# Sistemas de archivos distribuidos

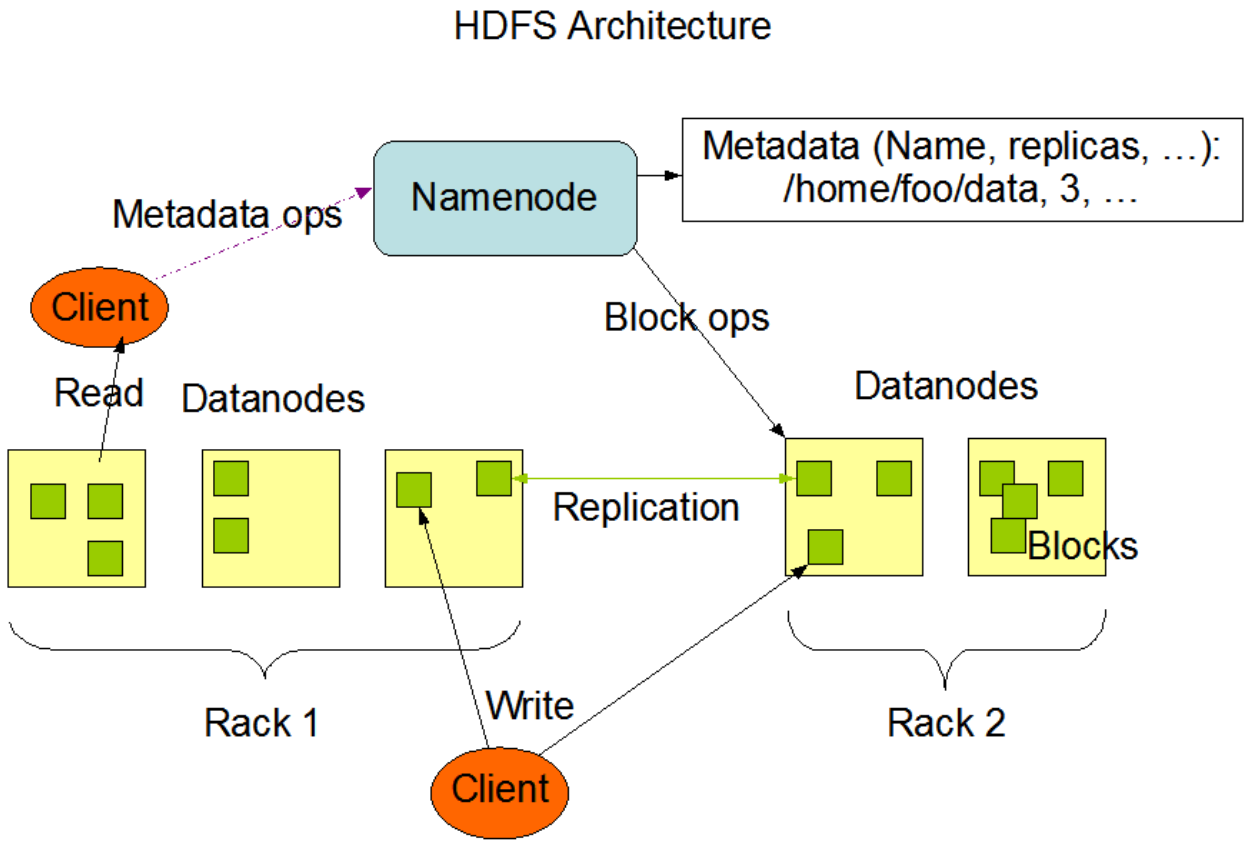
## Hadoop Distributed File System (HDFS)

HDFS es un sistema de archivos distribuido desarrollado como parte del proyecto Apache Hadoop. Está diseñado para proporcionar almacenamiento redundante confiable y de alto rendimiento para datos grandes. Es ampliamente utilizado en la industria para almacenar y gestionar grandes conjuntos de datos en un entorno distribuido.

Inspirado en Google File System (GFS), HDFS está optimizado para manejar grandes archivos y es utilizado extensivamente en la industria para soportar aplicaciones de big data y análisis.

## Arquitectura de Hadoop Distributed File System (HDFS)

HDFS sigue una arquitectura maestro-esclavo y está compuesto por varios componentes clave que trabajan juntos para proporcionar almacenamiento distribuido.




Los componentes principales son:

- Namenode
- Datanode
- Secondary Namenode
- Checkpoint Node

- Backup Node

**Namenode**

El Namenode es el nodo maestro en la arquitectura HDFS y tiene la responsabilidad de gestionar el sistema de archivos y la regulación del acceso a los datos por parte de los clientes. Sus funciones principales incluyen:

- Gestión de metadatos: El Namenode mantiene los metadatos de HDFS, que incluyen la estructura del sistema de archivos (árbol de directorios y archivos) y la ubicación de los bloques de datos. Estos metadatos se almacenan en la memoria para un acceso rápido y también se guardan en el disco para persistencia.
- Control de acceso: Gestiona las operaciones de creación, eliminación, y modificación de archivos y directorios. Autentica y autoriza las solicitudes de los clientes para acceder a los datos.
- Coordinación de bloques: Asigna bloques de datos a los Datanodes para almacenamiento y gestiona la replicación de bloques para garantizar la redundancia y tolerancia a fallos. También mantiene un registro de los Datanodes que contienen cada bloque.

**Datanode**

Los Datanodes son los nodos esclavos en HDFS y son responsables de almacenar los bloques de datos reales. Cada Datanode ejecuta una instancia de servicio que se comunica con el Namenode y realiza las siguientes funciones:

- Almacenamiento de datos: Los Datanodes almacenan y recuperan bloques de datos en respuesta a las solicitudes de lectura y escritura de los clientes del sistema de archivos.
- Gestión de bloques: Informan regularmente al Namenode sobre los bloques de datos que almacenan. Este informe incluye información sobre el estado y la integridad de los bloques.
- Replicación de bloques: En caso de fallo de un Datanode, el Namenode coordina la replicación de bloques a otros Datanodes para asegurar que se mantenga el nivel de replicación especificado.

**Secondary Namenode**

El Secondary Namenode no es un nodo de respaldo del Namenode, como su nombre podría sugerir. En cambio, su función principal es ayudar a aliviar la carga del Namenode mediante la creación de checkpoints del estado de HDFS. Sus responsabilidades incluyen:

- Creación de Checkpoints: Periódicamente, el Secondary Namenode toma una instantánea del sistema de archivos en el Namenode y fusiona los archivos de registro de edición (edit logs) con la imagen del sistema de archivos (fsimage) para crear una nueva imagen de checkpoint. Este proceso reduce el tamaño de los archivos de registro y ayuda a prevenir la sobrecarga del Namenode.

**Checkpoint Node y Backup Node**

El Checkpoint Node y el Backup Node son extensiones del concepto del Secondary Namenode. Ambos tienen roles similares en la creación de checkpoints, pero con algunas diferencias:

- Checkpoint Node: Similar al Secondary Namenode, el Checkpoint Node también crea checkpoints del sistema de archivos. Se diferencia en que puede ejecutarse en una máquina separada y se comunica con el Namenode para obtener los archivos de registro y la imagen del sistema de archivos.
- Backup Node: Actúa como un nodo de respaldo en tiempo real del Namenode. Almacena una copia de los metadatos del sistema de archivos en memoria y en disco, proporcionando una recuperación rápida en caso de fallo del Namenode. Sin embargo, no participa activamente en la creación de checkpoints como el Secondary Namenode o el Checkpoint Node.

**Funcionamiento de HDFS**

El funcionamiento de HDFS implica varios procesos críticos, incluyendo la lectura y escritura de datos, la replicación de bloques y la recuperación de fallos.

Escritura de datos

1. Solicitud del cliente: Un cliente solicita al Namenode la creación de un archivo. El Namenode verifica la disponibilidad del nombre del archivo y lo registra en sus metadatos.
2. Asignación de bloques: El Namenode asigna bloques para el archivo y selecciona un conjunto de Datanodes para almacenar cada bloque, basado en una política de replicación configurada (por defecto, tres copias).
3. Escritura de datos: El cliente divide el archivo en bloques y escribe cada bloque en los Datanodes seleccionados. Los Datanodes confirman la escritura de los bloques al Namenode.
4. Actualización de Metadatos: Una vez que todos los bloques han sido escritos y replicados, el Namenode actualiza sus metadatos para reflejar la ubicación de los bloques.

Lectura de datos

1. Solicitud del cliente: Un cliente solicita al Namenode la lectura de un archivo. El Namenode proporciona la ubicación de los bloques que componen el archivo.
2. Lectura de bloques: El cliente se comunica directamente con los Datanodes que almacenan los bloques y lee los datos. Si un bloque no está disponible en el primer Datanode, el cliente intenta leerlo de los otros Datanodes replicados.

**Replicación de bloques**

1. Monitoreo de Datanodes: El Namenode monitorea regularmente el estado de los Datanodes a través de informes de latidos (heartbeats) y registros de bloques.
2. Detección de fallos: Si un Datanode no responde en un tiempo configurado, el Namenode lo marca como fallido.
3. Replicación de bloques perdidos: El Namenode identifica los bloques almacenados en el Datanode fallido y coordina la replicación de estos bloques desde los Datanodes restantes a nuevos Datanodes.

**Recuperación de fallos**

1. Fallos del Namenode: En caso de fallo del Namenode, se puede restaurar desde el último checkpoint creado por el Secondary Namenode o el Checkpoint Node.
2. Fallos de Datanodes: Los datos son recuperables a través de las réplicas almacenadas en otros Datanodes. El Namenode coordina la replicación para asegurar la redundancia.

### Ventajas y desventajas de HDFS

**Ventajas**

- Escalabilidad: HDFS está diseñado para escalar a petabytes de datos distribuidos a través de miles de nodos.
- Tolerancia a fallos: La replicación de bloques y la detección proactiva de fallos aseguran una alta disponibilidad de datos.
- Economía de costos: Utiliza hardware común en lugar de servidores de alto costo, reduciendo los costos de infraestructura.

**Desventajas**

- Sobrecarga del Namenode: Mantener todos los metadatos en memoria puede convertirse en un cuello de botella, especialmente en grandes clústeres.
- Tamaño de bloque fijo: No es óptimo para archivos pequeños debido a la sobrecarga de metadatos y la gestión de bloques.
- Desempeño de escritura: Las operaciones de escritura son costosas debido a la replicación y la coordinación entre múltiples nodos.

La arquitectura de HDFS está diseñada para proporcionar un almacenamiento distribuido eficiente, escalable y tolerante a fallos para aplicaciones de big data. Aunque enfrenta desafíos como la sobrecarga del Namenode y la gestión de archivos pequeños, sus ventajas en términos de escalabilidad, redundancia y economía de costos lo convierten en una elección popular para manejar grandes volúmenes de datos en entornos distribuidos.

## Frameworks y herramientas relacionadas

1. Apache Hadoop: HDFS es un componente central del ecosistema Apache Hadoop, que también incluye MapReduce para procesamiento de datos y YARN para la gestión de recursos. Hadoop proporciona una plataforma completa para el procesamiento y almacenamiento de grandes volúmenes de datos distribuidos.
2. Apache Hive: Un sistema de data warehousing que se ejecuta encima de Hadoop y permite consultas SQL sobre grandes conjuntos de datos almacenados en HDFS.
3. Apache Pig: Un lenguaje de alto nivel para el análisis de datos en Hadoop. Permite escribir scripts para tareas complejas de procesamiento de datos.
4. Apache HBase: Una base de datos distribuida y no relacional que se ejecuta sobre HDFS, proporcionando capacidades de almacenamiento y acceso a datos de baja latencia.
5. Apache Flume: Un servicio para recopilar, agregar y mover grandes cantidades de datos de registros a HDFS.
6. Apache Spark: Un motor de análisis de datos de código abierto que puede procesar datos almacenados en HDFS de manera más rápida y eficiente que MapReduce, utilizando un modelo de programación más flexible.

7. Presto: Un motor de consultas distribuido de alto rendimiento desarrollado originalmente por Facebook, que permite realizar consultas SQL interactivas sobre HDFS.

**Integraciones y herramientas adicionales**

1. Apache Oozie: Un sistema de flujo de trabajo para la gestión de trabajos en Hadoop. Permite programar y coordinar trabajos de HDFS, MapReduce, Pig, y otros.
2. Apache Sqoop: Una herramienta diseñada para transferir datos entre HDFS y bases de datos relacionales.
3. Apache Kafka: Una plataforma de transmisión de datos distribuida que puede integrarse con HDFS para almacenar y procesar flujos de datos en tiempo real.
4. Hadoop Common: Bibliotecas y utilidades que soportan otros módulos Hadoop.

## Google File System (GFS)

GFS es un sistema de archivos distribuido propietario desarrollado por Google para manejar grandes cantidades de datos en aplicaciones que requieren un rendimiento alto y escalabilidad masiva. Fue diseñado para satisfacer las necesidades específicas de los servicios de Google y ha influido en el diseño de muchos otros sistemas de archivos distribuidos.

## Arquitectura de GFS

GFS sigue una arquitectura maestro-esclavo que incluye los siguientes componentes principales:

- Master Node
- Chunkservers
- Clients

**Master Node**

El Master Node es el nodo maestro en la arquitectura GFS y tiene la responsabilidad de gestionar los metadatos del sistema de archivos y coordinar las operaciones de acceso a los datos. Sus funciones principales incluyen:

- Gestión de metadatos: El Master Node mantiene metadatos sobre los archivos y directorios en GFS. Estos metadatos incluyen la estructura del sistema de archivos, la ubicación de los chunks, las versiones de los chunks, y la información de replicación. Los metadatos se almacenan en la memoria para un acceso rápido y también se guardan en el disco para persistencia.
- Control de acceso: El Master Node maneja las solicitudes de los clientes para crear, eliminar y modificar archivos y directorios. Autentica y autoriza las operaciones de los clientes y gestiona los permisos de acceso.
- Coordinación de chunks: El Master Node asigna chunks a los Chunkservers y gestiona la replicación de chunks para asegurar la redundancia y tolerancia a fallos. También mantiene un registro de los Chunkservers que almacenan cada chunk.

**Chunkservers**

Los Chunkservers son los nodos esclavos en GFS y son responsables de almacenar los chunks de datos reales. Cada Chunkserver ejecuta una instancia de servicio que se comunica con el Master Node y realiza las siguientes funciones:

- Almacenamiento de datos: Los Chunkservers almacenan y recuperan chunks de datos en respuesta a las solicitudes de lectura y escritura de los clientes del sistema de archivos.
- Gestión de chunks: Informan regularmente al Master Node sobre los chunks de datos que almacenan. Este informe incluye información sobre el estado y la integridad de los chunks.
- Replicación de chunks: En caso de fallo de un Chunkserver, el Master Node coordina la replicación de chunks a otros Chunkservers para asegurar que se mantenga el nivel de replicación especificado.

**Clients**

Los Clients son las aplicaciones que interactúan con GFS para leer y escribir datos. Los Clients se comunican primero con el Master Node para obtener la ubicación de los chunks y luego interactúan directamente con los Chunkservers para acceder a los datos. Esto reduce la carga sobre el Master Node y mejora la escalabilidad del sistema.

## Funcionamiento de GFS

El funcionamiento de GFS implica varios procesos críticos, incluyendo la lectura y escritura de datos, la replicación de chunks y la recuperación de fallos.

Escritura de datos

1. Solicitud del cliente: Un cliente solicita al Master Node la creación de un archivo. El Master Node verifica la disponibilidad del nombre del archivo y lo registra en sus metadatos.

2. Asignación de Chunks: El Master Node asigna chunks para el archivo y selecciona un conjunto de Chunkservers para almacenar cada chunk, basado en una política de replicación configurada (por defecto, tres copias).

3. Escritura de datos: El cliente divide el archivo en chunks y escribe cada chunk en los Chunkservers seleccionados. Los Chunkservers confirman la escritura de los chunks al Master Node.

4. Actualización de metadatos: Una vez que todos los chunks han sido escritos y replicados, el Master Node actualiza sus metadatos para reflejar la ubicación de los chunks.

Lectura de datos

1. Solicitud del cliente: Un cliente solicita al Master Node la lectura de un archivo. El Master Node proporciona la ubicación de los chunks que componen el archivo.
2. Lectura de chunks: El cliente se comunica directamente con los Chunkservers que almacenan los chunks y lee los datos. Si un chunk no está disponible en el primer Chunkserver, el cliente intenta leerlo de los otros Chunkservers replicados.

Replicación de Chunks

1. Monitoreo de Chunkservers: El Master Node monitorea regularmente el estado de los Chunkservers a través de informes de latidos (heartbeats) y registros de chunks.
2. Detección de fallos: Si un Chunkserver no responde en un tiempo configurado, el Master Node lo marca como fallido.
3. Replicación de chunks perdidos: El Master Node identifica los chunks almacenados en el Chunkserver fallido y coordina la replicación de estos chunks desde los Chunkservers restantes a nuevos Chunkservers.

Recuperación de fallos

1. Fallos del Master Node: En caso de fallo del Master Node, se puede restaurar desde el último checkpoint y los registros de operaciones (operation logs) guardados en el disco.
2. Fallos de Chunkservers: Los datos son recuperables a través de las réplicas almacenadas en otros Chunkservers. El Master Node coordina la replicación para asegurar la redundancia.

## Ventajas y desventajas de GFS

Ventajas

- Escalabilidad: GFS está diseñado para escalar a petabytes de datos distribuidos a través de miles de nodos.
- Tolerancia a fallos: La replicación de chunks y la detección proactiva de fallos aseguran una alta disponibilidad de datos.
- Optimización para lecturas y escrituras por anexado: GFS está optimizado para operaciones de lectura frecuentes y escrituras por anexado, lo cual es ideal para aplicaciones de procesamiento de datos masivos.

Desventajas

- Complejidad del master node: Al mantener todos los metadatos y coordinar las operaciones, el Master Node puede convertirse en un cuello de botella.
- Tamaño fijo de Chunks: Utilizar chunks de tamaño fijo (64 MB) puede no ser óptimo para todas las aplicaciones, especialmente aquellas que manejan archivos pequeños.
- Limitaciones en las escrituras aleatorias: GFS no está diseñado para operaciones de escritura aleatoria, lo que puede ser una limitación para ciertas aplicaciones.

## Frameworks y herramientas relacionadas

1. MapReduce: El modelo de programación y procesamiento de datos distribuido original desarrollado por Google, estrechamente integrado con GFS para el procesamiento eficiente de grandes conjuntos de datos.
2. Bigtable: Una base de datos distribuida de Google, diseñada para manejar grandes cantidades de datos estructurados y semiestructurados. Utiliza GFS para almacenamiento de datos.
3. Google Borg: Un sistema de gestión de clústeres que automatiza la ejecución, el seguimiento y la gestión de aplicaciones en los clústeres de Google, utilizando GFS para el almacenamiento de datos.

**Servicios y herramientas de Google Cloud**

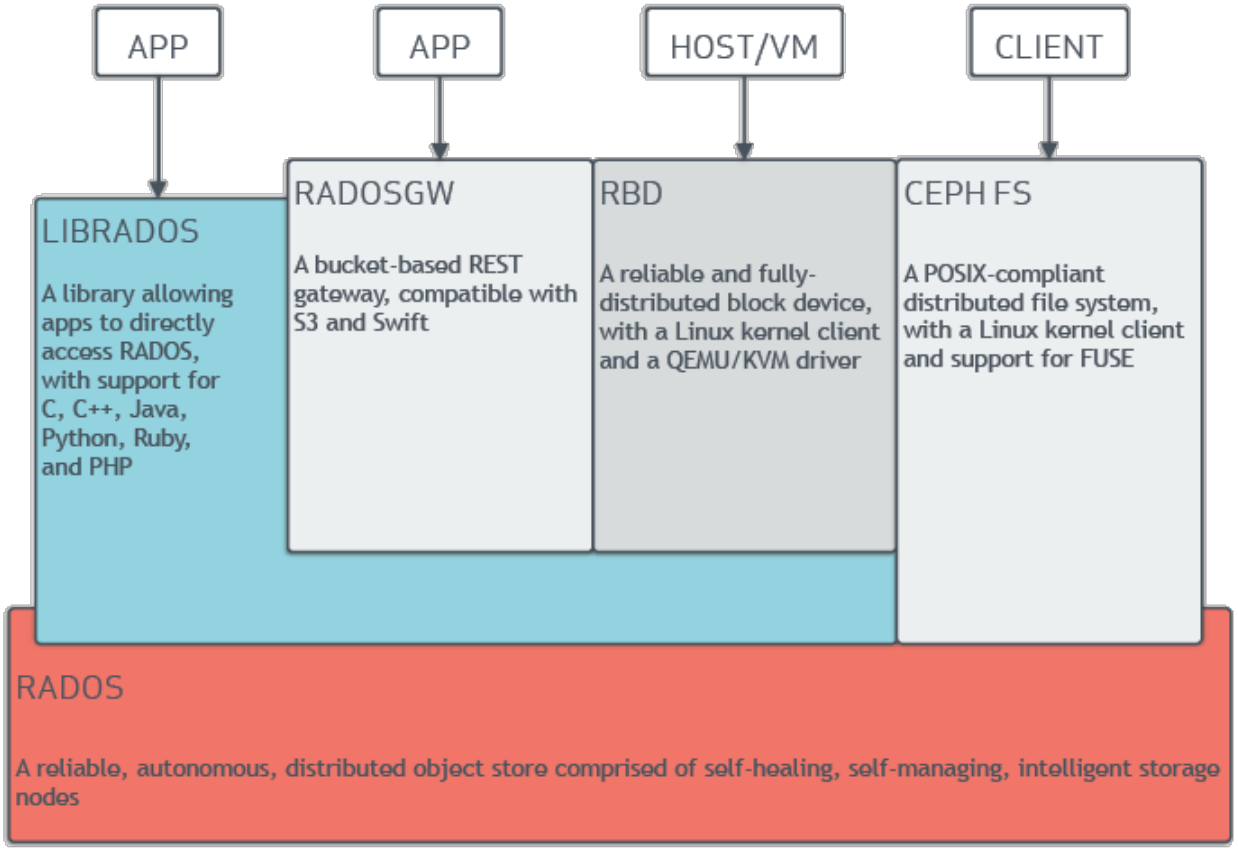
1. Google Cloud Storage: Un servicio de almacenamiento de objetos que proporciona almacenamiento escalable y de alta durabilidad. Está inspirado en los principios de GFS pero optimizado para la nube.
2. Google Cloud Dataproc: Un servicio gestionado de procesamiento de datos que permite ejecutar clústeres de Hadoop y Spark en Google Cloud Platform, facilitando la integración con GFS y otros servicios de Google Cloud.
3. Google BigQuery: Un almacén de datos sin servidor y altamente escalable que permite realizar análisis de datos a gran escala. Aunque no utiliza directamente GFS, se basa en tecnologías similares desarrolladas por Google.

## Arquitectura de Ceph

Ceph es un sistema de almacenamiento distribuido de código abierto diseñado para proporcionar almacenamiento escalable y de alto rendimiento para grandes volúmenes de datos. Ceph se destaca por su arquitectura completamente distribuida y descentralizada, lo que elimina los puntos únicos de fallo y mejora la resiliencia y escalabilidad del sistema. Ceph ofrece almacenamiento en bloque, almacenamiento de objetos y un sistema de archivos distribuido, todo en una única plataforma.

Ceph está compuesto por varios componentes clave, cada uno desempeñando un papel específico en el funcionamiento del sistema. Estos componentes incluyen:

- RADOS (Reliable Autonomic Distributed Object Store)
- Ceph Monitors (MONs)
- Ceph Managers (MGRs)
- Ceph OSD Daemons (OSDs)
- Ceph Metadata Servers (MDSs)
- Ceph Clients




RADOS es la capa de almacenamiento subyacente en Ceph, responsable de la distribución y replicación de datos. RADOS utiliza un algoritmo de mapeo llamado CRUSH (Controlled Replication Under Scalable Hashing) para distribuir los datos de manera eficiente y equilibrada a través de los nodos del clúster. Las principales funciones de RADOS incluyen:

- Almacenamiento de objetos: RADOS almacena datos como objetos, cada uno identificado por una clave única. Estos objetos se distribuyen automáticamente a través del clúster para equilibrar la carga y asegurar la redundancia.
- Replicación y recuperación de datos: RADOS asegura la redundancia mediante la replicación de objetos en múltiples nodos. En caso de fallo de un nodo, RADOS automáticamente replica los objetos afectados a otros nodos para mantener la disponibilidad de los datos.
- Escalabilidad: RADOS está diseñado para escalar a miles de nodos y petabytes de datos, ajustando dinámicamente la distribución de datos a medida que se añaden o eliminan nodos.

**Ceph monitors (MONs)**

Los Ceph Monitors (MONs) son responsables de mantener el estado del clúster y proporcionar servicios de consenso. Los MONs gestionan varios mapas que describen el estado del clúster, incluyendo el mapa del clúster, el mapa de monitores, el mapa de OSDs y el mapa de ubicaciones CRUSH. Sus funciones principales incluyen:

- Consenso y coordinación: Los MONs utilizan un protocolo de consenso para asegurar que todos los nodos del clúster tengan una visión coherente del estado del sistema.
- Autenticación y autorización: Gestionan las credenciales y permisos de los clientes y nodos del clúster para asegurar el acceso seguro a los datos.
- Gestión de fallos: Los MONs detectan y responden a los fallos de nodos, actualizando los mapas del clúster y coordinando la recuperación de datos.

**Ceph Managers (MGRs)**

Los Ceph Managers (MGRs) complementan a los MONs proporcionando servicios adicionales para la gestión del clúster. Los MGRs recopilan estadísticas, realizan tareas de supervisión y mantenimiento, y proporcionan interfaces de administración. Sus funciones incluyen:

- Monitorización y métricas: Recopilan y almacenan métricas detalladas sobre el rendimiento y estado del clúster, facilitando la monitorización y optimización del sistema.
- Interfaz de gestión: Proporcionan una interfaz web (Ceph Dashboard) y API REST para la administración y supervisión del clúster.
- Tareas de mantenimiento: Realizan tareas de mantenimiento y gestión, como la reequilibración de datos y la limpieza de objetos huérfanos.

**Ceph OSD Daemons (OSDs)**

Los Ceph OSD Daemons (OSDs) son los componentes que almacenan los datos reales en el clúster. Cada OSD gestiona un disco o una partición de disco y se encarga de las operaciones de lectura y escritura de datos. Las funciones principales de los OSDs incluyen:

- Almacenamiento y recuperación de Datos: Los OSDs almacenan objetos de datos y los recuperan en respuesta a las solicitudes de los clientes.
- Replicación de datos: Coordinan la replicación de objetos de datos a otros OSDs para asegurar la redundancia y disponibilidad.
- Reequilibración de datos: Ajustan la distribución de datos en el clúster para mantener un equilibrio de carga óptimo, especialmente cuando se añaden o eliminan nodos.
- Detección de fallos: Detectan fallos de discos y coordinan la recuperación de datos con otros OSDs y MONs.

**Ceph Metadata Servers (MDSs)**

Los Ceph Metadata Servers (MDSs) son responsables de gestionar los metadatos del sistema de archivos CephFS. Los MDSs permiten a CephFS escalar y proporcionar acceso rápido a los metadatos, mejorando el rendimiento para las operaciones de archivo. Sus funciones incluyen:

- Gestión de metadatos: Almacenan y gestionan la estructura del sistema de archivos, incluyendo nombres de archivos, directorios y permisos.
- Optimización del rendimiento: Aseguran que las operaciones de metadatos sean rápidas y eficientes, distribuyendo la carga entre múltiples MDSs en clústeres grandes.
- Consistencia y coherencia: Mantienen la coherencia de los metadatos en todo el sistema de archivos, asegurando que las operaciones concurrentes se gestionen correctamente.

**Ceph Clients**

Los Ceph Clients son las aplicaciones o servicios que interactúan con Ceph para leer y escribir datos. Ceph proporciona varias interfaces de cliente para diferentes casos de uso, incluyendo:

- Librados: Una biblioteca de cliente para interactuar directamente con RADOS. RBD (RADOS Block Device): Una interfaz de dispositivo de bloque que permite a las máquinas virtuales y contenedores utilizar Ceph como almacenamiento en bloque.
- CephFS: Un sistema de archivos distribuido que utiliza RADOS para el almacenamiento subyacente.
- RGW (RADOS Gateway): Una interfaz de almacenamiento de objetos compatible con Amazon S3 y OpenStack Swift.

## Funcionamiento de Ceph

El funcionamiento de Ceph se basa en la interacción coordinada de sus componentes para proporcionar almacenamiento distribuido eficiente y resiliente. A continuación se describen algunos procesos clave en Ceph.

**Escritura de datos**

1. Solicitud del cliente: Un cliente envía una solicitud de escritura de datos a un Ceph OSD. El cliente puede interactuar directamente con el OSD utilizando librados, RBD, CephFS o RGW.
2. Distribución de datos: El OSD utiliza el algoritmo CRUSH para determinar a qué otros OSDs replicar los datos. El algoritmo CRUSH permite una distribución uniforme y controlada de los datos a través del clúster.
3. Replicación de datos: El OSD escribe los datos en su disco y coordina la replicación con los otros OSDs determinados por CRUSH. Cada réplica se confirma para asegurar la durabilidad de los datos.
4. Confirmación al cliente: Una vez que los datos se han replicado satisfactoriamente, el OSD confirma la operación de escritura al cliente.

**Lectura de datos**

1. Solicitud del cliente: Un cliente envía una solicitud de lectura de datos a un Ceph OSD. El cliente puede interactuar directamente con el OSD utilizando librados, RBD, CephFS o RGW.
2. Recuperación de datos: El OSD identifica la ubicación de los datos solicitados y los recupera de su disco. Si el OSD original no está disponible, el cliente puede acceder a una réplica de los datos almacenada en otro OSD.
3. Entrega al cliente: Los datos se entregan al cliente una vez recuperados.

**Reequilibración de datos**

1. Añadir o eliminar nodos: Cuando se añaden o eliminan nodos en el clúster, el algoritmo CRUSH recalcula la distribución óptima de los datos.
2. Migración de datos: Los OSDs comienzan a mover datos según la nueva distribución calculada por CRUSH, asegurando que la carga esté equilibrada y que los datos estén replicados adecuadamente.
3. Actualización de metadatos: Los MONs actualizan los mapas del clúster para reflejar la nueva distribución de datos.

**Recuperación de fallos**

1. Detección de fallos: Los OSDs y MONs monitorizan continuamente el estado del clúster. Si un OSD falla, los MONs lo detectan a través de la falta de latidos (heartbeats).
2. Reasignación de réplicas: El algoritmo CRUSH recalcula la distribución de los datos afectados por el fallo del OSD y reasigna las réplicas a otros OSDs.
3. Recuperación de datos: Los OSDs replican los datos faltantes a partir de las réplicas disponibles, asegurando que el nivel de replicación se mantenga y que los datos sigan siendo accesibles.

**Monitorización y gestión**

1. Ceph Managers: Los MGRs recopilan y almacenan métricas detalladas sobre el rendimiento y estado del clúster. Estas métricas se utilizan para monitorizar el sistema y detectar problemas de rendimiento o fallos potenciales.
2. Ceph Dashboard: Proporciona una interfaz web intuitiva para la administración y supervisión del clúster, permitiendo a los administradores gestionar el almacenamiento, monitorizar el rendimiento y realizar tareas de mantenimiento.
3. Alertas y Notificaciones: Los MGRs pueden configurar alertas basadas en umbrales específicos de rendimiento o estado, notificando a los administradores sobre eventos críticos que requieren atención inmediata.

**Ventajas y desventajas de Ceph**

**Ventajas**

- Arquitectura descentralizada: Sin un único punto de fallo, lo que mejora la resiliencia y disponibilidad del sistema.
- Escalabilidad: Capaz de escalar a exabytes de datos y miles de nodos sin comprometer el rendimiento.
- Flexibilidad: Soporta múltiples tipos de almacenamiento (bloques, archivos y objetos) en una única plataforma.

- Algoritmo CRUSH: Proporciona una distribución eficiente y equilibrada de los datos a través del clúster.

**Desventajas**

- Complejidad de configuración: La instalación y configuración pueden ser complicadas, requiriendo una planificación cuidadosa y conocimientos técnicos avanzados.
- Requisitos de hardware: Necesita hardware robusto y de alto rendimiento para funcionar de manera óptima, lo que puede aumentar los costos de implementación.
- Curva de aprendizaje: La administración y operación de Ceph pueden tener una curva de aprendizaje pronunciada, especialmente para administradores sin experiencia previa en almacenamiento distribuido.

**Referencia:** **Intro Celph**

## Frameworks y herramientas relacionadas

1. RADOS: Reliable Autonomic Distributed Object Store es la capa de almacenamiento subyacente en Ceph. RADOS gestiona la distribución de datos y la replicación en el clúster Ceph.

2. CephFS: Un sistema de archivos distribuido que utiliza RADOS para el almacenamiento subyacente. CephFS proporciona una interfaz de sistema de archivos tradicional para aplicaciones que necesitan acceso a archivos.

3. RBD (RADOS Block Device): Un servicio de almacenamiento en bloque basado en Ceph que permite crear volúmenes de almacenamiento para máquinas virtuales y otros usos de almacenamiento en bloque.

4. RGW (RADOS Gateway): Un servicio de almacenamiento de objetos basado en Ceph que es compatible con las APIs de Amazon S3 y OpenStack Swift.

**Integraciones y herramientas adicionales**

1. OpenStack: Ceph se integra bien con OpenStack, proporcionando almacenamiento en bloque y almacenamiento de objetos para las máquinas virtuales y otros servicios de OpenStack.
2. Kubernetes: Ceph puede integrarse con Kubernetes a través de Rook, un operador de Kubernetes que facilita la implementación y gestión de clústeres de Ceph en entornos de contenedores.
3. Rook: Un operador de almacenamiento en la nube nativa que facilita la implementación y gestión de Ceph en Kubernetes.
4. Ansible: Una herramienta de automatización que se puede usar para desplegar y gestionar clústeres de Ceph a gran escala.
5. Prometheus y Grafana: Herramientas de monitoreo y visualización que pueden integrarse con Ceph para monitorear el rendimiento y la salud del clúster.

## Ejercicios

**Hadoop Distributed File System (HDFS)**

**Ejercicios teóricos**

**Ejercicio:** Arquitectura y funcionamiento de HDFS

- Describe la arquitectura de HDFS, explicando los roles y responsabilidades del Namenode, Datanode y Secondary Namenode.
- Explica cómo HDFS maneja la tolerancia a fallos y la replicación de datos. ¿Qué sucede si un Datanode falla? ¿Cómo se asegura HDFS de que los datos sean recuperables?

**Ejercicio:** Escalabilidad y rendimiento en HDFS

Analiza los desafíos de escalabilidad que enfrenta HDFS. ¿Cómo maneja HDFS la adición de nuevos nodos al clúster? Discuta las limitaciones del Namenode en términos de memoria y rendimiento. Proponga posibles soluciones para superar estos problemas.

**Ejercicio:** Integración de HDFS con MapReduce

- Explica cómo se integra HDFS con el modelo de programación MapReduce. ¿Cómo se optimiza la localización de datos para mejorar el rendimiento de MapReduce?
- Diseña un flujo de trabajo de MapReduce para procesar un conjunto de datos almacenado en HDFS, describiendo cada paso desde la lectura de datos hasta la escritura de resultados.

**Ejercicio:** Consistencia y coherencia en HDFS

- Explica cómo HDFS garantiza la consistencia de los datos. ¿Qué mecanismos utiliza para asegurar que los datos escritos sean consistentes y duraderos?
- Discute las implicaciones de las operaciones de actualización en HDFS. ¿Cómo maneja HDFS las escrituras concurrentes y las condiciones de carrera?

**Ejercicio:** Casos de uso de HDFS

- Identifica y describe dos aplicaciones que se beneficien significativamente del uso de HDFS. Analice las características específicas de HDFS que hacen que sea adecuado para estas aplicaciones.
- Discute cómo HDFS puede integrarse con otros componentes del ecosistema Hadoop, como Apache Hive y Apache Spark, para construir soluciones de análisis de datos de extremo a extremo.

**Ejercicios prácticos**

**Ejercicio:** Implementación de un sistema de análisis de Logs en HDFS

Descripción:

Desarrolla un sistema para procesar y analizar grandes volúmenes de logs almacenados en HDFS utilizando Apache Hadoop y MapReduce. El sistema debe:

1. Leer los archivos de log almacenados en HDFS.
2. Procesar los logs para extraer información relevante, como errores y advertencias.
3. Generar informes detallados sobre la frecuencia de distintos tipos de eventos en los logs.
4. Utilizar Apache Pig o Hive para consultas ad-hoc sobre los datos procesados.

**Ejercicio:** Integración de Apache Spark con HDFS para procesamiento de datos en tiempo real

Descripción:

Desarrolle una solución que integre Apache Spark con HDFS para procesar datos en tiempo real. El ejercicio debe incluir:

1. Configuración de un clúster Apache Spark que lea datos de HDFS.
2. Implementación de un trabajo de Spark Streaming que procese datos en tiempo real y almacene los resultados en HDFS.
3. Visualización de los resultados utilizando una herramienta como Apache Zeppelin o Jupyter Notebook.

```
## Tus respuestas
```

**Google File System (GFS)**

**Ejercicios teóricos**

**Ejercicio:** Diseño y principios de GFS

- Describw los principios de diseño que guían el desarrollo de GFS. ¿Cómo se diferencian estos principios de los sistemas de archivos tradicionales?
- Analiza la decisión de GFS de utilizar bloques de tamaño fijo de 64 MB. ¿Cuáles son las ventajas y desventajas de esta elección en términos de rendimiento y gestión de datos?

**Ejercicio:** Gestión de fallos en GFS

- Explica cómo GFS maneja la detección y recuperación de fallos. ¿Qué mecanismos utiliza para asegurar la consistencia y disponibilidad de los datos?
- Compara y contrasta el manejo de fallos en GFS con el de HDFS. ¿Qué lecciones aprendidas de GFS fueron aplicadas en el diseño de HDFS?

**Ejercicio:** Casos de uso y aplicaciones de GFS

- Identifica y describe dos aplicaciones de Google que utilizan GFS. ¿Cómo se beneficia cada aplicación de las características específicas de GFS?
- Discute las implicaciones de usar GFS en un entorno de nube pública versus un entorno de nube privada. ¿Qué consideraciones adicionales deben tenerse en cuenta?

**Ejercicio:** Seguridad y control de acceso en GFS

- Describe los mecanismos de seguridad implementados en GFS para proteger los datos almacenados. ¿Cómo maneja GFS la autenticación y autorización de los clientes?
- Analiza las implicaciones de seguridad de utilizar GFS en un entorno compartido por múltiples aplicaciones y usuarios. ¿Qué medidas adicionales pueden implementarse para asegurar los datos en estos escenarios?

**Ejercicio :** Optimización del rendimiento en GFS

- Explica las técnicas utilizadas por GFS para optimizar el rendimiento de las operaciones de lectura y escritura. ¿Cómo maneja GFS la localización de datos y la replicación para mejorar el rendimiento?

- Discute las consideraciones de diseño que deben tenerse en cuenta al configurar un clúster GFS para aplicaciones con diferentes perfiles de carga de trabajo.

**Ejercicios prácticos**

**Ejercicio:** Despliegue de GFS en un entorno de nube privada

Descripción:

Desarrolla una solución para desplegar un sistema GFS en un entorno de nube privada utilizando herramientas de automatización como Ansible y Terraform. El ejercicio debe cubrir:

1. Configuración de la infraestructura de nube privada utilizando Terraform.
2. Desarrollo de playbooks de Ansible para instalar y configurar GFS en la infraestructura provisionada.
3. Pruebas de la solución desplegada para asegurar su funcionalidad y rendimiento.

**Ejercicio:** Implementación de un servicio de almacenamiento de objetos sobre GFS

Descripción:

Desarrolla un servicio de almacenamiento de objetos compatible con S3 sobre GFS. El ejercicio debe incluir:

1. Implementación de una API RESTful para gestionar el almacenamiento de objetos.
2. Integración de la API con GFS para almacenar y recuperar objetos.
3. Pruebas de la API para asegurar su conformidad con el protocolo S3 y su rendimiento.

```
## Tus respuestas
```

**Ceph**

**Ejercicios teóricos**

**Ejercicio:** Comparación de CRUSH con otros algoritmos de distribución de datos

- Compara el algoritmo CRUSH utilizado por Ceph con otros algoritmos de distribución de datos, como consistent hashing. ¿Cuáles son las ventajas y desventajas de cada uno en términos de rendimiento, escalabilidad y equilibrio de carga?
- Analiza cómo CRUSH maneja la redistribución de datos cuando se añaden o eliminan nodos en el clúster. ¿Cómo se minimiza la cantidad de datos que necesitan ser movidos?

**Ejercicio:** Integración de ceph con herramientas de orquestación y gestión

- Explica cómo Ceph se integra con herramientas de orquestación como Kubernetes y OpenStack. ¿Qué beneficios y desafíos presenta esta integración en términos de despliegue y gestión?
- Discute las mejores prácticas para monitorizar y gestionar un clúster Ceph utilizando herramientas como Prometheus y Grafana . **Ejercicios prácticos**

**Ejercicio:** Despliegue de Ceph en un clúster Kubernetes con Rook

Descripción:

Desarrolla una solución para desplegar un clúster Ceph en Kubernetes utilizando Rook. El ejercicio debe incluir:

1. Configuración de Kubernetes y Rook Operator.
2. Despliega de Ceph Cluster utilizando Rook.
3. Pruebas de la solución desplegada para asegurar que el almacenamiento CephFS y RBD estén funcionando correctamente.

**Ejercicio:** Implementación de un sistema de almacenamiento de objetos con Ceph RGW

Descripción:

Desarrolla un sistema de almacenamiento de objetos utilizando Ceph RGW (RADOS Gateway). El ejercicio debe incluir:

1. Configuración de Ceph RGW para proporcionar almacenamiento de objetos compatible con S3.
2. Desarrollo de una aplicación cliente para interactuar con el almacenamiento de objetos Ceph RGW.
3. Pruebas de la solución para asegurar su rendimiento y conformidad con el protocolo S3.

**Ejercicio:** Automatización de despliegue y gestión de Ceph con Ansible

Descripción:

Automatiza el despliegue y la gestión de un clúster Ceph utilizando Ansible. El ejercicio debe incluir:

1. Desarrollo de playbooks de Ansible para instalar y configurar Ceph en una infraestructura provisionada.
2. Automatización de tareas de gestión, como la adición y eliminación de nodos, y la reconfiguración del clúster.
3. Validación del despliegue automatizado y pruebas de las tareas de gestión.

**Ejercicio:** Monitorización avanzada de Ceph con Prometheus y Grafana

Descripción:

Implementa una solución avanzada de monitorización para un clúster Ceph utilizando Prometheus y Grafana. El ejercicio debe incluir:

1. Configuración de Prometheus para recolectar métricas detalladas de Ceph.
2. Desarrollo de dashboards avanzados en Grafana para visualizar métricas de rendimiento, uso de recursos y estado del clúster.
3. Implementación de alertas avanzadas basadas en umbrales específicos y condiciones de rendimiento.

```
## Tus respuestas
```