# UNIVERSIDAD PRIVADA BOLIVIANA


# RESUMEN

## “SISTEMAS DISTRIBUIDOS”

**ESTUDIANTE:** **Joseph Anthony Meneses Salguero**

**CODIGO:** **55669**

**MATERIA:** **Sistemas Distribuidos**

**DOCENTE:** **Ing. Alejandro Contreras**

**LA PAZ 11/5 – BOLIVIA-2023**

# INDICE

# SISTEMAS DISTRIBUIDOS

## 1. Introducción

En el transcurso de este informe se explorarán en mayor detalle los diversos temas que son fundamentales en los sistemas distribuidos donde cada uno de ellos desempeña un papel importante para lograr el éxito en entornos distribuidos ya sea garantizando la disponibilidad de datos, la eficiencia en el almacenamiento o la seguridad en la comunicación entre nodos.

En primer lugar se hablara prácticamente sobre sistemas distribuidos, que es, como se aplcica y dentro de las explicaciones se analizará el concepto de RAID que permite combinar varios discos duros en un único dispositivo lógico. Se profundizará en las diferentes técnicas de almacenamiento redundante utilizadas en RAID y cómo contribuyen a la protección de datos y la tolerancia a fallos.

Ademas se abordará el tema de LVM una herramienta esencial en sistemas Linux para la gestión flexible de volúmenes lógicos. Se examinarán los protocolos y mecanismos utilizados para la comunicación entre los componentes de un sistema distribuido, así como los tipos de fallos que pueden ocurrir y cómo se abordan.

La seguridad de los datos en sistemas distribuidos será abordada a través del uso de llaves asimétricas para el cifrado, se explorará cómo se emplean estas llaves para garantizar la confidencialidad de la información transmitida entre los nodos de un sistema distribuido.

La replicación, otro tema importante, será discutida en relación con la disponibilidad de datos, se analizará cómo mantener copias idénticas de datos o servicios en diferentes nodos contribuye a la tolerancia a fallos y asegura la accesibilidad continua de la información.

La sincronización del tiempo entre los nodos a través de relojes, se examinará cómo se logra la sincronización precisa de los relojes en sistemas distribuidos y cómo se manejan los desafíos asociados.

El concepto de transacciones será presentado como una unidad de trabajo lógica que agrupa operaciones que deben ejecutarse de manera consistente. Se profundizará en los principios ACID que rigen las transacciones en sistemas distribuidos.

En cuanto a la infraestructura y servicios de soporte, se explorarán el sistema de nomenclatura jerárquica distribuida utilizado para asignar nombres a recursos, el servidor web y proxy inverso de alto rendimiento , la plataforma de virtualización a nivel de sistema operativo , la implementación de protocolos de red de Microsoft Windows , la plataforma de virtualización y gestión de máquinas virtuales y contenedores , así como las herramientas de supervisión y monitoreo como Zabbix y la solución de copia de seguridad y replicación de datos críticos Veeam Backup & Replication Console.

## 2. Sistemas distribuidos

Los sistemas distribuidos se basan en la interconexión de múltiples computadoras a través de una red, lo que les permite trabajar en conjunto para resolver problemas de manera eficiente. Estos sistemas dependen de una comunicación confiable entre los componentes distribuidos y deben abordar desafíos como la gestión de recursos compartidos y la sincronización. Al conectar computadoras autónomas y utilizar software distribuido se crea una entidad unificada capaz de abordar tareas complejas y ofrecer soluciones integrales.

En otras palabras, los sistemas distribuidos son arquitecturas de software que permiten a múltiples computadoras colaborar y compartir recursos a través de una red. Estos sistemas afrontan desafíos como la comunicación, la gestión de recursos y la sincronización, pero al trabajar en conjunto brindan la capacidad de resolver problemas complejos y ejecutar operaciones distribuidas de manera eficiente.

### 2.1. Características de los Sistemas Distribuidos

- Transparencia: Los sistemas distribuidos buscan ocultar la complejidad de la distribución de los recursos y servicios a los usuarios. Esto implica que los usuarios perciban el sistema como un todo unificado, sin necesidad de conocer los detalles de su implementación distribuida.
- Escalabilidad: Los sistemas distribuidos deben ser capaces de adaptarse y crecer para manejar un aumento en la carga de trabajo y el tamaño del sistema. La escalabilidad puede lograrse mediante la adición de nuevos nodos de procesamiento o recursos de almacenamiento.
- Fiabilidad y tolerancia a fallos: Los sistemas distribuidos deben ser robustos y capaces de resistir y recuperarse de posibles fallos en los componentes

individuales. La tolerancia a fallos implica la capacidad de mantener la continuidad del servicio a pesar de la aparición de errores o fallos en los nodos o en la red.

- Seguridad: Los sistemas distribuidos deben garantizar la confidencialidad, integridad y disponibilidad de los datos y servicios. Se deben implementar mecanismos de seguridad adecuados para proteger la información transmitida y almacenada en el sistema.
- Interoperabilidad: Los sistemas distribuidos pueden estar compuestos por diferentes plataformas, sistemas operativos y tecnologías. La interoperabilidad permite la comunicación y el intercambio de información entre estos componentes heterogéneos, facilitando la cooperación y la integración entre ellos.

**2.2. Tipos de Sistemas Distribuidos**

Existen varios tipos de sistemas distribuidos, cada uno con su enfoque particular y aplicación específica:

- Sistemas de archivos distribuidos: Permiten el acceso y la gestión de archivos distribuidos en múltiples nodos deun sistema. Proporcionan transparencia de acceso a los archivos, permitiendo a los usuarios acceder y compartir archivos de manera transparente, como si estuvieran almacenados localmente.

- Sistemas de información distribuidos: Estos sistemas se centran en el procesamiento y la gestión de grandes cantidades de información distribuida. Permiten el acceso y la manipulación de datos distribuidos en diferentes ubicaciones, garantizando la consistencia y la integridad de los datos en todo el sistema.

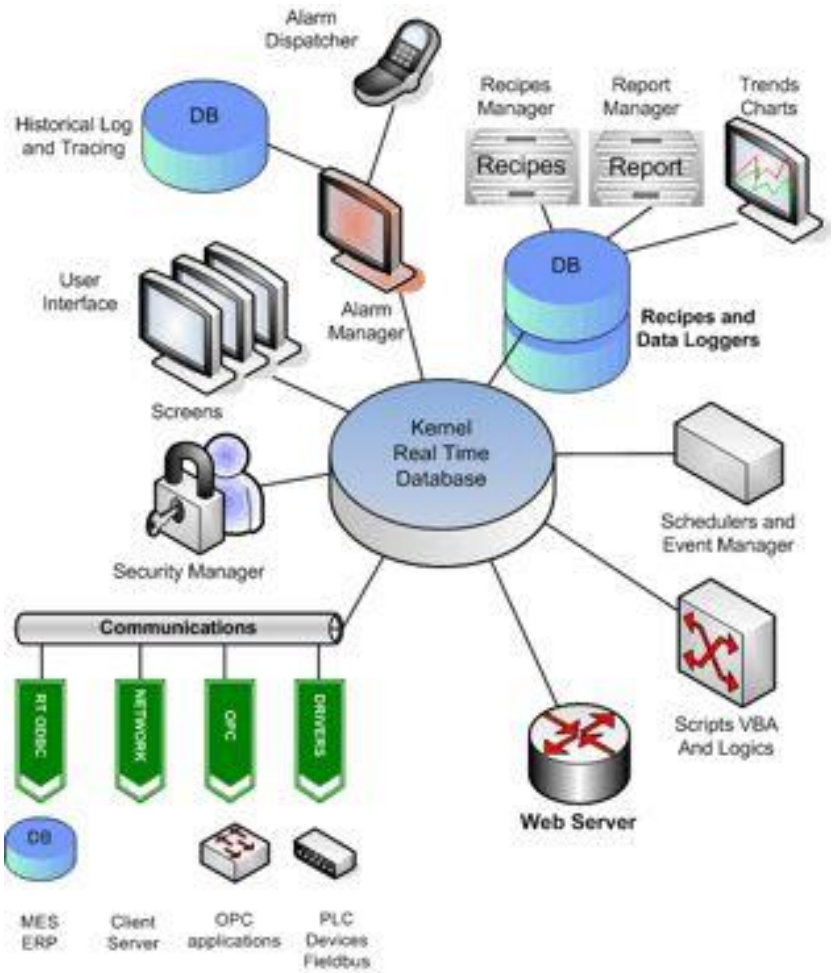
- Sistemas de bases de datos distribuidos: Estos sistemas distribuyen y replican datos en múltiples nodos de una red. Proporcionan capacidades de gestión de datos distribuidos, permitiendo el acceso eficiente y la manipulación de grandes volúmenes de datos distribuidos.

- Sistemas de comunicaciones distribuidos: Estos sistemas se centran en facilitar la comunicación y el intercambio de información entre los

componentes distribuidos de un sistema. Proporcionan servicios de comunicación confiables y eficientes, como el envío de mensajes y la transferencia de datos, para permitir la colaboración y la coordinación entre los nodos del sistema.

- Sistemas de computación distribuida: Estos sistemas utilizan múltiples nodos de procesamiento para realizar tareas computacionalmente intensivas. Dividen las tareas en subprocesos más pequeños y los distribuyen entre los nodos, lo que permite una mayor capacidad de procesamiento y la ejecución paralela de las tareas.

Cada tipo de sistema distribuido tiene sus propias características, desafíos y beneficios. La elección del tipo de sistema distribuido adecuado depende de los requisitos y objetivos específicos del sistema, así como de los recursos disponibles e el momento.

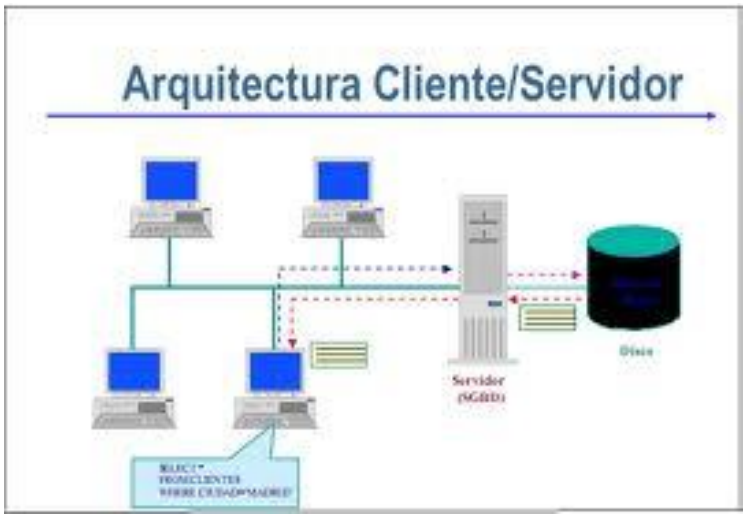


### 2.3. Ejemplo

Una posible estructura de sistema distribuido es la arquitectura cliente-servidor. En esta arquitectura, los componentes del sistema se dividen en dos categorías principales: los clientes y los servidores. Los clientes solicitan y consumen los servicios proporcionados por los servidores.

En el caso de una aplicación web, por ejemplo, los clientes son los navegadores web que se ejecutan en las computadoras de los usuarios, mientras que los servidores son las máquinas que alojan y procesan los recursos y servicios de la aplicación. Los clientes envían solicitudes a los servidores a través de la red, y los servidores responden proporcionando los datos solicitados o realizando las acciones requeridas.

En la siguiente imagen podemos ver un poco mas detallada la estructura




En la imagen, se muestra un cliente que envía una solicitud a un servidor a través de la red, y el servidor responde devolviendo la respuesta al cliente. Esta interacción entre cliente y servidor se puede repetir con múltiples clientes y servidores en un sistema distribuido más complejo.

## 3. RAID (Redundant Array of Independent Disks)

RAID (Redundant Array of Independent Disks) es un sistema de almacenamiento de datos que combina múltiples discos duros para crear un único disco virtual. Los discos que forman parte del RAID se denominan discos miembros y pueden tener diferentes tamaños y tipos. El objetivo del RAID es proporcionar mayor capacidad de

almacenamiento, mayor velocidad de acceso a los datos y una mayor fiabilidad en comparación con un disco duro individual.

### 3.1. Tipos de RAID

Hay una variedad de RAIDs pero vamos a mencionar los mas importantes y mas utilizados

- RAID 0: Ofrece mayor velocidad y tolerancia a fallos, aunque algunos no lo consideran como un verdadero RAID. Una configuración RAID 0 con dos discos puede ser hasta dos veces más rápida que un solo disco duro. Se utiliza comúnmente en aplicaciones que requieren una alta velocidad de transferencia de datos, como videojuegos, reproducción de video, audio, imágenes, bases de datos y aplicaciones en general.

- RAID 1: También conocido como Mirror o espejo, funciona agregando discos duros paralelos a los discos principales existentes en la computadora. Por ejemplo, si una computadora tiene 2 discos, se puede agregar un disco duro adicional para cada uno, lo que suma un total de 4 discos. Los discos agregados actúan como una copia del primero. El RAID 1 ofrece tolerancia a fallos y velocidad, pero no aumenta la capacidad de almacenamiento. Una configuración RAID 1 con dos discos puede ser hasta dos veces más rápida que un solo disco duro y se utiliza principalmente para instalar el sistema operativo.

- RAID 5: Este tipo de RAID ofrece tolerancia a fallos y optimiza la capacidad del sistema, permitiendo utilizar hasta el 80% de la capacidad total de los discos en conjunto. Una configuración RAID 5 con tres discos puede ser hasta tres veces más rápida que un solo disco duro y puede perder un disco sin perder la información. Es adecuada para aplicaciones que requieren un equilibrio entre rendimiento, capacidad y protección de datos.

- RAID 6: Al igual que el RAID 5, ofrece tolerancia a fallos y optimización de la capacidad del sistema. Sin embargo, una configuración RAID 6 con al menos cuatro discos puede ser hasta cuatro veces más rápida que un solo disco duro y

puede perder dos discos sin perder datos. Es recomendable para aplicaciones que necesitan un mayor nivel de protección y capacidad, como bases de datos, aplicaciones web, aplicaciones móviles y aplicaciones de escritorio. Sin embargo, no se recomienda tener más de 18 discos en un arreglo RAID 6 debido a la complejidad y el riesgo de fallos.

### 3.2. RAID hardware

Se recomienda utilizar RAID en su versión hardware, ya que ofrece mayor rendimiento y confiabilidad. El RAID hardware es un sistema de almacenamiento que se encuentra integrado en la placa base de la computadora. Aunque es más costoso, proporciona funcionalidades avanzadas de administración y configuración, así como mayor protección contra fallos, como la reconstrucción automática de datos y la detección de errores.

### 3.3. RAID software

El RAID software es un sistema de almacenamiento que se implementa a través del sistema operativo de la computadora. Es más económico y fácil de instalar, pero puede tener un rendimiento inferior debido a la carga adicional que impone al sistema operativo. Además, la fiabilidad del RAID software depende del sistema operativo y puede ser menos robusta en comparación con el RAID hardware. Sin embargo, el RAID software sigue siendo una opción viable para configuraciones de nivel básico o entornos donde no se requiere un alto rendimiento o una protección de datos crítica.

### 3.4. Ejemplo

Un ejemplo de uso de RAID 5 es en entornos empresariales que requieren almacenamiento de datos confiable y de alta capacidad. Supongamos que una empresa de desarrollo de software utiliza RAID 5 para su servidor de archivos. Tienen tres discos duros de 1 TB cada uno configurados en RAID 5. En este caso, el RAID 5 proporciona tolerancia a fallos y capacidad de recuperación en caso de que uno de los discos falle. Además, la capacidad total utilizable será de aproximadamente 2 TB (la capacidad de dos discos), mientras que el tercero se utiliza para la paridad.

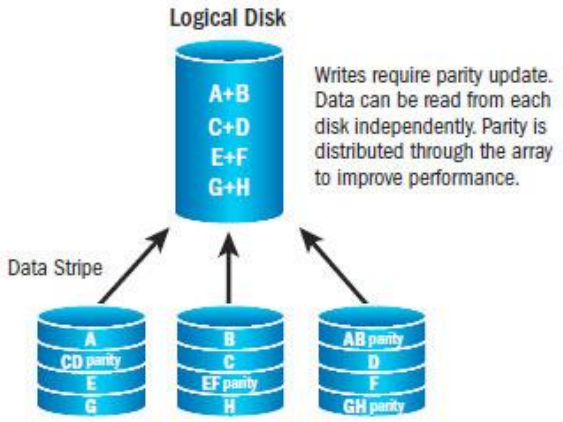



Figure 5. RAID 5 Array

Haciedo un recapitulacon podemos decir qyue el RAID es un sistema de almacenamiento que combina discos duros para crear un único disco virtual con mayor capacidad, velocidad y fiabilidad. Los diferentes tipos de RAID, como RAID 0, RAID 1, RAID 5 y RAID 6, ofrecen diversas combinaciones de tolerancia a fallos, velocidad y capacidad de almacenamiento. Se recomienda utilizar RAID hardware cuando sea posible debido a su mayor rendimiento y confiabilidad, mientras que el RAID software es una opción más económica pero menos eficiente.

## 4. LVM (Logical Volume Manager)

LVM (Logical Volume Manager) es una tecnología que proporciona una capa de abstracción entre el sistema operativo y los discos y particiones utilizados. En lugar de asignar directamente discos y particiones al sistema operativo mediante rutas como /dev/sda1 o /dev/sdb2, LVM permite combinar y administrar de forma flexible múltiples discos y particiones en un dispositivo lógico único. Esto brinda ventajas en cuanto a la gestión del almacenamiento, la flexibilidad y la capacidad de redimensionamiento.

### 4.1. Estructura de LVM

LVM sigue la siguiente estructura

I. Disk Drive (Unidad de Disco):

   Representa un disco físico en el sistema, como un disco duro o una unidad de estado sólido (SSD). Estos discos son la base sobre la cual se construye el almacenamiento en LVM.

II. Physical Volume (Volumen Físico):

Un Physical Volume es una porción de un disco físico que se asigna para su uso con LVM. Puede ser una partición completa o una sección de un disco. Los Physical Volumes actúan como contenedores para los datos y se combinan para formar un espacio de almacenamiento más grande y flexible.

III. Physical Partition (Partición Física):

Una Physical Partition es una subdivisión de un Physical Volume. Se utiliza para gestionar y asignar el espacio de almacenamiento de manera eficiente. Cada partición física tiene su propia identificación y tamaño, y puede contener datos o estar vacía.

IV. Logical Volume (Volumen Lógico):

Un Logical Volume es una abstracción de almacenamiento que se crea a partir de uno o más Physical Volumes. Los Logical Volumes se comportan como discos virtuales y se utilizan para almacenar datos. Pueden abarcar múltiples discos y particiones y ofrecen una mayor flexibilidad en términos de capacidad y rendimiento. Desde la perspectiva del sistema operativo, un Logical Volume es indistinguible de un disco físico individual.

V. Logical Partition (Partición Lógica):

Una Logical Partition es una subdivisión de un Logical Volume y se utiliza para organizar y asignar el espacio de almacenamiento dentro del volumen. Proporciona una forma de dividir lógicamente el espacio disponible en el Logical Volume en unidades más pequeñas y manejar diferentes tipos de datos de manera más eficiente.

La estructura jerárquica de LVM permite una administración más flexible y eficiente del almacenamiento en el sistema operativo. Los beneficios incluyen la capacidad de agregar o quitar discos fácilmente, redimensionar volúmenes lógicos sobre la marcha y migrar datos sin interrupciones. Esto brinda mayor capacidad de expansión, mejor aprovechamiento de los recursos y una gestión más simplificada del almacenamiento en comparación con los enfoques tradicionales de administración de discos.

### 4.2. Ejemplo

Supongamos que tienes tres discos físicos en tu sistema: /dev/sda, /dev/sdb y /dev/sdc. Quieres combinar estos discos en un único dispositivo lógico y administrar el almacenamiento de manera flexible. Utilizarás LVM para lograrlo.

I. **Creación de Physical Volumes:**

Utilizando el comando "pvcreate", creas los Physical Volumes para cada disco:

pvcreate /dev/sda

pvcreate /dev/sdb

pvcreate /dev/sdc

II. **Creación de un Volume Group:**

Utilizando el comando "vgcreate", creas un Volume Group que agrupe los Physical Volumes:

vgcreate myvg /dev/sda /dev/sdb /dev/sdc

III. **Creación de Logical Volumes:**

Utilizando el comando "lvcreate", creas Logical Volumes dentro del Volume Group:

lvcreate -n mylv1 -L 50G myvg

lvcreate -n mylv2 -L 100G myvg

IV. **Formateo y montaje de los Logical Volumes:**

Utilizando los comandos adecuados según el sistema de archivos, formateas los Logical Volumes y los montas en puntos de montaje deseados:

mkfs.ext4 /dev/myvg/mylv1

mkfs.ext4 /dev/myvg/mylv2

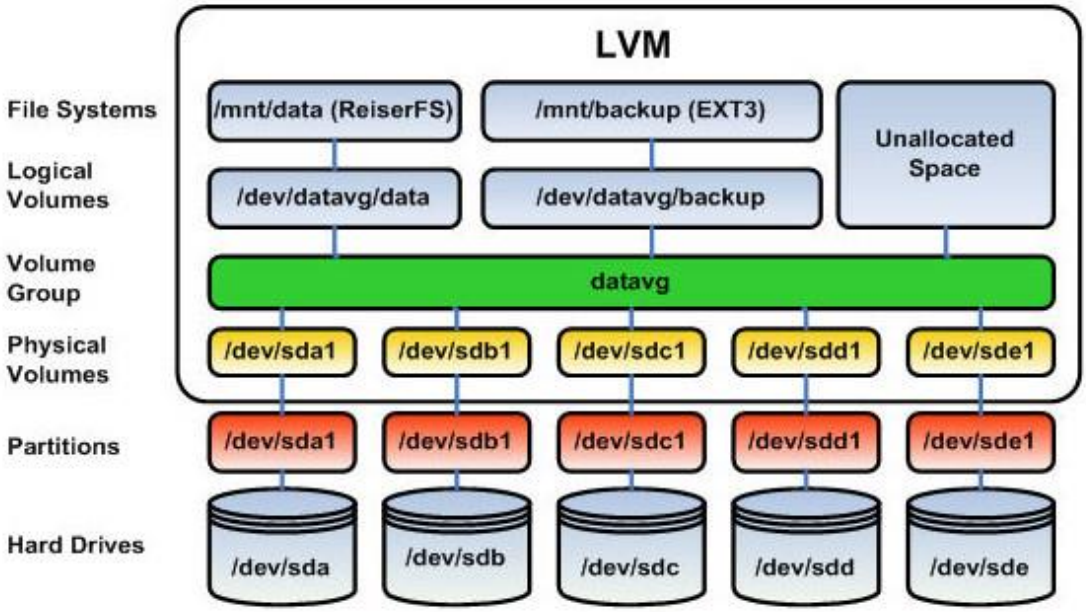
mount /dev/myvg/mylv1 /mnt/data1

mount /dev/myvg/mylv2 /mnt/data2

V. **Utilización del almacenamiento:**

Ahora puedes utilizar los puntos de montaje (/mnt/data1 y /mnt/data2 en este ejemplo) para almacenar tus datos. El sistema operativo los reconocerá como

discos individuales, pero en realidad están utilizando el almacenamiento combinado de los discos físicos que configuraste con LVM.




## 5. Modelo de Interaaccion

El modelo de interacción se refiere a la forma en que se establece la comunicación entre un cliente y un servidor en un sistema informático. Puede haber diferentes enfoques en la interacción entre el cliente y el servidor, como los modelos síncronos y asincrónos.

### 5.1. Modelo síncrono

En el modelo síncrono, la comunicación entre el cliente y el servidor ocurre de forma sincrónica y secuencial. El cliente envía una petición al servidor y espera a que este responda antes de continuar con la siguiente acción. El servidor procesa la petición y envía la respuesta de vuelta al cliente. Durante este proceso, el cliente y el servidor están sincronizados y se mantienen en espera hasta que se complete la transacción.

### 5.2. Modelo asíncrono

En contraste, en el modelo asíncrono, la comunicación entre el cliente y el servidor ocurre de forma no secuencial y sin necesidad de esperar una respuesta inmediata. El cliente puede enviar múltiples peticiones al servidor sin detenerse a esperar las respuestas. El servidor, a su vez, puede responder a las peticiones en cualquier momento, incluso en un orden no correlacionado con las peticiones recibidas. Esta comunicación asincrónica

permite una mayor flexibilidad y eficiencia en el intercambio de datos entre el cliente y el servidor.

En ambos modelos, el cliente envía peticiones al servidor y este responde a esas peticiones. La diferencia radica en cómo se maneja la sincronización y el flujo de información entre el cliente y el servidor. El modelo síncrono se basa en una comunicación secuencial y bloqueante, mientras que el modelo asíncrono permite una comunicación no secuencial y no bloqueante.

Es importante tener en cuenta que cada modelo de interacción tiene sus ventajas y desventajas, y la elección del modelo adecuado dependerá de los requisitos y características específicas del sistema informático en cuestión.

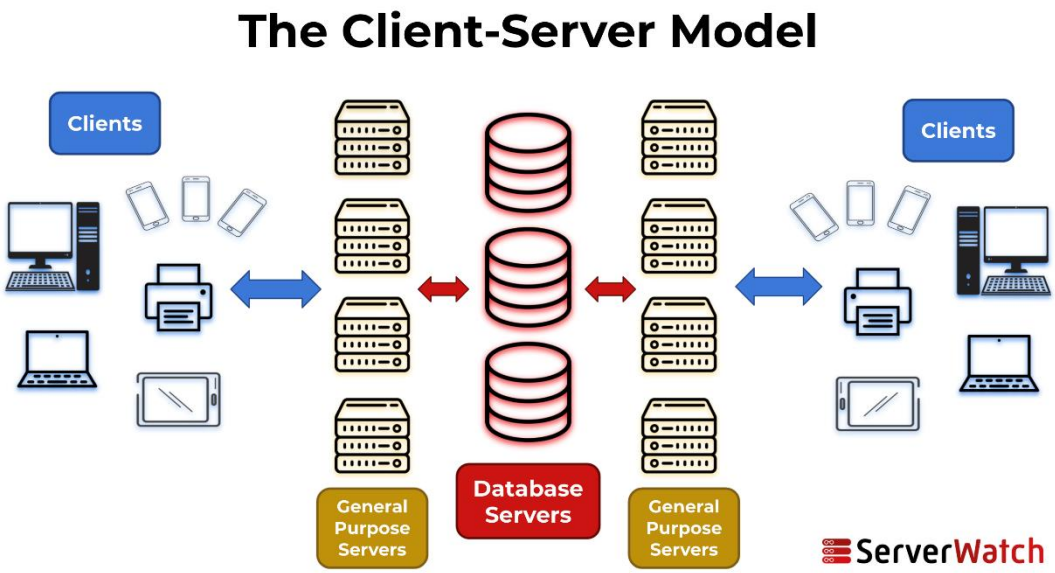
### 5.3. Ejemplo

Supongamos que tienes un cliente de correo electrónico y estás utilizando un modelo de interacción para enviar y recibir correos electrónicos desde un servidor de correo. Aquí se ilustra cómo podría funcionar este proceso:

I. Modelo síncrono:
- El cliente de correo electrónico envía una petición al servidor para verificar si hay nuevos correos electrónicos.
- El servidor procesa la petición y envía una respuesta al cliente, indicando si hay nuevos correos o no.
- Si hay nuevos correos, el cliente solicita descargar los mensajes específicos.
- El servidor responde proporcionando los mensajes solicitados.
- El proceso se repite para cada interacción entre el cliente y el servidor.

II. Modelo asíncrono:
- El cliente de correo electrónico envía una petición al servidor para verificar si hay nuevos correos electrónicos.
- En lugar de esperar una respuesta inmediata, el cliente continúa con otras tareas.
- El servidor procesa la petición y, cuando detecta nuevos correos, envía una notificación al cliente.
- El cliente recibe la notificación y solicita descargar los mensajes específicos.
- El servidor responde proporcionando los mensajes solicitados.

- Este proceso puede repetirse en cualquier momento, incluso si el cliente está realizando otras operaciones.




## 6. Modelo de Fallos

El modelo de fallos es una representación teórica que se utiliza para analizar y comprender los diferentes tipos de fallos que pueden ocurrir en un sistema o proceso. Estos fallos pueden afectar el funcionamiento normal del sistema, comprometer su seguridad, causar errores o producir resultados inesperados.

### 6.1. Tipos de fallos

I. Falla de congelación:

Este tipo de fallo ocurre cuando un proceso o sistema se bloquea y deja de responder. El sistema se queda "congelado" y no puede continuar ejecutando sus tareas o procesos normales. Puede ser causado por problemas en el software, recursos insuficientes o conflictos en el sistema.

II. Falla de omisión:

Las fallas de omisión se refieren a situaciones en las que se pierde la ejecución de una acción que debería haber ocurrido. Se pueden distinguir dos subtipos:

Omisión de recepción: Ocurre cuando un proceso o sistema no recibe un mensaje o señal que se esperaba recibir.

Omisión de envío: Se produce cuando un proceso o sistema no envía un mensaje o señal que debería haber enviado.

III. Falla de tiempo:

Este tipo de fallo está relacionado con problemas de sincronización y retrasos en la ejecución de tareas. Puede ocurrir cuando una tarea no se completa dentro de un tiempo establecido o cuando hay inconsistencias en los tiempos de ejecución de diferentes procesos.

IV. Falla de respuesta:

Las fallas de respuesta se dividen en dos categorías:

Falla de valor: Ocurre cuando un proceso o sistema produce un resultado incorrecto o inesperado debido a un error en el cálculo, procesamiento de datos o interpretación incorrecta.

Falla de transición de estado: Se produce cuando un sistema no cambia de estado o no realiza una transición de estado correctamente, lo que puede afectar su funcionamiento normal.

V. Falla arbitraria:

Este tipo de fallo engloba errores de cualquier tipo que no siguen un patrón o esquema predefinido. Estos fallos pueden ser impredecibles y pueden deberse a errores humanos, fallas aleatorias en los componentes del sistema o problemas imprevistos.

### 6.2. Ejemplo

Imaginemos un sistema de control de tráfico de una ciudad, donde varios semáforos coordinan el flujo de vehículos en diferentes intersecciones. En este escenario, pueden ocurrir diferentes tipos de fallos que afecten el funcionamiento normal del sistema.

I. Falla de congelación:

Ejemplo: Un semáforo se queda bloqueado en luz roja y no cambia su estado, lo que provoca que el tráfico se detenga en esa intersección y cause congestión.

II. Falla de omisión:

Ejemplo de omisión de recepción: El semáforo no recibe la señal de un sensor de detección de vehículos, por lo que no activa el cambio de luz verde correspondiente a la presencia de vehículos en espera.

Ejemplo de omisión de envío: El semáforo no envía la señal de cambio de luz verde a los semáforos adyacentes, lo que provoca una falta de coordinación en el flujo de tráfico.

III. Falla de tiempo:

Ejemplo: Los tiempos programados en los semáforos no se ajustan correctamente a los patrones de tráfico, lo que resulta en tiempos de espera prolongados o cambios de luz ineficientes.

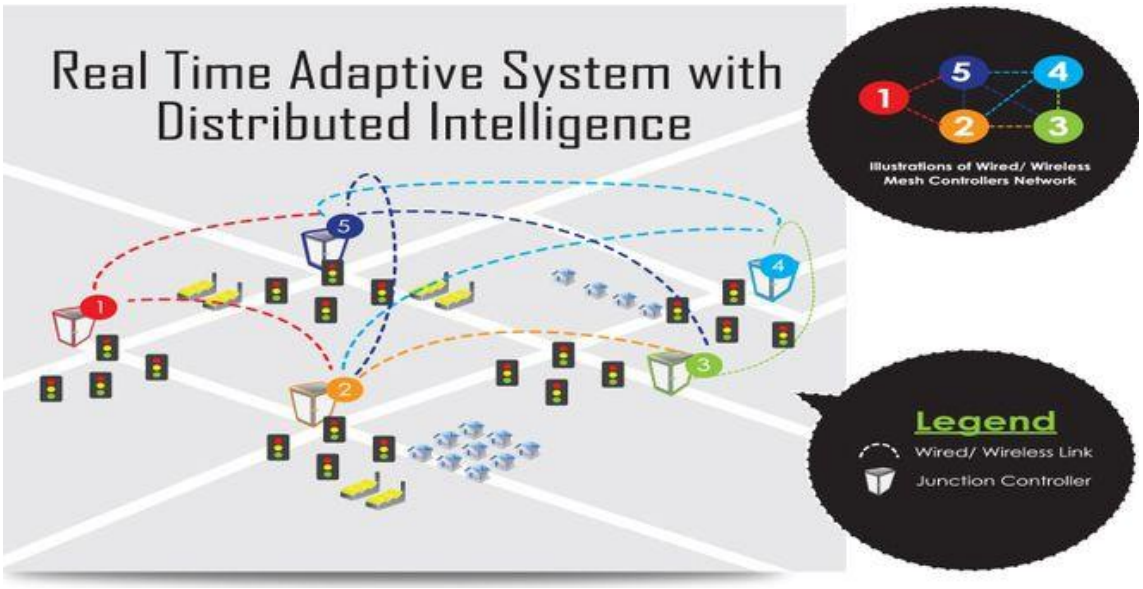
IV. Falla de respuesta:

Ejemplo de falla de valor: El semáforo muestra una luz verde cuando debería mostrar una luz roja, lo que crea una situación peligrosa y puede causar accidentes.

Ejemplo de falla de transición de estado: El semáforo no realiza la transición de luz amarilla a roja antes de cambiar a verde, lo que confunde a los conductores y genera riesgos de colisión.

V. Falla arbitraria:

Ejemplo: En un momento aleatorio, los semáforos de una intersección cambian rápidamente de luz verde a roja y viceversa sin seguir el patrón establecido, lo que genera confusión y caos en el tráfico.



## 7. Llaves Asimetricas

Una llave asimétrica, también conocida como criptografía de clave pública, es un sistema de cifrado que utiliza un par de claves matemáticamente relacionadas pero distintas: una llave pública y una llave privada. Este enfoque contrasta con la criptografía de clave simétrica, donde se utiliza una sola llave para cifrar y descifrar la información.

### 7.1. Funcionamiento de una llave asimétrica

En la criptografía de llave asimétrica, la llave pública se utiliza para cifrar los datos, mientras que la llave privada se utiliza para descifrarlos. El proceso se puede resumir de la siguiente manera:

- Generación de claves: El usuario genera un par de claves matemáticamente relacionadas: una llave pública y una llave privada. Estas claves se basan en algoritmos criptográficos seguros.
- Distribución de la llave pública: El propietario de la llave asimétrica comparte libremente su llave pública con otros usuarios o entidades que deseen enviarle mensajes cifrados
- Cifrado: Cuando alguien quiere enviar un mensaje al propietario de la llave asimétrica, utiliza la llave pública del destinatario para cifrar el mensaje. Solo el propietario de la llave privada correspondiente podrá descifrarlo.
- Descifrado: El propietario del par de llaves utiliza su llave privada para descifrar los mensajes cifrados que le son enviados. La llave privada se mantiene en secreto y solo el propietario tiene acceso a ella.

### 7.2. Ventajas de las llaves asimétricas

Las llaves asimétricas ofrecen varias ventajas en comparación con la criptografía de clave simétrica:

- Mayor seguridad: Al utilizar un par de claves distintas, el riesgo de exposición de la llave privada se reduce significativamente, lo que dificulta la descifrado no autorizado de la información.
- Intercambio seguro de información: La llave pública puede ser compartida abiertamente sin comprometer la seguridad de la comunicación. Esto permite un intercambio seguro de información en entornos no confiables.
- Autenticación y firma digital: Las llaves asimétricas también se utilizan para verificar la autenticidad de los mensajes y firmar digitalmente documentos. Esto

proporciona una forma de garantizar la integridad y la no repudiación de la información.

### 7.3. Ejemplo

- Generación de llaves:

  Para generar una llave privada RSA de 2048 bits:

openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048

Para extraer la llave pública correspondiente:

openssl rsa -pubout -in private_key.pem -out public_key.pem

- Cifrado y descifrado de un mensaje:

  Genera un archivo de texto llamado `message.txt` con el contenido del mensaje que deseas cifrar.

  Cifrado utilizando la llave pública:

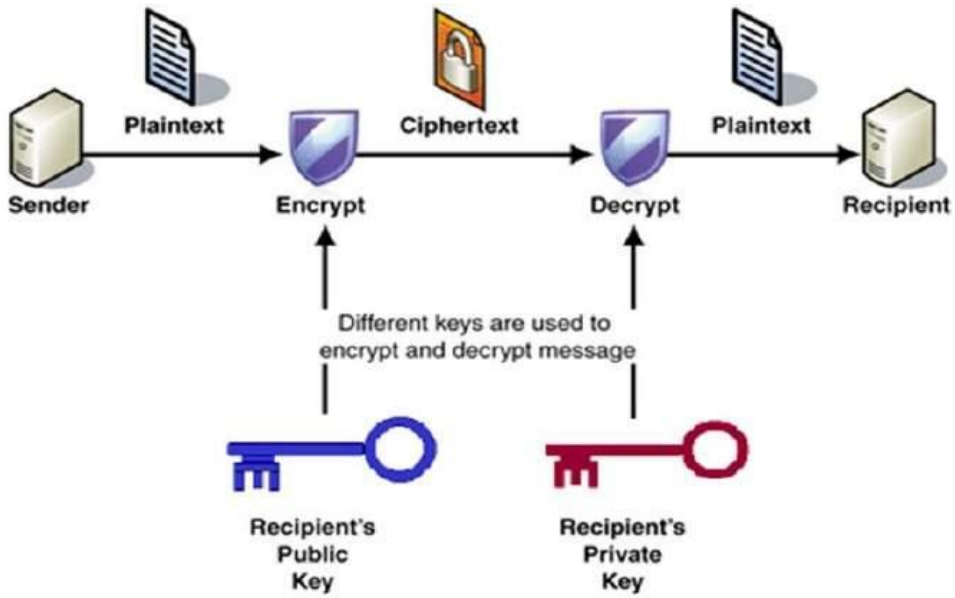
openssl rsautl -encrypt -pubin -inkey public_key.pem -in message.txt -out ciphertext.bin

Descifrado utilizando la llave privada:

openssl rsautl -decrypt -inkey private_key.pem -in ciphertext.bin -out plaintext.txt

- Verificación del resultado:

  Abre el archivo `plaintext.txt` para ver el mensaje descifrado.



## 8. Replicación

La replicación es un recurso ampliamente utilizado en sistemas distribuidos para mantener copias de información en múltiples computadoras. Consiste en crear y mantener réplicas exactas de los datos en diferentes nodos del sistema. Esta estrategia se utiliza para mejorar distintos aspectos de un sistema distribuido, como el rendimiento, la disponibilidad y la tolerancia a fallos.

### 8.1. Aplicaciones de la replicación

La replicación se aplica en diversos contextos y aplicaciones en sistemas distribuidos:

- Servidores web: Los servidores caché y los proxies son ejemplos comunes de replicación en el ámbito de los servidores web. Estos servidores almacenan copias de recursos web frecuentemente accedidos para reducir la latencia y mejorar el rendimiento de las solicitudes de los clientes.
- Sistemas de nombres de dominio (DNS): En el DNS, se utilizan copias replicadas de los mapeos URL-IP para permitir consultas rápidas y eficientes. Estas copias se actualizan regularmente para reflejar los cambios en la configuración de red.
- Google Data Centers: Google utiliza una amplia replicación en sus centros de datos para garantizar la disponibilidad y el rendimiento de sus servicios. Los datos y servicios clave se replican en múltiples ubicaciones geográficas para ofrecer un acceso rápido y confiable a los usuarios.

### 8.2. Beneficios de la replicación

La replicación aporta varios beneficios a los sistemas distribuidos:

- Rendimiento: Al mantener copias de los datos en nodos cercanos a los usuarios o en servidores caché, se reduce la latencia de acceso a la información. Esto mejora el rendimiento general del sistema al acelerar las respuestas a las solicitudes.
- Disponibilidad: La replicación permite distribuir la carga de trabajo entre varios nodos, lo que ayuda a evitar puntos únicos de fallo. Si uno de los nodos falla, los usuarios aún pueden acceder a los datos a través de las réplicas disponibles.
- Tolerancia a fallos: En caso de que un nodo falle o se desconecte, las réplicas garantizan que los datos sigan estando disponibles en otros nodos. Esto aumenta la tolerancia a fallos del sistema y contribuye a la continuidad del servicio.

### 8.3. Ejemplo

La replicación en sistemas distribuidos puede implementarse de diversas formas según el contexto y las herramientas utilizadas. A continuación, te mostraré un ejemplo básico de replicación en Linux utilizando el comando `rsync`:

- Configuración de los servidores:

- Servidor A: Contiene los archivos que se desean replicar.

- Servidor B: Actuará como la réplica del Servidor A.

- Instala `rsync`:

sudo apt-get install rsync

- Sincronización inicial de archivos desde Servidor A a Servidor B:

rsync -avz /ruta/archivos origen usuario@IP_Servidor_B:/ruta/destino

Reemplaza `/ruta/archivos` con la ubicación de los archivos en el Servidor A, `usuario` con el nombre de usuario en el Servidor B y `IP_Servidor_B` con la dirección IP del Servidor B. `/ruta/destino` representa la ubicación en el Servidor B donde se guardarán los archivos replicados.

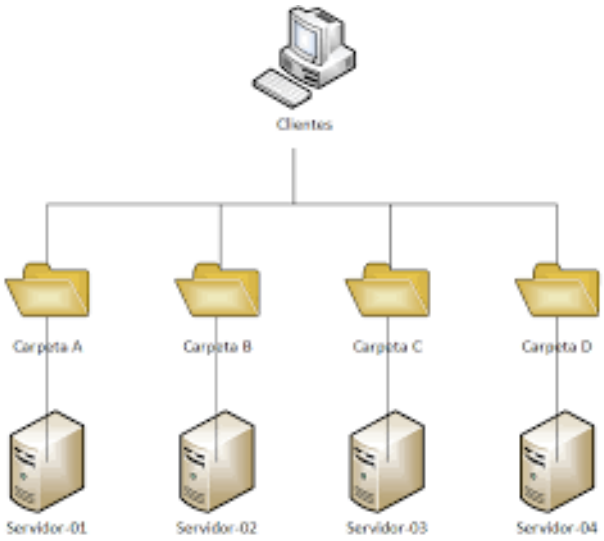
- Programar sincronización periódica:

Puedes programar tareas de sincronización periódica utilizando `cron` para asegurarte de que los cambios en el Servidor A se reflejen en el Servidor B.

crontab -e

Agrega la siguiente línea para ejecutar la sincronización cada 5 minutos:

*/5 * * * * rsync -avz /ruta/archivos origen usuario@IP_Servidor_B:/ruta/destino




## 9. Exclusión

La exclusión mutua distribuida es un concepto fundamental en sistemas distribuidos que se refiere a la necesidad de garantizar que solo un proceso o hilo pueda acceder a un recurso compartido en un momento dado. Esto es esencial para evitar conflictos y inconsistencias en los datos cuando varios procesos intentan acceder al mismo recurso simultáneamente.

La exclusión mutua distribuida aborda el desafío de coordinar el acceso a recursos compartidos en un entorno distribuido, donde los procesos se ejecutan en diferentes nodos de un sistema. A continuación, se presentan algunos subtemas importantes relacionados con la exclusión mutua distribuida.

### 9.1. Deadlock

Un deadlock es una situación en la que dos o más procesos quedan bloqueados permanentemente porque cada uno de ellos espera un recurso que está siendo utilizado por otro proceso. Esto puede resultar en una parálisis del sistema, ya que los procesos no pueden avanzar y liberar los recursos que necesitan otros procesos.

### 9.2. Mecanismos para garantizar la exclusión mutua

Existen diferentes mecanismos y algoritmos que se utilizan para garantizar la exclusión mutua distribuida. Estos mecanismos aseguran que solo un proceso pueda acceder a un recurso compartido en un momento dado. Algunos de los mecanismos comunes son los siguientes: semáforos, monitores, algoritmo de decker y de Peterson.

### 9.3. Semáforos

Los semáforos son variables especiales utilizadas en programación concurrente para coordinar y controlar el acceso a recursos compartidos. Tienen un valor entero y se utilizan principalmente para sincronizar procesos o hilos. Los semáforos pueden ser de dos tipos: binarios (0 o 1) o contadores (valores enteros positivos).

Características clave de los semáforos:

- Operaciones atómicas: Los semáforos ofrecen operaciones atómicas como "esperar" y "señalar". La operación "esperar" reduce el valor del semáforo y bloquea el proceso si el valor es cero, mientras que la operación "señalar" incrementa el valor del semáforo y desbloquea un proceso en espera si lo hay.
- Control de acceso: Los semáforos permiten controlar el acceso a recursos compartidos mediante la sincronización de procesos o hilos. Solo un proceso puede acceder al recurso compartido cuando el semáforo tiene un valor positivo, evitando así condiciones de carrera.
- Sincronización: Los semáforos permiten sincronizar la ejecución de procesos o hilos, ya que un proceso puede esperar a que se libere un recurso compartido antes de continuar su ejecución.
- Protección de secciones críticas: Los semáforos también se utilizan para proteger secciones críticas de código, evitando que múltiples procesos o hilos las ejecuten simultáneamente.

### **9.4. Monitores**

Los monitores son módulos de programación que encapsulan un recurso compartido y proporcionan métodos y condiciones para acceder a ese recurso de manera segura. Los monitores permiten el acceso exclusivo al recurso compartido, asegurando que solo un proceso o hilo pueda ejecutar una sección crítica del código a la vez.

Características clave de los monitores:

- Encapsulamiento: Los monitores encapsulan un recurso compartido, lo que significa que solo los métodos definidos en el monitor pueden acceder a ese recurso. Esto evita el acceso no autorizado y garantiza que el recurso se utilice correctamente.

- Acceso exclusivo: Los monitores permiten el acceso exclusivo al recurso compartido. Si un proceso o hilo está ejecutando un método del monitor, otros procesos o hilos deben esperar hasta que se libere el recurso antes de poder acceder a él.
- Condiciones y variables de condición: Los monitores pueden utilizar condiciones y variables de condición para gestionar la sincronización y la comunicación entre procesos o hilos. Las condiciones permiten a los procesos o hilos esperar hasta que se cumpla una determinada condición, mientras que las variables de condición proporcionan información adicional para coordinar la ejecución.

**9.5. Algoritmo de Decker:**

El algoritmo de Decker es un algoritmo clásico utilizado para lograr la exclusión mutua distribuida. Se basa en un conjunto de variables compartidas y reglas para coordinar el acceso a un recurso. El algoritmo garantiza que solo un proceso pueda acceder al recurso a la vez, evitando así condiciones de carrera.

Características clave del algoritmo de Decker:

- Variables compartidas: El algoritmo utiliza variables compartidas para controlar el acceso al recurso. Estas variables indican si el turno y el estado de cada proceso en relación con el recurso compartido.
- Reglas de coordinación: El algoritmo de Decker utiliza reglas específicas para coordinar el acceso al recurso compartido. Estas reglas aseguran que solo un proceso tenga el turno para acceder al recurso en un momento dado, mientras que los demás procesos deben esperar.
- Exclusión mutua: El algoritmo garantiza la exclusión mutua, lo que significa que solo un proceso puede acceder al recurso en un momento dado. Esto evita situaciones de conflicto donde múltiples procesos intentan acceder al recurso simultáneamente.

9.6. Algoritmo de Peterson:

El algoritmo de Peterson es otro algoritmo clásico utilizado para garantizar la exclusión mutua en sistemas distribuidos. Al igual que el algoritmo de Decker, utiliza variables compartidas y reglas de turno para coordinar el acceso al recurso compartido.

Características clave del algoritmo de Peterson:

- Variables compartidas: El algoritmo de Peterson utiliza variables compartidas para controlar el acceso al recurso. Estas variables representan el estado y el turno de cada proceso.
- Reglas de turno: El algoritmo utiliza reglas de turno para determinar qué proceso tiene prioridad para acceder al recurso compartido. Los procesos se turnan en base a una variable de turno, lo que garantiza que solo un proceso tenga acceso al recurso a la vez.
- Protocolo de petición y confirmación: El algoritmo de Peterson utiliza un protocolo de petición y confirmación entre los procesos para coordinar el acceso al recurso compartido. Los procesos solicitan el acceso al recurso y esperan la confirmación del otro proceso antes de acceder a él.
- Exclusión mutua: El algoritmo garantiza la exclusión mutua, evitando que múltiples procesos accedan al recurso simultáneamente. Solo un proceso puede acceder al recurso mientras los demás esperan su turno.

Ambos algoritmos, Decker y Peterson, son ejemplos clásicos de mecanismos utilizados para garantizar la exclusión mutua en sistemas distribuidos. Estos algoritmos aseguran que solo un proceso pueda acceder al recurso compartido en un momento dado, evitando condiciones de carrera y conflictos de acceso.

### 9.7. Ejemplo

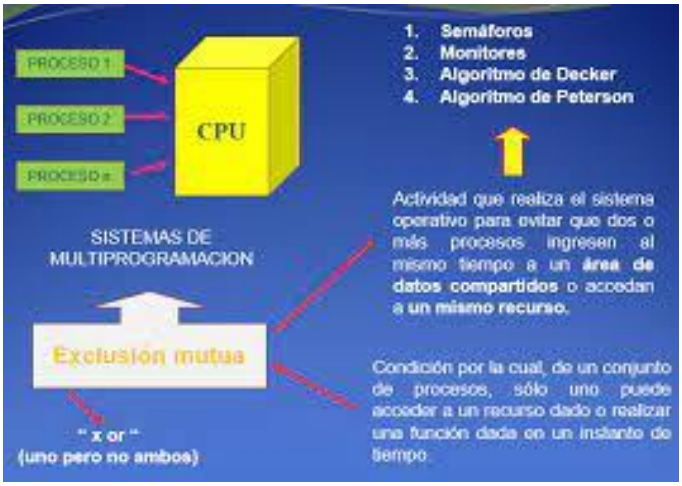
Supongamos que tenemos dos procesos, A y B, que comparten un recurso crítico llamado sección crítica. Ambos procesos desean acceder a la sección crítica, pero solo uno puede hacerlo a la vez para evitar condiciones de carrera.

Para garantizar la exclusión mutua, utilizamos un semáforo binario que actúa como una cerradura para el acceso a la sección crítica. Inicialmente, el semáforo se establece en 1, lo que significa que está desbloqueado y disponible para su uso.

Cuando un proceso quiere acceder a la sección crítica, debe adquirir el semáforo llamando a la operación "esperar". Si el semáforo está en 1, el proceso lo adquiere y accede a la sección crítica. Si el semáforo está en 0 (bloqueado), el proceso se bloquea y espera hasta que el semáforo se libere.

Una vez que el proceso ha terminado de usar la sección crítica, debe liberar el semáforo llamando a la operación "señalar". Esto permite que otros procesos adquieran el semáforo y accedan a la sección crítica.

El uso de semáforos garantiza que solo un proceso pueda acceder a la sección crítica en un momento dado, evitando así problemas de concurrencia y condiciones de carrera.




## 10. Reloges

En el contexto de los sistemas distribuidos, los relojes son dispositivos o mecanismos utilizados para medir el paso del tiempo. Cada nodo en un sistema distribuido puede tener su propio reloj local, que puede funcionar de manera autónoma y sin conocimiento de los demás relojes en el sistema. Sin embargo, debido a la falta de un reloj global común, los relojes locales pueden tener diferentes valores y experimentar desviaciones.

### 10.1. Distorsión de relojes

La distorsión de relojes en sistemas distribuidos puede ser causada por diversos factores. Uno de los principales factores es la diferencia en las velocidades de los relojes locales de los nodos. Debido a la variabilidad en la fabricación de los componentes de los relojes, es común que los relojes locales tengan velocidades ligeramente diferentes, lo que lleva a desviaciones en los valores medidos. Además, los errores de precisión inherentes a los relojes pueden contribuir a la distorsión. Por último, la variabilidad en los retrasos de la red también puede introducir desviaciones en los relojes, ya que los paquetes pueden experimentar diferentes retardos al viajar a través de la red.

### 10.2. Sincronización en sistemas distribuidos

La sincronización de relojes en sistemas distribuidos es esencial para garantizar la coherencia temporal de los eventos y facilitar la comunicación y coordinación entre los nodos. El objetivo de la sincronización es lograr que los relojes locales estén lo más cerca posible en términos de valores y orden temporal. Sin embargo, debido a la distorsión de

los relojes, es difícil lograr una sincronización perfecta. El objetivo es minimizar las desviaciones y asegurarse de que los eventos ocurran en un orden lógico dentro del sistema.

### 10.3. Método de Cristian

El método de Cristian es un algoritmo utilizado para la sincronización de relojes en sistemas distribuidos. En este método, un nodo envía una solicitud de tiempo a un servidor de tiempo centralizado. El servidor responde incluyendo su propio tiempo en la respuesta. El nodo que realiza la solicitud ajusta su reloj local utilizando la diferencia entre su tiempo local y el tiempo proporcionado por el servidor. Este método asume que la latencia de red es predecible y que los tiempos de respuesta son rápidos. Sin embargo, puede ser menos efectivo en entornos de red más complejos o con retrasos variables, ya que no tiene en cuenta las fluctuaciones en los retrasos de la red.

### 10.4. Algoritmo de Berkeley

El algoritmo de Berkeley es otro algoritmo utilizado para la sincronización de relojes en sistemas distribuidos. En este algoritmo, se selecciona un nodo coordinador que se encarga de recopilar los valores de los relojes de todos los nodos en el sistema. El coordinador calcula el valor promedio de los relojes y envía a cada nodo la diferencia entre su valor local y el valor promedio. Cada nodo ajusta su reloj local en función de esta diferencia para acercarse al valor promedio. Este algoritmo es más robusto que el método de Cristian, ya que tiene en cuenta las variaciones en los retrasos de la red y puede adaptarse a diferentes condiciones de red.

### 10.5. Precisión y exactitud de los relojes

En sistemas distribuidos, se busca tanto la precisión como la exactitud en la sincronización de los relojes. La precisión se refiere a la consistencia en la medición del tiempo, mientras que la exactitud se refiere a la cercanía de los valores de los relojes a un tiempo de referencia común, como el tiempo universal coordinado (UTC). La precisión puede mejorar mediante técnicas como el uso de relojes de alta resolución y algoritmos de compensación de la variabilidad en los retrasos de la red. La exactitud se puede lograr mediante la sincronización con fuentes de tiempo confiables, como servidores de tiempo basados en estándares como el Protocolo de Tiempo de Red (NTP).

### 10.6. Distribución geográfica y latencia

La sincronización de relojes en sistemas distribuidos puede ser más desafiante cuando los nodos están ubicados en diferentes lugares geográficos. La latencia de la red, es decir, el tiempo que lleva transmitir un paquete de un nodo a otro, puede variar según la distancia y la congestión de la red. La distorsión de los relojes puede ser más significativa en sistemas distribuidos geográficamente dispersos debido a las diferencias en la latencia de red. Los algoritmos de sincronización deben tener en cuenta estas variaciones en la latencia para lograr una sincronización precisa en todo el sistema.

### 10.7. Ejemplo

Imaginemos un sistema distribuido compuesto por tres nodos: A, B y C, donde cada nodo tiene su propio reloj local. Inicialmente, los relojes de los nodos pueden tener ciertas desviaciones entre sí debido a la variabilidad en la velocidad de los relojes o la distorsión causada por los retrasos de la red. El objetivo es sincronizar estos relojes para tener una noción consistente del tiempo en todo el sistema.

Supongamos que el nodo A actúa como el coordinador del proceso de sincronización. El algoritmo de Berkeley se utiliza para lograr la sincronización. El coordinador A recopila los valores de los relojes de todos los nodos y calcula el promedio. Supongamos que los valores de los relojes son los siguientes:

Nodo A: 10:00:00

Nodo B: 10:00:02

Nodo C: 10:00:05

El coordinador A calcula el promedio de estos valores:

(10:00:00 + 10:00:02 + 10:00:05) / 3 = 10:00:02.333

Despues el coordinador A envía a cada nodo la diferencia entre su valor local y el valor promedio. En este caso, las diferencias serían:

Para el nodo A: 10:00:00 - 10:00:02.333 = -00:00:02.333

Para el nodo B: 10:00:02 - 10:00:02.333 = -00:00:00.333

Para el nodo C: 10:00:05 - 10:00:02.333 = 00:00:02.667

Cada nodo ajusta su reloj local sumando la diferencia recibida al valor promedio. Después de este ajuste, los relojes de los nodos estarán más sincronizados:

Nodo A: 10:00:02.333

Nodo B: 10:00:01.667

Nodo C: 10:00:05.667

Esto representa una mayor consistencia y precisión en los valores de los relojes de los nodos en el sistema distribuido.

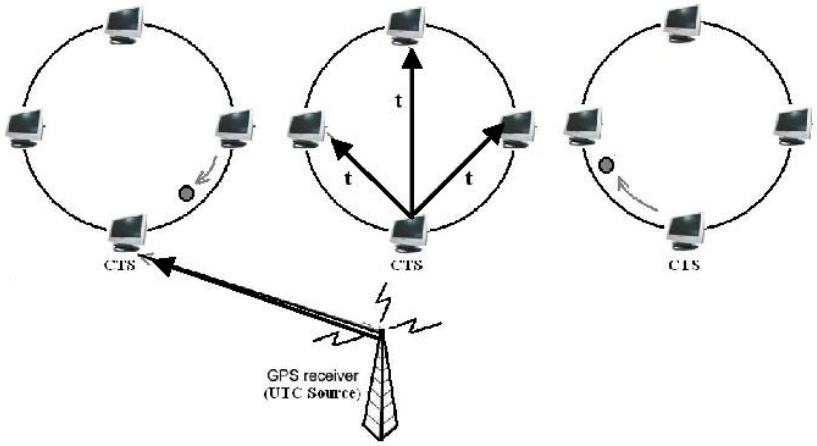



Figure 4. Clock synchronization

## 11. Transacciones

Las transacciones en sistemas de bases de datos son unidades lógicas de trabajo que agrupan un conjunto de operaciones relacionadas que deben ser ejecutadas de manera atómica, consistente, aislada y duradera. Las transacciones aseguran que las modificaciones en la base de datos se realicen de forma coherente y confiable, preservando la integridad de los datos.

### 11.1. Modelo de las transacciones

El modelo de las transacciones define las propiedades y características que deben cumplir las transacciones. Estas propiedades se conocen como las propiedades ACID, que son: Atomicidad, Consistencia, Aislamiento y Durabilidad.

### 11.2. Condiciones de terminación

Las condiciones de terminación de una transacción se refieren a los estados en los que puede finalizar una transacción. Estos estados incluyen la terminación exitosa de la transacción (commit) o su cancelación (rollback). El objetivo es garantizar que las transacciones se completen de manera coherente y sin dejar la base de datos en un estado inconsistente.

### 11.3. Propiedades de las transacciones ACID

Las propiedades ACID son fundamentales para asegurar la integridad y consistencia de las transacciones en sistemas de bases de datos:

- Atomicidad: Una transacción se trata como una unidad atómica de trabajo, lo que significa que todas las operaciones en la transacción se ejecutan como una sola entidad. Si alguna operación falla, se deshacen todas las operaciones anteriores (rollback), y si todas las operaciones son exitosas, se confirman todas (commit).
- Consistencia: Las transacciones deben llevar la base de datos desde un estado válido a otro estado válido. Esto implica que las operaciones dentro de una transacción deben cumplir con todas las restricciones de integridad y reglas definidas en la base de datos.
- Aislamiento: Cada transacción se ejecuta de manera aislada y no debe interferir con otras transacciones concurrentes. Esto significa que los efectos de una transacción en curso no deben ser visibles para otras transacciones hasta que se haya confirmado (commit). El aislamiento se logra mediante mecanismos de control de concurrencia.
- Durabilidad: Una vez que una transacción se ha confirmado (commit), sus efectos deben ser permanentes y sobrevivir a cualquier falla del sistema, incluyendo errores de hardware o software. Los cambios realizados por la transacción deben ser duraderos y estar disponibles incluso después de un reinicio del sistema.

### 11.4. Tipos de transacciones:

Existen diferentes tipos de transacciones en sistemas de bases de datos, que se adaptan a las necesidades y requerimientos específicos de las aplicaciones:

- Transacciones de lectura/escritura: Son transacciones que involucran operaciones tanto de lectura como de escritura en la base de datos. Permiten consultar y modificar los datos almacenados.

- Transacciones de solo lectura: Son transacciones que involucran solo operaciones de lectura en la base de datos. Estas transacciones no realizan modificaciones en los datos y son útiles cuando solo se necesita obtener información de la base de datos sin cambiarla.
- Transacciones anidadas: Son transacciones que pueden incluir otras transacciones dentro de ellas. Esto permite estructurar operaciones complejas y descomponerlas en transacciones más pequeñas y manejables.

### 11.5. Estructura de un sistema de manejo de transacciones:

La estructura de un sistema de manejo de transacciones en un sistema de bases de datos consta de varios componentes que trabajan juntos para garantizar la integridad y la correcta ejecución de las transacciones:

- Gestor de transacciones: Es el componente central del sistema de manejo de transacciones. Se encarga de coordinar y supervisar todas las transacciones en el sistema. Gestiona la creación, ejecución, confirmación o cancelación de transacciones, y garantiza el cumplimiento de las propiedades ACID.
- Registro de transacciones: El registro de transacciones es una estructura persistente en la que se registran todas las operaciones realizadas dentro de una transacción. El registro permite recuperar y reconstruir el estado de la base de datos en caso de fallas o interrupciones del sistema.
- Control de concurrencia: El control de concurrencia se ocupa de gestionar el acceso simultáneo de múltiples transacciones a la base de datos. Utiliza técnicas como bloqueo, control de versiones u otros mecanismos para garantizar el aislamiento entre transacciones y evitar conflictos y anomalías en la ejecución concurrente.
- Gestión de bloqueo: El gestor de bloqueo es responsable de administrar los bloqueos adquiridos por las transacciones en los objetos de la base de datos. Garantiza que no se produzcan conflictos de acceso y mantiene la consistencia y el aislamiento de las transacciones.
- Recuperación y rollback: El sistema de manejo de transacciones debe ser capaz de recuperar la base de datos a un estado consistente en caso de fallas o errores. Utiliza técnicas de recuperación y rollback para deshacer los cambios de las transacciones no confirmadas y restaurar la base de datos a un estado válido.

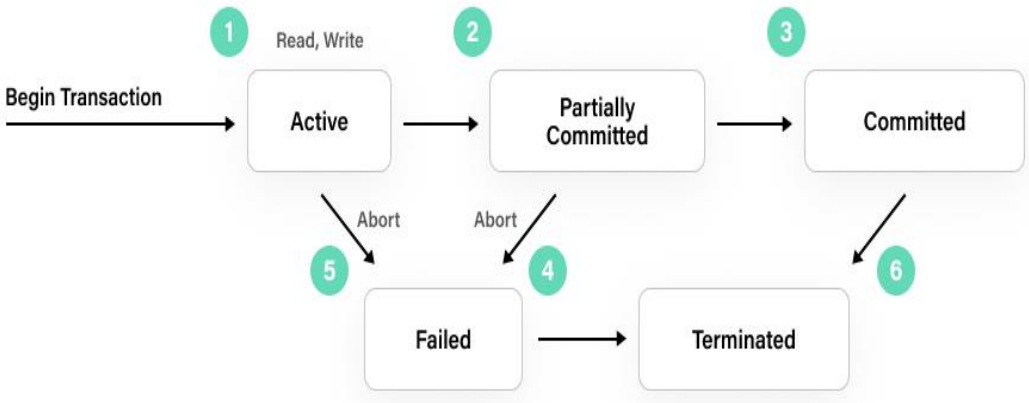
### 11.6. Ejemplo

Imaginemos una base de datos de una tienda en línea que registra los pedidos de los clientes y su información de pago. Supongamos que un cliente realiza un pedido y realiza el pago correspondiente. La operación de registrar el pedido y actualizar la información de pago debe tratarse como una transacción para garantizar la integridad y consistencia de los datos.

En este ejemplo, la transacción se puede dividir en varias operaciones:

- Verificación de disponibilidad: Se verifica si los productos solicitados están disponibles en el inventario.
- Reserva de productos: Se reserva la cantidad adecuada de productos en el inventario para el pedido.
- Registro del pedido: Se registra el pedido del cliente en la base de datos.
- Procesamiento del pago: Se realiza el procesamiento del pago y se actualiza la información de pago del cliente.

Si todas las operaciones se completan con éxito, la transacción se confirma y los cambios se vuelven permanentes en la base de datos. Sin embargo, si ocurre algún error durante cualquiera de las operaciones, la transacción se cancela y se deshacen los cambios realizados.



## 12. DNS

El Sistema de Nombres de Dominio o DNS en ingles es una infraestructura fundamental en Internet que se utiliza para traducir los nombres de dominio legibles por humanos en direcciones IP numéricas. Proporciona un mecanismo de resolución de nombres que permite a los usuarios acceder a recursos en línea utilizando nombres de dominio fáciles de recordar en lugar de tener que recordar las direcciones IP numéricas asociadas.

### 12.1. Funcionamiento del DNS

El DNS opera mediante una jerarquía de servidores distribuidos en todo el mundo. A continuación, se describen los componentes clave del funcionamiento del DNS:

- Nombres de dominio: Los nombres de dominio son las etiquetas legibles por humanos utilizadas para identificar recursos en Internet, como sitios web o servidores de correo electrónico. Ejemplos de nombres de dominio son "google.com" o "facebook.com".
- Servidores de nombres: Los servidores de nombres, también conocidos como servidores DNS, son los encargados de almacenar y proporcionar información sobre los nombres de dominio y sus correspondientes direcciones IP. Hay diferentes tipos de servidores de nombres, como los servidores raíz, los servidores de dominio de nivel superior (TLD) y los servidores autoritativos.
- Resolución de nombres: Cuando un usuario intenta acceder a un recurso en línea utilizando un nombre de dominio, su dispositivo realiza una consulta DNS para obtener la dirección IP correspondiente. Esta consulta se envía a un servidor de nombres, que busca en su base de datos la información asociada al nombre de dominio y devuelve la dirección IP al dispositivo del usuario.

### 12.2. Configuracion

- Servidor DNS: Se necesita un servidor físico o virtual con software de servidor DNS instalado. Algunas opciones populares son BIND, PowerDNS o Windows Server DNS.
- Zonas DNS: Las zonas DNS son áreas lógicas de autoridad dentro de un dominio. Se deben configurar las zonas DNS para especificar los registros de recursos, como las direcciones IP asociadas a los nombres de dominio.
- Registros de recursos: Los registros de recursos son entradas en la zona DNS que especifican cómo se traducen los nombres de dominio en direcciones IP. Algunos ejemplos comunes de registros de recursos son los registros A (que asocian un

nombre de dominio con una dirección IP), los registros MX (que especifican servidores de correo electrónico asociados a un dominio) y los registros CNAME (que permiten establecer alias o nombres alternativos para un dominio).

- Configuración de reenviadores: Los reenviadores DNS se utilizan para enviar las consultas DNS a otros servidores si el servidor DNS local no puede resolver la consulta. Se configuran los reenviadores para asegurar la resolución de nombres adecuada.

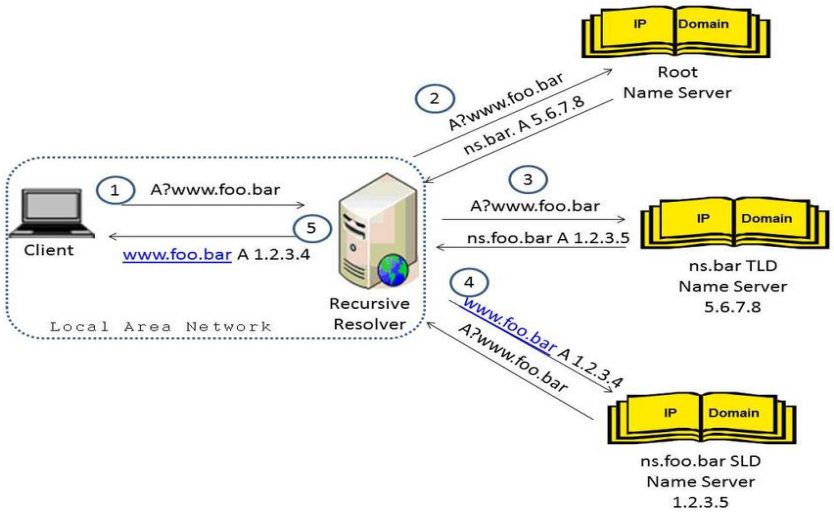
Es importante tener en cuenta que la configuración de un servidor DNS puede variar dependiendo del software utilizado y de los requisitos específicos de la red o del dominio.

### 12.3. Ejemplo

Supongamos que deseas acceder al sitio web de una popular empresa de tecnología llamada "TechCo". En lugar de recordar la dirección IP numérica asociada al sitio, utilizas el nombre de dominio "techco.com". El DNS se encarga de traducir ese nombre de dominio en la dirección IP correspondiente para que puedas acceder al sitio web.

Cuando escribes "techco.com" en el navegador, tu dispositivo realiza una consulta DNS para obtener la dirección IP asociada. La consulta se envía al servidor DNS configurado en tu dispositivo o al servidor DNS de tu proveedor de servicios de Internet (ISP). El servidor DNS busca en su base de datos local la información de "techco.com" y, si no la encuentra, consulta a otros servidores DNS en la jerarquía hasta obtener la respuesta.

Una vez que se encuentra la dirección IP asociada a "techco.com", el servidor DNS la devuelve a tu dispositivo. Ahora, tu dispositivo puede establecer una conexión directa con la dirección IP proporcionada para cargar el sitio web de TechCo.



## 13. NGINX

NGINX es un servidor web ligero, de alto rendimiento y escalable que se utiliza para servir contenido web estático y dinámico. También puede funcionar como un servidor proxy inverso, un equilibrador de carga y un servidor de caché. NGINX se destaca por su eficiencia, su capacidad para manejar grandes cantidades de tráfico y su capacidad para escalar de manera efectiva.

### 13.1. Configuración necesaria para un servidor de NGINX

1. Instalación de NGINX:

Para configurar un servidor de NGINX, debes comenzar por instalar NGINX en tu sistema. Esto puede hacerse mediante el uso de gestores de paquetes específicos del sistema operativo o descargando los binarios desde el sitio web oficial de NGINX.

2. Archivos de configuración:

NGINX utiliza archivos de configuración para definir cómo debe comportarse el servidor. El archivo principal de configuración es `nginx.conf`. Aquí puedes especificar los ajustes globales del servidor y cargar otros archivos de configuración para los sitios web específicos.

3. Configuración de sitios web:

NGINX puede servir múltiples sitios web en el mismo servidor. Para cada sitio web, debes crear un archivo de configuración separado que define la ubicación de los archivos estáticos, las reglas de enrutamiento y otros ajustes específicos.

4. Configuración de puertos y direcciones IP:

NGINX se puede configurar para escuchar en diferentes puertos y direcciones IP. Puedes especificar los puertos y las direcciones IP en la sección de configuración del servidor para controlar cómo NGINX acepta las conexiones entrantes.

5. SSL/TLS:

Si deseas habilitar HTTPS en tu servidor de NGINX y asegurar las comunicaciones, necesitarás configurar certificados SSL/TLS. Esto implica generar o adquirir un certificado SSL y configurar NGINX para utilizarlo.

6. Reinicio y recarga del servidor:

Después de realizar cambios en la configuración de NGINX, debes reiniciar o recargar el servidor para que los cambios surtan efecto. Puedes hacerlo ejecutando comandos específicos según tu sistema operativo.

### 13.2. Ejemplo

Imaginemos que deseas configurar un servidor de NGINX para alojar un sitio web estático. A continuación, se presenta un ejemplo de cómo podría ser la configuración:

I. Instalación:

Utiliza el gestor de paquetes de tu sistema operativo para instalar NGINX. Por ejemplo, en Ubuntu puedes ejecutar el siguiente comando: sudo apt-get install nginx.

II. Archivo de configuración:

Abre el archivo de configuración principal de NGINX ubicado en /etc/nginx/nginx.conf.

Asegúrate de que la directiva worker_processes esté configurada con el número adecuado de procesos para tu sistema.

Verifica la sección http para asegurarte de que los ajustes globales sean los adecuados, como el user y los include de archivos de configuración adicionales.

III. Configuración del sitio web:

Crea un nuevo archivo de configuración para tu sitio web en el directorio /etc/nginx/conf.d/. Por ejemplo, mi_sitio.conf.

Dentro de ese archivo, define la configuración específica de tu sitio web, como el nombre del servidor, los archivos de acceso y error, y las reglas de enrutamiento. Asegúrate de especificar la ubicación de los archivos estáticos del sitio web.

IV. Configuración de puertos y direcciones IP:

En el archivo de configuración principal de NGINX (nginx.conf), verifica la sección http y asegúrate de que el servidor esté configurado para escuchar en el puerto adecuado, como el puerto 80 para HTTP.

Si deseas especificar una dirección IP específica en la que NGINX escucha, puedes configurarlo utilizando la directiva listen seguida de la dirección IP deseada.

V. Reinicio y verificación:

Reinicia NGINX ejecutando el comando sudo service nginx restart o el comando específico de reinicio de tu sistema operativo.

Verifica que tu sitio web esté accesible ingresando la dirección IP o el dominio en un navegador web.


**14. Docker**

Docker es una plataforma de código abierto que permite crear, desplegar y ejecutar aplicaciones de forma aislada utilizando contenedores. Los contenedores son entornos ligeros y portátiles que encapsulan todas las dependencias y configuraciones necesarias para que una aplicación se ejecute de manera consistente en diferentes entornos.

- Contenedores:

  Los contenedores son unidades de ejecución aisladas que contienen todas las dependencias de una aplicación, incluidos el código, las bibliotecas y los archivos de configuración. Utilizando tecnologías de virtualización a nivel de sistema operativo, Docker proporciona una forma eficiente de empaquetar y distribuir aplicaciones, garantizando que se ejecuten de manera confiable en cualquier entorno.
- Imágenes:

  Las imágenes de Docker son plantillas de solo lectura que se utilizan para crear contenedores. Una imagen contiene un sistema de archivos completo con todos los componentes necesarios para ejecutar una aplicación. Las imágenes se crean

utilizando un archivo de configuración llamado Dockerfile, que especifica las instrucciones para construir la imagen.

- Docker Hub:

  Docker Hub es un registro de imágenes públicas y privadas mantenido por Docker. Es una plataforma centralizada donde los desarrolladores pueden compartir y descargar imágenes de Docker predefinidas para acelerar el proceso de desarrollo y despliegue de aplicaciones. También permite la colaboración y la implementación de imágenes personalizadas.

- Comandos de Docker:

  Docker proporciona una interfaz de línea de comandos para administrar y controlar contenedores, imágenes y otros recursos relacionados. Algunos comandos comunes incluyen `docker run` para crear y ejecutar contenedores, `docker build` para construir imágenes a partir de Dockerfiles, y `docker push` y `docker pull` para subir y descargar imágenes de Docker Hub.

### 14.1. Configuracion de instalacion de docker

La documentación oficial de Docker proporciona instrucciones detalladas sobre la configuración necesaria para instalar Docker en Debian, se puede revisar el siguiente link https://docs.docker.com/engine/install/debian/

A continuación, te resumo los pasos principales según la documentación oficial:

1. Actualiza el sistema:

```
sudo apt update

sudo apt upgrade
```

2. Asegúrate de tener los paquetes necesarios para permitir que el sistema apt utilice repositorios sobre HTTPS:

```
sudo apt install apt-transport-https ca-certificates curl gnupg lsb-release
```

3. Agrega la clave GPG oficial de Docker:

```
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

4. Configura el repositorio estable de Docker:

echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/debian $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

5. Instala Docker Engine:

sudo apt update

sudo apt install docker-ce docker-ce-cli containerd.io

6. Verifica la instalación:

sudo docker run hello-world

Estos son los pasos básicos para instalar Docker en Debian según la documentación oficial. Recuerda consultar la documentación oficial de Docker para obtener instrucciones más detalladas y actualizadas, ya que los pasos pueden variar según la versión específica de Debian y las configuraciones adicionales que desees aplicar.

### 14.2. Ejemplo

Aquí tienes un ejemplo de cómo utilizar el contenedor de Docker "dperson/samba" para crear un servidor Samba:

I. Descargar la imagen del contenedor de Docker desde Docker Hub:

docker pull dperson/samba

II. Crear un directorio para compartir archivos y asignar los permisos adecuados:

mkdir /ruta/a/directorio_compartido

chmod 777 /ruta/a/directorio_compartido

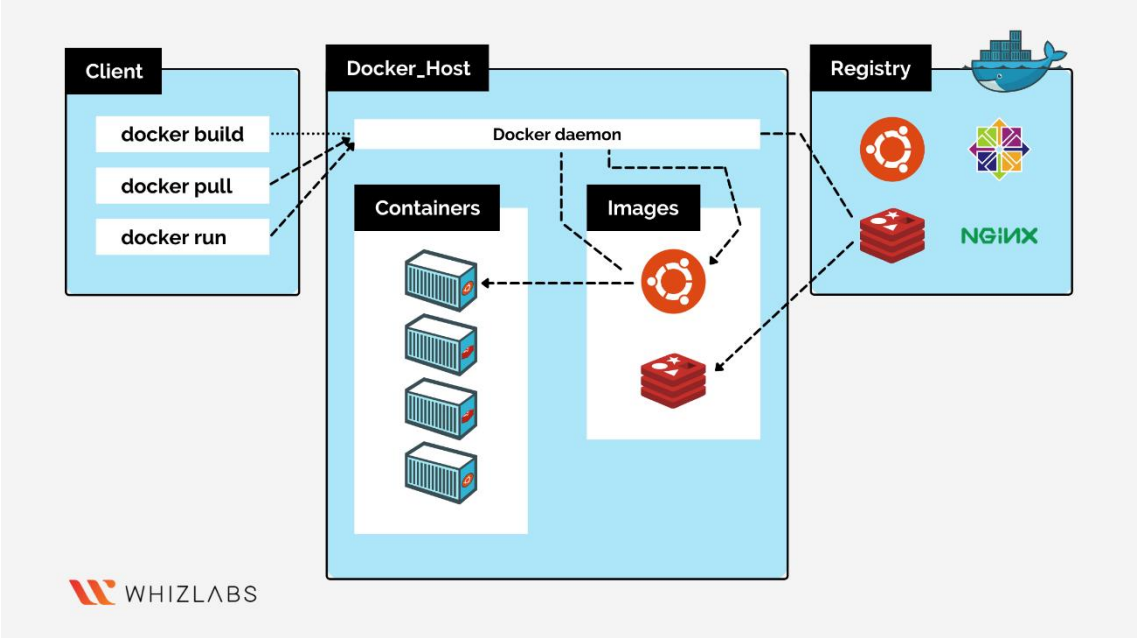
III. Ejecutar el contenedor de Docker, mapeando el directorio compartido y configurando los usuarios y contraseñas para Samba:

docker run -it -p 445:445 -p 139:139 -v /ruta/a/directorio_compartido:/mount -e USER=username -e PASSWORD=password --name samba-server dperson/samba

Asegúrate de reemplazar "/ruta/a/directorio_compartido" con la ruta al directorio que deseas compartir, "username" con el nombre de usuario que deseas utilizar y "password" con la contraseña correspondiente.

IV. Acceder al servidor Samba desde otro dispositivo en la red utilizando el nombre de host o dirección IP del servidor y las credenciales que configuraste.

Este esun ejemplo para ejecutar un servidor Samba utilizando Docker y la imagen "dperson/samba". Puedes consultar la documentación de la imagen en Docker Hub para obtener más detalles sobre las opciones de configuración adicionales disponibles. https://hub.docker.com/r/dperson/samba




## 15. PROXMOX

Proxmox es una plataforma de virtualización de código abierto que combina la virtualización basada en contenedores y la virtualización basada en hipervisor en un solo sistema. Utiliza tecnologías como KVM y LXC para ofrecer una solución integral de gestión de infraestructuras de servidores virtuales y contenedores. Esto permite a los usuarios crear y administrar de manera eficiente entornos virtuales, aprovechando las ventajas de ambas tecnologías en un entorno unificado.

### 15.1. Beneficios de Proxmox:

Gestión centralizadA: Proxmox ofrece una interfaz de administración centralizada que permite a los usuarios gestionar y monitorizar sus máquinas virtuales y contenedores desde un solo lugar. Esto simplifica la administración y proporciona una visión general de la infraestructura.

Flexibilidad: Proxmox ofrece opciones flexibles de virtualización, lo que permite a los usuarios elegir entre la virtualización basada en hipervisor (KVM) y la virtualización basada en contenedores (LXC) según sus necesidades. Esto brinda la posibilidad de utilizar la tecnología más adecuada para cada caso de uso.

Alta disponibilidad: Proxmox cuenta con funciones de alta disponibilidad que permiten la migración en vivo de máquinas virtuales y contenedores entre servidores físicos sin tiempo de inactividad. Esto garantiza la continuidad del servicio y mejora la fiabilidad de la infraestructura.

Seguridad: Proxmox ofrece medidas de seguridad integradas, como el aislamiento de recursos y la autenticación de dos factores, para proteger los entornos virtuales. Además, permite implementar políticas de seguridad personalizadas según los requisitos específicos de cada usuario.

### 15.2. Ejemplo

Supongamos que tienes un entorno empresarial y deseas utilizar Proxmox para virtualizar tus servidores y contenedores. A continuación, se muestra un ejemplo de cómo puedes utilizar Proxmox:

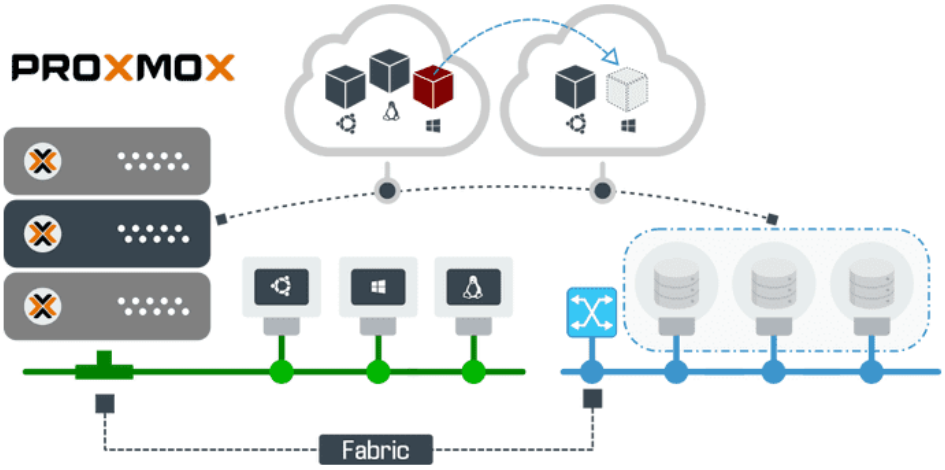
1. Instala Proxmox en un servidor físico dedicado.

2. Utiliza la interfaz de administración web de Proxmox para crear máquinas virtuales y contenedores según tus necesidades. Puedes asignar recursos como CPU, memoria y almacenamiento a cada máquina virtual o contenedor.

3. Configura la red virtual dentro de Proxmox para permitir la comunicación entre las máquinas virtuales y los contenedores.

4. Administra y monitoriza tus máquinas virtuales y contenedores a través de la interfaz de administración de Proxmox. Puedes realizar tareas como iniciar, detener, migrar y realizar copias de seguridad de tus instancias virtuales.

5. Aprovecha las funciones de alta disponibilidad de Proxmox para garantizar la continuidad del servicio. Puedes migrar máquinas virtuales y contenedores en vivo entre servidores físicos para evitar tiempos de inactividad.




## 16. Zabbix

Zabbix funciona mediante la instalación de agentes en los dispositivos que se desean monitorear, estos agentes recopilan datos y los envían al servidor central de Zabbix. El servidor procesa y almacena los datos, y ofrece una interfaz web donde los usuarios pueden visualizar y analizar la información recolectada. Zabbix también permite definir reglas de alerta y notificaciones para recibir avisos en caso de eventos o condiciones específicas.

### 16.1. Configuración de Zabbix

La configuración de Zabbix implica varios pasos. Primero, se debe instalar el servidor central de Zabbix, junto con la base de datos y el frontend web. Luego, se configuran los agentes en los dispositivos a monitorear para establecer la comunicación con el servidor. Se definen los elementos de monitoreo, que son los indicadores específicos que se desean supervisar, como el uso de CPU, la memoria, el ancho de banda, etc. Además, se pueden configurar triggers y acciones para definir umbrales y notificaciones en caso de eventos anormales.

### 16.2. Ejemplo con Zabbix

Supongamos que deseamos monitorear el rendimiento de una red de servidores en un entorno empresarial. Configuramos Zabbix instalando el servidor central en una máquina dedicada y los agentes en cada servidor de la red. Definimos elementos de monitoreo como el uso de CPU, la carga del sistema y el espacio de almacenamiento. Establecemos triggers para recibir alertas cuando los niveles de uso excedan ciertos umbrales. A través de la interfaz web de Zabbix, podemos visualizar gráficas y reportes en tiempo real, identificar cuellos de botella y tomar acciones correctivas para optimizar el rendimiento de la red de servidores.


## 17. Veeam Backup & Replication Community Edition

Veeam Backup & Replication Community Edition es una solución de respaldo y replicación de máquinas virtuales (VM) gratuita que ofrece una amplia gama de funcionalidades. A continuación, se detallan los diferentes aspectos de Veeam Backup & Replication Community Edition.

### 17.1. Cómo funciona Veeam Backup & Replication Community Edition

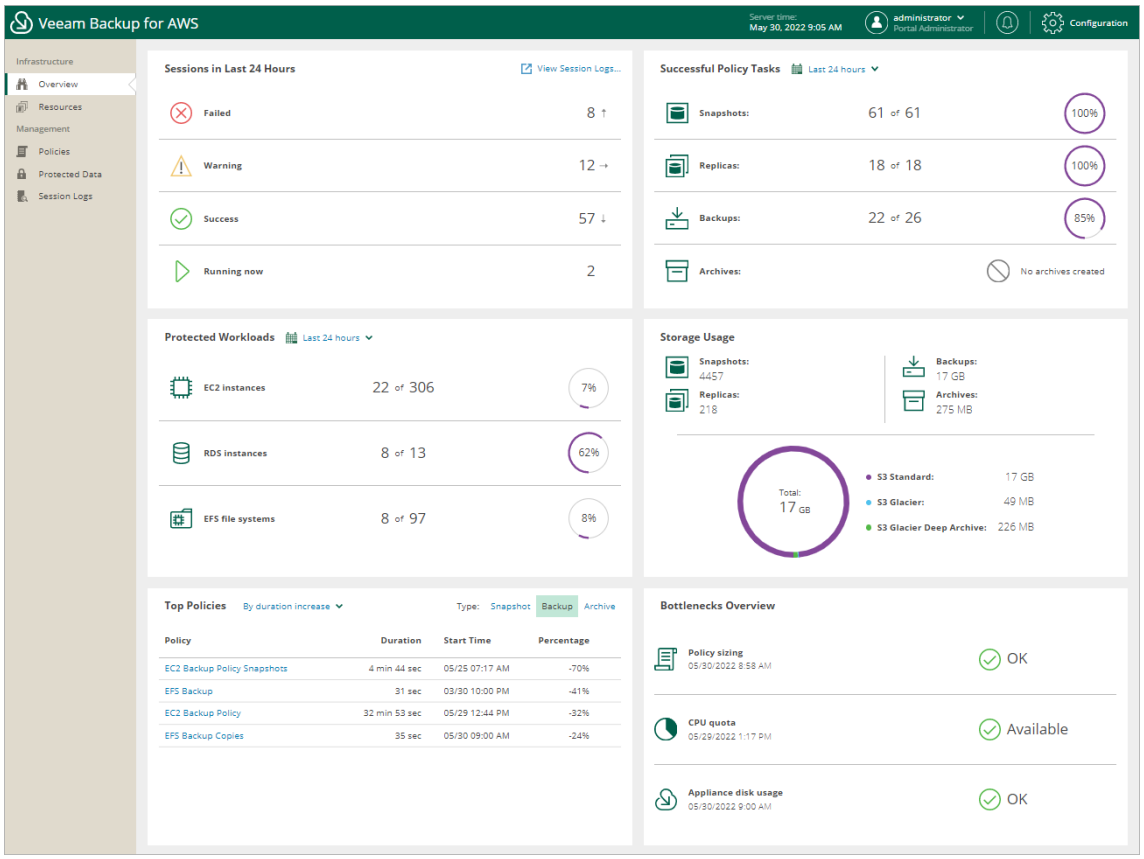
Veeam Backup & Replication Community Edition utiliza la tecnología de snapshot (instantánea) para realizar copias de seguridad de VM completas, incrementales y diferenciales. Esta solución se integra con las plataformas de virtualización líderes, como VMware vSphere y Microsoft Hyper-V, y permite la protección de VM en entornos virtuales. Utiliza agentes de respaldo instalados en los hosts de virtualización para realizar copias de seguridad de manera eficiente y confiable.

### 17.2. Configuración de instalación de Veeam Backup & Replication Community Edition

La instalación de Veeam Backup & Replication Community Edition implica seguir algunos pasos básicos. Se debe descargar el archivo de instalación desde el sitio oficial de Veeam y ejecutarlo en el servidor designado para el respaldo y la replicación. Durante el proceso de instalación, se deben configurar las opciones de conexión a la infraestructura de virtualización y las preferencias de almacenamiento para los respaldos. Una vez completada la instalación, se puede acceder a la interfaz de administración de Veeam para configurar y gestionar las tareas de respaldo.

### 17.3. Ejemplo de respaldo con Veeam Backup & Replication Community Edition:

Supongamos que queremos realizar una copia de seguridad de una VM crítica en nuestro entorno de virtualización VMware. Configuramos Veeam Backup & Replication Community Edition instalando el software en un servidor designado y conectándolo al entorno de VMware. A través de la interfaz de administración, configuramos una tarea de respaldo programada para la VM objetivo. Podemos especificar la frecuencia y el tipo de respaldo (completo, incremental, diferencial), así como las políticas de retención. Una vez configurada la tarea, Veeam realizará automáticamente las copias de seguridad según lo programado, asegurando la disponibilidad de los datos y la capacidad de recuperación ante fallos.




## 18. Conclusión

En conclusión, el estudio de los sistemas distribuidos abarca una amplia gama de temas fundamentales que son esenciales para poder implemntarlo en prodiccion. Durante el desarrollo de la materia de Sistemas Distribuidos, hemos explorado en detalle conceptos y tecnologías clave que desempeñan un papel fundamental en estos sistemas.

Hemos analizado el concepto de RAID y las técnicas de almacenamiento redundante utilizadas para garantizar la protección de datos y la tolerancia a fallos. Además, hemos profundizado en el uso de LVM como una herramienta esencial para la gestión flexible de volúmenes lógicos en sistemas Linux.

La comunicación entre los componentes de un sistema distribuido, los protocolos utilizados y los mecanismos para abordar posibles fallos también han sido vistos y estudiados, además hemos explorado la seguridad de los datos en sistemas distribuidos, haciendo hincapié en el uso de llaves asimétricas para garantizar la confidencialidad de la información transmitida entre los nodos.

La replicación ha sido otro tema importante, por el hecho de comprender el cómo mantener copias idénticas de datos o servicios en diferentes nodos además de mantener asegura la disponibilidad continua de la información.

La sincronización del tiempo entre los nodos y el concepto de transacciones han sido vists para lograr una coordinación precisa y consistente en sistemas distribuidos. Además, hemos explorado las propiedades ACID que rigen las transacciones en este entorno.

Aprate de los aspectos teóricos, es fundamental comprender la funcionalidad práctica de estas tecnologías. La búsqueda de documentación oficial y el acceso a recursos prácticos son aspectos esenciales para adquirir el conocimiento necesario y mejorar nuestras habilidades en la implementación y configuración de estas soluciones en entornos reales.

De manera general la materia de Sistemas Distribuidos nos ha brindado una sólida base de conocimientos para comprender y utilizar eficientemente los elementos clave en el diseño, configuración y administración de sistemas distribuidos. Es importante reconocer la importancia de la funcionalidad práctica y continuar explorando documentación y recursos adicionales para mejorar y ampliar nuestro dominio en este ambiente que va creciendo cada vez mas con nuevas tecnologías y de mas.