo*
     - *Parallel: para trabajos paralelos (MPI)*
     - *Docker: para contenedores*

Condor

2. Hacer el trabajo *batch-ready*:
   - Tiene que ejecutarse en background:
     - Sin E/S interactiva, interfaz, etc.
   - Se puede usar STDIN, STDOUT y STDERR:
     - Con redirección a ficheros
   - Se puede acceder a otros ficheros de datos

```
$ programa < entrada.txt > salida.txt 2>errores.txt &
```

batch-ready
≈
programa clásico de shell

Condor

3. Crear el fichero de descripción del trabajo:
   - Fichero de texto con la información sobre el trabajo:
     - Nombre del programa ejecutable
     - Argumentos del programa
     - Universo
     - Ficheros de entrada/salida
     - Variables de entorno
     - Requisitos
     - Preferencias

Fichero de descripción
≠
ClassAd

```
Executable = programa
Universe = vanilla
Input = entrada.txt
Output = salida.txt
Error = errores.txt
Log = log.txt
Queue
```

programa.sub

Condor

- ClassAds (Classified Advertisements)
  - Mecanismo para encajar tareas con máquinas:
    - Es la forma que utiliza Condor para que los trabajos y los recursos anuncien sus características y requisitos

ClassAD de un trabajo

```
MyType        = "Job"
TargetType    = "Machine"
Owner         = "ruf"
Requirements =
    (Arch == "INTEL")
&& (OpSys == "LINUX")
&& (Disk >= DiskUsage)
&& ((Memory * 1024)>=ImageSize)
...
```

ClassAD de una máquina

```
MyType = "Machine"
TargetType = "Job"
Name = "servidor.uniovi.es"
Machine = "servidor.uniovi.es"
Rank = 0.000000
OpSys = "LINUX"
Arch = "X86_64"
Activity = "Idle"
...
```

Condor

4. Enviar el trabajo:
   - El trabajo se envía utilizando el comando condor_submit
   - El comando condor_submit:
     - Procesa el fichero de descripción y crea el ClassAd
     - Envía el ClassAd al Schedd
   - El Schedd almacena el trabajo en la cola
     - El comando condor_q permite ver el estado de la cola

Condor

- Antes del enviar un trabajo se puede comprobar el estado del pool:

```
[ruf@glite-tutor ruf]$ condor_status

Name               OpSys      Arch   State     Activity LoadAv Mem   ActvtyTime

slot1@iceage-wn-01 LINUX      INTEL  Unclaimed Idle     0.000  1012  0+00:03:29
slot2@iceage-wn-01 LINUX      INTEL  Unclaimed Idle     0.000  1012  0+01:48:25
slot3@iceage-wn-01 LINUX      INTEL  Unclaimed Idle     0.000  1012191+16:48:32
slot4@iceage-wn-01 LINUX      INTEL  Unclaimed Idle     0.000  1012  0+02:05:07

...

slot3@iceage-wn-14 LINUX      INTEL  Unclaimed Idle     0.000  1012 16+21:19:09
slot4@iceage-wn-14 LINUX      INTEL  Unclaimed Idle     0.000  1012 16+21:19:10

                     Total Owner Claimed Unclaimed Matched Preempting Backfill

        INTEL/LINUX     56     0       0        56       0          0        0

               Total    56     0       0        56       0          0        0
```

Condor

- Ejemplo de envío:

```
[ruf@glite-tutor ruf]$ condor_submit programa.sub
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 111.

[ruf@glite-tutor ruf]$ condor_q


-- Submitter: glite-tutor.ct.infn.it : <193.206.208.141:9674> : glite-tutor.ct.infn.it
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
 111.0   ruf             1/24 13:48   0+00:00:00 R  0   0.0  programa

1 jobs; 0 idle, 1 running, 0 held

[ruf@glite-tutor ruf]$ condor_q


-- Submitter: glite-tutor.ct.infn.it : <193.206.208.141:9674> : glite-tutor.ct.infn.it
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD

0 jobs; 0 idle, 0 running, 0 held
```

Condor

- Log del trabajo:

```
[ruf@glite-tutor ruf]$ more log.txt
000 (112.000.000) 01/24 13:55:38 Job submitted from host: <193.206.208.141:9674>
...
001 (112.000.000) 01/24 13:55:41 Job executing on host: <193.206.208.205:9680>
...
005 (112.000.000) 01/24 13:55:45 Job terminated.
        (1) Normal termination (return value 0)
                Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
        56  -  Run Bytes Sent By Job
        5101  -  Run Bytes Received By Job
        56  -  Total Bytes Sent By Job
        5101  -  Total Bytes Received By Job
...
```

El log describe la vida del trabajo

Procesamiento distribuido

Procesamiento con Hadoop

Hadoop

- Hadoop
  - Proyecto de Apache Inspirado en *papers* de Google
  - Framework para el almacenamiento y el procesamiento distribuido de grandes cantidades de datos
    - Almacenamiento: HDFS
    - Procesamiento: MapReduce

  - Idea diferenciadora:
    - Mover el procesamiento al lugar donde están los datos
      - Moving computation/processing doser to data
    - No mover los datos al lugar de procesamiento usando la red

Hadoop

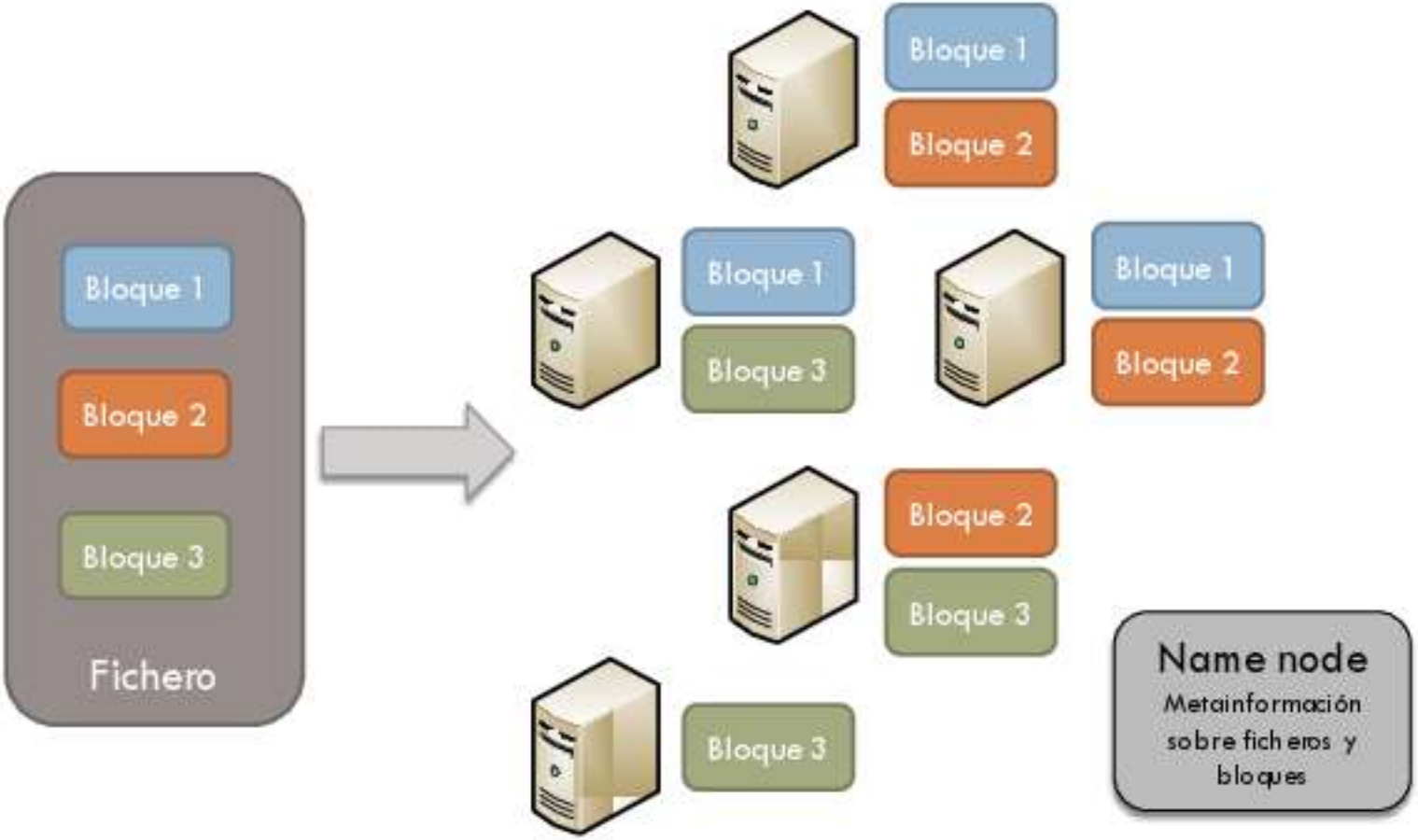
- Hadoop File System (HDFS):
  - Origen:
    - Proyecto de Apache Inspirado en GFS (Google File System)
  - Características:
    - Desarrollado para dar soporte a tareas MapReduce
    - Pensado para para grandes ficheros cuyo acceso sigue un patrón de una escritura y muchas lecturas
    - Tolerante a fallos gracias a la replica de información en los nodos
    - No permite montar el sistema de ficheros (si con FUSE)
      - Herramientas de acceso propias

Hadoop

- Hadoop File System (HDFS):
  - Cada fichero
    - Se divide en bloques (por defecto 64MB)
    - Se replica en los nodos (por defecto 3)
      - Durabilidad
      - Disponibilidad
      - Productividad
  - Se permite añadir
    - No se permite insertar o modificar
      - Mejor crear otro fichero

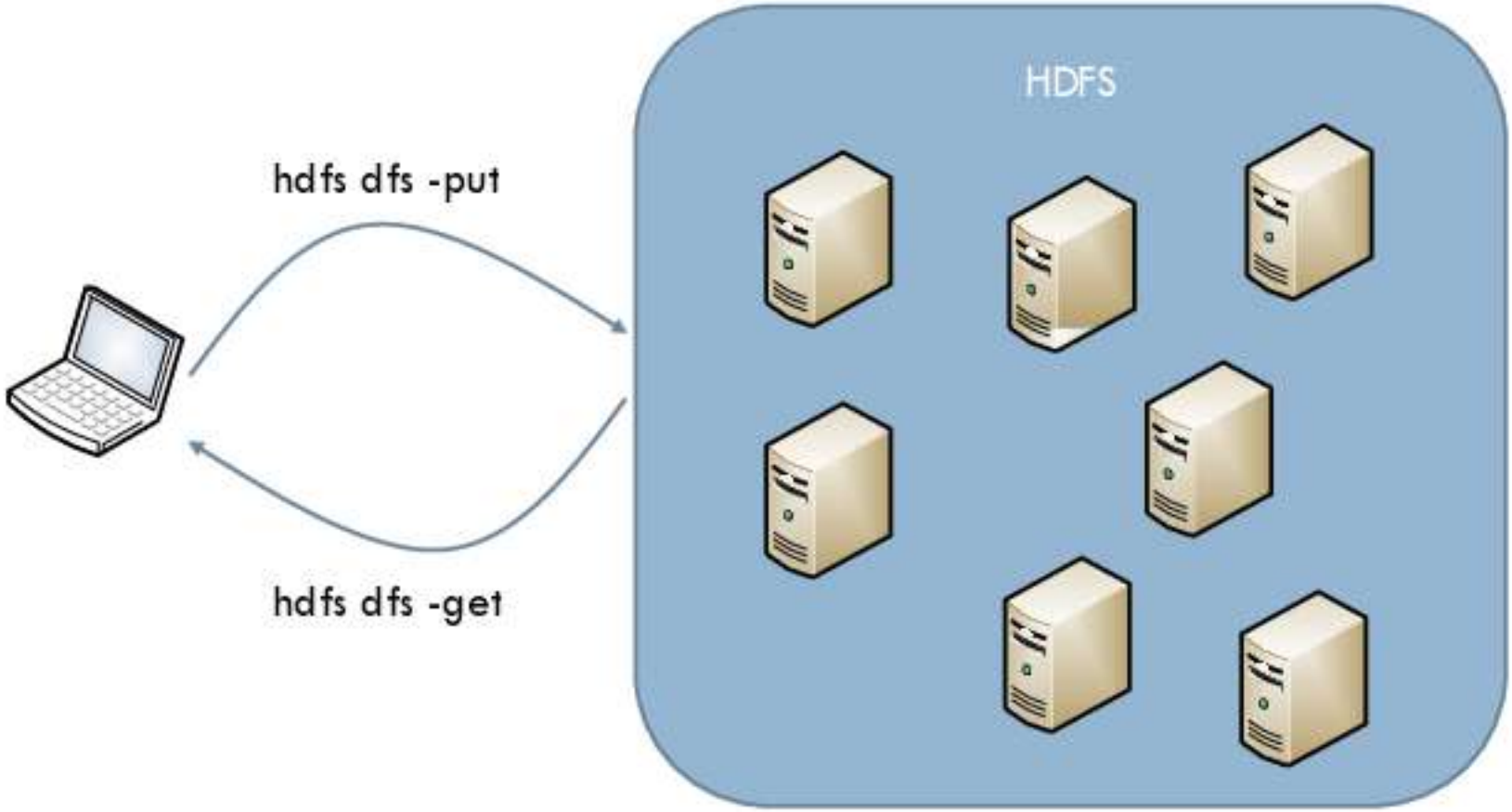
Hadoop

- Hadoop File System (HDFS):



Hadoop

- Hadoop File System (HDFS):



Hadoop

- MapReduce
  - Origen:
    - Proyecto de Apache Inspirado en Google MapReduce
  - Framework para ejecutar tareas en paralelo
    - El programador establece cómo procesar los datos
    - El resto de cuestiones ya están resultas
      - Paralelismo
      - Ejecución remota
      - Distribución de los datos
      - Balanceo de carga
      - Tolerancia a fallo

Hadoop

- MapReduce
  - Operaciones fundamentales:
    - Map
      - Procesa datos y produce resultados intermedios
      - Pares de (clave, valor)
    - Reduce
      - Se ejecuta una vez completados todos los Map
      - Procesa todos los resultados intermedios para una misma clave
      - Se llama una sola vez por clave

Hadoop

- MapReduce
  - Contar palabras en documentos

```
// key: nombre del documento
// value: contenido

map(String key, String value):
        for each word w in value:
                EmitIntermediate(w, "1");

// key: una palabra
// values: una lista con la cuenta

reduce(String key, Iterator values):
        result = 0;
        for each v in values:
                result += ParseInt(v);
        Emit(key, result);
```

MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Deanand and SanjayGhemawat, 2004

Hadoop

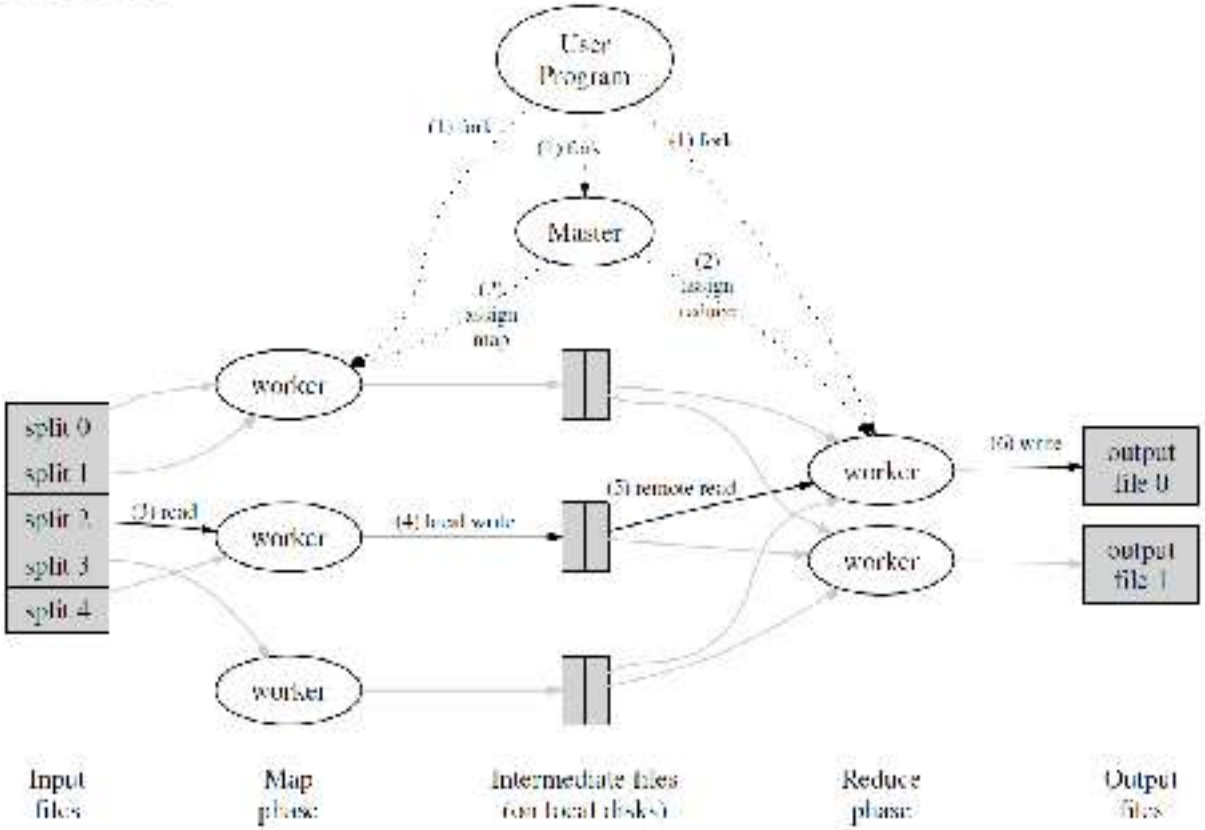
- MapReduce
  - Ejecución:
    - Se dividen los ficheros en bloques (splits)
    - Se crean los procesos de ejecución
      - 1 Máster
      - N Workers (dependiendo del número de nodos)
      - Cada worker ejecutará una tarea Map o Reduce
        - M tareas Map
        - R tareas Reduce

Hadoop

- MapReduce
  - Ejecución:
    - Un worker con una tarea Map
      - Lee los contenidos del bloque
      - Llama a la función Map
      - Notifica al master
    - El master notifica a las tareas Reduce cuando terminan los Map
      - Determina la tarea Reduce usando (Clave módulo R)
      - Ordenan la información y llaman a la función Reduce
      - Las tareas Reduce recopilan la información de las tareas Map
    - El master recupera la información de las tareas Reduce y la retorna

Hadoop

- MapReduce



MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Deanand and SanjayGhemawat, 2004

Hadoop

- MapReduce
  - Tolerancia a fallos
    - El master se encarga de verificar si los worker fallan
      - En caso de fallo replanifica el trabajo
    - No se contempla el fallo del master
      - Uno solo: menos probabilidad
  - Localidad
    - El master planifica el trabajo de Map en la máquina donde están almacenados los datos
  - Tareas de backup
    - Se tienen en cuenta las operaciones *straggler* (lentas)
    - Cuando se está cerca de terminar se ejecutan tareas de backup redundantes

Hadoop

- MapReduce
  - Ejemplos de trabajos
    - Grep distribuido
    - Procesamiento de logs y accesos
    - Detección de palabras importantes en documentos
    - Creación dé índices invertidos
    - Ordenación distribuida

Procesamiento distribuido

Procesamiento con Spark

Spark

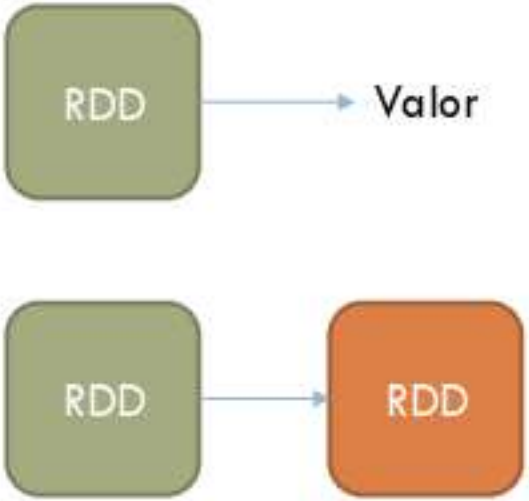
- Spark
  - Proyecto Apache con origen en University of California, Berkeley
  - Framework para procesamiento big data
  - Flexibiliza el modelo MapReduce
    - En MapReduce hay solamente una fase de Map y una fase de Reduce
  - Trabaja con los datos en memoria

Spark

- Spark
  - RDD (Resilient Distributed Dataset)
    - Unidad de datos fundamental en Spark
    - Las operaciones se realizan sobre RDDs
    - Se crean a partir de ficheros (ej: ficheros en HDFS)
      - O se generan a partir de datos u otros RDD
    - Almacenan cualquier tipo de elemento
      - Tipos primitivos, secuencias, objetos
    - Ciertos tipos de RDD permiten operaciones especiales
      - Los diccionarios permiten operaciones MapReduce

Spark

- Spark
  - Operaciones con RDD
    - Acciones
      - Count
      - Take(n)
    - Transformaciones
      - Filter
      - Map
      - Reduce



Spark

- Spark
  - Contar palabras en documentos

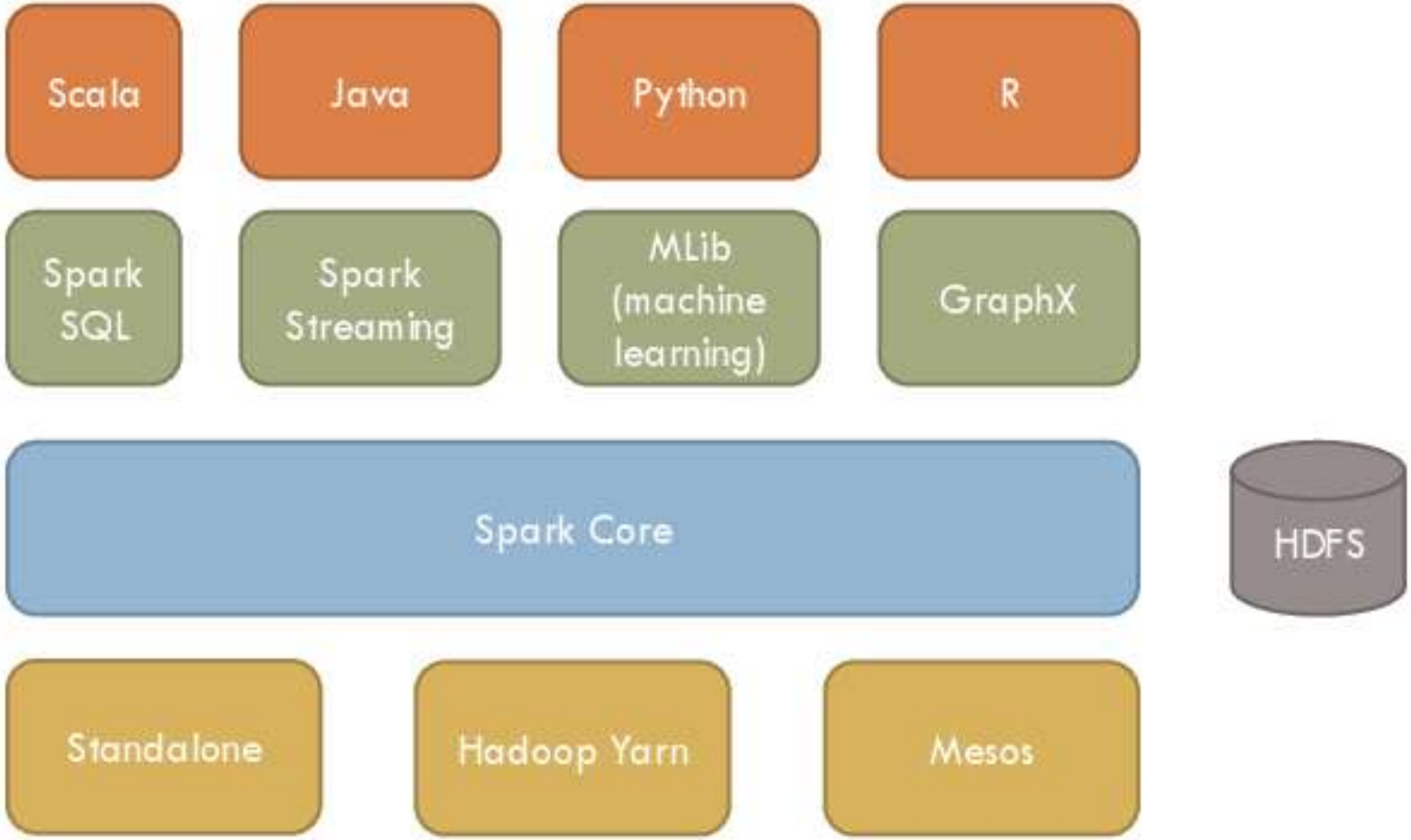
```
sc = SparkContext(...)

text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

Spark

- Spark
  - Contar líneas en un fichero

```
sc = SparkContext(...)

lines = sc.textFile("data.txt")
lineLengths = lines.map(lambda s: len(s))
totalLength = lineLengths.reduce(lambda a, b: a + b)
```

Spark


Scala
Java
Python
R
Spark SQL
Spark Streaming
MLib (machine learning)
GraphX
Spark Core
HDFS
Standalone
Hadoop Yarn
Mesos

Aplicaciones y Sistemas Distribuidos

Linkedin: https://www.linkedin.com/in/eriesgo/
X: @eriesgo
GitHub: eriesgo
enrique.riesgo@gmail.com

© 2024 Enrique Riesgo Canal. All rights reserved.

Enrique Riesgo Canal

Formación

1998 – Ingeniero Técnico en Infomática de Sistemas

2002 – Ingeniero en Informática

Carrera Profesional

2000 – 2003: AMS y CGI: Consultant

2003 – 2023: DXC (Previamente CSC), Master Technologist, Global Product Manager, Lead Architect, Product Developer, Solution Executive, People Manager y mucho más

2023 – : Plexigrid, Director of Software Engineering

Certificaciones

AWS Solutions Architect Associate, Dynatrace Solution Architect, ITIL

@eriesgo

© 2024 Enrique Riesgo Canal. All rights reserved.

“Real” Agenda

Topic
Integration Styles
REST APIs
Messaging and Event-Drive Architecture
Amazon Web Services
Microservices

© 2024 Enrique Riesgo Canal. All rights reserved.

¿Cómo diseñar un arquitectura de servicios basada en Plataforma sin caer en la trampa del monolito distribuido?

© 2024 Enrique Riesgo Canal. All rights reserved.

(1)

Entendiendo las dimensiones de la comunicación entre servicios

Photo by paul wallez on Unsplash

@eriesgo

© 2024 Enrique Riesgo Canal. All rights reserved.

Dimensiones de la comunicación

© 2024 Enrique Riesgo Canal. All rights reserved.

Ejemplos de la vida real

			Cardinalidad	
			Uno-a-uno (One-to-one)	**Uno-a-muchos** (One-to-many)
Respuesta	**Solo ida** (One-way)	Asíncrono (Asynchronous)	Escribo una carta a un amigo sobre mi ultimo viaje a Australia	Señales de humo
	Petición/Respuesta (Request-response)	**Sincrónico (Synchronous)**	Pregunto a alguien “¿qué hora es?”	Votación en el Congreso de los Diputados
		Asíncrono (Asynchronous)	Llamo a mi abogado y le digo “prepárame un contrato”	Solicito presupuesto para renovar mi cocina en un portal de profesionales

© 2024 Enrique Riesgo Canal. All rights reserved.

Ejemplos tecnológicos

			Cardinalidad	
			Uno-a-uno (One-to-one)	**Uno-a-muchos** (One-to-many)
Respuesta	**Solo ida** (One-way)	Asíncrono (Asynchronous)	• Messaging: queues (eg SQS) • REST API (202 Accepted)	• Messaging: Pub/Sub (eg SNS, Kafka)
	Petición/Respuesta (Request-response)	**Sincrónico** **(Synchronous)**	• RPI/RPC, gRPC, RMI • REST API	• Webhooks (HTTP)
		Asíncrono **(Asynchronous)**	• Messaging: 2 x queues • REST Hooks	• Scatter-Gather pattern with messaging

© 2024 Enrique Riesgo Canal. All rights reserved.

(2)

Seleccionando los estilos de comunicación adecuados

Photo by Jean-Philippe Delberghe on Unsplash

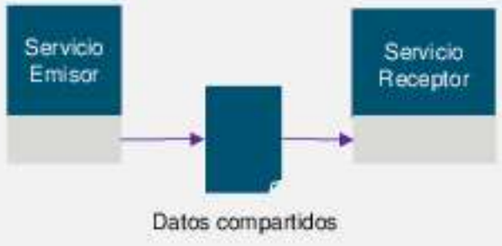
@eriesgo

© 2024 Enrique Riesgo Canal. All rights reserved.

(2) No todos los estilos de comunicación son iguales

Sofisticación

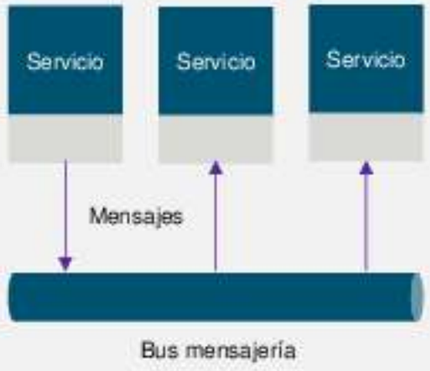
Transferencia de Ficheros



Base de Datos Compartida

Invocación de Procedimiento Remoto

Mensajería



Fuente: EAI Patterns

© 2024 Enrique Riesgo Canal. All rights reserved.

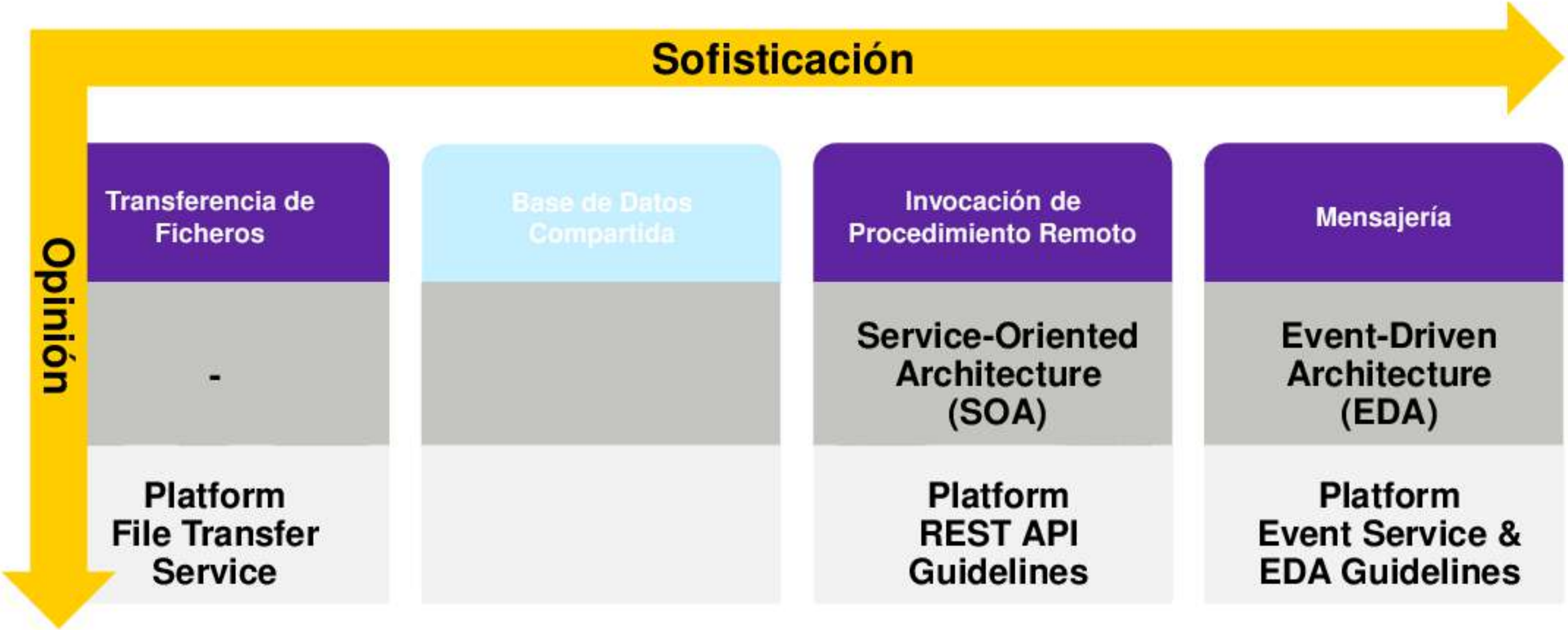
(3)

Arquitecturas y servicios de plataforma más allá de estilos y patrones

@eriesgo

© 2024 Enrique Riesgo Canal. All rights reserved.

Arquitecturas de integración y servicios según estilo



© 2024 Enrique Riesgo Canal. All rights reserved.

Architectural Approaches

REST APIs ➔ Service Oriented Architecture (SOA)?

- Client-driven, intentional, usually synchronous* **request-response** interactions
- Based in REST semantics or Commands/Queries**
- Typically, SOAP or REST APIs based (RPI)

Some limitations

- Client dictates which service will process the request (client-driven architecture). Clients highly coupled to the interface
- Client must wait for the response even if it could be doing other things ➔ **MONEY**

Events ➔ Event Driven Architecture (EDA)

- Event-driven, asynchronous **one-way** Publish/Subscribe interactions
- Based in events semantics:
  they report something that has already happened.
  No answer or reply expected.
- Messaging or event broker based

Benefits over synchronous APIs

- Reduces the physical and temporal coupling between request-response clients and services.
- Simple relationship between the components, which is **unidirectional** in nature.

(*) We can create asynchronous interactions (REST Hooks) or services using asynchronous messaging mechanisms, but could be more complex to develop than using REST API

At the conceptual level they are complementary,
and they all have a role to play in design and management of IT systems.

© 2024 Enrique Riesgo Canal. All rights reserved.

But wait… What exactly are Events?

- They report something that has already happened: reflect a change of state in an object
- They are not “requests”, but observations
- Events happen in the past (“party has changed address”)
- They are immutable
- Events are one-way messages. They are unidirectional.
  - Therefore, we cannot refuse an Event
  - If we are subscribed and you receive an event that you are not expecting or not interested in, you just ignore it.
- Events have a single source (publisher) that publishes the event. One or more recipients (subscribers) may receive events. Therefore, events can be processed multiple times, by multiple consumers

© 2024 Enrique Riesgo Canal. All rights reserved.

APIs REST

Don't forget good Software Engineering principles: modularity, cohesion, separation of concerns, information hiding and abstraction, and managing coupling

- Synchronous service communication based on SIX CONSTRAINTS… - Architectural Styles and the Design of Network-based Software Architectures
- Richardson Maturity Model - https://martinfowler.com/articles/richardsonMaturityModel.html
  - Level 2, more adopted by the industry: OpenAPI - https://www.openapis.org/what-is-openapi
  - Level 3
    - Django REST Framework - REST, Hypermedia & HATEOAS - Django REST framework
    - DXC Pragmatic Recommendations for API Design - https://developer.dxc.com/apis
      - JSON Schema et al - https://json-schema.org/
      - HAL (Hypertext Application Language) - https://tools.ietf.org/html/draft-kelly-json-hal-08
      - Tooling to use these guidelines: Halstack - https://dxc-technology.github.io/halstack/
      - DXC Developer Central API Catalog - https://developer.dxc.com/apis/catalog

- Good Practices
  - Design - Introduction - REST API Tutorial
  - Security First! - REST Security - OWASP Cheat Sheet Series
  - Why not using TDD?
  - API Expand and Contract - https://www.thoughtworks.com/radar/techniques/api-expand-contract

© 2024 Enrique Riesgo Canal. All rights reserved.

Messaging and Event Driven Architecture

- Asynchronous service communication
  - To design and architect services to react in response to events and to structure code/functions on this basis.
- What do you mean by “Event-Driven?” - https://martinfowler.com/articles/201701-event-driven.html
- My current choice
  - Event Service (Serverless)
  - Centralized. KISS.
  - Publish-subscribe pattern
  - REST API to create topics, register subscriptions and send events
  - Subscribers receive notifications (push) using HTTP or Lambda invocation
  - Thumb pipes, smart endpoints. No routing or ESB features
  - Events are stored => enabling event-sourcing pattern and event-driven microservices
- Avoid complex pieces of infra… unless no other option

© 2024 Enrique Riesgo Canal. All rights reserved.

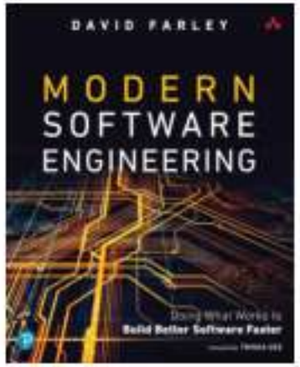
Amazon Web Services

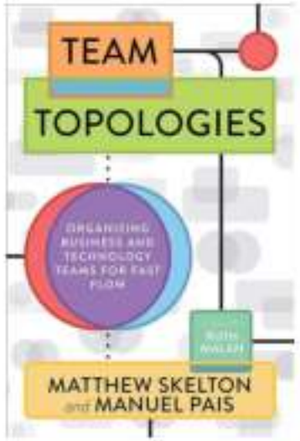
- Based on API Mandate - https://api-university.com/blog/the-api-mandate/
- My platform for choice to create SaaS ASD (Aplicaciones y Sistemas Distribuidos)
- Massive scaled and **distributed** platform
  - Global vs regional vs AZ scoped services
  - Every service has a public REST API (level 1 or level 2?) - sample
  - But internally they communicate each other asynchronously
- REST APIs implementation
  - Any back-end with HTTP interface or Lambda functions
  - API Gateway - https://aws.amazon.com/api-gateway/
  - CloudFront - https://aws.amazon.com/cloudfront/
- Platform Event Service implementation
  - Simple Notification Service (SNS) - https://aws.amazon.com/sns/
  - Simple Queue Service (SQS) - https://aws.amazon.com/sqs/
  - Distributed system challenges: incremental and unique numbers, message duplication, message ordering ➔ FIFO, but subscribers always idempotent
  - Future plans: AWS EventBridge? AWS Kinesis?

© 2024 Enrique Riesgo Canal. All rights reserved.

Microservices – The Vision

- David Farley, “Modern Software Engineering”
  - “Small, focused in one task, aligned with a bounded context, autonomous, independently deployable and loosely coupled”
  - “Microservices is an **organizational-scaling pattern**. If you don’t need to scale up development in your organization, you don’t need microservices”
- Amy Philips, “Testing Observability”
  - “If you have microservices but you wait and do end-to-end testing of a combination of them before a release, what you have is a distributed monolith”
- Microservices Guide by Martin Fowler - https://martinfowler.com/microservices/





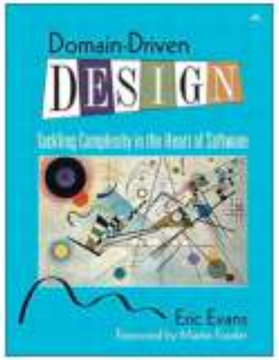
© 2024 Enrique Riesgo Canal. All rights reserved.

Microservices - The Implementation

- Standards and Principles for Microservices very well aligned with - https://microservices.io/index.html
- AWS vision
  - Technology choices - https://aws.amazon.com/microservices/
  - White paper - https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/introduction.html
- My choice is serverless - https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/serverless-microservices.html
  - Serverless microservices using AWS Lambda
  - Serverless microservices using AWS Fargate

© 2024 Enrique Riesgo Canal. All rights reserved.

DDD + SOA + API REST como punto de partida

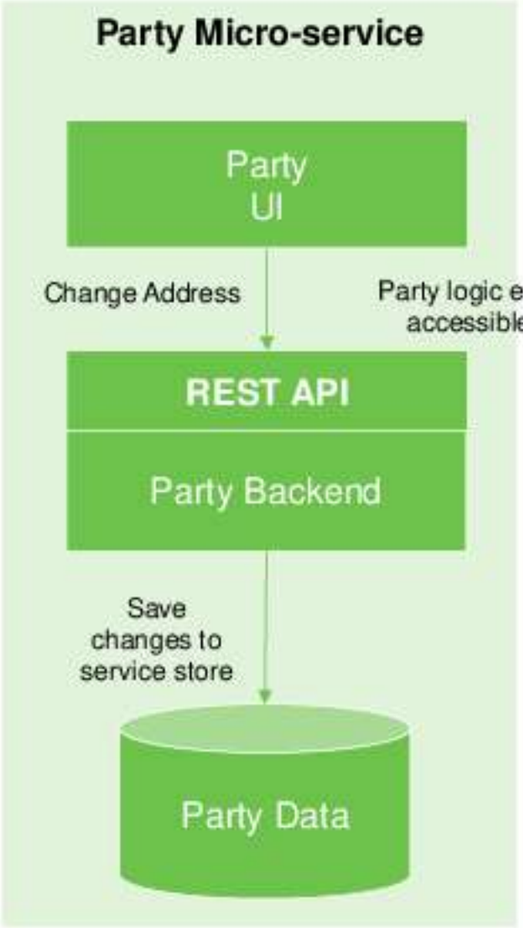


Business Domain

Notas:

SOA significa que se estructura una aplicación descomponiéndola en varios servicios (en nuestro caso como micro-servicios con API REST) que se pueden clasificar en tipos diferentes, como subsistemas o niveles. Fuente: Microsoft

Definimos servicios/micro-servicios de acuerdo a nuestros contextos delimitados (DDD bounded contexts)



Notas:

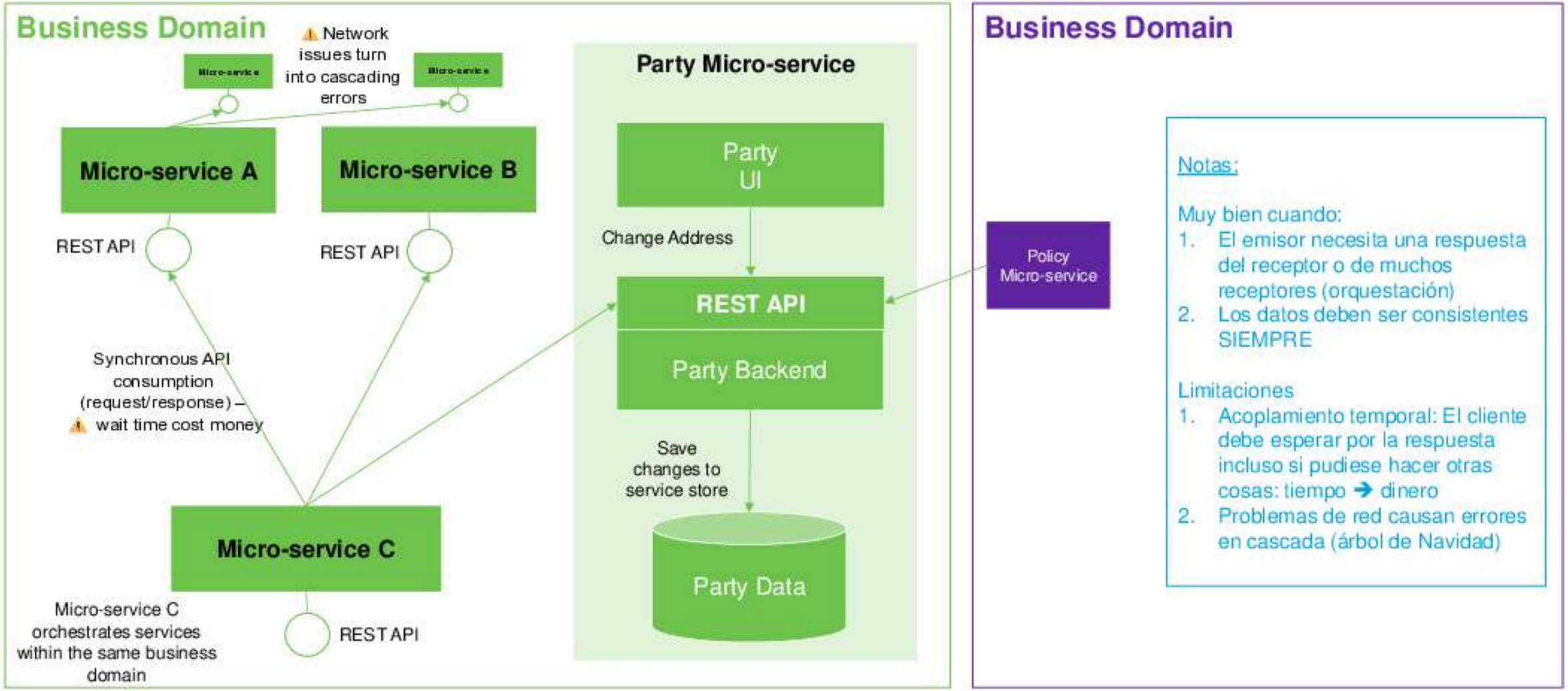
Las API REST (Nivel 3 RMM)

1. Son ligeras (mínimas dependencias externas y configuración)
2. No necesitan herramientas especiales
3. Usan contratos que fomentan bajo acoplamiento
4. Auto documentadas y descubribles

Más información: Roy T. Fielding: REST APIs must be hypertext-driven

© 2024 Enrique Riesgo Canal. All rights reserved.

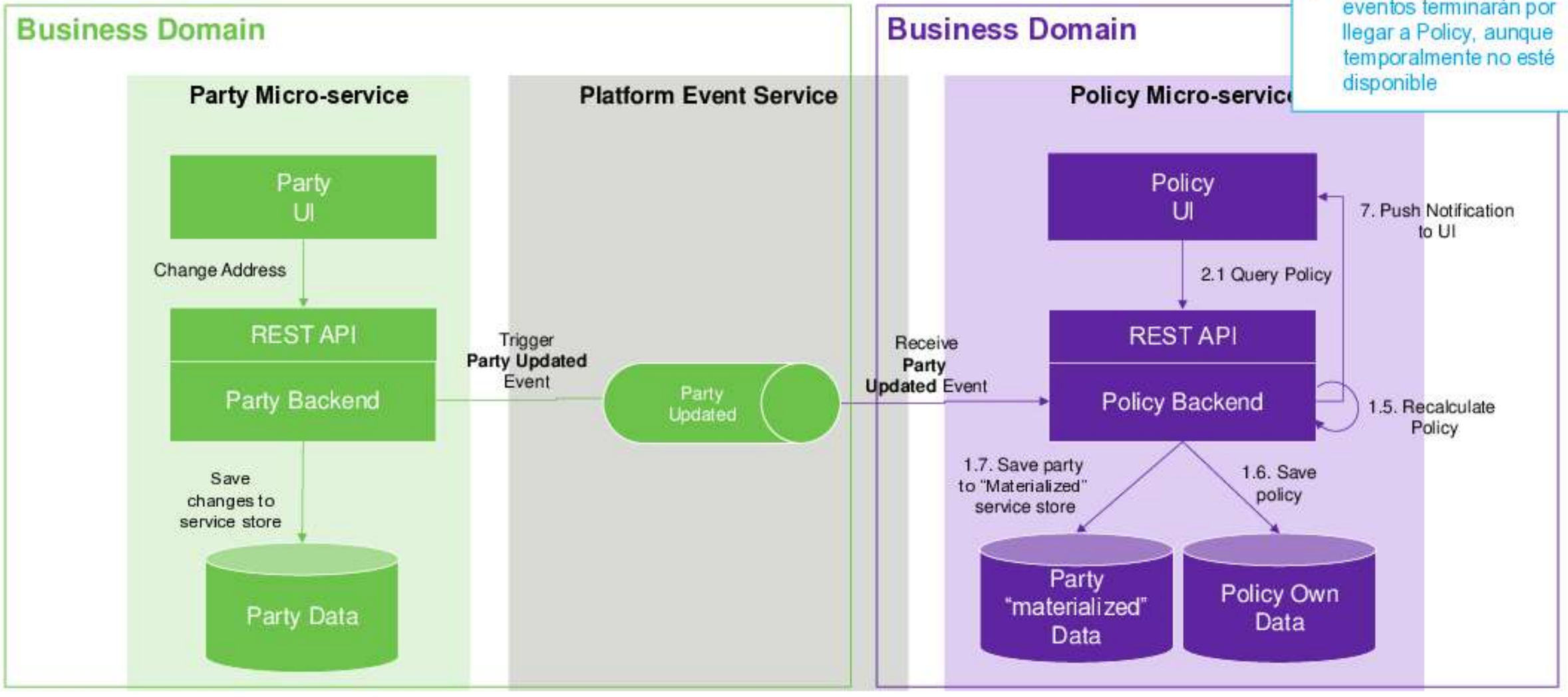
Integraciones dentro de un dominio de negocio



© 2024 Enrique Riesgo Canal. All rights reserved.

EDA como complemento a REST APIs

Notas:
1. Prevenimos que una ralentización o un fallo en Party impacte a Policy
2. Party sabe que sus eventos terminarán por llegar a Policy, aunque temporalmente no esté disponible



© 2024 Enrique Riesgo Canal. All rights reserved.

Plexigrid

Award winning start-up founded in Gijón in 2020

We are creating the BEST SOFTWARE PRODUCTS that are reinventing electricity grids for the energy transition

Four products and growing!

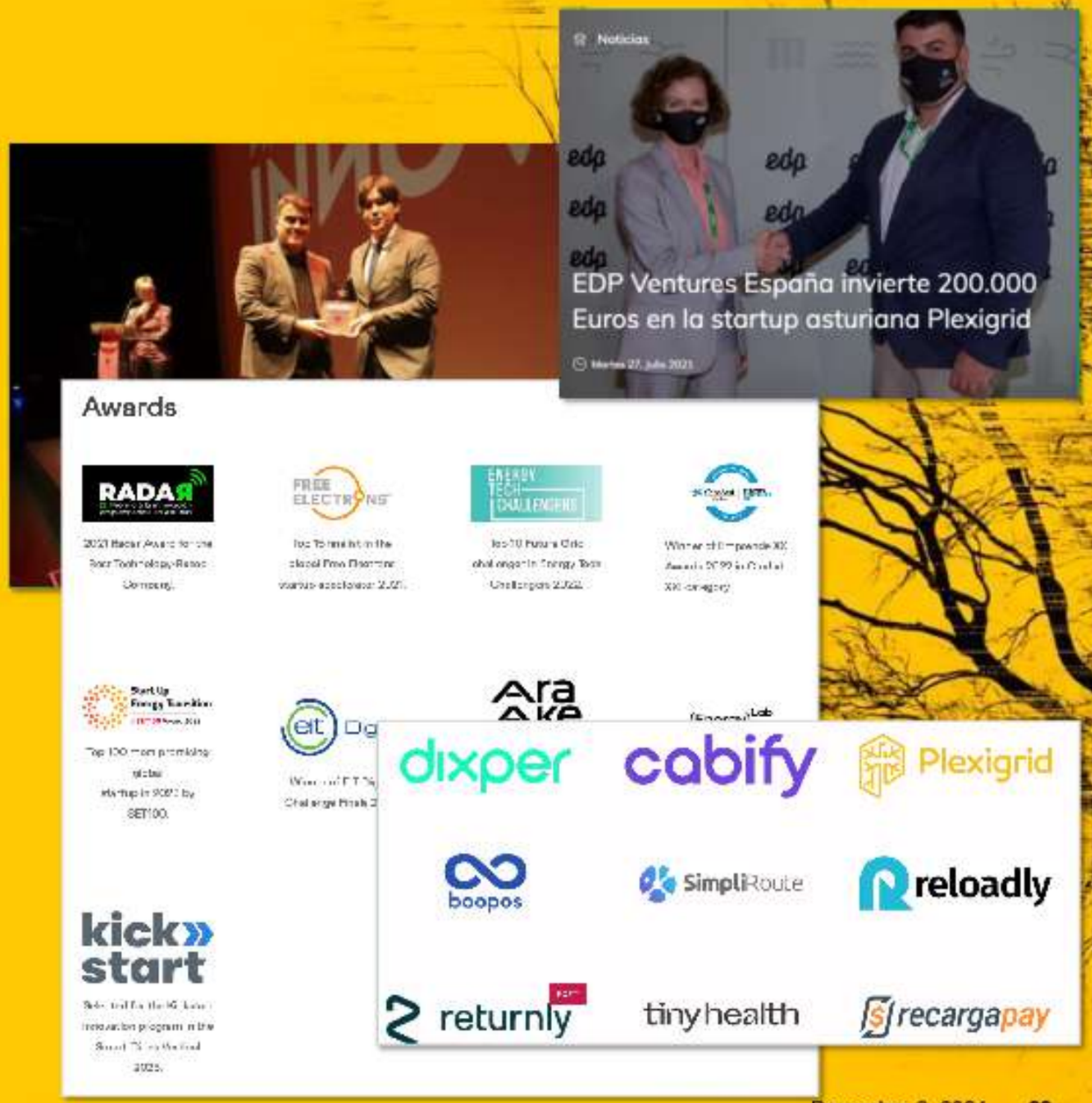
Offices in Gijón and Stockholm

Operations in five countries and growing!

AsturiasInnova+ Startup of this year (2023)

Investors including Polar, Vargas, Venture City and EDP Ventures

Featured Startup of Venture City



© 2024 Enrique Riesgo Canal. All rights reserved.

Plexi's Software Enginering Team

Joined in October 2023 as Director of Software Engineering to create the best Software Engineering team ever!

Lean Software Development culture, Modern Software Engineering practices and Team Topologies based teams

Hot Cloud native stack including like Vue, Python, Graph databases, Time series databases, Machine Learning

Platform Engineering and DevOps

Training budget, mentoring, "softwaresophy" workshops, hybrid work, regular team buildings, reading club

… And now hiring!

enrique.riesgo@plexigrid.com

© 2024 Enrique Riesgo Canal. All rights reserved.

Team B during Team Building in Malaga '23