Actividad 2: Arquitectura Operacional del Sw.

1. Diseña un esquema simple correspondiente a cada una de las arquitecturas estudiadas (centralizada, y distribuidas). Indica en el esquema el hardware implicado y el software que se ejecuta en cada equipo.

Arquitectura centralizada:

[Terminal/es tontos] --(red ligera)--> [Servidor central / Mainframe]

Terminal tonto / thin client

- Hardware: pantalla, teclado, NIC; CPU/RAM mínimas.
- Software: cliente ligero (X/Remote Desktop/VT100), navegador fino o app de terminal.

Servidor central (mainframe/servidor potente)

- Hardware: CPU multinúcleo/CPU mainframe, mucha RAM, almacenamiento RAID/SAN, NICs redundantes.
- Software: SO de servidor (z/OS, UNIX/Linux, Windows Server), DBMS, aplicaciones de negocio, servicios de autenticación, monitor de transacciones.

Arquitectura distribuida:

1) Cliente-Servidor (2 capas)

[Clientes] <----> [Servidor de Aplicación+BD]

Cliente

- Hardware: PC/portátil/tablet con NIC/Wi-Fi.
- Software: SO de escritorio/móvil, app cliente o GUI, drivers, middleware (ODBC/JDBC si aplica).

Servidor (app + base de datos en la misma máquina)

- Hardware: servidor x86_64, RAM/SSD, RAID, NIC.

- Software: SO de servidor, runtime (JRE/.NET/Node), servidor de apps ligero, DBMS (PostgreSQL/MySQL/SQL Server), servicio API.

2) Tres capas (presentación, lógica y datos)

[Clientes (web/app)] <--> [Servidor de Aplicaciones/API] <--> [Servidor de BD]

Cliente (presentación)

- Hardware: PC/móvil.
- Software: Navegador o app móvil/escritorio (HTML/JS/CSS).

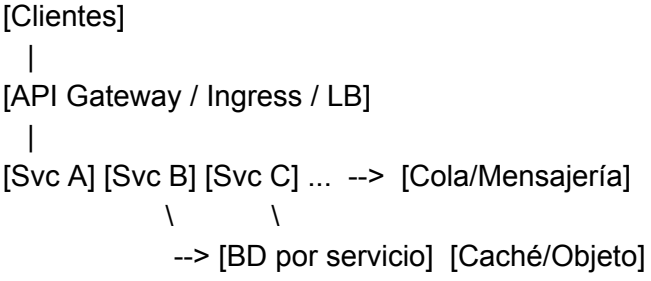
Servidor de Aplicaciones (capa lógica)

- Hardware: 1+ servidores; balanceador opcional.
- Software: SO de servidor, runtime (Java/.NET/Node/Python), contenedor (Tomcat, Kestrel, Express, Gunicorn), servicios REST/GraphQL, autenticación, caché local.

Servidor de Base de Datos (datos)

- Hardware: servidor con almacenamiento rápido (RAID/NVMe), posible réplica.
- Software: DBMS (PostgreSQL/MySQL/Oracle/SQL Server), motor de replicación/backup, herramientas de monitorización.

3) Microservicios (distribución fina + contenedores)



Cliente

- Hardware: PC/móvil.
- Software: App web/móvil.

Capa de entrada

- Hardware: balanceador/reverso (físico/virtual).
- Software: NGINX/Envoy/API Gateway.

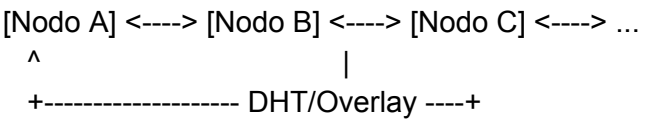
Nodos de microservicios

- Hardware: clúster de servidores o nube.
- Software: SO Linux, **contenedores** (Docker/CRI-O), **orquestador** (Kubernetes), cada servicio con su runtime (Go/Java/Node/Python), observabilidad (Prometheus/Grafana), CI/CD agent.

Datos y mensajería

- Hardware: nodos de BD/caché/cola dedicados.
- Software: BDs por servicio (PostgreSQL/Mongo/Redis), mensajería (Kafka/RabbitMQ), almacenamiento de objetos (S3/minio).

4) P2P (entre pares)



Cada nodo

- Hardware: PC/servidor con NIC.
- Software: SO general, **cliente P2P** (BitTorrent/IPFS/etc.), tabla distribuida (DHT), cifrado, NAT traversal (STUN/TURN).

2. En la actualidad, el desarrollo de aplicaciones web prácticamente monopoliza el desarrollo de aplicaciones, quedando el desarrollo “en ventana” relegado a un segundo plano. Indica, a tu juicio, algunas razones que justifiquen este hecho.

Creo que se debe a la accesibilidad de la web desde cualquier dispositivo.

3. Indica las ventajas e inconvenientes que plantea la arquitectura cliente/servidor.

Las ventajas son muchas como la centralización de datos por ejemplo, el fallo la mas grande que tiene es la dependencia del servidor.

4. Una de las ventajas de la arquitectura de aplicaciones distribuidas en la nube radica en una mayor flexibilidad para configurar, mantener y modificar el hardware que da soporte a la aplicación web. Indica por qué.

Virtualizacion y recursos de baja demanda.

5. Las aplicaciones P2P como eMule, ¿hacen uso de una arquitectura centralizada o distribuida?

Distribuida

6. Cuando un servidor recibe una petición de conexión de un cliente, crea un subproceso (thread) hijo para atender al cliente. ¿Por qué no atiende directamente el servidor al cliente?

Porque asi puedes atender a varios clientes a la vez y evitas bloqueos.

7. Uno de los elementos imprescindibles para crear una infraestructura en la nube es un balanceador de carga. Investiga en qué consiste este software, cuáles son sus características y qué tipos de distribución de carga pueden configurarse.

Es un software (o hardware virtualizado) que actúa como intermediario entre los clientes y los servidores de una aplicación.
Su función es repartir equitativamente las peticiones entre varios servidores para:

- evitar sobrecarga en uno solo,
- mejorar el rendimiento,
- aumentar la disponibilidad,
- y garantizar la tolerancia a fallos.

Ejemplos: NGINX, HAProxy, AWS ELB, Azure Load Balancer, Google Cloud Load Balancing.

8. Investiga qué relación mantienen las máquinas virtuales de sistema con las infraestructuras en la nube.

Las máquinas virtuales de sistema son **la unidad básica de computación en la nube**:

- Permiten que la infraestructura física del proveedor se **divida en recursos virtuales aislados**.
- Facilitan la **automatización, elasticidad y facturación por uso** que caracterizan a la nube.
- Son la **capa fundacional** sobre la que se construyen servicios más avanzados (contenedores, microservicios, serverless).

9. Busca en Internet algunos de los principales proveedores de soluciones de hosting (1&1, Acens, Arsys, Webfusion, etc.) e indica sus tarifas para planes de alojamiento con las principales tecnologías (PHP, JSP/Servlets, ASP.NET). ¿Te ha sido difícil encontrar servicios de hosting para PHP o ASP.NET? ¿Y para JSP/Servlets? ¿Por qué?

Hosting para PHP: abundante, estándar, muchos proveedores lo ofrecen en paquetes básicos.

Hosting para ASP.NET: existe, pero más limitado que para PHP; suelen estar en planes Windows más “premium”.

Hosting para JSP/Servlets: más difícil de encontrar, casi siempre mediante servicios especializados o VPS/servicios más avanzados; no es tan común en hosting compartido general.

10. Investiga en Internet las tecnologías denominadas **IaaS**, **PaaS** y **SaaS** e incluye a continuación una breve reseña de sus principales características.

IaaS (Infrastructure as a Service)
→ Te dan infraestructura virtual (máquinas, red, almacenamiento).
Tú gestionas el SO y las apps.
Ejemplo: AWS EC2, Azure VM.

PaaS (Platform as a Service)
→ Te dan plataforma lista (SO, runtimes, BD, servicios).
Tú solo subes tu código.
Ejemplo: Heroku, Google App Engine.

SaaS (Software as a Service)
→ Te dan la aplicación final lista para usar.
Solo consumes el servicio.
Ejemplo: Gmail, Office 365, Dropbox.

11. En la actualidad, numerosas empresas como Microsoft, Google, Canonical o Amazon ofrecen servicios en la nube que pueden ser contratados por desarrolladores bien como plataforma de desarrollo colaborativo, o bien como plataforma de alojamiento de aplicaciones. 'Azure' es el cloud que ofrece Microsoft. Averigua que características ofrece y el precio de sus servicios.

Microsoft Azure

Es la nube de Microsoft. Ofrece IaaS, PaaS y SaaS:

- Cómputo (máquinas virtuales, contenedores, funciones serverless).
- Almacenamiento (bases de datos, blobs, backup).
- Redes (VPN, balanceo, CDN).
- IA, analítica, IoT y seguridad integrados.
- Escalable, global y muy integrado con Windows, .NET y Office.

Precios orientativos

- VMs: desde céntimos/hora según tamaño (~0,5 $/h una VM media).
- Almacenamiento: ~0,018 $/GB al mes en nivel “Hot”.
- Soporte: desde 29 $/mes básico.
- DevOps / App Services: gratis con límites, luego pago por usuario o recursos.