Administración de Sistemas Operativos - Universidad de A Coruña

Zettabyte File System

Eduardo Lorenzo Bóveda

Xián Cotelo Varela

ÍNDICE

1. Introducción

1.1. Definición de sistema de archivos

Un **sistema de archivos** (a veces también un sistema de archivos escrito) es un método para estructurar datos que un sistema operativo utiliza para controlar cómo se almacenan y recuperan los datos. Organiza archivos y directorios para garantizar una asignación adecuada de espacio en el dispositivo.

El sistema de archivos organiza los datos en una estructura jerárquica y permite la creación, movimiento, modificación y eliminación de archivos y directorios. Los sistemas de archivos son esenciales para optimizar el rendimiento, garantizar integridad de los datos y facilitar la gestión de datos y backups.

Existen diferentes tipos de sistemas de archivos, cada uno diseñado con objetivos específicos. Por ejemplo, algunos priorizan la velocidad, otros la compatibilidad o la seguridad. Por ejemplo, **ext4** en sistemas Linux, **NTFS** en Windows o sistemas más avanzados como **ZFS**, que integran funcionalidades adicionales como protección contra corrupción de datos y gestión avanzada del almacenamiento.

1.2. ¿Qué es ZFS?

ZFS es un sistema de archivos avanzado diseñado para resolver los principales problemas encontrados en el software de subsistemas de almacenamiento anteriores.

Originalmente desarrollado en Sun™, el desarrollo ZFS como software de código abierto se ha movido al Proyecto OpenZFS.

Una de las características más destacadas de ZFS es su enfoque en la **integridad de los datos**. Utiliza mecanismos como checksums para detectar errores y, en configuraciones redundantes, puede incluso corregirlos automáticamente. Además, emplea el modelo *Copy-on-Write (CoW)*, que evita sobrescribir datos existentes, reduciendo el riesgo de corrupción.

ZFS también incorpora funcionalidades avanzadas como:

- **Snapshots**: copias instantáneas del sistema de archivos en un momento determinado.

- **Clones**: duplicados de snapshots que pueden modificarse independientemente.
- **Compresión y deduplicación**: optimizan el uso del espacio de almacenamiento.
- **Escalabilidad masiva**: permite gestionar grandes volúmenes de datos sin pérdida de rendimiento.

1.3. Objetivos y propósito de ZFS

ZFS fue diseñado con el objetivo de **superar las limitaciones de los sistemas de archivos tradicionales**, integrando en una sola solución tanto la gestión del sistema de archivos como la del almacenamiento. Su propósito principal es ofrecer un sistema robusto, seguro y altamente escalable

Entre los objetivos fundamentales de ZFS destacan:

- **Integridad de datos**: Todos los datos incluyen un checksum de datos. ZFS calcula los checksum y los escribe junto con los datos. Al leer esos datos más tarde, ZFS recalcula los checksum. Si los checksum no coinciden, lo que significa detectar uno o más errores de datos, ZFS intentará corregir automáticamente los errores cuando estén disponibles bloques ditto-, mirror- o de paridad.

- **Almacenamiento agrupado:** agregar dispositivos de almacenamiento físico a una agrupación y asignar espacio de almacenamiento de esa agrupación compartida. El espacio está disponible para todos los sistemas de archivos y volúmenes, y aumenta al agregar nuevos dispositivos de almacenamiento a la agrupación.

- **Rendimiento:** los mecanismos de caché proporcionan un rendimiento aumentado. ARC (Adaptive Replacement Cache) es una caché de lectura de memoria avanzada. ZFS proporciona una segunda capa de caché de lectura basada en disco con L2ARC, y una caché de escritura sincronizada basada en disco llamada ZIL.

- **Eliminar la complejidad de la gestión de almacenamiento**: ZFS unifica conceptos como volúmenes, particiones y RAID en un único modelo basado en pools de almacenamiento (zpools), simplificando la administración.

- **Mejorar la fiabilidad del sistema**: gracias a técnicas como el *Copy-on-Write*, se asegura que los datos nunca se sobrescriban directamente, reduciendo el riesgo de pérdida ante fallos.

- **Ofrecer alta escalabilidad**: ZFS está diseñado para manejar grandes cantidades de datos, desde entornos domésticos hasta infraestructuras empresariales con volúmenes masivos de información.

- **Facilitar la recuperación y gestión de datos**: mediante funcionalidades como snapshots y clones, permite realizar copias y restauraciones de forma rápida y eficiente.

2. Historia y evolución

A principios de los años 2000, el sistema de archivos ZFS surge como respuesta a las limitaciones de los sistemas de archivos tradicionales en una época de un gran crecimiento en el almacenamiento de datos.

El aumento en la capacidad de los discos duros y la necesidad de gestionar grandes volúmenes de información hicieron evidentes las limitaciones de los sistemas de archivos existentes en aquella época, como UFS o ext3, a la hora de escalar eficientemente y garantizar la integridad de los datos a largo plazo.

2.1. Origen en Sun Microsystems

El desarrollo de ZFS comienza en el año 2001, bajo la dirección de Jeff Bonwick y Matthew Ahrens. La filosofía de este proyecto se centraba en solucionar problemas fundamentales presentes en los sistemas existentes y que integrase múltiples capas en un único sistema coherente.

Entre las principales limitaciones que se buscaban solventar destacan:

- La dificultad para gestionar múltiples discos y volúmenes
- La falta de mecanismos de detección de corrupción de datos
- La complejidad de administración en sistemas con RAID y LVM

- Las limitaciones de escalabilidad (tamaño máximo de archivos y sistemas)

Finalmente, ZFS fue oficialmente presentado en el año 2004, causando gran interés en la comunidad tecnológica.

2.2. Desarrollo en Solaris

En 2005, ZFS fue integrado en el sistema operativo Solaris, desarrollado también por Sun Microsystems. Posteriormente, fue incluido en el proyecto OpenSolaris, que ofrecía una versión de código abierto del sistema.

Este suceso permitió aprovechar todas sus capacidades, destacando especialmente:

- La creación dinámica de pools de almacenamiento (zpools)
- La gestión automática del espacio sin necesidad de particiones
- La implementación de snapshots instantáneos
- La detección y corrección automática de errores mediante checksums

También se introdujeron innovaciones en esta fase, como el modelo Copy-on-Write (CoW), que elimina el riesgo de corrupción de datos al evitar sobrescribir bloques existentes.

Gracias a estas características y su buena integración con Solaris, ZFS comenzó a utilizarse en servidores de almacenamiento o sistemas empresariales, por ejemplo.

2.3. Nacimiento de OpenZFS

En el año 2010, Oracle adquirió Sun Microsystems. Como consecuencia, el desarrollo de OpenSolaris fue discontinuado y ZFS pasó a formar parte del ecosistema propietario de Oracle Solaris.

Como respuesta, la comunidad de código abierto creó OpenZFS en 2013, un proyecto independiente que continúa el desarrollo de ZFS fuera del control de Oracle.

Esta situación permitió unificar el desarrollo de ZFS en un proyecto común, adaptar ZFS a nuevos sistemas e incorporar mejoras en el rendimiento y nuevas funcionalidades.

OpenZFS ha evolucionado significativamente y es ampliamente utilizado en sistemas BSD, Linux y otros entornos debido a sus avanzadas características, como la detección y corrección automática de errores, instantáneas y almacenamiento escalable.

2.4. Situación actual y adopción

Actualmente, ZFS, a través de OpenZFS, es uno de los sistemas de ficheros consolidado en los ámbitos donde es crítica la integridad de los datos.

Principalmente se usa en sistemas NAS, entornos de virtualización, empresariales o CPDs. También sistemas de backup, gracias a su eficiente capacidad de replicación.

Sin embargo, su adopción en entornos domésticos es menor debido a su mayor consumo de recursos y complejidad. Además, su integración en Linux presenta limitaciones por cuestiones de licencias.

3. Arquitectura de ZFS

3.1. Concepto de zpool

Zpool es la unidad básica de almacenamiento sobre la que se construye todo el sistema. Se puede entender como un conjunto de dispositivos físicos (discos duros o SSD) agrupados para formar un **único espacio de almacenamiento lógico**.

A diferencia de los sistemas tradicionales, donde primero se crean particiones y luego sistemas de archivos, en ZFS el zpool integra directamente la **gestión del almacenamiento** y del **sistema de archivos**, eliminando la necesidad de particionado previo.

Las principales características de un zpool son:

- **Agregación de discos**: permite combinar varios dispositivos en un único pool, facilitando la gestión y ampliación del almacenamiento.
- **Asignación dinámica de espacio**: el espacio no se divide de forma fija entre sistemas de archivos, sino que se distribuye según las necesidades de los datasets.

- **Redundancia opcional:** mediante configuraciones como RAID-Z o espejado, el zpool puede proteger los datos frente a fallos de hardware.
- **Escalabilidad:** se pueden añadir nuevos dispositivos al zpool para aumentar su capacidad sin interrumpir el servicio.

Dentro de un zpool se crean los llamados datasets, que son los sistemas de archivos o volúmenes donde realmente se almacenan los datos. Todos ellos comparten el espacio disponible del pool, lo que permite una gestión mucho más flexible y eficiente.

3.2. vdev (Virtual Devices)

Para entender los vdev hace falta visualizar su jerarquía:

1. **ZPOOL (Piscina):** El nivel superior. Es el almacenamiento total combinado.
2. **VDEV (Dispositivo Virtual):** La capa intermedia (y la más importante para la integridad).
3. **Dispositivos físicos:** Los HDD o SSD individuales que conforman el VDEV.

Un vdev es una unidad lógica que **agrupa** uno o más discos físicos. ZFS no escribe datos directamente en los discos, sino que los distribuye a través de los VDEVs**.** Si un solo VDEV falla y es de tipo stripe (no tiene **redundancia interna**), pierdes todo el ZPOOL. No importa si tienes otros 10 vdev sanos; el **pool se corrompe por completo**.

A diferencia de otros sistemas de archivos como NTFS o EXT4, el uso de vdev en ZFS tiene la ventaja de que tras escribir un dato el VDEV actualiza los punteros para decir: "El archivo real es el nuevo, el viejo ya no sirve". Así, por ejemplo, si se va la luz, **siempre existirá o la versión vieja o la nueva**, pero nunca una mezcla corrupta como sí ocurriría si escribiese directamente en el disco. Con este método se evitan los *¨write hole¨.*

Los vdev pueden ser de varios tipos:

A. Stripe (Sin redundancia) o RAID 0

ZFS proporciona checksumming para evitar la corrupción de datos pero con este método no dispone de datos redundantes (ni paridad ni mirror) para reconstruir los datos en caso de fallo de un disco. Otra desventaja es que almacena los registros de metadatos en varias copias, independientemente de la configuración RAIDZ de nivel

superior. Sin embargo, los datos de usuario se almacenan en una sola copia en un striped pool ZFS. Aunque no se recomienda utilizar esta configuración de forma aislada, puede ser útil como parte de configuraciones redundantes como RAID10 (stripe + mirror).

Disco 1	Disco 2	Disco 3
1	5	3
6	2	7
4	8	9

B. Mirror (Espejo)

Requiere un mínimo de dos discos y, en general, es muy similar a un RAID 1. Su uso junto con el sistema de archivos ZFS ofrece ciertas ventajas, como la verificación automática de checksums. Si bien esta verificación solo permite detectar la corrupción de datos, es una característica de la que carecen la mayoría de los sistemas RAID tradicionales. Otra diferencia con un RAID 1 convencional es que, gracias a ZFS, se pueden crear sistemas de almacenamiento con múltiples espejos, en lugar de los espejos de 2 o 3 que es característica del RAID 1.

Un pool ZFS en espejo puede almacenar tantas copias de datos como se desee y sí utiliza bloques y franjas las copias se almacenan en filas específicas. También, utiliza la capacidad de (N-1) discos para mantener la tolerancia a fallos. Esto equivale a una pérdida de capacidad de (N-1) \ N para el array. Por ejemplo, si se combinan cuatro discos de 500 GB en un espejo ZFS, sólo se obtendrían 500 GB de espacio de disco utilizable y 1,5 TB se destinarían a la redundancia.

Gracias a esto, un mirror pool de ZFS es un sistema de almacenamiento extremadamente fiable.

Disco 1	Disco 2

1	2
2	1
3	3

C. RAID-Z (Paridad)

RAIDZ es una solución de almacenamiento amplia, fiable y relativamente económica.

RAIDZ (a veces denominado RAIDZ1, que apunta a una única paridad) es muy similar a un RAID5 tradicional y requiere un mínimo de dos discos. Al igual que en un RAID5 convencional, cada fila, junto con los bloques de datos, almacena una función de paridad calculada sobre dichos bloques, lo que permite que el sistema sobreviva a un fallo de un solo disco. La diferencia radica en el patrón de ubicación de las filas, que ya no es el mismo en todo el conjunto de discos; otra diferencia es el tamaño de los bloques, que puede variar entre las filas. Además, en los pools RAIDZ de ZFS ya no existen configuraciones RAID5 derecha/izquierda ni síncronas/asíncronas.

Si todos los discos están en buen estado, las solicitudes de lectura se distribuyen uniformemente entre ellos, proporcionando una velocidad de lectura similar a la de un pool con distribución de datos en franjas (striped pool). Teóricamente, para una matriz de N discos, un pool con distribución de datos en franjas ofrece lecturas N veces más rápidas, mientras que RAIDZ debería ofrecer lecturas (N-1) veces más rápidas. Sin embargo, el número de bloques en una fila determinada depende del tamaño del bloque de datos que se va a escribir, así como del tamaño del bloque de la unidad ZFS, por lo que solo podemos hablar de una aceleración de la velocidad de lectura de (N-1) veces como límite máximo. Si falla una de las unidades, la velocidad de lectura se reduce a la de una sola unidad, ya que se requieren todos los bloques de una fila para atender la solicitud.

La velocidad de escritura de un RAIDZ está limitada por las actualizaciones de paridad. Por cada bloque escrito, su bloque de paridad correspondiente debe leerse, actualizarse y luego volver a escribirse. Por lo tanto, no hay una mejora significativa en la velocidad de escritura en RAIDZ, si es que la hay.

La capacidad de una unidad miembro se utiliza para mantener la tolerancia a fallos. Por ejemplo, si tiene 10 unidades de 1 TB cada una, la capacidad resultante del RAIDZ sería de 9 TB.

Disco 1	Disco 2	Disco 3	Disco 4
P1,2	1	2	
P3,4,5	3	4	5
6	P6		
7	P7,8	8	

3.3. Datasets y volúmenes

Un sistema de archivos tradicional como ext4 se ejecuta dentro de una partición de disco. Esta es una estructura simple, pero inflexible si el crecimiento de sus datos cambia con el tiempo y posteriormente desea reorganizarlos. Con datasets se puede mover el espacio dinámicamente donde se necesite, sin necesidad de redimensionar ni reorganizar los datos.

Existen dos tipos de dataset diferentes:

- **Filesystem:** es el tipo de dataset por defecto, es el que se utiliza para almacenar los archivos, carpetas etc. Se puede establecer directamente el punto de montaje, sin editar el típico fstab de sistemas Linux.
- **ZVOL:** es un dataset que representa a un dispositivo por bloques, también lo podemos encontrar en los diferentes sistemas operativos como «Volumen». Este dataset permite crear un dispositivo por bloques, y posteriormente darle formato con sistemas de archivos como EXT4.os.

Las características principales son las siguientes:

- **Asignación dinámica de espacio**:
  Los datasets no tienen un tamaño fijo predefinido. Todos comparten el espacio

disponible del zpool y lo utilizan según sus necesidades, lo que evita el desperdicio de almacenamiento.

- **Jerarquía tipo árbol**:
  Los datasets se organizan en una estructura jerárquica similar a directorios, lo que facilita la administración y la organización lógica de los datos.
- **Configuración independiente**:
  Cada dataset puede tener propiedades propias, como compresión, cuotas, reservas o permisos, sin afectar a otros datasets dentro del mismo zpool.
- **Snapshots eficientes**:
  Permiten crear copias instantáneas del estado de un dataset en un momento determinado, ocupando muy poco espacio gracias al modelo Copy-on-Write.
  - “zfs snapshot <pool_name>/<zvol_name>@<snapshot_name>”
- **Clones**:
  A partir de un snapshot se pueden crear clones, que son copias editables que comparten bloques de datos con el original, optimizando espacio y tiempo.
  - “zfs clone <pool>/<dataset>@<snapshot> <destino_dataset>”
- **Compresión integrada**:
  Los datasets pueden activar la compresión de datos de forma transparente, reduciendo el espacio utilizado sin intervención del usuario.
- **Control de cuotas y reservas**:
  Se pueden establecer límites máximos (cuotas) o garantizar espacio mínimo (reservas), lo que resulta útil en entornos multiusuario o servidores.
- **Gestión sencilla**:
  La administración de datasets es directa mediante comandos simples, permitiendo crear, modificar o eliminar datasets rápidamente.
- **Compatibilidad con distintos usos**:
  Gracias a la existencia de sistemas de archivos y volúmenes (zvols), los datasets pueden adaptarse tanto a almacenamiento de archivos como a uso a nivel de bloque.
  - zfs create -V 20G pool1/volumen1

4. Funcionamiento Interno

Este es el elemento que lo diferencia del resto de sistemas de ficheros tradicionales, ya que, mientras otros se centran exclusivamente en el almacenamiento de datos, ZFS introduce mecanismos avanzados para garantizar su integridad, consistencia y eficiencia.

Se basa en un modelo transaccional, en el que todas las operaciones se realizan de forma segura y verificable, evitando la corrupción de datos incluso en situaciones de fallo del sistema.

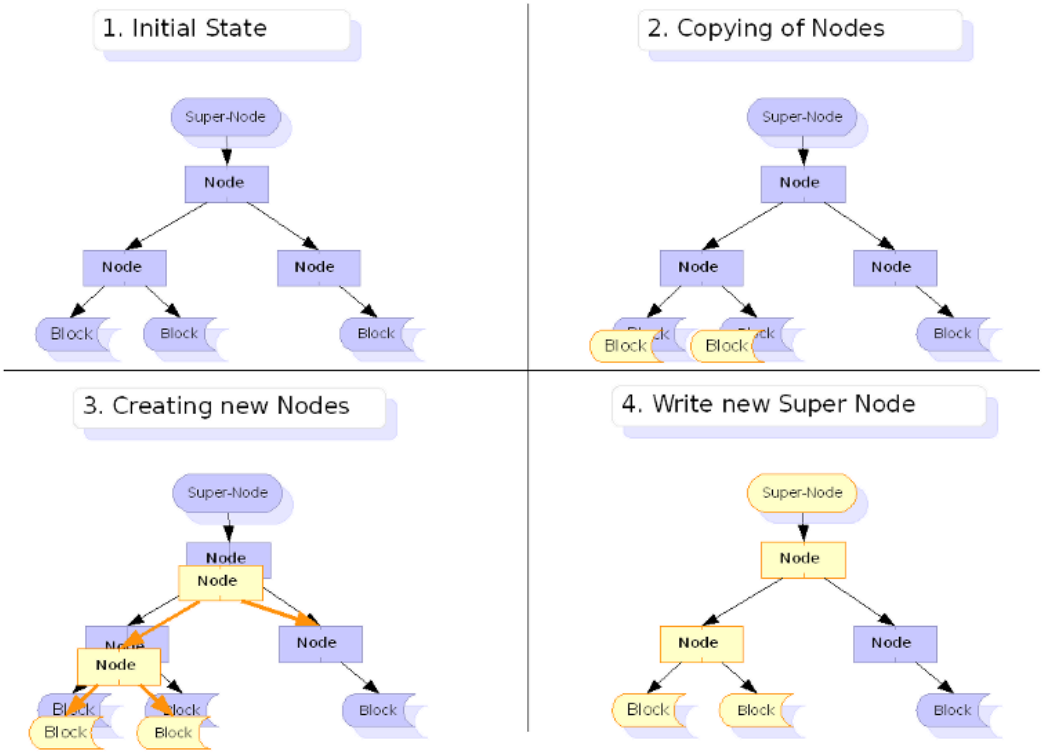
4.1. Copy-on-Write (CoW)

Este mecanismo es uno de los pilares fundamentales de ZFS.

En sistemas tradicionales, al modificar un archivo, los datos se sobrescriben en el mismo bloque del disco. Esto implica un riesgo; si ocurre un fallo durante la escritura, como un corte de energía, los datos pueden quedar en un estado inconsistente o corrupto.

ZFS ofrece la siguiente solución a este problema; cuando se modifica un dato, no se sobrescribe el bloque original, en su lugar, se escribe la nueva versión en un bloque distinto del disco. Se verifica que la escritura se ha realizado correctamente y, solo entonces se actualiza el puntero que referencia ese bloque y el original se marca como libre y se podrá reutilizar. Sin embargo, si hay snapshots del sistema de archivos, el bloque no se elimina inmediatamente porque sigue siendo referenciado por la instantánea.

En la siguiente imagen se puede ver que, cuando se actualiza un bloque de datos, también se deben ser actualizados el bloque hijo y sus padres con el nuevo hash. Los bloques amarillos son una copia de los datos, en otra parte del sistema de archivos, y los nodos hash padre se actualizan para apuntar a las nuevas ubicaciones de bloque.



De esta manera se garantiza que el sistema siempre tenga una versión válida de los datos.

4.2. Sistema de checksums

ZFS incorpora un sistema de verificación de integridad basado en checksums.

Cada bloque de datos tiene asociado un checksum, que se calcula cuando el dato se escribe en disco. Este checksum no se guarda en el propio bloque, sino en su bloque padre dentro de la estructura del sistema.

Al leer datos, se vuelve a calcular el checksum y se compara con el original. Ahora surgen dos posibles escenarios:

Si los checksums coinciden, se continúa con la operación con normalidad. Pero, si no coinciden, ZFS detecta a ese dato como corrupto. En este último caso, se intentará corregir el dato, por ejemplo, mediante mecanismos de redundancia como RAID o mirrors, recuperando la copia correcta y reparando el bloque dañado.

Este sistema permite detectar errores que otros sistemas no pueden o pueden, pero de una manera más costosa. Algunos ejemplos son: fallos de hardware, errores de lectura/escritura o el bit rot (corrupción silenciosa con el tiempo)

El uso de checksums está relacionado con el Copy-on-Write. Dado que los datos nunca se sobrescriben directamente, siempre existe una versión consistente del modelo, lo que facilita la detección y corrección de errores sin riesgo de corrupción adicional.

Otra característica importante es la verificación de integridad de extremo a extremo (end-to-end data integrity). Esto implica que los datos son verificados en todas las etapas del proceso, desde que se escriben hasta que se leen desde el disco, incluyendo la memoria, la caché y los dispositivos de almacenamiento. De este modo, ZFS es capaz de detectar errores que pueden producirse en cualquier punto del sistema.

4.3. Gestión de bloques

Los datos son organizados en bloques, los cuales son gestionados de una manera flexible y eficiente:

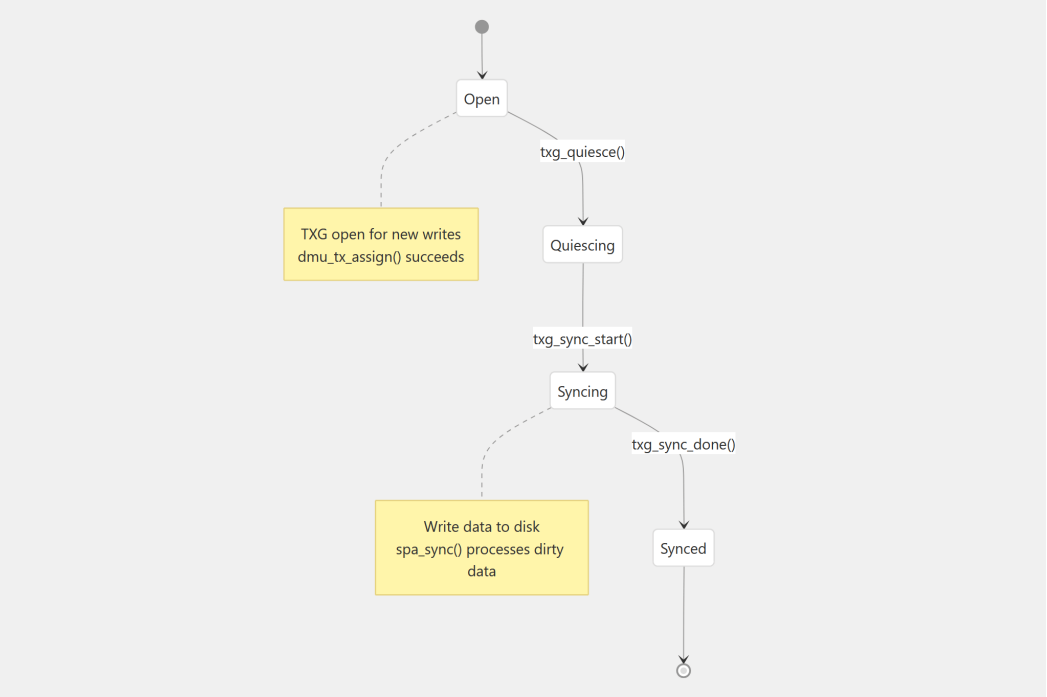
Los bloques tienen tamaño variable (hasta 128KB), lo que permite una mayor adaptabilidad, rendimiento y reduce el desperdicio de espacio.

Los datos se organizan en una estructura jerárquica en formato árbol, donde cada bloque tiene un puntero (el cual contiene el checksum de su bloque hijo) al siguiente. Esto crea una cadena de verificación desde la raíz a cada hoja del árbol, de manera que, si un bloque se corrompe, es detectado y corregido rápidamente.

ZFS agrupa las operaciones en conjuntos llamados transaction groups (TXG). Estas agrupan modificaciones en unidades atómicas que son sincronizadas al disco de manera periódica.

El sistema mantiene tres estados de TXG simultáneamente en un pipeline, que se ejecutan de manera secuencial: open (acepta nuevas escrituras), quiescing (que completa operaciones sin que unas afecten a otras) y syncing (que escribe a disco).

Debido al CoW, la fragmentación es inevitable, pero esto se puede mitigar gracias a una caché avanzada y a una distribución eficiente de bloques.



4.4. Caché (ARC)

El rendimiento de este sistema de archivos depende en gran medida de su sistema de cacheado.

ARC significa Adaptive Replacement Cache (caché de reemplazo adaptable) y es una caché dentro de la DRAM para el sistema de archivos y los datos de volumen. Almacena tanto datos como metadatos, siendo estos últimos especialmente importantes, ya que su acceso frecuente puede tener un gran impacto en el rendimiento del sistema.

Guarda los datos más utilizados (MRU) y los accedidos recientemente (MFU), cumpliendo el principio de localidad temporal y la localidad basada en frecuencia. Destaca gracias a su algoritmo adaptativo, el cual es excelente a la hora de seleccionar los datos a cachear.

ARC ajusta dinámicamente la relación entre MRU y MFU en función del patrón de acceso actual. Un servidor de base de datos con muchos accesos repetidos se beneficia de la parte de MFU, mientras que un servidor de archivos con patrones de acceso secuencial favorece la parte de MRU.

Cuando la RAM no es suficiente, ZFS puede usar un segundo nivel de caché: la L2ARC (Level 2 ARC). Se implementa en un SSD o NVMe y almacena datos desalojados de la ARC.

Cabe destacar que, a diferencia de la caché ARC, la L2ARC no es persistente, así que los datos almacenados ahí se perderán tras el reinicio del sistema.

Una contraparte de este sistema es la gran demanda de capacidad de memoria. La ARC utiliza una gran parte de la RAM disponible para maximizar el rendimiento, lo que puede suponer un problema en sistemas con recursos limitados. Aunque ZFS gestiona dinámicamente el tamaño de la caché y puede liberar memoria si es necesario, en entornos con poca RAM, la ARC puede competir con otras aplicaciones, provocando degradación del rendimiento general.

4.5. ZIL (ZFS Intent Log)

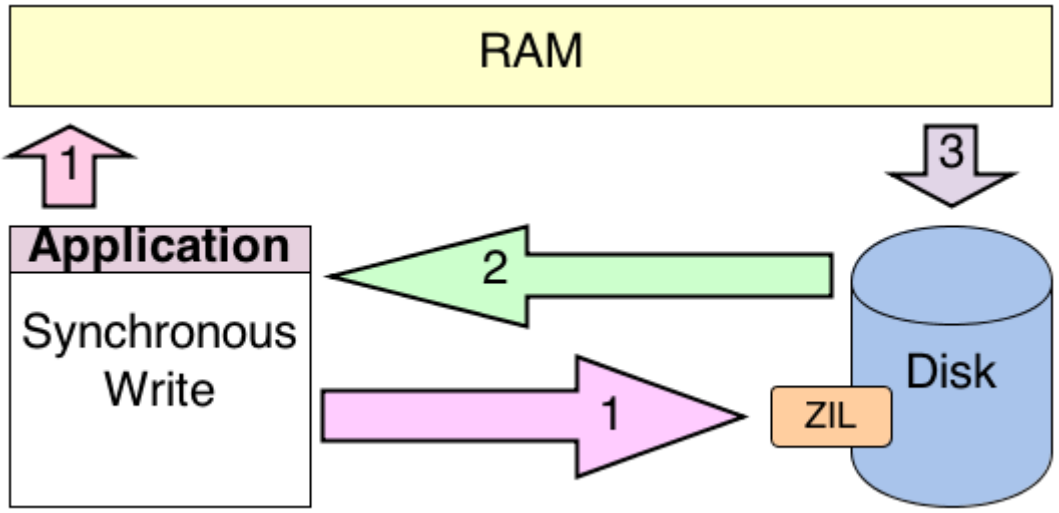
El ZFS Intent Log (ZIL) es un mecanismo diseñado para garantizar la durabilidad y consistencia de las operaciones de escritura síncronas. No interviene en las escrituras síncronas, que son gestionadas directamente mediante los *transaction groups (TXG)*.

El ZIL actúa como un registro temporal que complementa a los TXG. Mientras que estos agrupan las escrituras y las envían periódicamente a disco para optimizar el rendimiento, el ZIL asegura que las operaciones síncronas queden registradas inmediatamente. Así, antes de confirmar una escritura síncrona, ZFS registra la operación en el ZIL y posteriormente la integra en el almacenamiento definitivo junto con el resto de datos.

En caso de fallo del sistema (como un corte de energía), el ZIL se utiliza durante el proceso de recuperación para reproducir las operaciones que aún no habían sido consolidadas, consiguiendo así la consistencia del sistema de archivos. Durante el funcionamiento normal del sistema, el ZIL no se utiliza para lectura, sino únicamente para registrar las operaciones pendientes.

El ZIL puede almacenarse en el propio pool de almacenamiento o en un dispositivo dedicado llamado SLOG (Separate Intent Log), normalmente un SSD de buen rendimiento. Su uso puede mejorar significativamente el rendimiento en sistemas donde predominan las escrituras síncronas, como bases de datos o servidores NFS.

Otra característica es que no está diseñado como un sistema de almacenamiento permanente, sino como un registro temporal. Una vez que los datos han sido escritos correctamente en el almacenamiento principal, la información contenida en el ZIL deja de ser necesaria.



5. Características principales

5.1. Snapshots

Los **snapshots** son copias instantáneas que **capturan el estado exacto de un dataset** en un momento determinado. Se trata de una funcionalidad fundamental dentro de ZFS, ya que permite preservar versiones históricas de los datos de forma eficiente, segura y prácticamente inmediata. A diferencia de los métodos tradicionales de copia de seguridad, los snapshots no implican la duplicación completa de la información en el momento de su creación. Gracias al modelo **Copy-on-Write (CoW)**, ZFS simplemente marca los bloques actuales como pertenecientes al snapshot. Cuando se realizan modificaciones posteriores en los datos, los nuevos cambios se escriben en bloques diferentes, mientras que los bloques originales permanecen intactos para conservar el estado del snapshot.

Las principales ventajas de los snapshots son:

1. **Eficiencia en el uso del espacio**. Inicialmente, no ocupan espacio adicional significativo, ya que comparten los mismos bloques de datos que el dataset original. Solo a medida que los datos cambian, el sistema necesita almacenar nuevas versiones de los bloques modificados, lo que incrementa progresivamente el espacio utilizado.

2. **Inmutabilidad**. Una vez creado, un snapshot no puede modificarse ni alterarse, lo que garantiza que los datos almacenados reflejen fielmente el estado del sistema en

el momento de su creación. Esta propiedad resulta especialmente útil para tareas de auditoría, control de versiones y recuperación ante errores o fallos del sistema.

3. **Recuperación de datos muy flexible**. Es posible restaurar archivos individuales, directorios completos o incluso revertir todo un dataset a un estado anterior. Esto resulta especialmente útil en situaciones como borrados accidentales, corrupción de datos o errores en actualizaciones de software.

5.2. Clones

Los **clones** en ZFS son copias editables de un dataset que se crean a partir de un snapshot. Representan una funcionalidad avanzada que permite duplicar datos de forma eficiente, rápida y con un consumo mínimo de espacio, lo que los convierte en una herramienta muy útil en distintos entornos.

A diferencia de una copia tradicional, un clon no implica duplicar inmediatamente toda la información. En su lugar, el clon y el snapshot original **comparten los mismos bloques de datos** gracias al modelo **Copy-on-Write (CoW)**. Esto significa que el clon apenas ocupa espacio adicional. Solo cuando se realizan modificaciones en el clon, ZFS escribe nuevos bloques para esos cambios, manteniendo intactos los datos originales.

Una característica importante de los clones es que son **totalmente independientes en cuanto a uso y modificación**, pero dependen del snapshot del que fueron creados. Esto implica que el snapshot original no puede eliminarse mientras existan clones asociados, ya que contiene los datos base compartidos.

Entre las principales ventajas de los clones destacan:

- **Eficiencia en el uso del espacio**: al compartir datos con el snapshot, se evita la duplicación innecesaria.
- **Creación rápida**: se generan de forma casi instantánea, independientemente del tamaño del dataset original.
- **Entornos de prueba y desarrollo**: permiten crear copias de sistemas reales para experimentar sin afectar al original.
- **Flexibilidad**: los clones pueden modificarse libremente sin alterar el snapshot ni el dataset original.

Los clones son especialmente útiles en escenarios donde se necesitan múltiples versiones de un mismo conjunto de datos, como en desarrollo de software, pruebas de sistemas o despliegue de entornos virtualizados.

Desde el punto de vista de la gestión, los clones se administran como cualquier otro dataset, pudiendo aplicarles propiedades, límites de espacio o configuraciones específicas. Sin embargo, es importante tener en cuenta su dependencia del snapshot, ya que esto puede influir en la organización y mantenimiento del sistema.

5.3. Comprensión

La compresión es una característica que permite **reducir el tamaño** de los datos almacenados de forma **automática** y **transparente**, sin necesidad de herramientas externas. Su objetivo principal es optimizar el uso del espacio de almacenamiento y, en muchos casos, mejorar el rendimiento del sistema.

Una vez habilitada, todos los datos que se escriben se comprimen automáticamente antes de almacenarse en disco y se descomprimen al ser leídos, sin que el usuario o las aplicaciones tengan que intervenir.

ZFS soporta varios algoritmos de compresión, siendo uno de los más utilizados LZ4, debido a su equilibrio entre velocidad y eficiencia. Este algoritmo permite comprimir y descomprimir datos rápidamente, lo que en muchos casos reduce la cantidad de datos que se escriben en disco y, por tanto, mejora el rendimiento general del sistema.

5.4. Deduplicación

La **deduplicación** es una función opcional de ZFS diseñada para eliminar copias redundantes de datos y así reducir de forma significativa el uso del almacenamiento. En un sistema tradicional si se almacenan dos archivos idénticos de 1 GB cada uno, estos ocuparían 2 GB en el disco duro. Pero con la deduplicación activada ZFS es capaz de almacenar una única copia de los datos. Así, ambos archivos aunque comparten los mismos bloques, el espacio total utilizado se reduce aproximadamente a 1 GB.

Este proceso no se realiza a nivel de archivo completo, sino a nivel de **bloques de datos**. ZFS divide la información en fragmentos (bloques) de tamaño variable y, para cada uno de ellos, calcula un **hash** (checksum). Este identificador es único para el contenido del bloque.

Cuando se va a escribir un nuevo bloque, el sistema compara su hash con los ya existentes en una estructura interna llamada **tabla de deduplicación (DDT, Deduplication Table)**. Si el sistema detecta que un bloque con el mismo contenido ya está almacenado, en lugar de escribirlo de nuevo en disco, simplemente crea un **puntero o referencia** al bloque existente. De esta forma, múltiples archivos pueden compartir físicamente los mismos datos sin duplicarlos. Cuando un archivo necesita ser leído, ZFS reconstruye su contenido siguiendo estas referencias, de manera totalmente transparente para el usuario y las aplicaciones.

Aunque esto es una gran ventaja a la hora de gestionar el almacenamiento, también tiene una notable desventaja. El mantenimiento de la tabla de deduplicación requiere una gran cantidad de **memoria RAM**, ya que el sistema necesita acceder rápidamente a los hashes para comprobar duplicados. Si esta tabla no cabe en memoria y debe consultarse en disco, el rendimiento ya no va ser tan bueno.

5.5. Escalabilidad

ZFS es una buena opción para gestionar grandes volúmenes de datos y adaptarse al crecimiento del almacenamiento durante el tiempo. A diferencia de los sistemas tradicionales, ZFS permite ampliar la capacidad añadiendo nuevos discos a un **zpool** sin necesidad de detener el sistema ni realizar configuraciones complejas.

Gracias a su arquitectura de **64 bits**, puede manejar cantidades de datos muy grandes lo cual es bastante útil para infraestructuras empresariales. Además, su sistema de **asignación dinámica de espacio** permite que todos los datasets compartan el almacenamiento disponible, evitando la fragmentación y el desaprovechamiento típico de las particiones fijas. Otra ventaja importante es que mantiene un buen **rendimiento incluso a gran escala**, apoyándose en mecanismos como la caché y una gestión eficiente de los datos. Esto permite que el sistema siga siendo rápido y fiable aunque el volumen de información crezca considerablemente.

6. Gestión del almacenamiento

El zpool es la unidad principal de almacenamiento en ZFS. Agrupa discos en un único espacio lógico.

A continuación se muestran algunos comandos básicos para el manejo de dichas pools:

- Crear una nueva pool:
  zpool create [-fn] [-o property=value] ... [-O file-system-property=value] ... [-m mountpoint] [-R root] pool vdev ...

- Añadir nuevos discos (vdevs) a un pool existente:
  zpool add [-fn] pool vdev …

- Elimina un dispositivo del pool:
  zpool remove pool device ...

- Destruir pool y sistemas de ficheros asociados:
  zpool destroy pool device …

- Limpiar errores;
  zpool clear [-F [-n]] pool [device]

- Importar pool (útil para mover pools entre sistemas)
  zpool export [-f] pool ...

- Importar pool (útil para mover pools entre sistemas)
  zpool import [-d dir] [-D]

- Ver espacio (tamaño, uso, espacio libre, ...):
  zpool list [-H] [-o property[,...]] [pool] ...

- Verificar datos:
  zpool scrub [-s] pool ...

- Ver estado de un pool:
  zpool status [-xv] [pool] …

Ahora que conocemos los comandos básicos, veremos un ejemplo de cómo usar algunos para una configuración básica de pools en ZFS. Se hará en el sistema operativo FreeBSD, ya que en él, ZFS es nativo, viene integrado y es estable.

Antes de empezar, vemos los discos con los que trabajaremos:

```
root@ASO_DEMO:~ # ls /dev | grep ada
ada0
ada0p1
ada0p2
ada0p3
ada1
ada2
ada3
ada4
ada5
root@ASO_DEMO:~ #
```

Como se puede apreciar, ada0 está particionado, ya que en él es donde está el sistema operativo y en consecuencia no se usará para esta demostración. Trabajaremos con el resto de discos disponibles (ada1, ada2, ada3, ada4 y ada5), cada uno con un tamaño de 10 GB.

Primero, crearemos la pool sobre la que se hará este ejemplo. Se llamará tank y será un mirror, lo que nos da redundancia: los datos se duplican en ambos discos. Para ello ejecutaremos el siguiente comando: “zpool create tank mirror ada1 ada2”. A continuación miraremos su estado con “zpool status”.

```
root@ASO_DEMO:~ # zpool create tank mirror ada1 ada2
root@ASO_DEMO:~ # zpool status
  pool: tank
 state: ONLINE
config:

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            ada1    ONLINE       0     0     0
            ada2    ONLINE       0     0     0

errors: No known data errors
```

El pool se creó correctamente, sin errores y ya aparece como online. Por defecto, su punto de montaje es /tank.

Si ejecutamos “zpool list” se podrán ver más detalles:

```
root@ASO_DEMO:~ # zpool list
NAME    SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  AL
TROOT
tank   9.50G   384K  9.50G        -         -     0%     0%  1.00x    ONLINE  -
zroot  13.5G  1.62G  11.9G        -         -     0%    12%  1.00x    ONLINE  -
```

Vemos que tank aparece junto a zroot, el pool por defecto, el cual fue creado durante la instalación del sistema operativo.

Ya que el pool que hemos creado ha sido de tipo mirror (RAID 1) y hemos usado los discos ada1 y ada2, el tamaño usable son de ~10GB, puesto que cada disco es el espejo del otro. Se pueden apreciar otros detalles, como el espacio libre (que, como la acabamos de crear, es casi todo), la fragmentación, la deduplicación y su estado (ONLINE), entre otros.

Es posible añadir dispositivos al pool mediante “zpool add”. En este caso, como estamos en un pool de tipo mirror, no podemos hacer “zpool add tank ada1”, por ejemplo, ya que estaríamos mezclando un mirror con un disco suelto (stripe), la solución es ejecutar “zpool add tank mirror ada3 ada4”. Como resultado añadimos otro mirror al pool, compuesto de ada3 y ada4:

```
root@ASO_DEMO:~ # zpool add tank mirror ada3 ada4
```

```
config:

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            ada1    ONLINE       0     0     0
            ada2    ONLINE       0     0     0
          mirror-1  ONLINE       0     0     0
            ada3    ONLINE       0     0     0
            ada4    ONLINE       0     0     0

errors: No known data errors
```

En ZFS, la eliminación de dispositivos tiene limitaciones. Mientras que es posible añadir nuevos vdevs fácilmente, la eliminación de dispositivos individuales no está generalmente permitida, especialmente en configuraciones como mirror o RAID-Z.

Para eliminarlos del pool, tendrá que ser de forma conjunta (eliminar ada3 y ada4) y se usará “zpool remove tank mirror-1” y verificaremos con “zpool status” :

```
root@ASO_DEMO:~ # zpool remove tank mirror-1
```

```
  pool: tank
 state: ONLINE
remove: Removal of vdev 1 copied 64K in 0h0m, completed on Mon Apr 20 23:57:07 2
026
        48 memory used for removed device mappings
config:

        NAME          STATE     READ WRITE CKSUM
        tank          ONLINE       0     0     0
          mirror-0    ONLINE       0     0     0
            ada1      ONLINE       0     0     0
            ada2      ONLINE       0     0     0

errors: No known data errors
```

Ahora veremos dos casos de uso para la herramienta “zfs”, que es otra parte crucial de la gestión de este sistema de archivos.

Dentro de un pool, ZFS permite crear datasets, que funcionan como sistemas de archivos independientes. A diferencia de otros sistemas, no necesitamos particionar ni asignar tamaños, ya que todos los datasets comparten dinámicamente el espacio del pool.

Además, cada dataset puede tener propiedades propias, como compresión, cuotas o permisos. Podemos crearlo mediante el comando “zfs create tank/datos”, lo que lo montará en /tank/datos, y ver información suya mediante “zfs list”.

```
root@ASO_DEMO:~ # zfs create tank/datos
root@ASO_DEMO:~ # zfs list
NAME                 USED  AVAIL  REFER  MOUNTPOINT
tank                 480K  9.20G    96K  /tank
tank/datos            96K  9.20G    96K  /tank/datos
zroot                1.62G  11.5G    96K  /zroot
zroot/ROOT           1.62G  11.5G    96K  none
zroot/ROOT/default   1.62G  11.5G  1.62G  /
zroot/home            224K  11.5G    96K  /home
zroot/home/usuario    128K  11.5G   128K  /home/usuario
zroot/tmp              96K  11.5G    96K  /tmp
zroot/usr             288K  11.5G    96K  /usr
zroot/usr/ports        96K  11.5G    96K  /usr/ports
zroot/usr/src          96K  11.5G    96K  /usr/src
zroot/var             632K  11.5G    96K  /var
zroot/var/audit        96K  11.5G    96K  /var/audit
zroot/var/crash        96K  11.5G    96K  /var/crash
```

Se hará mediante el comando “zfs snapshot tank/datos@snap1”. También puede listar todas las snapshots ejecutando “zfs list -t snapshots” y ver cuanto ocupa la snapshot y el dataset con “zfs -o space tank/datos”:

```
root@ASO_DEMO:~ # zfs snapshot tank/datos@snap1
root@ASO_DEMO:~ # zfs list -t  snapshot
NAME                USED  AVAIL  REFER  MOUNTPOINT
tank/datos@snap1     0B      -    96K  -
root@ASO_DEMO:~ # zfs list -o space tank/datos
NAME         AVAIL   USED  USEDSNAP  USEDDS  USEDREFRESERV  USEDCHILD
tank/datos  9.20G    96K        0B     96K             0B         0B
root@ASO_DEMO:~ #
```

Se usará “zpool offline tank ada1”. Este comando marca el dispositivo indicado como fuera de servicio (*offline*). A partir de este momento, el pool continúa funcionando, pero en un estado degradado.

Al consultar el estado del pool mediante “zpool status”, se observa que el sistema pasa a un estado DEGRADED y ada1 se muestra como offline, lo que indica que uno de los dispositivos no está disponible, pero que el sistema sigue siendo funcional gracias a la redundancia existente.

```
root@ASO_DEMO:~ # zpool offline tank ada1
root@ASO_DEMO:~ #
```

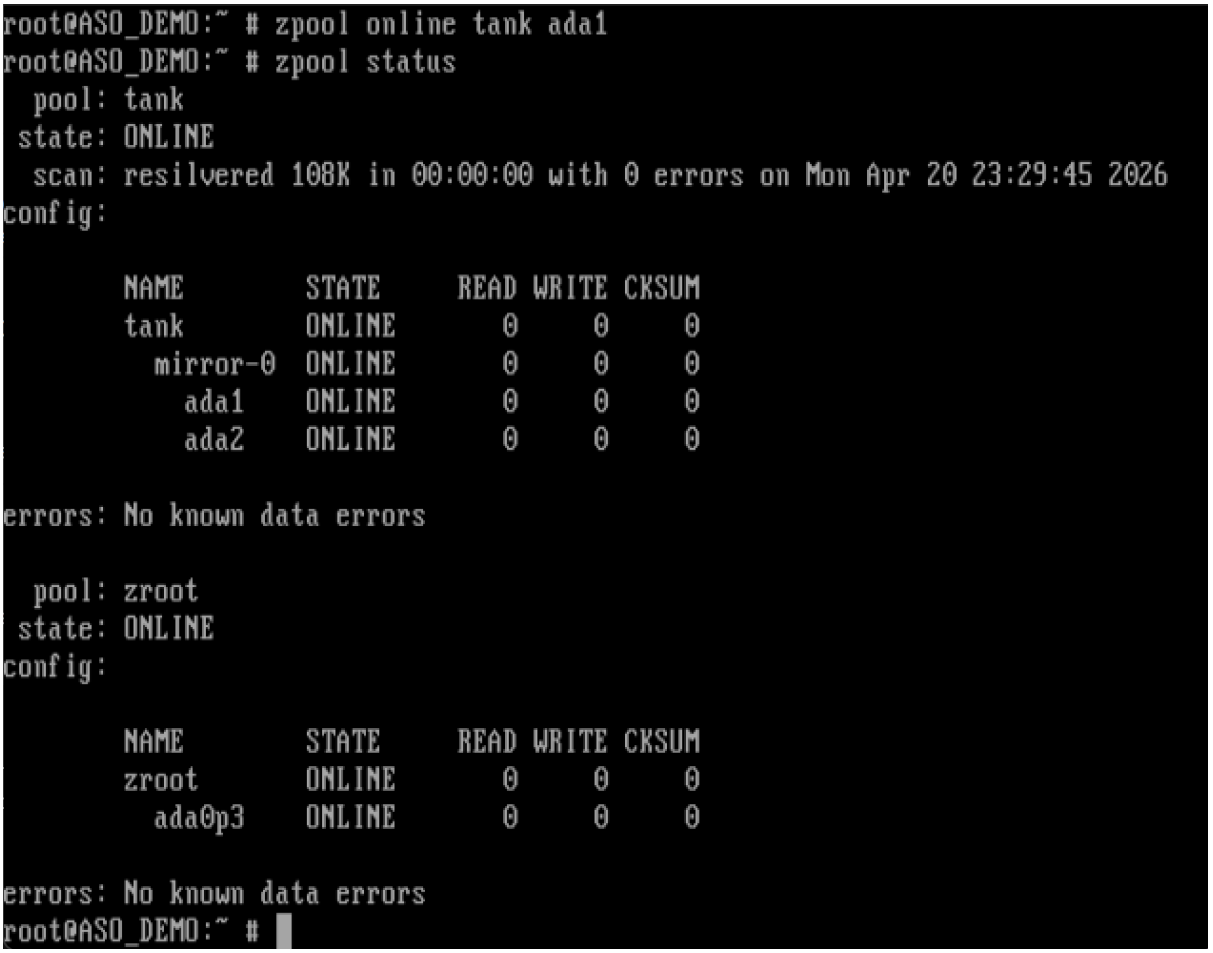
```
config:

        NAME        STATE     READ WRITE CKSUM
        tank        DEGRADED     0     0     0
          mirror-0  DEGRADED     0     0     0
            ada1    OFFLINE      0     0     0
            ada2    ONLINE       0     0     0

errors: No known data errors
```

Una vez simulado el fallo, es posible restaurar el disco al estado operativo mediante el comando “zpool online tank ada1”, el cual vuelve a poner el dispositivo en línea, permitiendo que el sistema recupere su estado original.

Tras ejecutar de nuevo“zpool status” se puede comprobar que el pool vuelve al estado ONLINE, lo que indica que todos los dispositivos están operativos y que la redundancia ha sido restaurada:



En sistemas reales, cuando un disco se reincorpora tras un fallo o se sustituye por uno nuevo, ZFS inicia automáticamente un proceso llamado resilvering, mediante el cual reconstruye los datos necesarios en el dispositivo para restaurar la redundancia. Este proceso es parecido a la reconstrucción en sistemas RAID más tradicionales, pero se realiza de forma más eficiente gracias a la arquitectura de ZFS.

Otra funcionalidad es el proceso de scrubbing, el cual es uno de los mecanismos fundamentales de mantenimiento en ZFS. Consiste en una verificación completa de todos los datos almacenados en el sistema con el objetivo de detectar y corregir posibles errores.

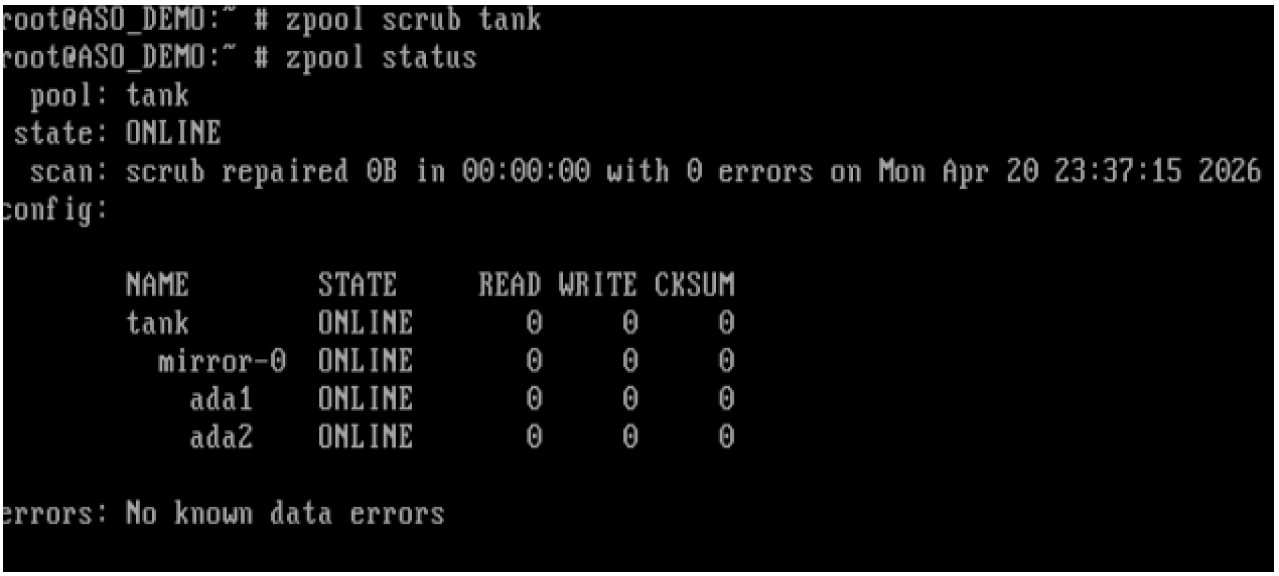
Se usa el comando “zpool scrub tank”. Durante el scrubbing, ZFS recorre todos los bloques de datos del pool y verifica su integridad utilizando los checksums almacenados en los metadatos. Si el checksum calculado y el almacenado no coinciden, el sistema identifica el bloque como corrupto.

En configuraciones con redundancia, ZFS puede recuperar automáticamente el bloque a partir de otra copia válida y reparar el error sin intervención del administrador. Este

mecanismo de autocorrección es una de las principales ventajas de ZFS frente a sistemas de archivos tradicionales.

Esta herramienta presenta ventajas frente a otras como fsck, como que se puede ejecutar con el sistema en funcionamiento, sin necesidad de desmontar el sistema de archivos ni interrumpir el servicio.

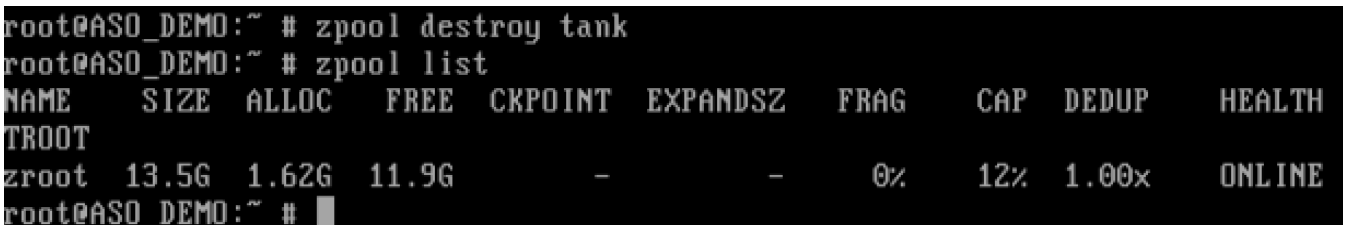
El progreso y estado del scrubbing pueden consultarse mediante “zpool status”:



En la práctica, el scrubbing se realiza de forma periódica, como medida preventiva para garantizar la integridad de los datos a largo plazo.

Para acabar, ZFS permite eliminar completamente un pool de almacenamiento mediante el comando “zpool destroy tank”

El resultado es la eliminación tanto de la estructura del pool como todos los datos almacenados en él, liberando los discos para su reutilización. Se trata de una operación irreversible, por lo que debe utilizarse con precaución.



Como podemos ver, tras el destroy, tank se elimina del sistema y, al ejecutar comandos como “zpool list” o “zpool status”, la pool que creamos ya no aparece.

Esta demostración ha sido para un pool de tipo mirror, pero hay más configuraciones que no exploraremos, pero sí serán comentadas brevemente.

La primera será stripe (sin redundancia). Los datos se reparten entre los discos para mejorar el rendimiento, pero no existe protección: si falla un disco, se pierden todos los datos.

Se crea mediante “zpool create tank ada1 ada2”. Es la opción por defecto.

Otra es RAIDZ1. Utiliza paridad para proteger los datos. Permite el fallo de un disco, aprovechando mejor el espacio que un mirror y se creará con “zpool create tank raidz ada1 ada2 ada3”.

Por último, RAIDZ2. Similar a RAIDZ1, pero con doble paridad. Permite el fallo de hasta dos discos, ofreciendo mayor seguridad en sistemas críticos.

El comando será “zpool create tank raidz2 ada1 ada2 ada3 ada4”.

7. Desventajas de ZFS

A pesar de ser uno de los sistemas de archivos más avanzados disponibles, ZFS presenta una serie de desventajas que pueden dificultar su uso en determinados entornos.

7.1. Consumo de recursos

ZFS está diseñado para ofrecer alta integridad y rendimiento, pero a costa de un gran uso de memoria RAM. Esto tiene varias razones que lo explican (y que imposibilitan que sea de otra manera).

La primera razón son los checksums y metadatos usados por ZFS. Esto se debe a que cada operación de lectura/escritura implica cálculos adicionales para validar la integridad de los datos. Además, los metadatos (snapshots, registros de transacciones, …) se almacenan en memoria. En discos grandes con millones de archivos, la gestión de estos metadatos pueden consumir una cantidad considerable de memoria.

La principal causa para este elevado consumo, es la caché ARC (Adaptive Replacement Cache). El algoritmo que implementa esta memoria es una espada de doble filo, ya que, pese a su gran desempeño, también es altamente demandante en cuanto a uso de memoria RAM, pudiendo llegar a ocupar más del 50% de esta si se dan las circunstancias.

Otra cuestión es la deduplicación, cuya función es que bloques de datos idénticos se almacenen una sola vez, lo que reduce considerablemente el uso en disco.

Si varios archivos contienen los mismos bloques de datos o cualquier otro dato aparece varias veces en la misma pool, ZFS almacena solo una copia. En lugar de almacenar muchas copias de un dato, almacena una copia y un número arbitrario de punteros a esa copia. Sólo cuando ningún archivo utiliza los datos estos se eliminan.

Esto perjudica al uso de memoria, ya que el hash de cada bloque se almacena en RAM para verificar si existe en el sistema.

Para mitigar estos problemas, en sistemas con limitaciones de tamaño en memoria RAM, es recomendable desactivar la deduplicación, reducir el uso de snapshots y limitar el tamaño que puede ocupar la ARC, modificando el valor que toma el parámetro zfs_arc_max (lo que se puede hacer creando el fichero de configuración /etc/modprobe.d/zfs.conf)

7.2. Complejidad técnica

La potencia de ZFS como sistema de ficheros va de la mano con lo complejo que es de manejar correctamente, sobre todo para un público no familiarizado con este tipo de sistemas. Mientras que sistemas como ext4 o NTFS están diseñados con un enfoque más

simple y directo, ZFS introduce una arquitectura avanzada que requiere una comprensión más profunda por parte del administrador.

En primer lugar, ZFS presenta una curva de aprendizaje considerable. Para poder utilizarlo de manera eficiente, es necesario comprender una serie de conceptos que no existen en otros sistemas de archivos convencionales. Entre ellos destacan los zpools (pools de almacenamiento), los vdevs (dispositivos virtuales), los diferentes niveles de RAID-Z, así como funcionalidades avanzadas como los snapshots, los clones o las propiedades configurables de los datasets (compresión, deduplicación, cuotas, etc.). Estos elementos aportan una gran flexibilidad, pero también obligan al administrador a adquirir conocimientos adicionales antes de poder gestionar el sistema de forma correcta.

Además, ZFS exige una planificación previa del almacenamiento mucho más cuidadosa. A diferencia de otros sistemas donde es posible modificar la estructura de almacenamiento con relativa facilidad, en ZFS muchas decisiones deben tomarse en el momento inicial de creación del sistema. Por ejemplo, la elección del tipo de RAID-Z o la organización de los vdevs dentro de un pool son aspectos críticos que determinan tanto el rendimiento como la tolerancia a fallos. Estas decisiones, una vez implementadas, no pueden modificarse fácilmente sin recrear el pool y migrar los datos, lo que puede resultar costoso en términos de tiempo y recursos. Por este motivo, una mala planificación inicial puede condicionar negativamente el funcionamiento del sistema a largo plazo.

Por otro lado, la complejidad de ZFS incrementa el riesgo de configuraciones incorrectas. Una configuración inadecuada puede tener consecuencias importantes, como una reducción del rendimiento, un uso ineficiente del almacenamiento o incluso una menor tolerancia a fallos de la esperada. Por ejemplo, una elección incorrecta del tipo de vdev o una distribución desigual de los discos puede generar cuellos de botella o limitar la capacidad de recuperación ante fallos. Asimismo, el uso de funcionalidades avanzadas como la deduplicación sin disponer de los recursos adecuados puede provocar un impacto negativo en el rendimiento del sistema.

Otro aspecto a considerar es que la administración de ZFS requiere una mayor familiaridad con herramientas específicas como zpool y zfs, así como una comprensión clara de cómo interactúan entre sí los distintos componentes del sistema. Esto implica que, frente a sistemas más simples, el administrador debe dedicar más tiempo tanto al aprendizaje inicial como al mantenimiento del sistema.

7.3. Limitaciones en ciertos sistemas

ZFS presenta una serie de limitaciones relacionadas principalmente con su integración en distintos sistemas operativos y su dependencia de determinadas condiciones de hardware.

Uno de los aspectos más relevantes está relacionado con las licencias de software. ZFS está distribuido bajo la licencia CDDL (Common Development and Distribution License), mientras que el kernel de Linux se rige por la licencia GPL (General Public License). Ambas licencias son incompatibles desde el punto de vista legal, lo que impide que ZFS se incluya directamente dentro del kernel de Linux. Como consecuencia, en sistemas Linux ZFS se distribuye como un módulo externo que debe cargarse de forma independiente.

Esta situación tiene varias implicaciones prácticas. Por un lado, puede requerir una instalación manual o adicional por parte del usuario, especialmente en distribuciones donde ZFS no viene preinstalado. Además, la compatibilidad entre versiones del kernel y el módulo de ZFS puede generar ciertos problemas, especialmente tras actualizaciones del sistema, donde es necesario recompilar o actualizar el módulo correspondiente. Esto añade un nivel extra de complejidad en la administración en comparación con sistemas de archivos integrados de forma nativa.

Además, el nivel de integración de ZFS no es uniforme en todos los sistemas operativos. En Solaris, donde fue originalmente desarrollado, ZFS está completamente integrado en el sistema, lo que permite aprovechar todas sus funcionalidades de forma óptima. En FreeBSD, ZFS también presenta una integración muy sólida y estable, siendo ampliamente utilizado en entornos profesionales. Sin embargo, en Linux, aunque el soporte ha mejorado considerablemente gracias a proyectos como OpenZFS, su integración sigue siendo menos directa al no formar parte del kernel del sistema. Esta diferencia puede influir en aspectos como la facilidad de instalación, el mantenimiento del sistema o el soporte a largo plazo.

Por otro lado, ZFS también presenta ciertas limitaciones relacionadas con el hardware. Para funcionar de forma óptima, requiere configuraciones relativamente específicas. A parte de la notoria demanda de memoria RAM, que comentamos anteriormente, se aconseja el uso de discos lo más homogéneos posible en cuanto a capacidad y rendimiento, ya que la mezcla de dispositivos con características muy diferentes puede afectar negativamente al comportamiento del sistema.

En entornos críticos, también se recomienda el uso de memoria ECC (Error-Correcting Code), ya que permite detectar y corregir errores en la RAM. Esto es especialmente

importante en ZFS, dado que el sistema confía en la integridad de los datos en memoria para garantizar la corrección de ellos en disco. En caso de no usarla, existe un mayor riesgo de que errores en RAM se propaguen al almacenamiento.

Cuando ZFS se ejecuta en hardware no adecuado o con recursos insuficientes, pueden aparecer problemas como una disminución del rendimiento, una mayor latencia en operaciones de entrada/salida o menor eficiencia en el uso del almacenamiento.

En conjunto, estas limitaciones hacen que, aunque ZFS sea una solución muy potente, su implementación debe evaluarse cuidadosamente en función del sistema operativo y del hardware disponible.

8. Comparativa con otros sistemas de archivos

8.1. Comparación con ext4

En primer lugar, una de las principales diferencias es que **ext4 es únicamente un sistema de archivos**, mientras que ZFS combina tanto el sistema de archivos como el gestor de volúmenes. Esto quiere decir que en ext4 es necesario utilizar herramientas adicionales como LVM (Logical Volume Manager) para gestionar el almacenamiento de forma avanzada, mientras que ZFS integra estas funciones de manera nativa.

En cuanto a la **integridad de los datos**, ZFS ofrece una protección mucho más avanzada. ext4 incluye mecanismos básicos como journaling para evitar inconsistencias tras fallos del sistema pero no tiene un mecanismo de verificación mediante checksums en todos los datos. Esto hace que sea más susceptible a la corrupción de datos en caso de interrupciones o apagones durante una operación de escritura. En cambio, ZFS gracias a la tecnología de copy-on-write garantiza que los datos se **escriben de forma completa**, no quedando a medias. Si se produce alguna interrupción durante el proceso de escritura, esta se completará con éxito o se cancelará por completo. Esto lo hace más fiable y evita cualquier daño a los datos. Además teniendo en cuenta las funciones de redundancia como RAIDZ (que permiten almacenar los datos en varios discos, lo que facilita la recuperación de los datos en caso de que uno de los discos falle) lo hace más potente que ext4 en este aspecto.

Respecto al **rendimiento**, ext4 aunque no cuenta con todas las características avanzadas de ZFS, en general es **más rápido y más ligero** en su funcionamiento. Por eso se

considera una buena opción para muchas aplicaciones y recursos, ya que tiene menor consumo de memoria y CPU. ZFS, por su parte, puede ofrecer mejor rendimiento en sistemas bien dimensionados gracias a su sistema de caché (ARC y L2ARC), aunque requiere más recursos para funcionar de manera óptima.

Otra diferencia importante es la **gestión del almacenamiento**. En ext4, se debe planificar de antemano cómo dividir el espacio de un disco duro en particiones.

Esto provoca varios problemas:

- ➔ **Infraflexibilidad:** Si se crea una partición para el directorio /home de 200 GB y luego los usuarios ocupan 250 GB, el sistema se quedará sin espacio aunque el disco duro tenga otros 300 GB libres en otra partición. No hay forma de que ext4 "tome prestado" espacio de otra partición.

- ➔ **Desperdicio:** Normalmente se tiende a asignar particiones más grandes de lo necesario, dejando espacio sin usar (por ejemplo, una partición de 500 GB que solo usa 100 GB). Ese espacio desperdiciado no puede ser aprovechado por otras particiones.

- ➔ **Dificultad de cambio:** Cambiar el tamaño de las particiones o redistribuir el espacio suele requerir desmontar el sistema de archivos, usar herramientas de terceros (como gparted o fdisk) y correr el riesgo de corrupción de datos si ocurre un fallo durante el proceso.

ZFS al trabajar con zpools (grupos de almacenamiento) sobre los cuales se crean los datasets, estos compiten por el espacio total del pool de forma dinámica. Esta característica ya explicada anteriormente nos aporta:

- ➔ **Flexibilidad total:** Si se tienen dos datasets (por ejemplo, tank/datos y tank/backups) dentro del mismo zpool de 1 TB, no hay límites fijos. El dataset tank/datos puede ocupar 800 GB y tank/backups 200 GB, o bien 950 GB y 50 GB, o cualquier combinación mientras la suma no supere 1 TB. No hay que redimensionar nada; ZFS asigna espacio sobre la marcha.

- ➔ **Sin desperdicio:** Todo el espacio del zpool está disponible para cualquier dataset que lo necesite. No hay particiones infrautilizadas que no puedan ceder su espacio sobrante a otras. El espacio que no se usa en un dataset queda automáticamente disponible para los demás.

- ➔ **Crecimiento sin molestias:** Si se necesita más capacidad, se pueden añadir discos nuevos al zpool (si el diseño de redundancia lo permite, por ejemplo, añadiendo un par de discos en espejo o un conjunto de discos en RAID-Z). Una vez añadidos, el espacio extra se integra al pool y todos los datasets pueden aprovecharlo inmediatamente, sin necesidad de redimensionar nada, desmontar sistemas de archivos ni reiniciar.

- ➔ **Reservas y límites opcionales:** Aunque por defecto es todo dinámico, ZFS permite al administrador establecer cuotas (límite máximo que puede ocupar un dataset) y reservas (espacio mínimo garantizado para un dataset). Esto ofrece control cuando se necesita, pero sin imponer la rigidez de las particiones fijas.

En cuanto a la **facilidad de uso**, EXT4 resulta más accesible para usuarios **sin experiencia** o para entornos donde se prioriza la **simplicidad y la rapidez** de configuración. ZFS, aunque incomparablemente más potente y con más funcionalidades, introduce una capa adicional de complejidad debido a su nueva terminología, sus numerosas opciones y su forma de administración diferente. Por ello, ZFS es más **adecuado para administradores** con experiencia o para entornos donde se necesitan sus ventajas (integridad, snapshots, compresión, etc.). Para un escritorio doméstico o un servidor pequeño con necesidades básicas, EXT4 sigue siendo la opción más simple y práctica.

8.2. Comparación con NTFS

El sistema de archivos NTFS (New Technology File System) es el estándar utilizado en sistemas Windows y está diseñado para ofrecer fiabilidad, seguridad y compatibilidad en entornos de uso general.

Al igual que ocurre con otros sistemas tradicionales, **NTFS** es únicamente un **sistema de archivos**, mientras que ZFS integra también funciones de gestor de volúmenes. Esto implica que NTFS depende de herramientas externas o configuraciones adicionales (al igual que lo hace ext4) para gestionar el almacenamiento de forma avanzada, mientras que ZFS lo hace de manera nativa mediante zpools.

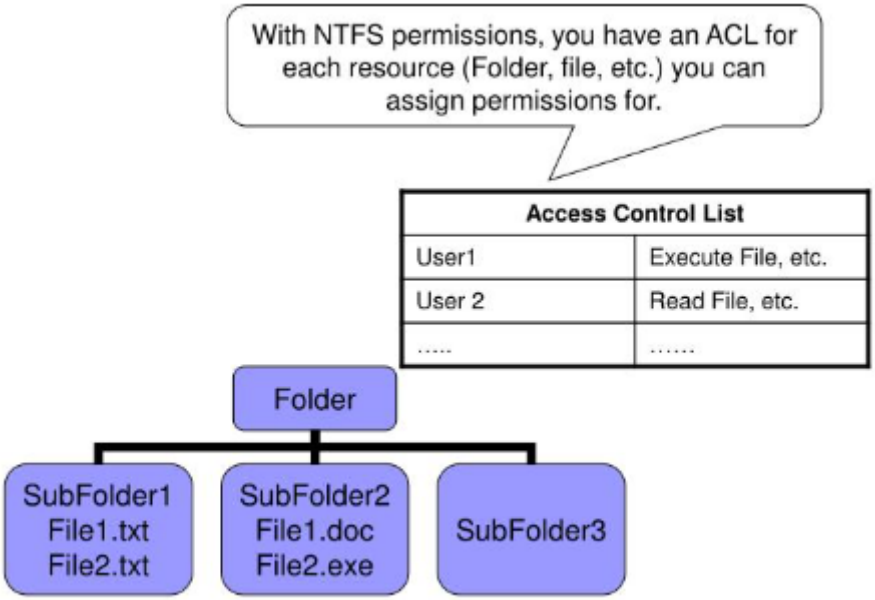
En cuanto a la **integridad de los datos**, NTFS incluye mecanismos como el journaling, que ayudan a mantener la consistencia del sistema tras fallos inesperados. (igual que lo hace ext4). No obstante, no ofrece un sistema completo de verificación mediante checksums para todos los datos como ZFS, por lo que no puede detectar ni corregir ciertos tipos de corrupción silenciosa.

Respecto a la **seguridad**, NTFS dispone de características avanzadas como listas de control de acceso (ACL), cifrado de archivos (EFS) y compresión.

- ➔ **Listas de control de acceso (ACL)**: NTFS ofrece un sistema de permisos muy granular y maduro. Se pueden asignar permisos específicos a usuarios y grupos individuales a nivel de archivo o directorio, con derechos como lectura, escritura, ejecución, modificación, control total, etc. Las ACL de NTFS son compatibles con Active Directory, lo que permite una gestión centralizada de permisos en dominios Windows.

- ➔ **Cifrado de archivos (EFS):** El Encrypting File System de NTFS permite cifrar archivos y carpetas de forma transparente para el usuario. El cifrado está integrado en el sistema de archivos y se basa en certificados del usuario. Está especialmente diseñado para proteger datos en equipos portátiles o en unidades extraíbles contra accesos no autorizados en caso de pérdida o robo.

- ➔ **Compresión:** NTFS incluye compresión transparente a nivel de archivo o directorio. Los datos se comprimen y descomprimen automáticamente al leerlos o escribirlos, ahorrando espacio en disco a costa de cierto impacto en el rendimiento. Aunque no es propiamente una característica de seguridad, a menudo se menciona junto a ellas por estar integrada en el mismo sistema.



ZFS también permite un control detallado de permisos y, en algunas implementaciones, soporte de cifrado, aunque su enfoque principal está más orientado a la integridad y gestión del almacenamiento.

- ➔ **Permisos y ACL:** ZFS soporta permisos Unix tradicionales (propietario, grupo, otros con lectura/escritura/ejecución) y también ACL NFSv4. Estas ACL son muy potentes y permiten un control detallado comparable al de NTFS, con herencia de permisos, acceso para usuarios o grupos específicos, y derechos como leer datos, escribir datos, ejecutar, borrar, cambiar permisos, etc. Sin embargo, su uso es menos común fuera de entornos empresariales o servidores NAS.

- ➔ **Cifrado:** El soporte de cifrado en ZFS depende de la implementación. En sistemas como FreeBSD, TrueNAS o ZFS en Linux (OpenZFS) las versiones recientes incluyen cifrado nativo a nivel de dataset. Este cifrado es transparente, eficiente y se integra con otras características como la compresión y los snapshots. No obstante, históricamente el cifrado llegó más tarde a ZFS que en NTFS y su adopción no es tan universal. Algunas implementaciones antiguas de ZFS carecen de cifrado o requieren capas externas (como geli en FreeBSD o LUKS en Linux).

En términos de **rendimiento**, NTFS ofrece un **rendimiento muy estable** en tareas cotidianas como el arranque del sistema, la apertura de aplicaciones, la copia de archivos de tamaño pequeño o mediano y la navegación por directorios. En cuanto al consumo de recursos, NTFS funciona perfectamente con equipos de escritorio que dispongan de 2 a 4 GB de RAM.

El comportamiento de NTFS es predecible, se comporta de manera consistente sin generar grandes picos de latencia. Además, el rendimiento se degrada de **forma gradual** a medida que se llena el disco, sin caídas bruscas.

Pero tiene ciertos límites: su principal cuello de botella es el rendimiento con directorios que contienen una **cantidad masiva de archivos**, del orden de decenas o cientos de miles. En estos casos, operaciones como listar o buscar dentro de dichos directorios pueden volverse lentas. También es menos eficiente en entornos con alta concurrencia, es decir, con muchos accesos simultáneos, como ocurre en servidores de archivos atendiendo a cientos de usuarios.

ZFS, por su parte, puede ofrecer un rendimiento superior en sistemas bien configurados, especialmente en servidores, gracias a su sistema de caché y optimización de datos ya explicados anteriormente.

En cuanto a **funcionalidades avanzadas**, ZFS destaca claramente sobre NTFS. ZFS incluye de forma nativa un conjunto completo de características potentes: snapshots (instantáneas), clones, compresión en tiempo real y deduplicación. Todas estas funcionalidades están profundamente integradas en su arquitectura, funcionan a nivel de bloque y son altamente eficientes en el uso de espacio y recursos.

NTFS, por su parte, dispone de ciertas características que podrían considerarse análogas, pero con importantes limitaciones. La más relevante es el servicio de copias de sombra (**Volume Shadow Copy**), que permite crear instantáneas del volumen en momentos determinados. Sin embargo, estas copias de sombra no alcanzan el nivel de integración, flexibilidad y eficiencia que ofrecen los snapshots de ZFS. Por ejemplo, las copias de sombra de NTFS **consumen espacio reservado de antemano**, no se pueden replicar con la misma facilidad, y carecen de la capacidad de generar clones a partir de ellas de forma instantánea y con mínimo consumo de almacenamiento adicional.

Por último, en lo relativo a la **compatibilidad**, NTFS tiene la ventaja de ser el **sistema de archivos estándar en Windows**, lo que lo hace ampliamente soportado en ese entorno.

ZFS, en cambio, está más extendido en sistemas tipo Unix/Linux y no cuenta con soporte nativo en Windows, lo que puede limitar su adopción en ciertos casos.

9. Casos de uso

9.1. Servidores empresariales

Uno de los principales ámbitos de aplicación de ZFS es el entorno de los servidores empresariales. En este tipo de sistemas, la pérdida o corrupción de datos puede tener consecuencias graves tanto a nivel económico como operativo, por lo que la fiabilidad del almacenamiento es un requisito fundamental.

ZFS destaca en este contexto gracias a su capacidad para garantizar la integridad de los datos mediante mecanismos como los checksums y la autocorrección, comentados anteriormente. Esto permite detectar y corregir errores de forma automática, incluso en situaciones donde otros sistemas de archivos no serían capaces de identificar la corrupción de datos o no hacerlo de manera tan eficiente.

Además, su arquitectura basada en pools de almacenamiento facilita la gestión de grandes volúmenes de información. En entornos empresariales, donde el almacenamiento crece de forma constante, la posibilidad de añadir nuevos discos sin interrumpir el servicio supone una ventaja significativa.

Otro aspecto relevante es su capacidad para gestionar configuraciones RAID por software (RAID-Z), eliminando la necesidad de controladoras hardware específicas. Esto reduce costes y simplifica la infraestructura, al tiempo que mantiene un alto nivel de tolerancia a fallos.

Por todo ello, ZFS es ampliamente utilizado en servidores que requieren alta disponibilidad, como servidores de bases de datos, servidores de aplicaciones o sistemas de almacenamiento corporativo.

9.2. Sistemas NAS

ZFS es popular en sistemas NAS (Network Attached Storage), cuyo objetivo principal es proporcionar almacenamiento centralizado accesible desde múltiples dispositivos a través de la red. Es utilizado para soluciones de almacenamiento en red como TrueNAS.

En este contexto, ZFS aporta numerosas ventajas, como poder gestionar grandes cantidades de datos de forma eficiente y segura. Además, la combinación de RAID-Z y checksums garantiza que los datos almacenados están protegidos frente a fallos de hardware y corrupción.

La funcionalidad de snapshots resulta especialmente útil en estos sistemas, ya que permite realizar copias de seguridad instantáneas del sistema de archivos sin afectar al rendimiento ni consumir grandes cantidades de espacio. Esto facilita la recuperación de datos en caso de errores o circunstancias imprevistas.

Además, ZFS permite aplicar políticas de almacenamiento a nivel de dataset, como cuotas, compresión o deduplicación, proporcionando gran flexibilidad en la gestión de recursos.

Gracias a estas características, ZFS se ha convertido en una de las soluciones más utilizadas en entornos NAS tanto a nivel doméstico avanzado como en pequeñas y medianas empresas.

9.3. Backup y recuperación

Uno de los casos de uso más importantes de ZFS es su aplicación en sistemas de backup y recuperación de datos. La protección de la información es un aspecto crítico en cualquier sistema, y ZFS proporciona herramientas muy eficientes para este propósito.

Gracias al mecanismo CoW, podemos aprovecharlo para la toma de snapshots. Debido a que hemos hecho una copia del bloque en otra parte del sistema de archivos, el bloque antiguo sigue existiendo, a pesar de que ha sido marcado como libre. Con CoW, el sistema de archivos está llegando lentamente hasta el final del disco. Puede pasar mucho tiempo antes de que los viejos bloques liberados se reescriban. Si se ha tomado una instantánea, se trata como un sistema de archivos real. Si un bloque se sobrescribe después de que se ha tomado una snapshot, se copia en el sistema de archivos de instantáneas. Esto es posible, porque la instantánea es una copia del árbol de hash en ese momento exacto. Gracias a esto, las instantáneas son considerablemente baratas, y a menos que los bloques con instantáneas se sobrescriban, apenas ocupan espacio.

Además, ZFS permite realizar replicaciones de datos mediante los comandos zfs send y zfs receive. Estos comandos permiten transferir snapshots entre sistemas, lo que facilita la implementación de copias de seguridad remotas y planes de recuperación ante desastres.

Otra ventaja importante es la capacidad de restaurar rápidamente versiones anteriores de los datos. En caso de fallo, corrupción o error humano, es posible volver a un estado anterior del sistema de forma sencilla, lo que reduce significativamente los tiempos de recuperación.

9.4. Big Data

ZFS también encuentra aplicación en entornos de Big Data, donde se manejan grandes volúmenes de información que requieren sistemas de almacenamiento escalables, fiables y eficientes.

En este ámbito, es habitual trabajar con grandes cantidades de datos distribuidos, lo que hace necesario un sistema capaz de gestionar el almacenamiento de forma flexible. ZFS, gracias a su arquitectura basada en pools, permite añadir capacidad de almacenamiento de forma progresiva sin necesidad de rediseñar la estructura existente.

Además, la integridad de los datos es un aspecto crítico en Big Data, ya que los errores pueden propagarse rápidamente a través de grandes conjuntos de datos. El uso de checksums en ZFS permite detectar y corregir errores de forma automática, lo que garantiza la fiabilidad de la información almacenada.

Otra característica relevante en este contexto es la compresión de datos, que permite reducir el espacio necesario para almacenar grandes volúmenes de información, mejorando además el rendimiento al reducir las operaciones de entrada/salida.

Gracias a la caché ARC, también se consigue una gran eficiencia a la hora del acceso a datos.

Sin embargo, es importante tener en cuenta que en entornos de Big Data, ZFS suele utilizarse en combinación con otras tecnologías y sistemas distribuidos, como Hadoop o sistemas de almacenamiento en red, formando parte de una arquitectura más compleja.

10. Bibliografía

https://docs.freebsd.org/en/books/handbook/zfs/
https://www.raidz-calculator.com/raidz-types-reference.aspx
https://tadeubento.com/2024/aarons-zfs-guide-appendix-visualizing-the-zfs-intent-log-zil/

https://deepwiki.com/openzfs/zfs/5.4-zfs-properties-and-delegation#interaction-between-txg-and-zil
https://datazone.de/en/aktuelles/zfs-arc-l2arc-cache-tuning/
https://wiki.ubuntu.com/ZFS/ZPool
https://linux.die.net/man/8/zpool
https://avidandrew.com/understanding-zfs-datasets.html
https://www.cyberciti.biz/faq/how-to-set-up-zfs-arc-size-on-ubuntu-debian-linux/
https://cr0x.net/es/compresion-zfs-zstd-niveles-sin-quemar/
https://www.hostragons.com/es/blog/comparacion-de-sistemas-de-archivos-ntfs-ext4-apfs-y-zfs/
https://www.partitionwizard.com/news/zfs-vs-ext4.html