# Fundamentos de los Sistemas Distribuidos

### 1. Descripción de Arquitecturas y Sus Características

#### **a. Cliente/Servidor (C/S)**

* **Descripción**: Una arquitectura donde el cliente solicita servicios y el servidor los proporciona.
* **Características**:
  1. **Centralización**: Los datos y procesos principales residen en el servidor.
  2. **Simplicidad**: Fácil de implementar y gestionar.
  3. **Ejemplo**: Aplicaciones web, como un navegador solicitando una página a un servidor.

#### **b. Arquitectura Orientada a Servicios (SOA)**

* **Descripción**: Organización basada en servicios independientes que se comunican mediante protocolos estándar (SOAP, REST).
* **Características**:
  1. **Desacoplamiento**: Los servicios son autónomos y están diseñados para ser reutilizables.
  2. **Interoperabilidad**: Compatible con plataformas heterogéneas.
  3. **Ejemplo**: Sistemas empresariales complejos con múltiples servicios (banca, comercio electrónico).

#### **c. Middleware Orientado a Mensajes (MOM)**

* **Descripción**: Facilita la comunicación asíncrona entre aplicaciones mediante el uso de intermediarios (brokers).
* **Características**:
  1. **Comunicación desacoplada**: No depende de la sincronización de tiempo.
  2. **Ejemplo**: Sistemas de mensajería como RabbitMQ o Apache Kafka.

#### **d. Microservicios**

* **Descripción**: Aplicaciones divididas en pequeños servicios independientes, cada uno con una responsabilidad única.
* **Características**:
  1. **Escalabilidad**: Cada servicio puede escalarse individualmente.
  2. **Flexibilidad tecnológica**: Se pueden usar diferentes tecnologías para cada servicio.
  3. **Ejemplo**: Netflix, que utiliza microservicios para streaming, recomendación y autenticación.

#### **e. Cluster**

* **Descripción**: Conjunto de máquinas conectadas que actúan como un solo sistema.
* **Características**:
  1. **Alta disponibilidad**: Si un nodo falla, otros toman su lugar.
  2. **Rendimiento mejorado**: Uso conjunto de recursos.
  3. **Ejemplo**: Servidores web en paralelo para manejar grandes volúmenes de tráfico.

#### **f. Grid**

* **Descripción**: Recursos distribuidos y geográficamente separados que trabajan juntos.
* **Características**:
  1. **Heterogeneidad**: Integra diferentes plataformas y tecnologías.
  2. **Uso intensivo**: Proyectos científicos como procesamiento de datos del CERN.
  3. **Ejemplo**: [SETI@home](mailto:SETI@home).

#### **g. Peer-to-Peer (P2P)**

* **Descripción**: Arquitectura sin roles fijos; cada nodo puede actuar como cliente o servidor.
* **Características**:
  1. **Descentralización**: Mayor tolerancia a fallos y escalabilidad.
  2. **Ejemplo**: Redes blockchain, BitTorrent.

#### **h. Cloud Computing**

* **Descripción**: Provisión de recursos bajo demanda a través de internet.
* **Características**:
  1. **Escalabilidad automática**: Pago por uso.
  2. **Modelos**: SaaS, PaaS, IaaS.
  3. **Ejemplo**: Amazon Web Services (AWS), Google Cloud.

#### **i. Edge Computing**

* **Descripción**: Procesamiento de datos cerca del lugar donde se generan.
* **Características**:
  1. **Baja latencia**: Respuesta más rápida en tiempo real.
  2. **Ejemplo**: Dispositivos IoT como cámaras de seguridad inteligentes.

### 2. Características de los Sistemas Distribuidos

Un sistema distribuido se define por las siguientes características clave:

#### **a. Heterogeneidad**

* Integra hardware, software y redes de diferentes tipos.
* **Ejemplo**: Un sistema que utiliza servidores Linux y Windows.

#### **b. Escalabilidad**

* Capacidad de crecer horizontalmente (añadiendo nodos) o verticalmente (mejorando nodos existentes).
* **Ejemplo**: Servicios en la nube que escalan según la demanda.

#### **c. Tolerancia a Fallos**

* Los sistemas distribuidos pueden continuar funcionando incluso si uno o varios nodos fallan.
* **Mecanismos**: Replicación y redundancia.

#### **d. Transparencia**

* **De Acceso**: Los recursos son accesibles de manera uniforme, sin importar su ubicación.
* **De Fallos**: Oculta los fallos al usuario.
* **De Movilidad**: Los recursos pueden cambiar de ubicación sin afectar al sistema.

#### **e. Concurrencia**

* Permite la ejecución simultánea de múltiples procesos en nodos diferentes.
* **Ejemplo**: Bases de datos distribuidas que soportan múltiples usuarios.

#### **f. Seguridad**

* Protección contra accesos no autorizados y ataques externos.
* Uso de protocolos como SSL y Kerberos.

### 3. Diferencias entre Sistemas Operativos en Red, Sistemas Operativos Distribuidos y Middleware

| **Aspecto** | **SOR (Sistemas Operativos en Red)** | **SOD (Sistemas Operativos Distribuidos)** | **Middleware** |
| --- | --- | --- | --- |
| **Definición** | Sistemas independientes conectados en red. | SO único que controla múltiples nodos. | Capa de abstracción entre SO y aplicaciones. |
| **Arquitectura** | Descentralizada. | Centralizada y homogénea. | Mixta; agrega servicios y abstracción. |
| **Heterogeneidad** | Alta. | Baja (SO homogéneo). | Alta; oculta la heterogeneidad. |
| **Transparencia** | Limitada. | Alta (fallos, ubicación, acceso). | Media; servicios como RPC, MOM. |
| **Ventajas** | Flexibilidad y adaptabilidad. | Alta integración y transparencia. | Flexibilidad, integración y escalabilidad. |
| **Ejemplos** | Linux, Windows en red. | Amoeba, Mach. | CORBA, JEE, gRPC. |

### Resumen

Los **Sistemas Distribuidos** abarcan arquitecturas y tecnologías diversas para resolver problemas complejos mediante la colaboración de múltiples nodos. Desde arquitecturas como C/S hasta modelos modernos como Edge y Cloud, cada enfoque tiene aplicaciones específicas según las necesidades de escalabilidad, tolerancia a fallos y transparencia. Además, la evolución desde los **SOR** hasta los **Middleware** refleja cómo se ha simplificado la gestión y comunicación en estos entornos.

# Tecnologías en Sistemas Distribuidos

### 1. Evolución de Tecnologías para la Abstracción de Sockets

La comunicación en sistemas distribuidos ha evolucionado desde mecanismos básicos como **sockets** hacia soluciones más sofisticadas que añaden niveles de abstracción:

#### **a. Sockets**

* Los **sockets** proporcionan comunicación bidireccional entre procesos en una red. Son el nivel más básico de comunicación.
* **Ventajas**:
  + Control total sobre el flujo de datos.
  + Alta flexibilidad.
* **Desventajas**:
  + Complejidad en la implementación y manejo de errores.
  + Carecen de abstracciones que simplifiquen la programación.

#### **b. Llamadas a Procedimientos Remotos (RPC)**

* **RPC** permite a un programa ejecutar procedimientos en otro sistema como si fueran locales.
* Introduce abstracciones para ocultar detalles de red (como el uso de sockets) y facilita la comunicación remota.
* **Componentes**:
  + Cliente: Llama al procedimiento remoto.
  + Stub: Actúa como proxy para convertir la llamada en una solicitud de red.
  + Servidor: Ejecuta el procedimiento solicitado y envía el resultado.
* **Ventajas**:
  + Transparencia en la comunicación.
  + Facilidad en la programación distribuida.
* **Limitaciones**:
  + No es adecuado para sistemas orientados a objetos.

#### **c. Invocación de Métodos Remotos (RMI)**

* Extiende el paradigma de RPC al mundo de la **programación orientada a objetos**, permitiendo que los objetos remotos sean accesibles como locales.
* **Características**:
  + Soporte para Java.
  + Registro de objetos remotos (RMI Registry).
  + Serialización de objetos para su transmisión.
* **Ventajas**:
  + Transparencia en la invocación de métodos.
  + Integración nativa en aplicaciones Java.
* **Limitaciones**:
  + Dependencia exclusiva del lenguaje Java.

#### **d. Middleware y ORB**

* **Middleware** introduce una capa de abstracción para simplificar la comunicación entre aplicaciones heterogéneas.
  + Ejemplo: **ORB (Object Request Broker)**, utilizado en **CORBA**, que permite la comunicación entre objetos distribuidos en múltiples lenguajes.
* Evoluciones:
  + **JEE (Java Enterprise Edition)**: Framework para aplicaciones distribuidas en Java.
  + **.NET**: Plataforma para sistemas distribuidos en múltiples lenguajes.
* **Ventajas**:
  + Soporte para entornos heterogéneos.
  + Transparencia de localización y lenguaje.

### 2. Tecnología Web

#### **a. HTTP Request y Response**

**Petición HTTP (HTTP Request):**

* **Estructura**:
  + Línea de solicitud: Método, URL, y versión del protocolo (por ejemplo: GET /index.html HTTP/1.1).
  + Cabeceras: Información adicional como tipo de contenido, autenticación, etc.
  + Cuerpo (opcional): Datos enviados al servidor en métodos como POST o PUT.
* **Métodos más comunes**:
  + **GET**: Recuperar datos.
  + **POST**: Enviar datos para su procesamiento.
  + **PUT**: Actualizar recursos.
  + **DELETE**: Eliminar recursos.

**Respuesta HTTP (HTTP Response):**

* **Estructura**:
  + Línea de estado: Código de respuesta y mensaje (ejemplo: 200 OK).
  + Cabeceras: Detalles del contenido, como el tipo MIME (Content-Type).
  + Cuerpo (opcional): Información adicional como HTML, JSON, etc.
* **Códigos de estado**:
  + **2xx**: Éxito (ejemplo: 200 OK).
  + **4xx**: Error del cliente (ejemplo: 404 Not Found).
  + **5xx**: Error del servidor (ejemplo: 500 Internal Server Error).

#### **b. Llamadas de Navegador y Recursos**

Cuando un navegador solicita una URL, realiza múltiples solicitudes para cargar recursos adicionales (CSS, JavaScript, imágenes).  
**Ejemplo de flujo**:

1. **GET /index.html** → Solicita el documento HTML principal.
2. Dentro del HTML, el navegador encuentra referencias a recursos externos y realiza solicitudes adicionales:
   * **GET /style.css** → Para el diseño.
   * **GET /script.js** → Para la lógica del cliente.
   * **GET /imagen.jpg** → Para imágenes.

#### **c. Cabeceras y MIME**

* **Content-Type**: Indica el tipo de contenido en el cuerpo de la respuesta (usando MIME):
  + text/html: Documento HTML.
  + application/json: Datos JSON.
  + image/png: Imagen en formato PNG.
* MIME ayuda a identificar cómo procesar el contenido.

#### **d. Cliente Web y Servidor Web**

* **Cliente Web**: Realiza solicitudes HTTP. Ejemplo: Navegadores como Chrome o Firefox.
* **Servidor Web**: Responde a las solicitudes del cliente, proporcionando recursos o ejecutando servicios (Apache, Nginx).

### 3. Servicios Web

#### **a. Características Generales**

* Basados en **HTTP**, facilitan la comunicación entre aplicaciones en diferentes plataformas.
* Ejemplos: Servicios RESTful, SOAP, gRPC.

#### **b. SOA y Arquitectura Tradicional**

* **SOA (Service-Oriented Architecture)**:
  + Organización en torno a servicios independientes.
  + Utiliza estándares como **SOAP** y **WSDL**.
  + Incluye un **registro de servicios (UDDI)** para que los clientes descubran dinámicamente servicios disponibles.
* **Ventajas**:
  + Separación clara entre cliente y servidor.
  + Independencia de implementación.
* **Desventajas**:
  + Complejidad en la gestión de servicios.

#### **c. Simplificación con REST**

* **REST (Representational State Transfer)**:
  + Arquitectura ligera basada en HTTP.
  + Define **endpoints** para interactuar con recursos.
  + **Ejemplo**:
    - **GET /api/productos**: Listar productos.
    - **POST /api/productos**: Crear un producto.
  + No requiere un servicio de registro como UDDI.
* **Ventajas**:
  + Simplicidad.
  + Compatibilidad con aplicaciones modernas.
  + Uso extensivo en APIs.

#### **d. Comparación de Tecnologías**

| **Tecnología** | **Uso Ideal** |
| --- | --- |
| **SOA** | Aplicaciones complejas con descubrimiento dinámico. |
| **REST** | APIs ligeras y operaciones CRUD simples. |
| **GraphQL** | Consultas personalizadas y optimización. |
| **gRPC** | Comunicación eficiente entre microservicios. |
| **WebSocket** | Comunicación en tiempo real (chat, juegos). |

**Ejemplo de Uso**:

* **REST**: API de un ecommerce para CRUD de productos.
* **GraphQL**: Optimización de consultas en aplicaciones con múltiples vistas.
* **gRPC**: Servicios internos de microservicios en empresas tecnológicas.
* **WebSocket**: Juegos en línea o aplicaciones de chat en tiempo real.

# Tema de Seguridad en Sistemas Distribuidos

#### **1. La Triada de Seguridad: Confidencialidad, Integridad y Disponibilidad**

La triada **CID** es el eje fundamental de la seguridad de la información:

1. **Confidencialidad**:
   * Protección contra accesos no autorizados.
   * Garantiza que solo las personas autorizadas puedan acceder a la información.
   * Ejemplos: Encriptación de datos, contraseñas seguras.
2. **Integridad**:
   * Garantiza que la información no sea modificada de manera no autorizada.
   * Uso de **funciones hash** para verificar si los datos han sido alterados.
3. **Disponibilidad**:
   * Asegura el acceso a los recursos y la información en el momento que se necesiten.
   * Implementación de **sistemas redundantes** y protección frente a ataques DDoS.

#### **2. Blockchain: Tecnología Base**

* **Definición**: Un libro de registros distribuido y seguro que almacena transacciones en bloques encadenados.
* **Características**:
  1. **Descentralización**: No depende de una entidad central.
  2. **Inmutabilidad**: Una vez registrado, el bloque no se puede alterar.
  3. **Transparencia**: Los nodos mantienen una copia completa de la cadena.
  4. **Seguridad**: Uso de criptografía y funciones hash.

#### **3. Bitcoin y el Problema del Doble Gasto**

* **Doble Gasto**: Consiste en intentar usar la misma moneda más de una vez.
* **Soluciones en Bitcoin**:
  1. **Blockchain**: Mantiene un registro único y verificable de transacciones.
  2. **Red P2P**: Todos los nodos validan y acuerdan las transacciones.
  3. **Proof of Work (PoW)**: Garantiza la validación de transacciones a través de un consenso computacional.

#### **4. Funciones Hash**

* **Definición**: Algoritmos que transforman una entrada en una cadena fija de caracteres.
* **Características**:
  1. **Determinismo**: La misma entrada produce siempre el mismo hash.
  2. **Resistencia a colisiones**: Es difícil encontrar dos entradas que generen el mismo hash.
  3. **Uso Criptográfico**: Ejemplo: SHA-256 utilizado en Bitcoin.

#### **5. Firma Digital y Cifrado Asimétrico**

* **Firma Digital**:
  + Garantiza la autenticidad e integridad de un mensaje.
  + Utiliza una clave privada para firmar y una clave pública para verificar.
* **Proceso**:
  1. Generación del hash del mensaje.
  2. Firma del hash con la clave privada.
  3. Verificación con la clave pública del emisor.

#### **6. Criptomonedas y sus Elementos**

1. **Transacción**:
   * Movimiento de fondos entre direcciones.
   * Estructura básica: Entradas, salidas y firma digital.
2. **UTXO (Unspent Transaction Output)**:
   * Salidas de transacciones anteriores que no han sido gastadas.
   * Sirve como entrada para nuevas transacciones.
3. **Smart Contracts**:
   * Contratos autoejecutables programados en la blockchain.
   * Ejemplo: Transferencias de fondos basadas en condiciones predefinidas.

#### **7. Proof of Work (PoW): Consenso en Bitcoin**

* **Definición**: Proceso en el que los mineros compiten para resolver un problema criptográfico.
* **Proceso**:
  1. Los mineros agrupan transacciones y crean un bloque.
  2. Resuelven un problema computacional buscando un hash válido (nonce).
  3. El bloque se valida y se añade a la cadena.

#### **8. Proceso de Minado**

* **Pasos**:
  1. Recopilación de transacciones pendientes.
  2. Validación de transacciones.
  3. Resolución del PoW.
  4. Propagación del bloque validado.
  5. Recompensa al minero (Bitcoin recién creado y comisiones).

#### **Conclusión**

La seguridad en los sistemas distribuidos como blockchain y Bitcoin se basa en mecanismos robustos de criptografía, consenso y descentralización, garantizando integridad, confidencialidad y disponibilidad.