CAPÍTULO I: INTRODUCCIÓN A LOS SISTEMAS DISTRIBUIDOS

DR.-ING. RAÚL MONGE ANWANDTER
DEPARTAMENTO DE INFORMÁTICA
UTFSM, VALPARAÍSO - CHILE

1

VISIÓN GENERAL DEL CAPÍTULO

Objetivo:

- Revisar en general el área disciplinar de sistemas distribuidos.
- Precisar conceptos, identificar desafíos y mostrar tendencias en el área.

Contenido específico:

- Revisión histórica y aplicaciones de sistemas distribuidos.
- Conceptos básicos y caracterización de los sistemas distribuidos.
- Estrategias y técnicas de diseño.
- Arquitecturas de sistemas distribuidos.

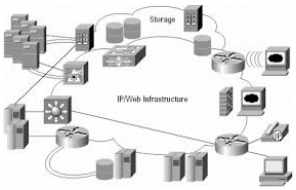
Resultados de aprendizaje (que desarrolla):

- Distingue diferentes estilos arquitectónicos y técnicas de diseño de sistemas distribuidos, aplicándolos en el desarrollo de este tipo de sistemas.

2

EVOLUCIÓN, APLICACIONES Y TENDENCIAS
EN SISTEMAS DISTRIBUIDOS

1.1 MOTIVACIÓN



RAZONES PARA DISTRIBUIR

SISTÉMICAS Y DE CALIDAD

Compartición de recursos

- Acceder y usar recursos compartidos a través de una red de comunicación.

Aumento del desempeño computacional

- Procesamiento paralelo, distribuir carga, explotar recursos ociosos, agregar nuevas capacidades, etc.

Mayor disponibilidad y resiliencia (robustez)

- Separación física para mayor confiabilidad, replicación, sitios de contingencia, etc.
- Aumento de componentes complejiza lograr este objetivo.

Seguridad del sistema

- Segmentar por dominios de confianza (compartimentar): contener y controlar el acceso.
- Aumento de superficie de ataque complejiza este objetivo.

ORGANIZACIONALES Y ESTRUCTURALES

Necesidad funcional de una organización

- Clientes remotos, recolección y procesamiento de datos (centralizado), integración con otros (e.g. *e-commerce*, *e-business*)

Distribución inherente a la aplicación

- Distribución geográfica, movilidad, interacción con el ambiente (sensible al contexto), sistemas colaborativos (e.g. Juegos, CAD)

Económicas (tb. heterogeneidad)

- Integrar sistemas heredados (*legacy*), usar diferentes proveedores (e.g. Cloud), ofrecer servicios externos.

PRIMEROS SISTEMAS DISTRIBUIDOS (DÉCADAS DEL '60 AL '80)

Década '60: Computador central con terminales (teleprocesamiento, primeras redes)

- Usuarios se comunican con un único computador (*mainframe*)
- Terminales remotos conectados por redes de telecomunicaciones (*modems* y *multiplexing*).
- Se desarrolla la ARPANET para mayor robustez y desempeño (1968-69).

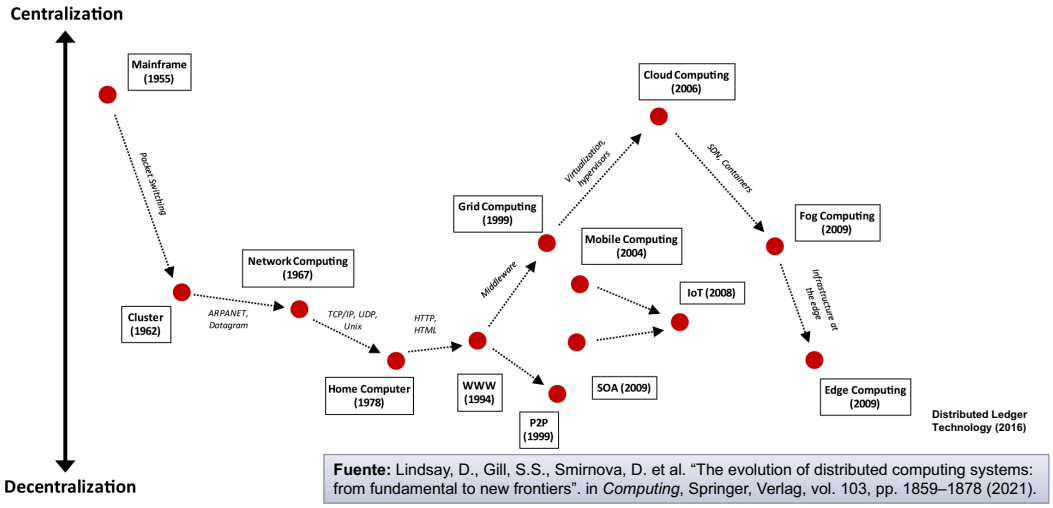
Década '70: Primeras redes de computadores (basadas en *mainframes; tb. minicomputadores*)

- Servicios de red para transferencia de archivos, *login* remoto, *e-mail* y directorio (capa aplicación).
- Definición de una arquitectura de protocolos de comunicación (TCP/IP: 1974; ISO/OSI 1980).

Década '80: Arquitecturas Cliente-Servidor (primeros servicios realmente distribuidos: computación en red)

- Estaciones de trabajo y computadores personales conectadas a servidores en redes LAN y WAN.
- Tecnologías claves: Ethernet, Unix BSD, Sockets BSD, Diskless Workstations, X Window System, RPC, NFS, etc.
- Destacan proyectos SUN-Xerox (1979) - Sun Microsystems (1982), y proyecto Athena del MIT (1983).

EVOLUCIÓN HISTÓRICA DEL PARADIGMA DE SISTEMAS DISTRIBUIDOS



Fuente: Lindsay, D., Gill, S.S., Smirnova, D. et al. "The evolution of distributed computing systems: from fundamental to new frontiers". in *Computing*, Springer, Verlag, vol. 103, pp. 1859–1878 (2021).

A) SISTEMAS DE INFORMACIÓN EMPRESARIAL (EIS): INTEGRACIÓN Y DESARROLLO

Características:

- Integración de diferentes sistemas/aplicaciones en un único y gran sistema organizacional.
- Estandarización de estructuras de datos y protocolos para intercambio de información.

Generaciones tecnológicas:

- **1980-1990:** Redes LAN, Cliente-Servidor, RPC, servidores de archivos (ONC) y de datos (ODBC).
- **1990-2000:** Objetos distribuidos (Corba), aplicaciones Web con CGI, Java Applets.
- **2000-2010:** Aplicaciones Web y móviles, tecnología de componentes y servicios web.
- **2010-hoy:** Cloud, Contenedores, Microservicios, Serverless, IoT y Blockchain.

Desafíos relevantes:

- Ingeniería para desarrollar software distribuido
- Almacenamiento y procesamiento masivos de datos
- Trasacciones distribuidos y consistencia de datos.
- Alto desempeño y alta disponibilidad.
- Heterogeneidad e integración de componentes o sistemas.
- Ciberseguridad y resiliencia.

B) SISTEMAS WEB

Características:

- Arquitectura abierta implementada sobre Internet, su desarrollo comienza en 1994 con creación de W3C.
- Modelo de recursos compartidos usando URL/URI como referencia y protocolo HTTP(S) para conexión y transporte de datos.
- Integración con aplicaciones de negocio para generación de páginas dinámicas (*front-end*).
- Acceso universal a aplicaciones usando sólo el navegador (*browser*) y HTML como lenguaje de presentación.

Desafíos relevantes:

- Integración de sistemas de información empresarial y sistemas legados (*backend*).
- Usabilidad e interactividad para usuarios.
- Representación estándar de datos (XML) y búsqueda de información.
- Código móvil y seguridad.

C) SISTEMAS DE COMPUTACIÓN MÓVIL Y UBICUA

Características:

- Basadas en redes inalámbricas y dispositivos inteligentes (*tablets*, teléfonos móviles, drones, etc.).
- Incrustación de componentes computacionales (*Pervasive Computing*) para inteligencia ambiental (RFID, sensores y actuadores).
- Integración masiva de componentes conectados a Internet (*Internet of Things*).

Desafíos relevantes:

- Capacidad de proceso y almacenamiento de elementos computacionales.
- Limitaciones de energía, estabilidad de la comunicación, disponibilidad de componentes, seguridad, entre otros.
- Concentración de grandes volúmenes de datos (*big data*) y extracción de información.

9

DESARROLLO DE SOFWARE Y APLICACIONES DISTRIBUIDAS

Servidores de archivos NFS (1986) y servidores de datos SQL (1987)

- RPC (ONC y DCE) y sentencias SQL remotas (ODBC o JDBC).

Sistemas de Objetos Distribuidos (1989)

- Plataformas típicas de middleware: Corba/OMG (1991) y DCOM/Microsoft (1993)

Sistemas Web (1993)

- Sitios y Aplicaciones Web (HTTP, HTML, URL, CGI, etc.)
- W3C (1994), XML (1996), JavaScript (1995)

Arquitecturas multi-tier y de componentes (2001)

- Middleware de desarrollo: J2EE (2001) y .NET (2002).

Arquitectura de servicios (2000-hoy)

- Arquitecturas de servicios (2000), W3C Web Service Architecture (2002), Microservicios (2004), etc.

Plataformas de desarrollo en la Nube (2006-hoy)

- AWS (2006), Google Cloud (2008), Microsoft Azure (2008), FaaS (2010), AWS Lambda (2016), etc.
- Soporte para IA, Data analytics, Blockchain, etc.

10

EJEMPLOS DE TECNOLOGÍAS ACTUALES

SISTEMAS DE COMUNICACIÓN Y MULTIMEDIALES

Características:

- Fuertemente basados en Internet.
- Fuentes de datos heterogéneos que requieren sincronización en tiempo real.
- Servicios distribuidos con múltiples consumidores (*multicast+streaming*).

Ejemplos de aplicaciones:

- Bibliotecas digitales (multimediales).
- Herramientas de educación a distancia.
- Videoconferencia (Skype, Zoom, Google Meet, Microsoft Team, etc.)
- Video y audio por demanda (Youtube, Netflix, Spotify)
- Videojuegos distribuidos

LIBRO MAYOR DISTRIBUIDO (DISTRIBUTED LEDGER)

Características:

- Blockchain como una cadena de bloques inmutable y creciente.
- Datos replicados y compartidos en una red P2P, sincronizados por consenso.
- No existe autoridad central.
- Seguridad lograda mediante criptografía.
- Aseguran alta disponiblidad e integridad de la información.

Ejemplos:

- Criptomonedas como Bitcoin (2009).
- Ethereum y contratos inteligentes (2015)
- Existen redes públicas y privadas (e.g. Hyperledger, 2015)

11

DEFINICIONES Y CARACTERIZACIONES SOBRE SISTEMAS DISTRIBUIDOS; MIDDLEWARE Y VIRTUALIZACIÓN; CLIENTE-SERVIDOR Y P2P.

1.2 CONCEPTOS BÁSICOS SOBRE SISTEMAS DISTRIBUIDOS

12

DEFINICIÓN: "SISTEMA DISTRIBUIDO"

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

[Coulouris 2011]

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

[Tanenbaum & van Steen 2017]

"*Es un sistema en el cual la falla en un computador (u otro componente), que desconocíamos de su existencia, causa que vuestro computador (o equipo) quede inutilizado*".

[Leslie Lamport 1987]

CARACTERÍSTICAS PRINCIPALES DE UN SISTEMA DISTRIBUIDO

Elementos de computación autónomos.

- Multiplicidad introduce concurrencia / paralelismo (requiere coordinación distribuida).

Subred de comunicación compartida.

- Con paso de mensajes entre componentes distribuidos (de *hardware* y/o *software*). En principio, no existe memoria compartida.

Estado distribuido del sistema.

- "Estado global" está distribuido entre sus componentes; ningún componente tiene conocimiento preciso sobre cuál es el real estado del sistema en un determinado momento.

Inexistencia de tiempo global.

- Múltiples relojes sin sincronización perfecta. Por lo tanto, no existe un patrón de tiempo confiable.

Fallas independientes.

- Pueden existir múltiples puntos de falla, no necesariamente correlacionados. Por lo tanto, fallas pueden ser parciales y perderse parte del estado del sistema.

LAS FALACIAS DE LA COMPUTACIÓN DISTRIBUIDA

- La red es confiable
- La latencia es cero
- El ancho de banda es infinito
- La red es segura
- La topología no cambia
- Existe un único administrador
- El costo de transporte es cero
- La red es homogénea

Fuente: [Peter Deutsch, 1994; James Gosling, 1997]

15

DIFICULTADES PARA SU REALIZACIÓN

Propiedades intrínsecas:

- Tener globalmente un estado compartido.
- Mayor latencia y fiabilidad de la red de comunicaciones.
- Falta de sincronismo perfecto entre relojes y asincronismo.
- Mayor probabilidad de fallas y existencia de fallas parciales.

Escala:

- Gran número de componentes y cambios frecuentes de la topología.
- Incremento de costos de coordinación, con más posibilidades de fallas y mayor latencia.
- Inconsistencia de información o estado por manejo de múltiples copias de datos.

Seguridad:

- Aumento de superficie de ataque.
- Mayores posibilidades de existencia de amenazas y vulnerabilidades (mayores riesgos).

Gestión:

- Dificultad para administrar componentes y servicios (distribuidos).
- Heterogeneidad de componentes y problemas de integración.
- Gestión de configuraciones.
- Respuesta a incidentes y recuperación (sobrecarga, fallas, ataques, etc.)

16

MODELOS FUNDAMENTALES DE SISTEMAS DISTRIBUIDOS

Modelo de sincronismo

- Modelo asincrónico vs. sincrónico.
- Tiempo, relojes, causalidad, concurrencia y ordenamiento de eventos.

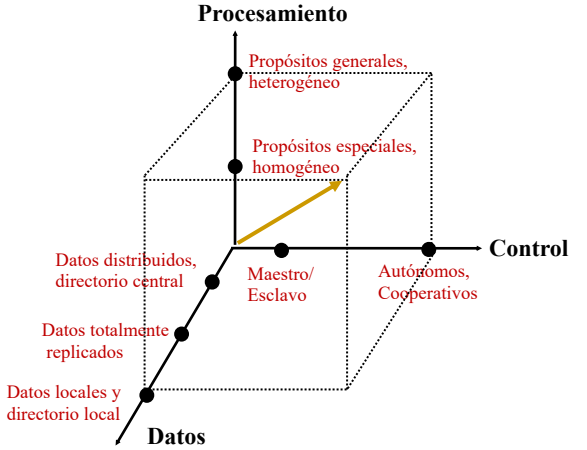
Modelo de fallas

- Tipos de fallas (procesos y canales de comunicación)
- Modelos de fallas de omisión, temporales y fallas arbitrarias (bizantinas).

Modelo de seguridad

- Tipos de amenazas y principios de seguridad.
- Contramedidas (o defensas).

¿QUÉ SE DISTRIBUYE?



FUENTE ORIGINAL: P.H. Enslow, "What is 'Distributed' Data Processing System?", CACM, January 1978

En general se distribuye capacidades de:

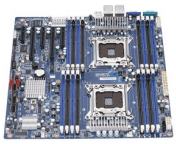
- **Procesamiento** (procesos o funciones).
- **Almacenamiento de datos** (eventualmente con múltiples copias).
- **Control del sistema**. (Descentralización de las decisiones).

OBSERVACIÓN: Esto ocurre a diferentes niveles de abstracción, y de escala o tamaño.

ARQUITECTURAS DE HARDWARE DE COMPUTACIÓN DE ALTO DESEMPEÑO

Sistema de multiprocesamiento con memoria compartida

- Procesadores comparten memoria en una máquina
- Es poco escalable
- Ejemplos: SMP (UMA) y Multicore



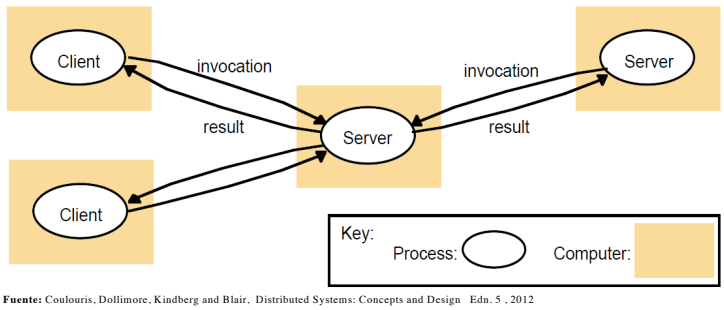
Sistema multicomputador homogéneo

- Procesadores autónomos conectados por red de comunicación de alto desempeño (memoria distribuida)
- Alto desempeño y disponibilidad, paralelismo de computación
- Ejemplos: NUMA; Clusters (Beowulf); Datacenters (Cloud).

Sistema multicomputador heterogéneo

- Similar al anterior, pero heterogéneo y posiblemente más masivo y distribuido
- Ejemplo: Grid Computing y Peer-to-Peer Computing

MODELO CLIENTE-SERVIDOR: DISTRIBUCIÓN DE SERVICIOS



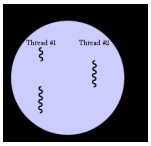
Fuente: Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5 , 2012

Ejemplos:

- Puertos estándares en TCP/IP: FTP (21), SMTP (25), SSH (26), HTTP (80), etc.
- URL para una página Web.

ESTRUCTURAS DE PROCESO

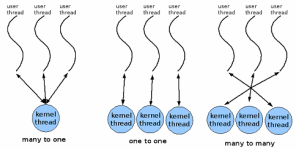
Proceso (o Tarea)



- Instancia de un programa en ejecución, gestionado por el Sistema Operativo, que representa a un "computador virtual".
- Procesos pueden realizar simultáneamente varias tareas (concurrentemente).

Thread (hebra o hilo)

- Representan a un "procesador virtual", una instancia de ejecución (flujo de control).
- Hebras comparten recursos en un mismo proceso (e.g. memoria, código, datos, entre otros), luego se requiere coordinación para mantener la consistencia del proceso.
- Cambio de contexto es mucho más eficiente y pueden explotar arquitecturas *multicore*, lo que mejora el desempeño.
- Existen implementaciones a nivel de usuario y de kernel (o híbrido)
- Ejemplos:
  - Estándar para C y UNIX: IEEE POSIX 1003.1c (Pthread: SO)
  - Threads de Java (lenguaje)



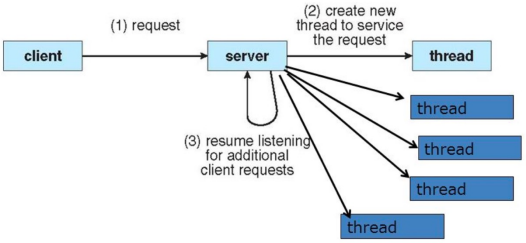
SISTEMAS MULTIHEBRA EN CLIENTE-SERVIDOR

Clientes Multihebra

- Permitir manejar la comunicación remota mientras se desarrollan múltiples actividades locales (sin bloquear al proceso).
- Crear múltiples conexiones (posiblemente a múltiples servidores).

Servidores Multihebra

- Simplifica el desarrollo de software que explota el paralelismo.
- Cada petición de cliente se puede procesar con una hebra independiente (*worker*).
- Hebras se deben sincronizar cuando acceden a recursos compartidos de exclusión mutua.



SISTEMAS OPERATIVOS EN AMBIENTES DISTRIBUIDOS

A) Sistema Operativo de Red (NOS)

- Red con Sistemas Operativos (SO) posiblemente heterogéneo e independientes.
- Acoplamiento débil.
- Sin transparencia de distribución (ubicación), existen múltiples imágenes.
- Ejemplos: UNIX y Windows.

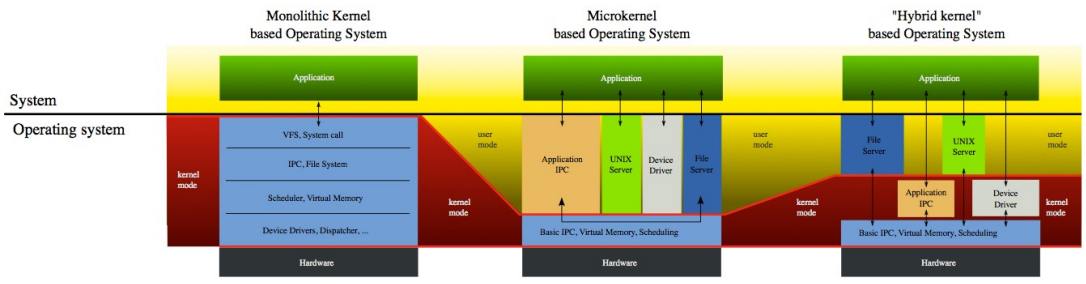
B) Sistema Operativo Distribuido (DOS)

- El SO gestiona muchos computadores, siendo ello transparente para el usuario.
- Basados en tecnología de *microkernel* (con copias iguales en cada máquina) y emulación de SO de red.
- Servicios del SO se pueden distribuir entre muchas máquinas.
- Ejemplos: V-Kernel, Mach, Chorus (todos experimentales en los 80); Datacenters e Infraestructura TIC.

C) Middleware

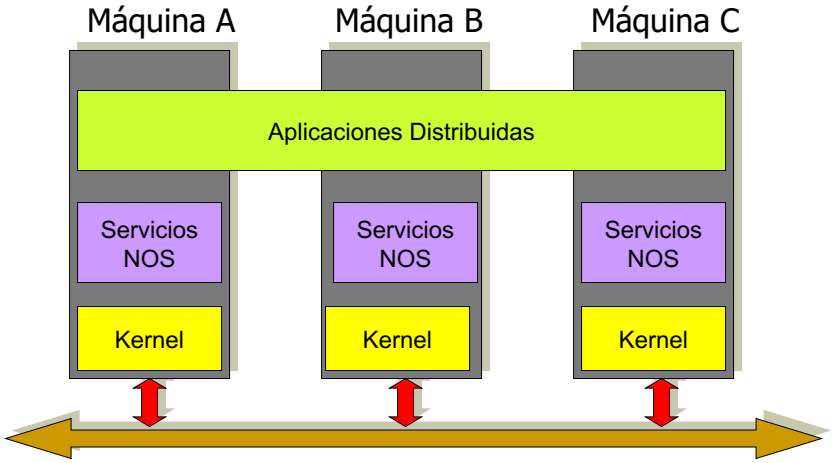
- Basado en NOS, pero con transparencia de distribución para la aplicación.
- Corresponde a estrategia más popular para distribuir aplicaciones.
- Ejemplo: .NET y JEE (década de los 2000).

MICROKERNELS



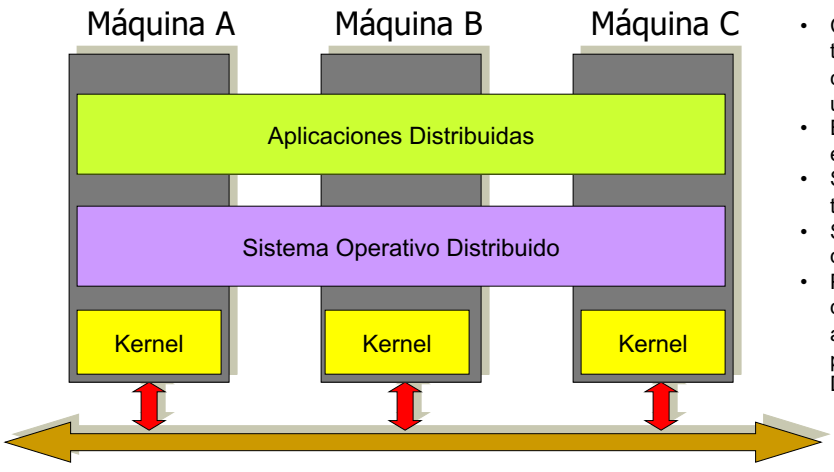
FUENTE: http://en.wikipedia.org/wiki/Distributed_operating_system

A) SISTEMAS OPERATIVOS DE RED (NOS)



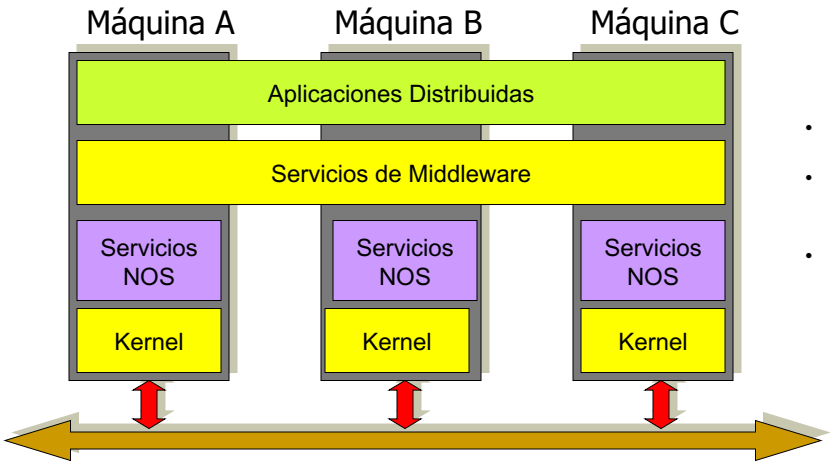
- Servicios de red de los SOs permiten compartir recursos
- La distribución no es trasparente.
- Agrega complejidad a los desarrollodores de software.
- Ejemplos: Unix, Windows NT server, MacOS, etc.

B) SISTEMAS OPERATIVOS DISTRIBUIDOS (DOS)



- Comparte recursos de una red transparentemente y se comporta como un único SO, típicamente para un sistema multicomputador.
- Existe mecanismo de IPC nativo entre máquinas.
- Servicios del SO se acceden con transparencia de ubicación.
- Se dispone políticas y mecanismos de distribución de carga.
- Por su gran complejidad, no están disponibles para desarrollar aplicaciones distribuidas de propósitos generales (si interés para Datacenters, Blockchain, IoT).

C) MIDDLEWARE (MW)



- Cada nodo ejecuta un SO de red (NOS) estándar.
- Una capa adicional de software abstrae de la complejidad de los SO de red interconectados (NOS).
- Programadores disponen de API para tener transparencia de acceso y facilitar el desarrollo de software distribuido.

DEFINICIÓN DE MIDDLEWARE

DEFINICIÓN: "En un sistema (de computación) distribuido, es una capa de software que se encuentra entre el Sistema Operativo y las Aplicaciones en cada sitio del sistema"

[Sacha Krakowiak, 2003]

Características:

- Permite conectar componentes de software, o a personas y sus aplicaciones.
- Consiste en conjunto de servicios que permite interacción de múltiples procesos, ejecutándose en una o más máquinas.
- Provee interoperabilidad basada en una arquitectura distribuida coherente, que generalmente apoya a simplificar el desarrollo de aplicaciones distribuidas complejas.

EJEMPLO: TIPOS DE MIDDLEWARE

- **Middleware de RPC (e.g. ONC/RPC, OSF DCE)**
- **Middleware de Base de Datos (e.g. ODBC, JDBC)**
- **Middleware de Objetos (e.g. Corba)**
- **Middleware orientado a Mensajes (e.g. MSMQ, MQSeries)**
- **Middleware Transaccional (e.g. Tuxedo)**
- **Middleware orientado a Servicios (e.g. Web services y Microservicios)**

VIRTUALIZACIÓN

DEFINICIÓN: Crear una versión virtual de un recurso, tales como un computador (máquina), dispositivos de almacenamiento y redes de computadores. Estrategia original en *Cloud Computing* para compartir recursos y proveer servicios.

Tipos virtualización de máquinas:

- Completo. Sistema que crea máquinas virtuales (VM) instaladas directamente sobre el hardware (e.g. *Hypervisor* o *VM manager*). Cada VM puede cargar un SO diferente.
- Paravirtualizado. Se ejecuta como proceso sobre un SO normal (e.g. VirtualBox). Corresponde a primera aproximación a virtualización de máquinas.

Incorporan mecanismos de *snapshot*, migración y *failover* para tolerar fallos y distribuir carga.

Otra forma de virtualización es a nivel de Desktop (VDI) y de Redes (SDN).

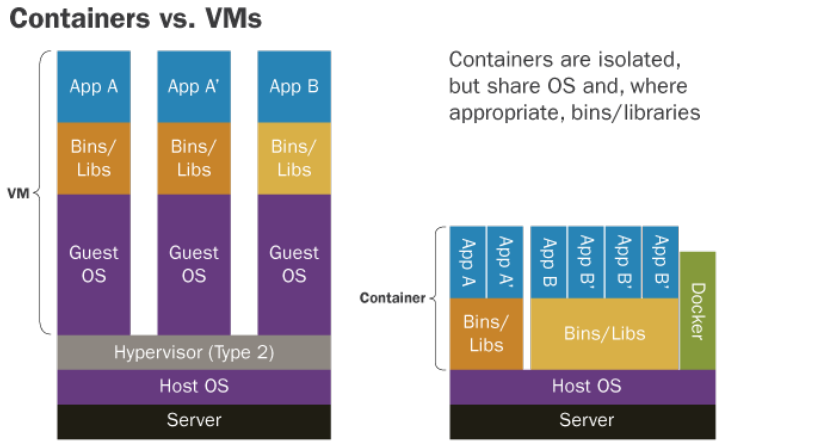
CONTENEDORES

Permite agrupar uno o más procesos, que se ejecutan en un mismo sistema operativo. Sin embargo, se puede limitar los recursos que ven de una máquina.

Es más liviano que tener una máquina virtual, haciendo un uso más eficiente de los recursos de un Datacenter.

Fuente: M. J. Scheepers, "*Virtualization and Containerization of Application Infrastructure: A Comparison*", University of Twente, 2014.

MÁQUINAS VIRTUALES VS. CONTENEDORES

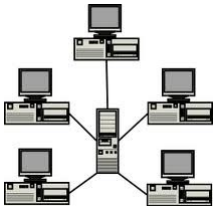


Fuente: https://www.microcontrollertips.com/containerization-differs-virtual-machines-faq/

C/S VS. P2P

Cliente/Servidor

- Estructura simple y muy popular.
- Facilita distribuir roles y responsabilidades.
- Servidor controla acceso a recursos o servicios.



Peer-to-Peer (P2P)

- Procesos tienen roles y privilegios similares.
- Cooperación sin coordinación central (control distribuido).
- Logra mejores tiempos de respuesta y puede ser más robusto.

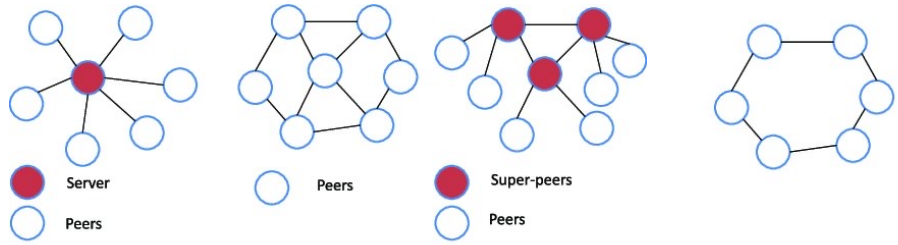
MIDDLEWARE PARA PEER-TO-PEER (P2P)

DEFINICIÓN: Es una arquitectura que distribuye las tareas en múltiples nodos llamados pares (*peers*), siendo todos iguales, es decir tienen la misma función. Definen redes de *overlay* (superpuestas).

Clasificación:

- No estructurada
  - Simples y robustas, pero pueden ser ineficientes en la búsqueda
  - No existe una autoridad centralizada,
  - Ejemplos: Gnutella (2001), Blockchain (2008), etc.
- Estructurada
  - Topología específica (anillo, hipercubo, jerárquico)
  - Las más comunes son con DHT (Distributed Hash Table), como P2P BitTorrent lo usa para su buscador o Tracker (2005).

TIPOS DE ARQUITECTURA DE REDES P2P



(a) P2P Centralizada. (b) P2P Pura. (c) P2P Híbrida (d) P2P Estructurada.

Fuente: T. Yeferny et al., "Query Learning-Based Scheme for Pertinent Resource Lookup in Mobile P2P Networks", *IEEE Access*, April 2019.

35

PRINCIPIOS, TÉCNICAS Y DESAFÍOS DE DISEÑO; MODELOS DE INTERACCIÓN.

1.3 ESTRATEGIAS Y TÉCNICAS DISEÑO DE SISTEMAS DISTRIBUIDOS

36

SERVICIOS COMUNES Y COMPARTIDOS EN SISTEMAS DISTRIBUIDOS

- Procesamiento distribuido
- Almacenamiento distribuido de información o datos
- Comunicación entre procesos e invocaciones remotas
- Identificación de componentes (nombres)
- Búsqueda y localización de componentes (descubrimiento)
- Seguridad y alta disponibilidad
- Tiempo y sincronización de relojes
- Administración y monitoreo de componentes

TÉCNICAS ÚTILES PARA CONSTRUIR SISTEMAS DISTRIBUIDOS

- Replicar para mejorar disponibilidad y desempeño
- Buscar compromiso entre disponibilidad y consistencia.
- Explotar localidad usando *caching.*
- Usar invocaciones remotas para facilitar desarrollo de sistemas.
- Usar criptografía para autenticar y proteger datos.
- Usar *timeout* para revocar uso de recurso asignados.
- Código móvil al cliente para mejorar tiempo de reacción (*responsiveness*).
- Código móvil para reducir costos administrativos de mantención de software.

DESAFÍOS DE DISEÑO

39

DESAFÍOS Y "METAS IDEALES" EN EL DISEÑO DE SISTEMAS DISTRIBUIDOS

Aspectos de calidad de servicio (QoS):

- Desempeño: Mejor tiempo de respuesta y productividad (*throughput*).
- Robustez: Mayor disponibilidad con consistencia funcional y de datos (aún en presencia de fallas; requiere manejo automático de fallas).
- Seguridad: Control de acceso, privacidad, disponibilidad y movilidad de código.

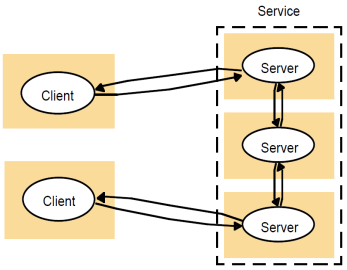
Propiedades arquitectónicas:

- Concurrencia: Controlarla para garantizar alto desempeño y consistencia.
- Escalabilidad: Tamaño, distancia y gestión.
- Transparencia: Acceso, ubicación, migración, replicación, fallas, etc.
- Heterogeneidad: Facilitar acceso y interoperabilidad entre componentes en ambientes heterogéneos.
- Apertura (*Openness*): Facilidad de extensión y reimplementación. Separación de la política del mecanismo. Desarrollo de estándares.

Lectura complementaria: [Coulouris et al.]: sección 1.5

40

DESEMPEÑO: ARQUITECTURA DE MÚLTIPLES SERVIDORES



Características:

- Un servicio lo implementan varios servidores.
- Permite distribuir carga.
- Se le denomina a veces distribución horizontal.
- Ejemplo: *Clustering*.

Estrategias:

- Partición (requiere función de distribución).
- Replicación (requiere coordinación, si existen datos replicados).

Ejemplos:

- La Web particiona los datos (páginas) en millones de servidores distribuidos por el planeta.
- Replicación para alto desempeño y alta disponibilidad en sistemas de directorio y base de datos.

ROBUSTEZ: MANEJO DE FALLAS

Modelo de fallas:

- Fallas en SD normalmente ocurren parcialmente.
- Conceptos: averías o fallas (*failures*), errores (*errors*) y fallos (*faults*)
- Tipos de fallas: fail-stop, omisión, bizantina, etc.

Principios para manejar fallas:

- Redundancia: temporal (repetición) o espacial (réplicas)
- Atomicidad: operaciones se organizan en unidades atómicas.

Técnicas de manejo de fallas:

- Detección de fallas
- Enmascaramiento de fallas
- Tolerancia a fallos
- Recuperación de errores

Anuncio: Se profundizará en capítulo 3 (fundamentos teóricos) y 5 (Confiabilidad toelenncia a fallos).

Lectura complementaria: [van Steen et al.]: sección 1.5.5

SEGURIDAD

Principios básicos (CIA):

- Confidencialidad (C)
- Integridad (I)
- Disponibilidad (A)

Servicios típicos:

- Identificación & autenticación
- Autorización y auditoría
- Cifrado de datos
- Control de integridad
- No repudio
- Firma digital

Posibles amenazas:

- Intrusión deliberada, sin autorización.
- Espionaje industrial y robo de propiedad intelectual.
- Vandalismo, sabotaje y terrorismo.
- Fraude financiero.
- Ingeniería social.
- Robo de información transaccional e identidad digital.
- Denegación de servicio.

ESCALABILIDAD

DEFINICIÓN: Capacidad de un sistema para manejar el crecimiento de trabajo; o potencial de ampliar el sistema para acomodar este crecimiento.

Dimensiones del escalamiento:

- Tamaño: Permite agregar más usuarios y recursos al sistema, sin impactar en su desempeño.
- Geográfico: Usuarios y recursos se pueden distribuir a mayor distancia, sin impactar en su desempeño (e.g. retardo no se nota).
- Administrativo. Aun cuando el sistema se extienda a un mayor número de organizaciones o usuarios, sigue siendo fácil administrarlo.

Fuente. [van Steen, et al.]: Section 1.2 Design Goals: "Being scalable"

TÉCNICAS DE ESCALAMIENTO

Ocultamiento de latencia de comunicación: Introducir comunicación asincrónica y/o mover funcionalidad cerca del usuario (e.g. código).

- Ejemplo: Procesamiento *batch* de mensajes con recepción asincrónica de respuestas; o JavaScript (código móvil).

Partición & distribución: Particionar datos y/o funciones entre nodos, distribuyendo la carga.

- Ejemplo: DNS, WWW; Applets y JavaScript en el navegador

Caching: Llevar copias de datos cerca del usuario.

- Ejemplo: Páginas web y archivos; Edge computing

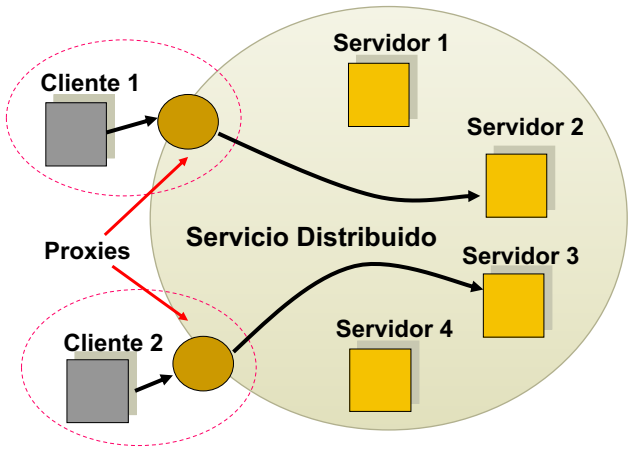
Replicación: Hacer varias copias de datos y/o funciones.

- Ejemplo: DFS, BDD, Mirrors; balanceador de carga.

Se debe manejar problema de consistencia.

Fuente. [van Steen, et al.]: Section 1.2 Design Goals: "Being scalable"

PRINCIPIO PROXY



Funciones típicas:

- Caching
- Autenticación
- Control de acceso (autorización)
- Validación local de datos
- Funcionalidad local

Ejemplos:

- Stubs y API´s
- Servidores Proxy en la Web
- CDN (*Content Delivery Network*)

Fuente: Marc Shapiro, "Structure and encapsulation in distributed systems: the proxy principle", in *Proc. 6th IEEE Int. Conf. on Distributed Computing Systems*, pp. 198-204, 1986.

TRANSPARENCIA

Fuente: ISO/IEC 10746: RM-ODP

- **Acceso.** Oculta diferencias en representación de datos y método de acceso a recursos u objetos remotos.
- **Ubicación**. Oculta dónde están localizados físicamente los objetos al accederlos.
- **Reubicación**. Oculta que un objeto pueda ser movido a otra ubicación, mientras está siendo usado y sin afectar la operación.
- **Migración**. Oculta que un objeto se pueda mover a otra ubicación.
- **Replicación**. Oculta que un objeto esté replicado para aumentar la disponibilidad (confiabilidad) o el desempeño (balance de carga).
- **Concurrencia**. Oculta que un objeto pueda ser compartido entre varios usuarios independientes que compiten por su uso (sin interferencia).
- **Fallas**. Oculta la ocurrencia de una falla y la recuperación de un objeto, para seguir operando y completar tareas.
- **Persistencia**. Oculta la desactivación y reactivación del objeto (para liberar recursos).
- etc.

Lectura: [Coulouris et al.]: sección 1.5.7

HETEROGENEIDAD

Fuentes de heterogeneidad:

- Computadores, sistemas operativos, lenguajes de programación, protocolos de comunicación, diferentes proveedores y vendedores

Algunas estrategias:

- Middleware (e.g. Corba, DCOM y RMI; SOA)
- Máquinas virtuales y código móvil (e.g. JVM)

Preocupaciones mayores:

- Extensiones, mejoras e integración
- Estandarización de interfaces y componentes

MODELOS DE INTERACCIÓN

50

ENTIDADES DE INTERACCIÓN

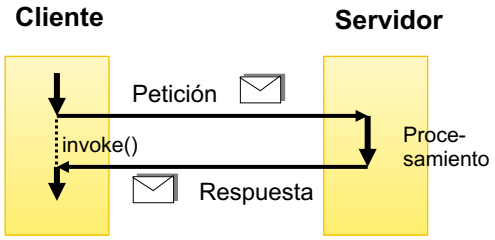
Los componentes arquitectónicas que interactúan pueden corresponder a diferentes tipos de entidades, como por ejemplo:

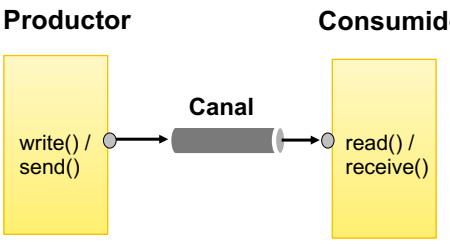
- **Máquinas o nodos**. Comunicación ocurre a un nivel más primitivo que IPC (por ejemplo en capa de red o física en ISO/OSI).
- **Procesos**. Sistema consiste de un conjunto de procesos que interactúan usando mecanismo de IPC (*interprocess communication*).
- **Objetos**. Sistema es conjunto de objetos interactuantes, usando una interfaz estándar (IDL) para invocar remotamente sus métodos.
- **Componentes**. Evolución de modelo anterior, que despliegue componentes en contenedores con soporte para controlar ciclo de vida, acceso, persistencia, etc.
- **Servicios (Web)**. Relacionado con objetos y/o componentes, típicamente integrados a Internet con tecnología Web.

51

ESTILOS DE INTERACCIÓN DIRECTA

Interacciones sincrónicas entre entidades conocidas





a) Paso de mensaje (directo)

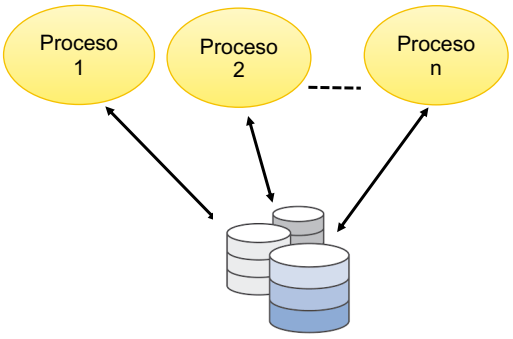
- Comunicación directa por paso de mensajes
- No existe memoria o estado compartido.
- Servidor controla sus recursos.
- Existen diversos patrones de interacción como tipo *request-reply* (figura), y notificación o señal (modelo simple: emisor y receptor).

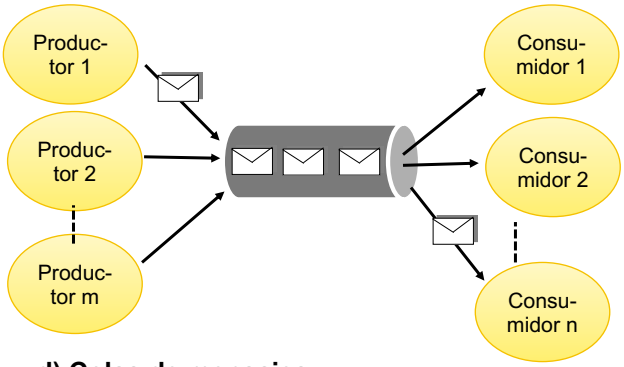
b) Data streaming (o flujo de datos)

- Comunicación directa por un flujo continuo de unidades de datos.
- Usualmente se controla el flujo, a veces con restricciones temporales.
- Típicamente asociado a aplicaciones multimediales; tradicionalmente transferencia de archivos.

ESTILOS DE INTERACCIÓN INDIRECTA

Interacciones asincrónicas entre entidades anónimas





c) Repositorio de datos compartidos

- Comunicación indirecta a través de datos compartidos (con dirección conocida).
- Requiere controlar el acceso concurrente para garantizar consistencia.
- Ejemplos: DSM, Archivos, BDs, Espacio de tuplas.

d) Colas de mensajes

- Comunicación indirecta a través de entidad pasiva intermedia (e.g. buzón, grupo, cola o bus de mensajes).
- Consumo de mensajes puede ser asíncrono o por *polling*.
- Ejemplos: Multicasting, Message Queue, Publish-Suscribe, Eventos.

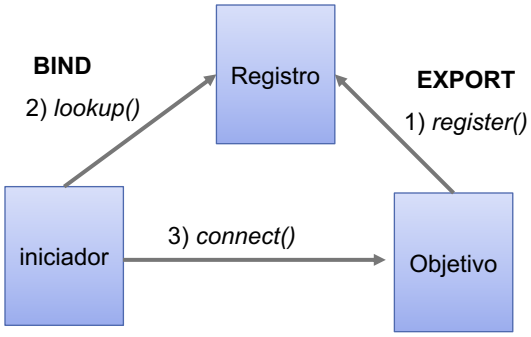
DIRECCIONAMIENTO DE ENTIDADES

Entidades activas interactuantes requieren, según estilo:

- Interacción directa: Iniciador debe conocer dirección de entidad objetivo.
- Interacción indirecta: Entidades activas sólo debe conocer dirección de una entidad pasiva intermedia.

El conocimiento sobre la dirección de la **entidad objetivo o intermedia** puede ser:

- Estático: Se conoce con anterioridad a la interacción (e.g. puerto 80 para HTTP).
- Dinámico: Se debe descubrir cuando se inicia la interacción, usando un nombre preestablecido (soporta configuraciones dinámicas del sistema).

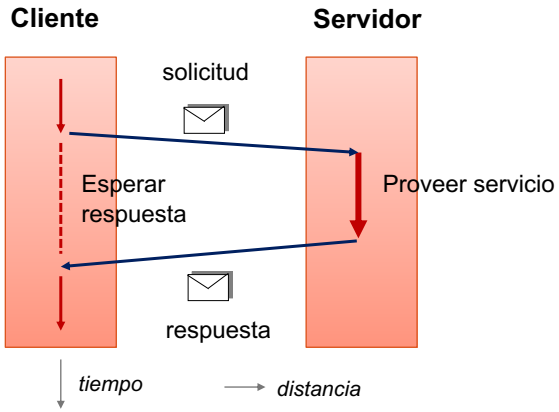


Patrón típico para resolución dinámica de dirección y descubrimiento de servicio.

MODELO CLIENTE-SERVIDOR

Características:

- Servidor es entidad que ofrece un servicio.
- Cliente es otra entidad que solicita el servicio.
- La interacción se basa en un protocolo tipo *request-reply* (solicitud-respuesta).
- Ambas entidades pueden estar distribuidas.
- Este modelo básico se extiende a invocación remota de procedimientos, métodos (objetos) o servicios.



VARIACIONES AL MODELO C-S

Código y agentes móviles

- Código migra de una máquina a otra.
- Ejemplos: Applets, JavaScript y agentes móviles.

Clientes delgados

- Recursos mínimos para manejar interfaz con usuario (presentación).
- Fácil de administrar, pero carga la red y los servidores
- Ejemplos: Network Computers, Estaciones X

Servidores proxies

- Permite reducir tráfico y mejorar tiempos de respuesta.
- Ejemplos: Web caching y file caching.

Dispositivos terminales móviles

- Conexión temporal, posiblemente con bajo BW e inestable.
- Ejemplos: portables, tablets, smartphones, drones, vehículos, etc.

ESTILOS, PATRONES, COMPUTACIÓN DISTRIBUIDA

1.4 ARQUITECTURAS EN SISTEMAS DISTRIBUIDOS

ARQUITECTURA (DE SOFTWARE)

DEFINICIÓN: Corresponde a la organización y estructura fundamental de un sistema, que incluye:

- Componentes como bloques de construcción.
- Relaciones entre componentes (conectores e interfaces) y con el entorno.
- Principios que guían el diseño (reglas y restricciones para componentes y conectores) y su evolución.

ESTILO ARQUITECTÓNICO

DEFINICIÓN: Es un marco de trabajo que define:

- Patrón de organización estructural que comparte una familia de sistemas.
- Componentes y conectores que pueden ser usados en instancias del estilo.
- Conjunto de restricciones de cómo combinar componentes y conectores (topológicas, de semántica de ejecución, etc.).

Permite identificar especializaciones comunes, y evaluar ventajas y desventajas.

ESTILOS Y PATRONES EN SISTEMAS DISTRIBUIDOS

Estilos arquitectónicos

- Pipes & Filters
- Abstracción de datos y objetos
- Multinivel (*multi-layer*)
- Multicapa (*multi-tier*)
- Basada en eventos
- Orientada a servicios o a microservicios

Patrones arquitectónicos y/o estructurales

- Proxy
- Broker y Bridge
- Fachada
- Adaptador o wrapper
- Model-View-Controller (MVC)
- Registro
- Interceptor
- Object-Relational Mapping (ORM)
- etc.

Observación: existen muchos enfoques diferentes y distintas clasificaciones. En esta asignatura solo veremos algunos estilos y patrones.

60

ESTILOS ARQUITECTÓNICOS DE SISTEMAS DISTRIBUIDOS

Estilo Arquitectónico	Descripción
Arquitectura multinivel (*multi-layered*)	Particiona problemas de la aplicaciones en grupos apilados (*layers*), que representan niveles de abstracción (desde hardware hasta aplicación).
Cliente-Servidor	Separa la funcionalidad del sistema en dos aplicaciones, donde el cliente realiza peticiones al servidor.
Arquitectura multicapa (*multi-tier*)	Separa la funcionalidad en múltiples segmentos (*tiers*), que pueden ser localizados en computadores físicamente separados.
Arquitectura orientada a objetos	Divide responsabilidades del sistema en unidades reusables y autosuficientes denominadas objetos, que contienen datos y comportamiento relevante.
Arquitectura basada en componentes	Descompone el diseño de la aplicación en componentes reusables que exponen interfaces de comunicación bien definidas.
Bus de mensajes	Interacción entre aplicaciones es por paso de mensajes por uno o más canales, sin necesidad de conocer detalles específicos de las aplicaciones.
Arquitectura orientada a servicios (SOA: *Service oriented Arch.*)	Aplicaciones exponen y consumen funcionalidades como un servicio, usando contratos y mensajes (usando un lenguaje de descripción de las interfaces).

FUENTE: "*Microsoft Application Architecture Guide*", 2nd Edition, 2008, msdn.microsoft.com

61

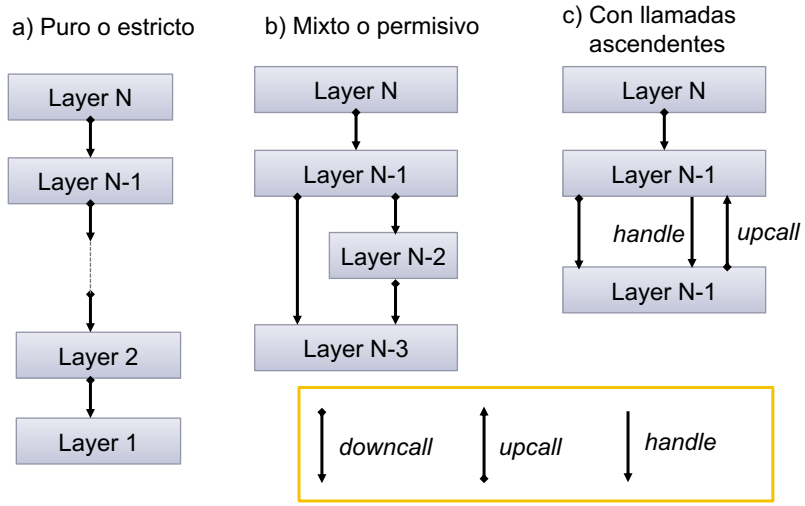
CAPAS DE ABSTRACCIÓN

ARQUITECTURAS MULTI-NIVEL (MULTI-LAYERED)

62

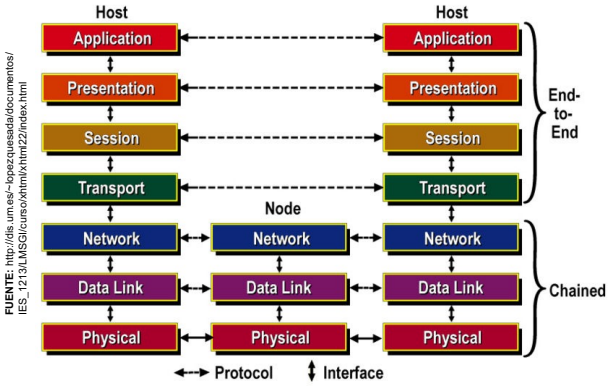
ARQUITECTURA MULTI-NIVEL (MULTI-LAYERED)

- Permite organizar sistemas complejos.
- Arquitectura se organiza por capas o niveles de abstracción.
- Cada nivel usa servicios de niveles inferiores.
- Es posible registrar una función superior (handle) para recibir llamadas de niveles inferiores.



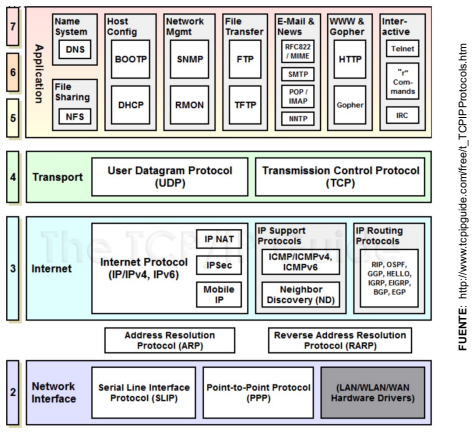
63

EJEMPLOS: ARQUITECTURA DE REDES DE COMUNICACIÓN



FUENTE: http://dis.um.es/~lopezquesada/documentos/IES_1213/LMSGI/curso/xhtml/xhtml22/index.html

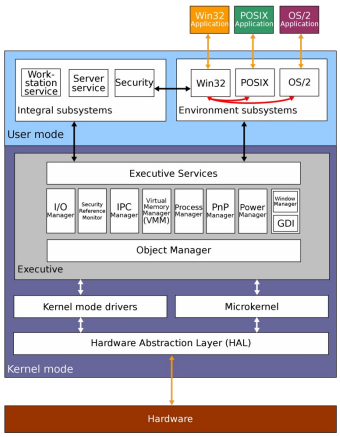
a) Modelo ISO/OSI



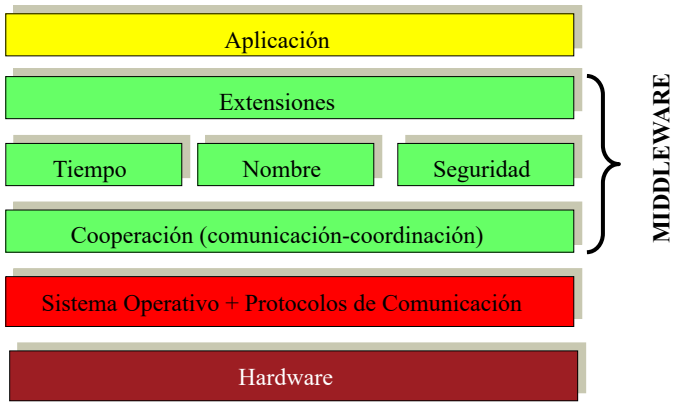
FUENTE: http://www.tcpipguide.com/free/t_TCPIPProtocols.htm

b) Modelo TCP/IP

EJEMPLOS: ARQUITECTURA MULTI-NIVEL

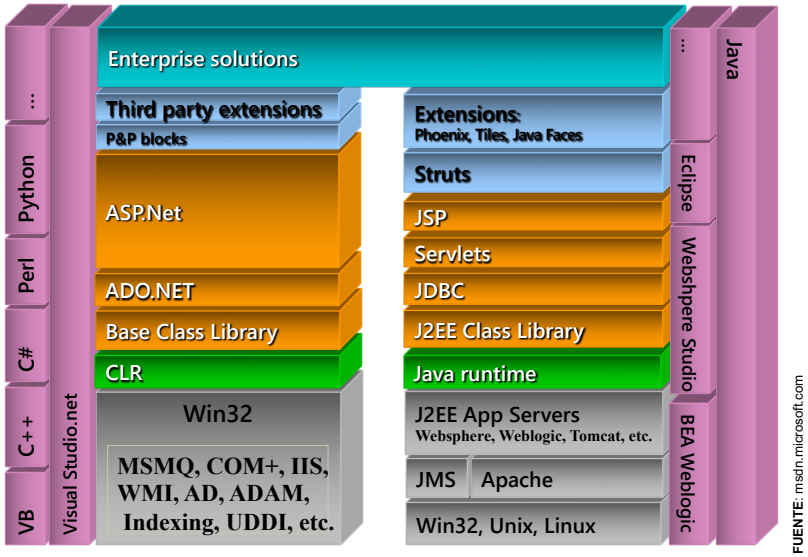


a) Arquitectura de Windows NT



a) Arquitectura de Middleware

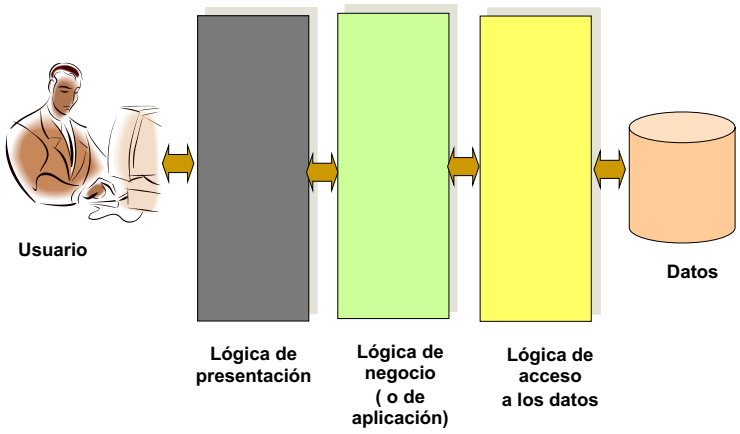
EJEMPLOS: TECNOLOGÍA .NET VS. J2EE



FUENTE: msdn.microsoft.com

ARQUITECTURAS MULTICAPA (MULTI-TIERED)

ESTILO DE ARQUITECTURA MULTICAPA (N-TIER)



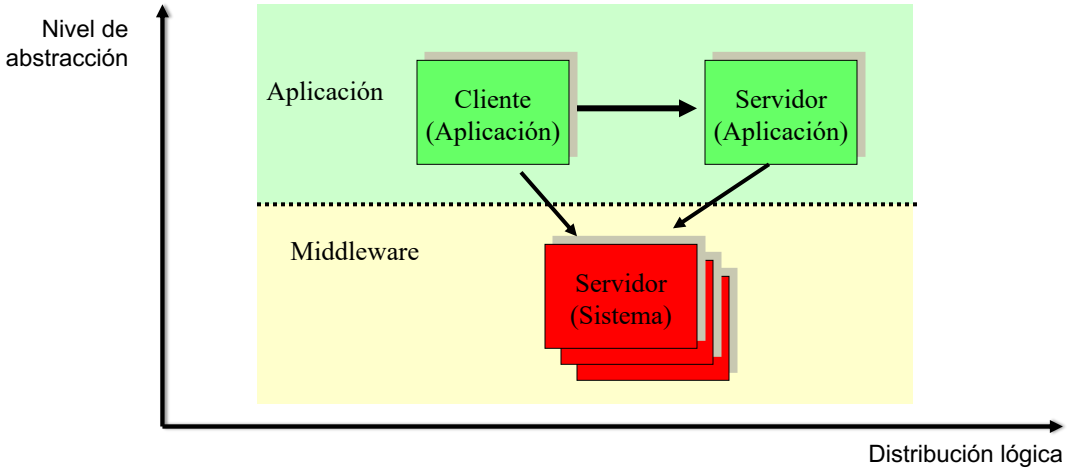
Características:

- La aplicación se segmenta en varias capas lógicas (para su distribución física).
- Normalmente estamos a un mismo nivel de abstracción (*layer*).

Observación:

- Algunos autores usan "multi-tiered architecture" y otros "multi-layered architecture" para referirse a lo mismo.
- ¡¡No confundirse con niveles o capas de abstracción!!

NIVEL DE ABSTRACCIÓN VS. DISTRIBUCIÓN FUNCIONAL



ARQUITECTURA MULTICAPA (MULTI-TIERED)

DEFINICIÓN: Sistemas se distribuyen físicamente en varios sitios, particionando su lógica en 2 o más capas (*tiers*).

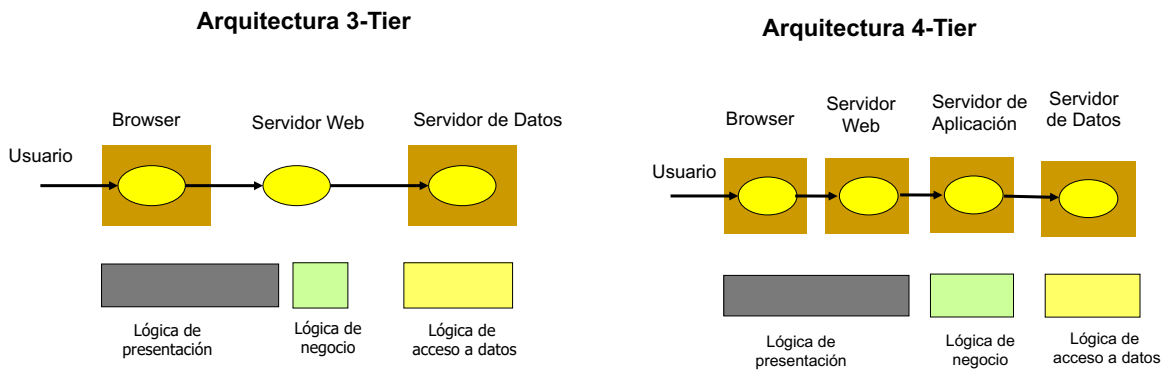
- Número de capas se define según requerimientos del sistema o aplicación.
- Un punto de vista es ser una distribución vertical a un mismo nivel de abstracción (*layer*).

Ejemplos:

- Aplicaciones C/S tradicionales son 2-tier
- Aplicaciones Web simples tienden a ser 3-tier (implementadas sobre nivel 4, de transporte de datos).

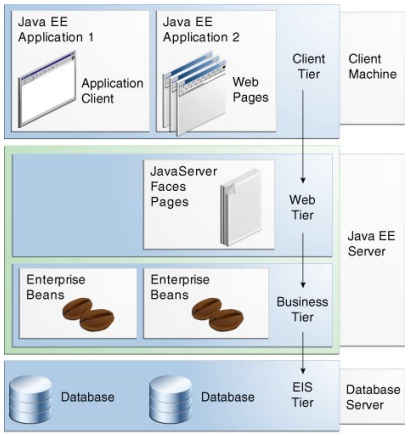
70

EJEMPLO: APLICACIÓN WEB CON SERVIDOR DE APLICACIÓN

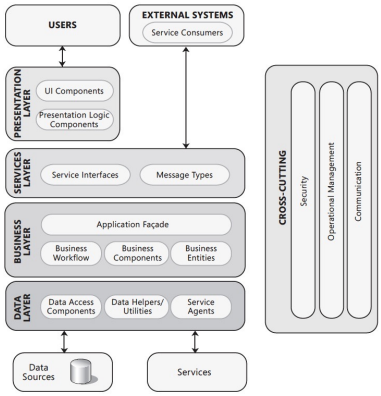


71

EJEMPLO: TECNOLOGÍAS MULTI-TIER



Fuente: www.oracle.com/technetwork/java/javaee

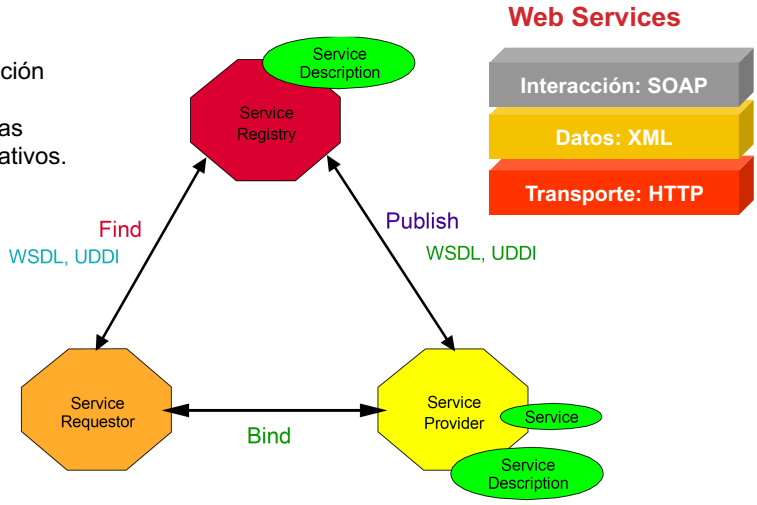


Fuente: Microsoft, "Microsoft Application Architecture Guide", 2009

ARQUITECTURAS ORIENTADAS A SERVICIOS

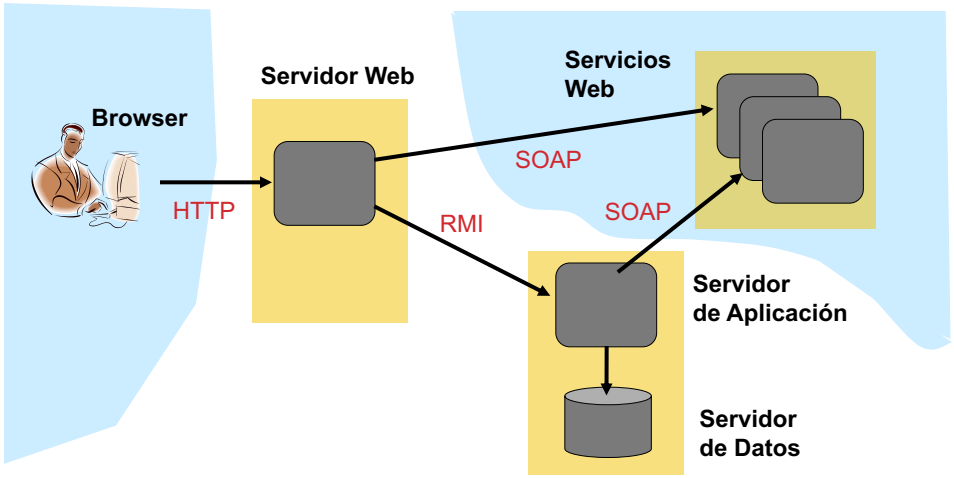
SOA: ARQUITECTURA ORIENTADA A SERVICIOS

- Sistema se construye como composición de servicios.
- Se pueden consumir servicios de otras organizaciones o dominios administrativos.



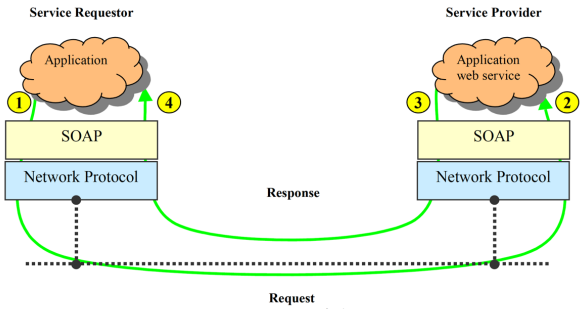
74

EJEMPLO: WEB SERVICES



75

MENSAJES SOAP



REQUEST:

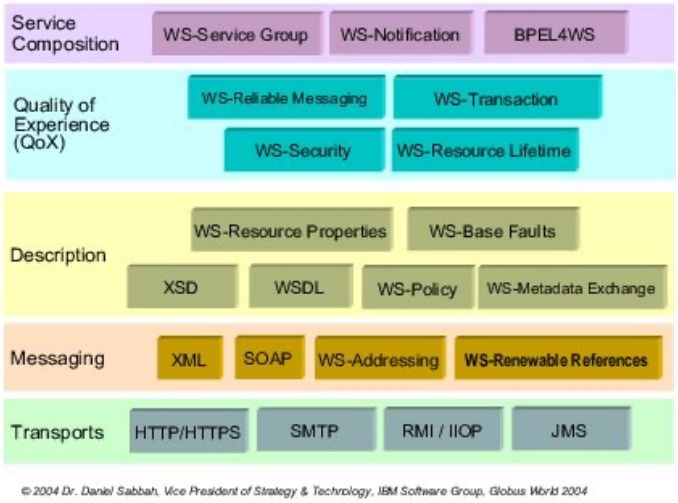
```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>
```

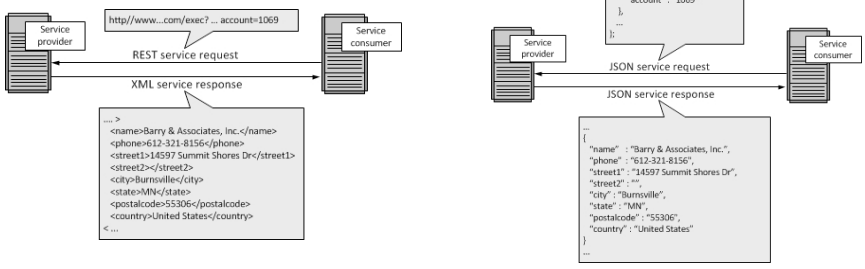
Fuente: https://www.w3schools.com/xml/xml_soap.asp

STACK DE ESTÁNDARES DE WEB SERVICE (W3C + OASIS)



© 2004 Dr. Daniel Sabbah, Vice President of Strategy & Technology, IBM Software Group, Globus World 2004

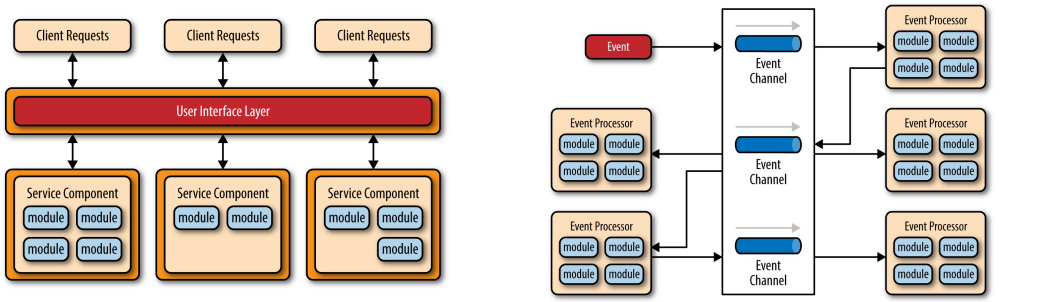
SERVICIOS WEB: TECNOLOGÍAS ALTERNATIVAS A SOAP



a) Representation State Transfer (REST)

b) JavaScript Object Notation (JSON)

ARQUITECTURAS DE MIDDLEWARE CON TOPOLOGÍA DE BROKER O BUS



a) Arquitectura de microservicios

b) Arquitectura basada en eventos

Fuente: Mark Richards, "*Software Architecture Patterns*", O'Reilly Media, Inc., 2015.

ARQUITECTURAS DE COMPUTACIÓN DISTRIBUIDA

80

CLOUD COMPUTING

DEFINICIÓN: "A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"

Fuente: "*The NIST Definition of Cloud Computing*", NIST SP 800-145

ATRIBUTOS CLAVES:

- Auto-servicio por demanda (consumidor se aprovisiona)
- Acceso de banda ancha (capacidades disponibles por la red)
- Recursos compartidos (entre múltiples consumidores)
- Elasticidad rápida (capacidades aprovisionadas elásticamente)
- Servicios medidos

81

MODELOS DE CLOUD

MODELO DE SERVICIO

- **IaaS (Infraestructura)**
  - Máquinas y redes virtuales
- **PaaS (Plataforma)**
  - Herramientas de desarrollo e Integración contínua
- **SaaS (Software)**
  - Aplicaciones

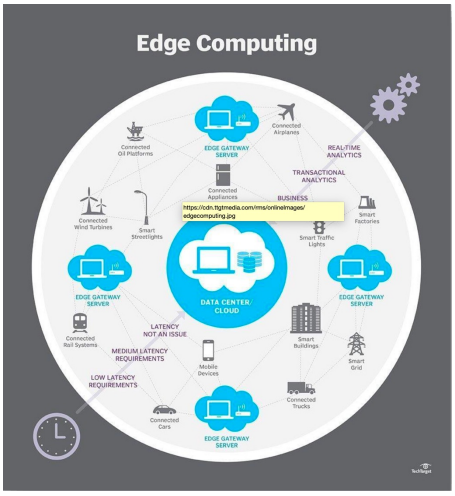
MODELO DE DESPLIEGUE

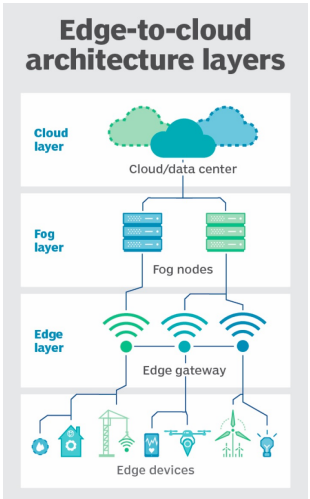
- **Cloud Privado**
- **Cloud Comunitario**
- **Cloud Público**
- **Cloud Híbrido**

82

Fuente: https://www.techtarget.com/searchdatacenter/pro/What-is-Edge-Computing-Everything-You-Need-to-Know

CASO: EDGE & FOG COMPUTING





83

SINOPSIS DEL CURSO

- Paradigmas de programación distribuida y comunicación en sistemas de software distribuido (Cap. 2)
- Fundamentos de computación distribuida y algoritmos distribuidos básicos (Cap. 3)
- Coordinación y algoritmos distribuidos básicos (Cap. 4)
- Confiabilidad y tolerancia a fallos (Cap. 5)
- Distribución de datos, transacciones y replicación de datos (Cap. 6)
- Seguridad en sistemas distribuidos (Cap. 7)*

CAPÍTULO I: INTRODUCCIÓN A LOS SISTEMAS DISTRIBUIDOS

DR.-ING. RAÚL MONGE ANWANDTER
DEPARTAMENTO DE INFORMÁTICA
UTFSM, VALPARAÍSO - CHILE