



A.S.O. - ZFS

(Zettabyte File System)

Administración de Sistemas Operativos - Universidad de La Coruña
Abril, 2025

Nicolás Villar Philippon
Rubén Barba Paz

ÍNDICE DE LA EXPOSICIÓN SOBRE ZFS

1. Introducción - *Págs. 1-3*

- ¿Qué es un sistema de archivos?
- Breve historia de ZFS (creado por Sun Microsystems en 2001, incluido en OpenSolaris, evolución hacia OpenZFS).
- Características que lo diferencian de otros sistemas de archivos como ext4, XFS o Btrfs.

2. Conceptos Fundamentales de ZFS - *Págs. 4-8*

- **Pooled Storage:** ZFS gestiona almacenamiento como un pool en lugar de volúmenes tradicionales.
- **Copy-on-Write (CoW):** Protección contra corrupción de datos.
- **Integridad de datos:** Checksums y detección/corrección de corrupción.
- **Snapshots y Clones:** Instantáneas y copias eficientes.

3. Estructura y Componentes - *Págs. 9-24*

- **Zpools:** La base del almacenamiento en ZFS.
- **Vdevs (Virtual Devices):** Tipos de vdevs (RAID-Z (los tres niveles), espejos, discos individuales). *Striping* en ZFS: Uso de discos individuales sin redundancia (*RAID 0-like*), ventajas y riesgos.
- **Datasets:** Tipos de datasets en ZFS (file systems, volúmenes y snapshots).
- **Propiedades de ZFS:** Compresión, deduplicación, encriptación y cuotas.

4. Gestión y Administración - *Págs. 25-30*

- **Comandos básicos:**
 - `zpool create`, `zpool status`, `zpool scrub`, `zfs create`, `zfs set`.
- **Scrubbing y Mantenimiento:** Cómo funciona la verificación y reparación de datos en ZFS.
- **RAID-Z vs RAID tradicional:** Explicación de los niveles RAID-Z1, RAID-Z2 y RAID-Z3.
- **Snapshots y backups eficientes con *zfs send* y *zfs receive*.**

5. Casos de Uso y Desventajas - Págs. 31-34

- **Casos de uso recomendados:**

- Servidores de almacenamiento y NAS (TrueNAS, Proxmox).
- Copias de seguridad y archivado de datos.
- Sistemas empresariales y bases de datos.

- **Desventajas de ZFS:**

- Alto consumo de RAM (recomendado 1 GB de RAM por TB de almacenamiento).
- Problemas de compatibilidad con Linux en el pasado (licencias CDDL vs GPL).

6. Conclusión - Págs. 35-36

7. Bibliografía - Pág. 37

1. Introducción

¿Qué es un sistema de archivos?

Un sistema de archivos es la estructura y método utilizados por un sistema operativo para organizar, almacenar, recuperar y administrar archivos en un dispositivo de almacenamiento, como un disco duro, una unidad SSD o una memoria USB. Actúa como una interfaz entre el hardware de almacenamiento y el usuario, permitiendo la creación de archivos, la gestión de permisos y la organización en directorios.

Los sistemas de archivos pueden diferir en características como seguridad, redundancia, velocidad y compatibilidad. Algunos ejemplos comunes incluyen FAT32, NTFS (Windows), ext4 (Linux) y APFS (MacOS). En entornos más avanzados, sistemas como ZFS y Btrfs ofrecen mayor integridad de datos y funciones avanzadas de gestión.

Breve historia de ZFS

ZFS (Zettabyte File System) fue creado en 2001 por Sun Microsystems como un sistema de archivos de nueva generación para su sistema operativo Solaris. Su desarrollo fue liderado por Jeff Bonwick y Matthew Ahrens, con el objetivo de diseñar un sistema de archivos altamente escalable, confiable y fácil de administrar.

En 2005, ZFS fue lanzado como parte de OpenSolaris, el proyecto de código abierto de Solaris. Sin embargo, cuando Oracle adquirió Sun Microsystems en 2010, dejó de mantener OpenSolaris y convirtió ZFS en un software propietario dentro 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. 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.

Características que lo diferencian de otros sistemas de archivos como ext4, XFS o Btrfs.

ZFS se diferencia de otros sistemas de archivos como ext4, XFS y Btrfs por su enfoque en integridad de datos, escalabilidad y facilidad de administración. Aquí están sus principales características distintivas:

- Integridad de datos y corrección de errores

Usa sumas de verificación en cada bloque de datos para detectar y corregir errores silenciosos (bit rot). Otros sistemas como ext4 y XFS no tienen esta capacidad integrada.

- Almacenamiento en conjunto (Pooled Storage)

No usa particiones tradicionales. En su lugar, administra el almacenamiento como un pool dinámico. Permite agregar discos sin necesidad de reformatar.

- Snapshots y clones

Permite crear instantáneas (snapshots) de todo el sistema de archivos sin impacto en el rendimiento. Los clones son instantáneas editables, útiles para pruebas y backups.

- Compresión y deduplicación

Soporta compresión transparente para ahorrar espacio en disco. Puede evitar duplicación de datos con deduplicación (aunque consume más RAM).

- RAID-Z: Alternativa mejorada al RAID tradicional

ZFS integra su propio sistema de RAID (RAID-Z), eliminando la dependencia de hardware RAID. Mejora la protección de datos y evita problemas como el "write hole" de RAID-5.

- Escalabilidad extrema

Puede gestionar hasta 256 billones de zettabytes, mucho más que ext4 o XFS. No necesita herramientas externas como fsck para corregir errores.

- Autogestión y facilidad de administración

Comandos como zfs y zpool facilitan la gestión sin requerir configuraciones complejas. Es un sistema de archivos y gestor de volúmenes en uno, a diferencia de ext4 y XFS, que dependen de LVM.

ZFS es ideal para servidores y almacenamiento de alto rendimiento, mientras que ext4 y XFS son más adecuados para sistemas tradicionales. Btrfs tiene características similares a ZFS, pero con menos madurez y estabilidad en comparación.

2. Conceptos Fundamentales de ZFS

Pooled Storage

En los sistemas de archivos tradicionales (como ext4 o NTFS), el almacenamiento se organiza en volúmenes o particiones fijas. Cada volumen está vinculado a un dispositivo físico específico y su tamaño es estático, lo que dificulta la gestión y la escalabilidad.

Sin embargo, ZFS introduce un nuevo paradigma: el almacenamiento basado en pools. En lugar de asignar discos a volúmenes individuales, estos se combinan en un zpool, creando una única reserva de almacenamiento expandible dinámicamente.

¿Cómo funciona en la práctica?

En ZFS no se formatean discos individuales como en otros sistemas de archivos, sino que se agrupan en vdevs (Virtual Devices), que luego se combinan para formar un zpool.

Un zpool puede contener varios discos organizados en diferentes configuraciones (RAID-Z, espejos, discos individuales).

A diferencia de los volúmenes tradicionales, un pool puede crecer dinámicamente al añadir más discos, sin necesidad de reformato ni de mover datos ni particiones manualmente.

Esto presenta numerosas ventajas:

- **Flexibilidad extrema:** No es necesario definir tamaños fijos, se pueden asignar recursos según la demanda.
- **Mejor gestión del espacio:** Todos los sistemas de archivos dentro del zpool comparten el espacio libre disponible, no se desperdicia almacenamiento.
- **Escalabilidad sin interrupciones:** Se pueden añadir nuevos discos al pool sin afectar a los datos existentes. Fácil de escalar en servidores y sistemas de almacenamiento grandes.
- **Redundancia y seguridad de datos:** ZFS permite configuraciones de redundancia (RAID-Z, espejos).

Entonces, viendo esta serie de ventajas, **¿por qué no lo implementan Windows y Linux?**

→ En Linux:

Tiene dos motivos principales por los que no implementa ZFS:

- 1.- ZFS fue desarrollado por Sun Microsystems bajo la licencia CDDL (Common

Development and Distribution License), que es incompatible con la GPL (General Public License) de Linux. La GPL exige que cualquier código vinculado al kernel de Linux sea también GPL, pero la CDDL no permite esto.

2.- Linux ya tiene varios sistemas de archivos avanzados como ext4, XFS (potente para servidores, optimizado para alto rendimiento y escalabilidad) y Btrfs (desarrollado por Oracle, similar a ZFS, pero con una implementación compatible con la GPL), que cumplen muchas de las funciones de ZFS.

→ En Windows:

Microsoft tiene su propio sistema de archivos avanzado: ReFS (Resilient File System), diseñado para mejorar la integridad de datos en servidores y reemplazar NTFS en algunas aplicaciones.

Además, Microsoft no tiene interés en adoptar un sistema de archivos que no controla. Prefiere mantener su ecosistema cerrado y optimizarlo para su propio software y hardware.

Copy-on-Write (CoW)

Se trata de un mecanismo de escritura que evita la corrupción de datos al garantizar que los bloques originales nunca se sobrescriben directamente.

Los datos modificados se escriben en una nueva ubicación y, solo después de que la escritura sea exitosa, se actualizan los punteros para que apunten al nuevo bloque. Esto no ocurre en ext4 o NTFS.

¿Qué pasos sigue ZFS para realizar el CoW?

1.- **Lectura del bloque original:** Antes de modificar un archivo, ZFS localiza el bloque de datos original. En sistemas tradicionales como ext4 o NTFS, este bloque sería modificado directamente, lo que podría causar corrupción si la operación de escritura falla (por ejemplo, debido a un corte de energía).

2.- **Escritura en un nuevo bloque:** En lugar de sobrescribir el bloque original, ZFS escribe la nueva versión en un bloque diferente del disco. Esto garantiza que la información previa se mantenga intacta hasta que la operación se complete. Si la escritura falla, el sistema aún tiene el bloque original y no hay pérdida de datos.

3.- **Verificación de integridad:** Se genera un checksum para el nuevo bloque y se compara con los datos escritos. Si el checksum no coincide, significa que ha habido un error durante la escritura, y el sistema intentará escribir en otro bloque. Esta verificación detecta errores en el

almacenamiento (bit rot, fallos en discos, errores de memoria, etc.).

4.- Actualización del puntero: Si la escritura es exitosa, se actualiza la estructura de metadatos de ZFS para que el sistema de archivos apunte al nuevo bloque. Este cambio se realiza de forma atómica, es decir, o se actualizan correctamente los punteros, o no se cambia nada. Gracias a esto, el sistema de archivos nunca queda en un estado inconsistente, incluso en caso de un fallo repentino del sistema.

5.- Eliminación del bloque antiguo: Una vez confirmado el cambio, el espacio del bloque original se marca como libre. Sin embargo, si hay snapshots del sistema de archivos, el bloque no se elimina inmediatamente porque sigue siendo referenciado por la instantánea.

CoW es una de las mayores fortalezas de ZFS que no solo protege contra la corrupción de datos, sino que también facilita snapshots eficientes y no compromete al rendimiento.

Es por esto que es un sistema de archivos ideal para servidores, almacenamiento crítico y bases de datos, donde la integridad de los datos es esencial.

Integridad de datos

La integridad de datos es un punto de gran relevancia. Los discos pueden corromper datos debido a errores en la memoria RAM, caché, degradación a lo largo del tiempo (bit rot), controladores o cables defectuosos, errores de escritura o lectura...

ZFS resuelve este problema mediante el empleo de checksums en cada bloque de datos.

Cada bloque de datos incluye un checksum SHA-256. Es importante recalcar que este checksum no se almacena en el propio bloque, sino en la estructura superior del árbol de datos:

- Cuando se escribe un bloque, ZFS calcula el checksum y lo almacena en la estructura padre.
- Cuando se lee un bloque, se recalcula el checksum y se compara con el original.
- Finalmente, si no coinciden, se detecta la corrupción. En caso de no haber redundancia de datos, se registra el error y se informa al usuario, evitando la lectura de datos erróneos. Sin embargo, en caso de existir redundancia (RAID-Z, espejo, etc), se recuperan los datos desde dicha copia y se corrige el bloque dañado.

Además, ZFS incluye una función llamada scrubbing, consistente en un proceso de verificación periódica de los datos almacenados. Se encarga de recorrer todo el sistema de archivos, verificando los checksums.

En caso de encontrar un error, lo intenta corregir con copias redundantes.

Esta función es similar a los comandos “chkdsk” o “fsck”, pero mucho más eficiente, automática y sin necesidad de desmontar el sistema de archivos.

Snapshots y Clones

Un snapshot, como sabemos, es una copia de un sistema de archivos/volumen en un momento específico. ZFS cuenta con esta función, pero mejorada.

Aquí, los snapshots son instantáneos (sin importar el tamaño del sistema de archivos), no afectan al rendimiento y pueden ser restaurados en cualquier momento rápidamente.

Sin embargo, cuenta con una gran característica con respecto a los snapshots convencionales: son muy eficientes en espacio.

Esta gran eficiencia es debida al mecanismo Copy-on-Write (CoW) anteriormente explicado. ZFS no copia los datos, sino que simplemente guarda referencias a los bloques existentes. Esto se traduce en que solo ocupan espacio si se modifican archivos. Si un bloque no cambia, el snapshot y el sistema de archivos lo comparten. Si se modifica, ZFS escribe los nuevos datos en un bloque diferente, sin afectar al original. En caso de eliminar archivos del sistema de archivos, los bloques originales no se eliminan mientras el snapshot siga existiendo. De esta forma, gracias a que mantiene una referencia al bloque original, se asegura que se pueda restaurar en cualquier momento.

Su gran eficiencia y velocidad convierten a los snapshots en excelentes herramientas para backups y recuperación ante fallos, aunque no hay que olvidar la importancia de eliminar snapshots antiguos, que acabarán siendo innecesarios y ocuparán tanto espacio como lo que ocupen aquellos archivos modificados (ya sea agregados o eliminados) desde su creación.

Ejemplo de creación de snapshot: `zfs snapshot pool/dataset@snapshot1`

Se crea un snapshot del dataset “pool/dataset”, llamado “snapshot1”

Ejemplo de restauración de snapshot: `zfs rollback pool/dataset@snapshot1`

Como dato extra, destacar la posibilidad de poder restaurar un único archivo desde el snapshot sin necesidad de restaurar el snapshot entero desde `/pool/.zfs/snapshot/...`

Cualquiera de las dos opciones ofrece un bajo tiempo de espera, son operaciones prácticamente instantáneas.

Cabe mencionar en este punto la utilidad de los clones. Un clon es una copia creada a partir de un snapshot que puede modificarse de forma independiente. Ambos son igual de eficientes en espacio, pues emplean referencias a bloques originales (hasta que, en el caso del clon, se modifican).

A diferencia de los snapshots, los clones son ideales para pruebas de software, creación de entornos de desarrollo y despliegue rápido de copias de datos sin duplicación.

Ejemplo de creación de clon: `zfs clone pool/dataset@snapshot1 pool/clone1`

crea “clone1” a partir de “snapshot1”, permitiendo modificarlo sin afectar el snapshot original

Sin embargo, el clon es dependiente del snapshot del que fue creado. En caso de querer convertirlo en un dataset independiente podría promocionarse con:

`zfs promote pool/clone1`

Promocionar un clon permite convertirlo, si fue creado para pruebas, en una nueva versión principal; o bien si se desea borrar un snapshot antiguo, pero no perder el clon creado a partir de él.

3. Estructura y componentes

Zpools

Un zpool (ZFS Storage Pool) es la base del sistema de archivos ZFS (Zettabyte File System). En términos simples, un zpool es el almacenamiento subyacente sobre el cual se crean sistemas de archivos ZFS. Es un conjunto de dispositivos físicos (discos) combinados en una única entidad lógica, que proporciona almacenamiento gestionado por ZFS.

En cuanto a la estructura, es un conjunto de vdevs (uno o más de almacenamiento (single disk, mirror y RAID-z) y cero o más vdevs de soporte (caché, log...)).

En cuanto a la utilidad, es un objeto de almacenamiento que puede dividirse en datasets (parecidos a carpetas) y vols (similares a dispositivos de bloque/caracteres simples, como discos sin procesar).

Es importante mencionar que un zpool no contiene discos reales directamente, estos están contenidos en los vdevs.

1. Características Principales de un Zpool

a) Uso de dispositivos físicos

Un zpool está compuesto por uno o más dispositivos físicos, que pueden ser:

- Discos enteros (/dev/sdX, /dev/nvmeXnY, etc.).
- Particiones de discos.
- Archivos de imagen (zpool create pool_name /ruta/archivo.img).
- Dispositivos en red (iSCSI, NFS).

b) Redundancia y protección contra fallos

ZFS ofrece varios niveles de protección para los zpools mediante VDEVs (Virtual Devices). Se pueden configurar de distintas maneras:

- **Single Disk (sin redundancia):** un solo disco, sin protección contra fallos.
- **Mirror:** equivalente a RAID 1, los datos se replican en varios discos.
- **RAID-Z1:** similar a RAID 5, con una paridad que permite tolerar la falla de un disco.
- **RAID-Z2:** equivalente a RAID 6, con dos discos de paridad, permitiendo la falla de dos discos.
- **RAID-Z3:** soporta la falla de hasta tres discos.
- **Striping (RAID 0):** se divide la carga entre varios discos, pero sin redundancia.

Cada una de estas configuraciones afecta el rendimiento, la disponibilidad y la capacidad efectiva del almacenamiento.

c) Expansión y gestión dinámica

- Se pueden agregar discos a un zpool en caliente.
- Soporta eliminación de dispositivos en ciertas configuraciones.
- La capacidad del zpool se expande automáticamente al añadir discos.
- Se pueden realizar snapshots y clones sin impacto en el rendimiento.

2. Capacidad

ZFS es un sistema de archivos de 128 bits, por lo que tiene mucha mayor capacidad de almacenamiento que otros sistemas de 64 bits, como por ejemplo NTFS (sistema de ficheros de Windows). ZFS fue diseñado de forma que los límites fuesen tan grandes, que fuesen encontrados sólo teóricamente en vez de en la práctica. Así, los límites teóricos son:

- número de entradas en una carpeta : 248
- número de atributos en un archivo: 248

- bytes como tamaño máximo de un archivo: 264
- número de dispositivos en un zpool: 264
- número máximo de zpool en un sistema: 264
- número máximo de sistemas de archivos en un zpool: 264
- bytes como tamaño máximo de algún zpool: 278

3. Ventajas de Usar ZFS y Zpools

- **Autocorrección de datos:** Gracias a su modelo de checksums, detecta y corrige errores silenciosos.
- **Snapshots y clones instantáneos:** Permite tomar snapshots sin impacto en el rendimiento y clones para pruebas o backups.
- **Compresión y deduplicación:** Soporta compresión transparente (lz4, gzip, zstd) y deduplicación de datos para ahorrar espacio.
- **Escalabilidad ilimitada:** Soporta hasta 256 zettabytes, ideal para almacenamiento masivo.
- **No requiere fsck:** ZFS mantiene la integridad del sistema de archivos sin necesidad de chequeos manuales.
- **Caché avanzado (ARC y L2ARC):** Optimiza el acceso a datos en RAM (ARC) y SSD (L2ARC) para mejorar el rendimiento.

Vdevs (Virtual Devices)

ZFS elimina del todo la administración de volúmenes. En vez de tener que crear volúmenes virtualizados, ZFS agrega dispositivos a una agrupación de almacenamiento. Esta describe las características físicas del almacenamiento (organización del dispositivo, redundancia de datos, etc.) y actúa como almacén de datos arbitrario en el que se pueden crear sistemas de archivos. Estos, se limitan a dispositivos individuales y les permite compartir espacio en el disco con todos los sistemas de archivos de la agrupación. Ya no es necesario predeterminedir el tamaño de los mismos, crece automáticamente en el espacio asignado a la agrupación de almacenamiento. Al incorporar un nuevo almacenamiento, todos los sistemas de archivos de la agrupación pueden usar de inmediato el espacio en el disco adicional sin procesos complementarios. En muchos sentidos, esta funciona del mismo modo que un sistema de memoria virtual: si se agrega al sistema un módulo de memoria DIMM (módulo de memoria RAM que se conecta directamente en las ranuras de la placa base de los ordenadores

personales y está constituido por pequeños circuitos impresos que contienen circuitos integrados de memoria), el sistema operativo no obliga a ejecutar comandos para configurar la memoria y asignarla a los procesos individuales. Todos los del sistema utilizan automáticamente la memoria adicional.

A diferencia de los sistemas de ficheros tradicionales que residen encima de un solo dispositivo subyacente y por lo tanto requieren un gestor de volúmenes separado, ZFS se apoya en espacios de almacenamiento virtuales (virtual storage pools).

Los espacios se construyen a partir de uno o más dispositivos virtuales, o vdevs (la forma en la que se referencia cualquier tipo de dispositivo de almacenamiento, ya sea local o remoto). Un pool puede ser de tipo simple (uno o más vdevs sin redundancia), mirror (dos o más vdevs en pares, en modalidad espejo), o RAID Z (tres o más vdevs con paridad).

- ❖ **Stripe:** todos los discos se meten dentro de un pool y se suman las capacidades de los diferentes discos. En caso de fallo de un disco, se pierde toda la información. Este tipo de pool es como el RAID 0 pero permite incorporar múltiples discos en él.
- ❖ **Mirror:** todos los discos se meten dentro de un pool y se replican, siendo su capacidad máxima, la misma que la menor capacidad de uno de los discos. Todos están replicados en el mirror, por tanto, solamente se perderá información si se rompen todos los discos del mirror. Este tipo de pool es como un RAID 1, pero permite incorporar múltiples discos en él. Es el vdev más rápido con tolerancia a fallos.
- ❖ **RAID Z1:** todos los discos se meten dentro del pool. Tiene un disco por fila reservado para paridad. Suponiendo que tengan la misma capacidad, se suma la capacidad de todos ellos menos la de uno (si tenemos 3 discos de 4TB, el espacio de almacenamiento efectivo sería de 8TB). Permite que uno se pueda romper y la información quede intacta. El funcionamiento es como un RAID 5. En un RAID Z1 se recomienda tener 3, 5 o 9 discos en cada vdev. Si falla uno, no tendremos pérdida de datos, en caso de fallar un segundo disco, perderemos toda la información.
- ❖ **RAID Z2:** todos los discos se meten dentro del pool. Utiliza paridad dual. Suponiendo que tengan la misma capacidad, se suma la capacidad la de todos menos la de dos discos (si tenemos 4 discos de 4TB, el espacio de almacenamiento efectivo sería de 8TB). Permite que dos de los discos se puedan romper, y la información quede intacta. El funcionamiento es similar a un RAID 6 que todos conocemos. En un

RAID Z2 se recomienda tener 4, 6 o 10 discos en cada vdev. Si fallan dos, no tendremos pérdida de datos, en caso de fallar un tercer disco, perderemos toda la información.

- ❖ **RAID Z3:** todos los discos se meten dentro del pool. Utiliza paridad triple. Suponiendo que tengan la misma capacidad, se suma la de todos menos la de tres (si tenemos 5 discos de 4TB, el espacio de almacenamiento efectivo sería de 8TB). Permite que tres de los discos se puedan romper, y la información quede intacta. En un RAID Z3 se recomienda tener 5, 7 o 11 discos en cada vdev. Si fallan tres, no tendremos pérdida de datos, en caso de fallar un cuarto disco, perderemos toda la información.

Otras configuraciones que se podrán realizar con ZFS, son definir un disco como Hot Spare para que, en caso de fallo de un disco, automáticamente entre en funcionamiento este de respaldo y comience el proceso de resilvering (regeneración de los datos utilizando este nuevo disco que acabamos de introducir al pool). Obviamente, un SPARE debe tener al menos el tamaño de un disco fallido para poder reemplazarlo automáticamente. Si un pool solo tiene un vdev de almacenamiento, no se necesitan SPAREs. Su utilidad reside en que puede dar servicio a cualquier vdev del pool que se degrade. En uno con un solo vdev, es mucho más lógico ajustar la paridad que implementar SPAREs.

También tenemos la posibilidad de definir un disco como caché. El caché vdev (también conocido como L2ARC) es solo un búfer de lectura. El caché vdev tiene una limitación de escritura bastante estricta para evitar que consuma los SSD, lo que también significa que la probabilidad de que tenga disponible un bloque que ya se ha leído es menor de lo esperado. Si a esto le sumamos las altísimas tasas de aciertos del ARC (algoritmo de caching para almacenar datos y metadatos de sistemas de ficheros en DRAMs, para acelerar las operaciones de los servidores) principal, los caché vdev no suelen obtener muchos aciertos en la mayoría de las cargas de trabajo. Este nunca tendrá una alta tasa de aciertos, ya que, una vez que se produce uno, ese bloque se devuelve al ARC principal. La pérdida de un vdev de caché no afecta negativamente al pool, salvo por la pérdida de la aceleración que se podría haber obtenido del propio caché. ZFS se ayuda del caché para acelerar enormemente las transferencias de datos, tanto en lectura como escritura secuencial y aleatoria.

Otra posibilidad con este sistema de archivo es definir un disco como LOG (ZIL (ZFS Intent Log)) para almacenar los registros de escritura que deberían haber ocurrido. Esto es de ayuda en caso de un corte en el suministro eléctrico. Comencemos hablando del LOG aclarando un error común: los vdevs LOG no son una caché ni un búfer de escritura. En funcionamiento normal, ZFS guarda las escrituras sincronizadas en disco dos veces: una, inmediatamente, en el registro de intentos de ZFS y, posteriormente, en el almacenamiento principal. En ausencia de un vdev LOG, ambas escrituras se realizarán en los vdevs del almacenamiento principal. Aunque parezca contradictorio, estas escrituras dobles mejoran el rendimiento de las escrituras síncronas, principalmente al reducir la latencia mediante el uso de bloques ZIL preasignados, lo que evita perder tiempo buscando una ubicación para los datos y reduce la fragmentación y, por lo tanto, la latencia de búsqueda. Un vdev LOG es simplemente un lugar para guardar el ZIL, que no es el almacenamiento principal. Normalmente, el vdev LOG es un dispositivo de muy baja latencia, por lo que la sincronización de datos con él será significativamente más rápida que con el almacenamiento principal. Las escrituras en el LOG no se vuelven a leer a menos que se produzca un fallo del sistema o del kernel, por lo que no se debe considerar como un "búfer de escritura". Aunque un vdev LOG rápido puede acelerar enormemente las cargas de trabajo con alta sincronización, no tendrá ningún efecto en las cargas de trabajo sin muchas escrituras síncronas, y la mayoría de las cargas de trabajo, incluso las de servidor, no incluyen muchas. Ejemplos de cargas de trabajo con alta sincronización incluyen exportaciones de NFS (Network File System: protocolo de nivel de aplicación, según el Modelo OSI. Es utilizado para sistemas de archivos distribuidos en el entorno de una LAN), bases de datos e imágenes de máquinas virtuales. En las primeras versiones de ZFS, perder un vdev LOG significaba perder todo el pool. En el moderno, su pérdida no pone en peligro el pool ni sus datos; lo único en riesgo son los datos corruptos que contiene, que aún no se han enviado al disco. Un LOG puede usar topologías individuales o espejo (de cualquier tamaño); no puede usar RAIDz ni DRAID. Se pueden tener tantos como se desee, si se tienen tantas escrituras síncronas que se prefiere distribuir la carga entre varios LOG.

Por otro lado, cabe mencionar los SPECIAL vdev. Es la clase de soporte más reciente, introducida para compensar las desventajas de los vdevs DRAID (que abordaremos más adelante). Al asociar un SPECIAL vdev a un pool, todas las escrituras de metadatos futuras en ese pool se realizarán en él, no en el almacenamiento principal. La pérdida de un dispositivo virtual especial, al igual que la de un dispositivo virtual de almacenamiento,

implica la pérdida de todo el pool. Por ello, este dispositivo debe tener una topología tolerante a fallos. Por lo tanto, un pool con almacenamiento RAIDz3 necesita un vdev especial con cuatro réplicas: el dispositivo virtual de almacenamiento puede sobrevivir a tres fallos de disco, por lo que el SPECIAL vdev también debe ser capaz de sobrevivir a tres fallos de disco.

Una última configuración que se podría hacer en ZFS es el DRAID. Esta topología se diseñó para sistemas con 60 o más unidades y reemplaza un grupo tradicional de dispositivos virtuales de almacenamiento y SPARE por un único y enorme dispositivo virtual que incorpora funcionalidad RAIDz y SPARE. Supongamos que se tienen exactamente sesenta discos y se quieren integrar en un DRAID. El primer paso es decidir el ancho de cada banda y el nivel de paridad, que a diferencia de RAIDz, no están estrictamente limitados por la cantidad de unidades disponibles. Se podrían haber usado nueve vdevs RAIDz2 de seis anchos (4 discos para datos y dos para paridad, pudiéndose recuperar de fallos en dos discos) más seis SPARE. Con esto, el número de vdevs queda fijo, no pudiéndose colocar más de diez vdevs de seis anchos en un total de sesenta discos.

Sin embargo, DRAID ofrece mayor flexibilidad: si bien se puede elegir un DRAID2:4:4 (dos bloques de paridad por banda, cuatro bloques de datos por banda y cuatro discos de repuesto distribuidos en el conjunto), también se puede elegir un DRAID3:4:4. Obviamente, tres bloques de paridad y cuatro bloques de datos por banda no suman un divisor par de las sesenta unidades totales, pero no es necesario, ya que DRAID ajustará la banda (conjunto de bloques de datos y paridad que se distribuyen entre los discos del grupo) según sea necesario para que se ajuste a la matriz de unidades (disposición estructurada de los discos dentro del conjunto (*zpool*)).

Hay un problema en esos detalles: establecer un ancho de banda DRAID que no se divida uniformemente entre el número de discos probablemente afectará al rendimiento. Pero cuando se tienen 60 discos o más, este tipo de problema de rendimiento suele desaparecer: el cuello de botella en un sistema de sesenta unidades suele residir en las propias controladoras de las unidades, no en las unidades individuales.

En una configuración DRAID, en lugar de asignar un disco específico a una franja específica dentro de un RAIDz en particular, cada uno de los miembros del DRAID vdev se distribuye

entre los 60 discos. Cuando uno falla, en lugar de perder una franja completa, la pérdida es de 1/60 de cada uno de los 60 miembros.

Si una unidad falla en nuestro DRAID vdev, los fragmentos de bloque que pertenecían a la unidad fallida se reconstruyen a partir de la paridad y se redistribuyen en la capacidad de repuesto distribuida dentro del DRAID. Esta redistribución es mucho más rápida que una redistribución tradicional en un vdev de repuesto. A modo de comparación, consideremos el ejemplo anterior: seis RAIDz2 de nueve anchos con seis repuestos. Cuando falla un disco, se leen de los ocho restantes y se escribe en uno solo de repuesto. En cambio, con DRAID, dado que el principal cuello de botella de un resilver tradicional (la velocidad de escritura) ahora se distribuye entre las cincuenta y nueve unidades restantes. Terminamos leyendo de cincuenta y nueve unidades y escribiendo en cincuenta y nueve, lo que se completará mucho más rápido que el proceso de ocho y uno de un RAIDz tradicional.

Sin embargo, los vdevs DRAID tienen sus desventajas. La más importante es que pierden la capacidad de los vdevs RAIDz tradicionales para almacenar datos en anchos de banda dinámicos. Si se tiene un DRAID2:4:4, cada banda debe tener seis discos de ancho (cuatro de datos más dos de paridad), incluso si esa banda solo almacena un bloque de metadatos de 4 KiB.

Aquí es donde entra en juego el vdev SPECIAL: si bien puede ser útil en un pool tradicional, es prácticamente necesario en un pool con un vdev DRAID amplio. Al colocar los metadatos (y, opcionalmente, otros bloques muy estrechos) en el SPECIAL, no es necesario desperdiciar cantidades excesivas de espacio de disco y rendimiento almacenándolos en bandas de ancho completo.

Datasets:

Un dataset en ZFS es el término genérico que engloba cualquiera de las entidades que se pueden crear dentro de un zpool. Esto incluye:

- **Sistemas de archivos (filesystems):** Son datasets que se montan en el sistema operativo y se comportan como sistemas de archivos tradicionales, pero con características avanzadas (como compresión, deduplicación, cuotas, reservas y herencia de propiedades). La raíz del zpool ya es un dataset de tipo filesystem, y

puedes crear otros (por ejemplo, para separar datos de usuarios, medios, backups, etc.) de forma muy rápida y sencilla.

- **Volúmenes (zvols):** Son datasets que se exponen como dispositivos de bloques en el directorio `/dev/zvol/`. Estos se usan cuando necesitas un dispositivo de bloques virtual, por ejemplo, para swap, para correr máquinas virtuales o para exportarlos vía iSCSI. Aunque también pueden tener propiedades como compresión y snapshots, no se montan como sistemas de archivos.
- **Snapshots:** Son instantáneas inmutables de un dataset (ya sea filesystem o zvol) que capturan el estado completo en un punto del tiempo. Gracias a la arquitectura de copy-on-write de ZFS, se crean casi al instante y usan espacio solo para los bloques que cambian después de la instantánea.
- **Clones:** Son snapshots modificables. Se crean a partir de un snapshot y, en ese momento, comparten bloques con el original, pero a medida que se realizan modificaciones, se asignan nuevos bloques, lo que permite tener dos sistemas de archivos independientes que parten de la misma base.

Características clave de los datasets en ZFS

1. Herencia de propiedades:

Los datasets permiten asignar propiedades (como compresión, deduplicación, cuotas, reservas, punto de montaje, etc.) que se pueden heredar a los datasets hijos. Esto significa que, al crear un nuevo dataset dentro de otro, hereda las configuraciones del padre, aunque se pueden sobrescribir.

2. Administración flexible:

- Crear, destruir y renombrar datasets es muy rápido (casi tan rápido como crear un directorio en un sistema de archivos tradicional).
- Los comandos principales son:
 - `zfs create <pool>/<dataset>` para crear uno nuevo.
 - `zfs list` para ver los datasets existentes.
 - `zfs set` y `zfs get` para configurar y consultar propiedades.

- zfs destroy para eliminar datasets, lo cual es casi instantáneo.

3. Segmentación lógica del almacenamiento:

Gracias a los datasets, puedes dividir el almacenamiento de un zpool en partes lógicas, asignando diferentes políticas o configuraciones a cada uno. Por ejemplo, podrías tener un dataset para datos multimedia con compresión y otro para bases de datos con un tamaño de registro (recordsize) ajustado a sus necesidades.

4. Snapshots y clones para copias de seguridad y versionado:

La capacidad de crear snapshots permite recuperar versiones anteriores de datos sin necesidad de realizar copias completas. Los clones, que son derivados modificables de snapshots, facilitan la experimentación o la creación de entornos de prueba sin afectar al dataset original.

5. Aplicaciones y delegación:

Los datasets permiten una administración granular. Puedes delegar la administración de ciertos datasets a usuarios o grupos, lo que es muy útil en entornos multiusuario o en sistemas NAS donde cada usuario puede tener su propio espacio con cuotas y configuraciones personalizadas.

¿Por qué usar datasets en ZFS?

Utilizar datasets te ofrece múltiples ventajas:

- **Gestión granular:** Permite separar y configurar distintos tipos de datos con propiedades específicas (por ejemplo, diferentes niveles de compresión, cuotas, reservas, etc.).
- **Flexibilidad y escalabilidad:** Puedes crear y destruir datasets según las necesidades, sin preocuparte por particionar físicamente el disco.
- **Snapshots y recuperación:** Los datasets hacen posible el uso de snapshots y clones para protección y recuperación rápida de datos.

- **Optimización de rendimiento:** Al separar los datos en diferentes datasets, puedes ajustar parámetros como el tamaño de bloque (recordsize) para adaptarlos a distintos tipos de carga de trabajo.

En esencia, un dataset es la forma en que ZFS organiza el almacenamiento lógico dentro de un zpool. Ya sea que se trate de un sistema de archivos, un volumen, un snapshot o un clon, todos se gestionan de forma unificada y se benefician de las características avanzadas de ZFS como la integridad de datos, la compresión transparente y la administración dinámica del espacio.

Esta arquitectura integrada —donde el almacenamiento físico (el zpool) y la estructura lógica (los datasets) trabajan conjuntamente— es una de las principales fortalezas de ZFS, ya que simplifica la administración y mejora la eficiencia y la resiliencia de los datos.

Propiedades de ZFS

1. Compresión

La compresión en ZFS se aplica de forma transparente a nivel de dataset. Esto significa que cuando se escribe nueva información, ZFS intenta comprimirla automáticamente antes de almacenarla en disco, sin que la aplicación o el usuario tenga que realizar ninguna acción adicional. Si la compresión resulta efectiva (es decir, el tamaño comprimido es menor que el original), se almacena la versión comprimida; de lo contrario, se guarda la versión sin comprimir.

Algoritmos de compresión

ZFS soporta varios algoritmos de compresión, cada uno con características distintas:

- **lzjb:** Es el algoritmo original de ZFS; es rápido y ofrece una compresión moderada.
- **gzip:** Disponible en varios niveles (por ejemplo, gzip-1 hasta gzip-9); a niveles bajos es rápido y a niveles altos ofrece mejor compresión a costa de mayor consumo de CPU.

- **lz4:** Actualmente es el algoritmo recomendado en la mayoría de los casos porque ofrece un excelente equilibrio entre velocidad y ratio de compresión.
- **zstd:** Recientemente se ha incorporado, ofreciendo una gama amplia de niveles para balancear velocidad y eficiencia.

Como ejemplo, podemos mencionar el comando para habilitar la compresión lz4 en un dataset: `zfs set compression=lz4 pool/dataset`

Puntos a tener en cuenta:

- **Aplicación a nuevos datos:** La compresión sólo afecta a los datos que se escriben después de haber activado la propiedad; los datos ya existentes no se comprimen retroactivamente.
- **Impacto en el rendimiento:** Al disminuir el tamaño físico de los datos, se pueden reducir las operaciones de I/O, lo que puede mejorar el rendimiento, aunque en sistemas con CPU limitada, algoritmos más pesados como gzip-9 podrían afectar al rendimiento.

Casos de uso

- **Datos altamente redundantes o repetitivos:** Por ejemplo, archivos de texto o logs, donde la compresión puede ahorrar mucho espacio.
- **Sistemas con discos de baja capacidad:** Habilitar la compresión puede prolongar la vida útil del almacenamiento al reducir el espacio ocupado.

2. Deduplicación

La deduplicación es una función que evita almacenar múltiples copias de bloques idénticos de datos. En vez de escribir el mismo bloque varias veces, ZFS guarda una única copia y referencia esa copia desde múltiples ubicaciones (incluso a través de snapshots y clones).

Cómo funciona

- **Tabla de deduplicación (DDT):** Cada vez que se escribe un bloque, ZFS calcula un hash (checksum) y lo compara con la DDT para determinar si ya existe un bloque idéntico.
- **Memoria intensiva:** Esta operación requiere mantener una tabla en RAM para realizar búsquedas rápidas de bloques duplicados. Por ello, en sistemas donde no se repiten muchos datos o cuando la memoria es limitada, la deduplicación puede tener un impacto negativo en el rendimiento. Para habilitar la deduplicación en un dataset, se ha de realizar el siguiente comando: `zfs set dedup=on pool/dataset`

Debido al alto consumo de memoria y CPU, se recomienda habilitar la deduplicación solo cuando se espere un alto grado de datos duplicados (por ejemplo, en entornos de virtualización donde muchas máquinas virtuales usan el mismo sistema operativo).

Casos de uso

- **Máquinas virtuales:** Donde muchas imágenes comparten bloques idénticos (por ejemplo, sistemas operativos o aplicaciones comunes).
- **Backups y archivos de datos redundantes:** Situaciones en las que se almacenen versiones o copias similares de archivos.

3. Encriptación

La encriptación en ZFS se aplica a nivel de dataset y cifra los datos en reposo, protegiéndolos contra accesos no autorizados en caso de pérdida o robo de discos. La encriptación es completamente transparente para el usuario: los datos se cifran al escribirse y se descifran al leerse.

Características y funcionamiento

- **Configuración a nivel de dataset:** Se establece cuando se crea el dataset o se puede modificar posteriormente.
- **Algoritmos y formatos:** Aunque la implementación puede variar según la plataforma (por ejemplo, Oracle Solaris o OpenZFS en Linux), generalmente se usa un algoritmo de cifrado robusto (como AES).
- **Administración de claves:** Las claves de encriptación pueden establecerse mediante passphrase o mediante llaves en disco. Es posible cambiar la llave de encriptación sin necesidad de reescribir todos los datos, ya que ZFS utiliza un esquema de “envoltura de llaves” (wrapping keys).

Para crear un dataset encriptado con passphrase usaremos el siguiente comando: `zfs create -o encryption=on -o keyformat=passphrase -o keylocation=prompt pool/secure_dataset`

Consideraciones importantes

- **Herencia:** Los datasets hijos heredan la configuración de encriptación del padre, lo que facilita la administración.
- **Snapshots y clones:** Los snapshots y clones de un dataset encriptado mantienen la encriptación y comparten la misma clave, lo que garantiza la coherencia de la seguridad.
- **Rendimiento:** La encriptación puede impactar el rendimiento, pero en sistemas modernos y con hardware adecuado (por ejemplo, CPU con aceleración de AES), el impacto suele ser mínimo.

Casos de uso

- **Datos sensibles:** Sistemas que manejan información confidencial (datos financieros, personales, etc.).
- **Dispositivos portátiles o NAS:** Donde el riesgo de robo o pérdida es mayor, la encriptación protege la información almacenada.

4. Quotas

Las quotas son límites que se pueden imponer en los datasets para restringir la cantidad de espacio de almacenamiento que pueden consumir. Esto permite una gestión muy granular y evita que un solo conjunto de datos consuma todo el espacio disponible en el zpool.

Tipos de quotas y reservas

- **Quota:** Limita el espacio total que puede ser usado por un dataset y sus descendientes (incluyendo snapshots).
- **Refquota:** Limita únicamente el espacio usado por el dataset en sí, sin incluir sus datasets hijos.
- **Reservation:** Reserva una cantidad mínima de espacio para un dataset, garantizando que siempre estará disponible ese espacio, sin importar el uso de otros datasets en el mismo pool.
- **Refreservation:** Reserva espacio para el dataset, excluyendo el espacio que pueda ser usado por sus hijos.
- **User y group quotas:** En versiones recientes de OpenZFS, es posible asignar quotas específicas a usuarios y grupos para controlar el uso de espacio a nivel granular.

Funcionamiento

Las quotas y reservas en ZFS se aplican de manera dinámica y son conscientes de otras propiedades como la compresión y la deduplicación. Esto significa que el espacio "real" consumido en disco puede ser menor que el tamaño lógico de los

archivos debido a la compresión, y las cuotas se calculan en función de ese espacio referenciado.

Casos de uso

- **Entornos multiusuario:** Limitar el espacio que cada usuario puede consumir, evitando que un solo usuario agote la capacidad total del pool.
- **División de servicios:** Separar el espacio para diferentes servicios (por ejemplo, bases de datos, almacenamiento de medios, directorios personales) para facilitar la administración y garantizar que cada área disponga de espacio suficiente.
- **Control de crecimiento:** Prevenir que datasets críticos crezcan sin control, manteniendo el sistema estable y evitando la sobreutilización del pool.

Resumen y Conclusiones

Las propiedades de compresión, deduplicación, encriptación y cuotas son herramientas fundamentales en ZFS que permiten:

- **Optimización del uso del almacenamiento:** La compresión y la deduplicación pueden reducir significativamente la cantidad de espacio físico utilizado.
- **Seguridad de los datos:** La encriptación protege la información en reposo, garantizando que sólo usuarios autorizados puedan acceder a ella.
- **Gestión y control:** Las cuotas y reservas permiten distribuir el espacio de manera justa y garantizar que cada dataset tenga los recursos necesarios.

Estas propiedades se pueden configurar de forma independiente en cada dataset, lo que ofrece una gran flexibilidad para adaptar el sistema de almacenamiento a las necesidades específicas de cada entorno, ya sea en un NAS doméstico, en servidores empresariales o en sistemas de virtualización.

4. Gestión y Administración

Comandos básicos: *zpool create, zpool status, zpool scrub, zfs create, zfs set.*

ZFS combina la administración del sistema de archivos y del gestor de volúmenes, lo que simplifica la gestión del almacenamiento. Su administración se realiza con dos comandos principales:

- ***zpool***: Gestiona los pools de almacenamiento.
- ***zfs***: Gestiona los sistemas de archivos dentro del pool.

1. Crear un pool de almacenamiento

Crear un pool llamado "mi_pool" en el disco /dev/dsk/:

```
zpool create mi_pool /dev/dsk/
```

Se pueden agregar varios discos para RAID-Z, por ejemplo crear un RAID-Z con 3 discos.:

```
zpool create mi_pool raidz /dev/dsk/X /dev/dsk/Y /dev/dsk/Z
```

2. Ver el estado del pool

Mostrar el estado de los pools activos, discos disponibles y posibles errores:

```
zpool status
```

3. Realizar una verificación de integridad (scrub)

Escanear el pool para detectar y corregir errores de datos:

```
zpool scrub mi_pool
```

Se recomienda ejecutar este comando periódicamente en servidores.

4. Crear un sistema de archivos dentro del pool

Crear un sistema de archivos llamado "mis_datos" dentro del pool "mi_pool".

```
zfs create mi_pool/mis_datos
```

Útil para organizar almacenamiento en múltiples carpetas gestionadas por ZFS.

5. Configurar propiedades en un sistema de archivos

Activar la compresión en el sistema de archivos "mis_datos":

```
zfs set compression=on mi_pool/mis_datos
```

Otras configuraciones útiles:

```
zfs set mountpoint=/mnt/datos mi_pool/mis_datos # Cambia el punto de montaje
```

```
zfs set atime=off mi_pool/mis_datos # Desactiva registro de accesos (mejora rendimiento)
```

Scrubbing y mantenimiento

Uno de los mayores beneficios de ZFS es su capacidad para detectar y corregir errores de datos automáticamente. Esto se logra mediante el scrubbing, un proceso de verificación que compara los datos con sus sumas de verificación y repara cualquier corrupción detectada.

¿Cómo funciona el Scrubbing en ZFS?

Lectura y Verificación

ZFS recorre todos los bloques de datos en un pool y verifica su integridad mediante checksums (sumas de verificación SHA-256).

Si un bloque está corrupto, intenta recuperarlo desde una copia redundante (si existe RAID-Z o un espejo).

Corrección de errores (si es posible)

Si ZFS encuentra errores en un disco, pero tiene una copia válida en otro disco (por ejemplo, en RAID-Z o espejo), reemplaza el bloque dañado automáticamente.

Si no hay copia disponible, marca el bloque como dañado para evitar su uso.

Registro de Problemas

ZFS guarda información sobre los errores detectados en el comando `zpool status`, permitiendo análisis posteriores.

RAID-Z vs RAID tradicional

RAID-Z es la implementación de ZFS del sistema RAID, pero con mejoras significativas sobre el RAID tradicional. A diferencia de éste, que suele depender de un controlador de hardware, RAID-Z se gestiona a nivel de software dentro de ZFS, lo que permite una mejor integración con sus características avanzadas, como Copy-on-Write (CoW) y la verificación de integridad con checksums.

Antes de tratar los 3 niveles distintos de RAID-Z es conveniente destacar las diferencias más importantes:

- **No usa un disco dedicado para la paridad:** En RAID-Z, a diferencia de RAID 5 o 6, la paridad se distribuye entre todos los discos, lo que evita cuellos de botella y mejora la resistencia a fallos.
- **No tiene el problema del “write-hole”:** El “write-hole” es un escenario de fallo relacionado con RAID tradicional producido a raíz de un corte de energía en medio de una escritura en el que los datos pueden quedar en un estado inconsistente. ZFS evita esto con Copy-on-Write, garantizando que los datos siempre sean válidos.
- **Mejor recuperación de errores** gracias a los checksums en cada bloque. En RAID tradicional, si un bloque se daña sin que el sistema lo detecte, la corrupción se propaga mientras que ZFS puede identificar errores y corregirlos de forma automática.

A continuación se explican los 3 tipos de RAID-Z distintos:

RAID-Z1: Protección contra la falla de un disco

RAID-Z1 es el equivalente a RAID 5. Se usa una única capa de paridad, es decir, puede tolerar la pérdida de un disco sin perder datos.

Ejemplo con 4 discos en RAID-Z1: Si cada disco tiene 1TB, el espacio utilizable será de 3TB, pues un disco se usa para la paridad. Si un disco falla, los datos pueden reconstruirse usando la paridad almacenada en los demás discos.

Para crearlo: `zpool create pool_raidz1 raidz1 /dev/dsk/X /dev/dsk/Y /dev/dsk/Z.`

Desventaja: Si un segundo disco falla antes de que el primero se haya reconstruido, se pierde toda la información del pool. Esto es aún más probable con discos de gran capacidad, en los que su reconstrucción puede tardar horas.

RAID-Z2: Protección contra la falla de dos discos

RAID-Z2 es el equivalente a RAID 6. Utiliza dos capas de paridad, es decir, puede tolerar hasta dos fallos de disco simultáneos.

Ejemplo con 6 discos en RAID-Z2: Si cada disco tiene 2TB, el espacio utilizable será de 8TB, ya que dos discos se destinan a paridad. Si un disco falla, el sistema sigue funcionando sin problemas. Si un segundo disco falla antes de la reconstrucción, los datos seguirán a salvo.

Para crearlo: `zpool create pool_raidz2 raidz2 /dev/dsk/X /dev/dsk/Y /dev/dsk/Z /dev/dsk/W`.

Ventajas: Mucho más seguro que RAID-Z1, permitiendo dos fallos de disco antes de que se pierdan datos. Esto lo hace ideal para discos grandes donde la reconstrucción lleva mucho tiempo.

Desventaja: Se sacrifica más espacio, ya que dos discos se usan exclusivamente para la paridad.

RAID-Z3: Protección contra la falla de tres discos

RAID-Z3 es un nivel aún más avanzado que usa tres capas de paridad, lo que permite que el sistema tolere hasta tres fallos de disco simultáneos.

Ejemplo con 8 discos en RAID-Z3: Si cada disco tiene 4TB, el espacio utilizable será de 20TB, ya que tres discos se destinan a paridad. Se pueden perder hasta tres discos sin afectar la disponibilidad de los datos.

Para crearlo: `zpool create pool_raidz3 raidz3 /dev/dsk/X /dev/dsk/Y /dev/dsk/Z /dev/dsk/W /dev/dsk/V`.

Ventajas: Máxima seguridad para datos críticos. Ideal para grandes sistemas de almacenamiento empresarial o backups de largo plazo.

Desventaja: Se sacrifica aún más espacio que con RAID-Z2. No es necesario en la mayoría de los casos, salvo en entornos con gran cantidad de discos.

Tras crear cualquiera de estos pools RAID-Z, es posible verificar su estado: *zpool status datos*.

Snapshots y backups eficientes con *zfs send* y *zfs receive*.

Uno de los mayores beneficios de ZFS es su capacidad para gestionar snapshots y realizar copias de seguridad de manera eficiente. Gracias a los comandos *zfs send* y *zfs receive*, es posible transferir datos de un sistema a otro sin copiar archivos manualmente ni realizar copias de seguridad completas cada vez, lo que facilita la replicación y el respaldo de información sin interrupciones ni consumo excesivo de recursos:

- *zfs send*: Extrae un snapshot y lo envía en formato binario.
- *zfs receive*: Recibe ese snapshot y lo almacena en otro sistema de archivos ZFS.

Así, se pueden transferir snapshots de manera eficiente entre servidores, discos o dispositivos externos.

Ejemplo de backup sencillo:

```
zfs send tank/datos@snapshot1 | ssh usuario@backupserver zfs receive backup/datos
```

En este comando, *zfs send* genera un flujo de datos con la información del snapshot, el flujo se envía a través de *ssh* a otro servidor y *zfs receive* recibe y almacena el snapshot en el sistema remoto.

Uno de los puntos fuertes de *zfs send* es que permite realizar backups incrementales. Es decir, en lugar de copiar todo el sistema de archivos cada vez, solo se transfieren los cambios desde el último snapshot.

Ejemplo de backup incremental:

```
zfs send -i tank/datos@snapshot1 tank/datos@snapshot2 | ssh usuario@backupserver zfs receive backup/datos
```

La opción *-i snapshot1* indica que solo se envíen los cambios ocurridos desde snapshot1 hasta snapshot2, reduciendo drásticamente el tiempo y el espacio necesario para los backups.

De esta forma, con estos comandos conseguimos una gran eficiencia:

- **No hay deduplicación innecesaria de datos:** Los snapshots solo almacenan cambios, por lo que *zfs send* y *zfs receive* trabajan con datos mínimos.

- **Es mucho más rápido que copiar archivos manualmente:** Al trabajar a nivel de bloques, evita la sobrecarga de recorrer estructuras de directorios y archivos.
- **Permite la replicación en vivo:** Se pueden programar backups frecuentes sin afectar el rendimiento del sistema.
- **Evita la corrupción de datos:** Gracias a los checksums y la verificación de integridad de ZFS, los backups son confiables y se detectan errores antes de que se propaguen.

5. Casos de uso y desventajas

Casos de uso recomendados de ZFS.

1.- Servidores de almacenamiento y NAS (TrueNAS, Proxmox).

ZFS es ampliamente utilizado en servidores de almacenamiento y soluciones NAS (Network Attached Storage) debido a su fiabilidad, integridad de datos y facilidad de gestión.

Son varios los motivos que convierten a ZFS en una gran aliado para estos servidores, como su autocorrección de datos con checksums, sus snapshots y clones rápidos sin interrupción del servicio, su optimización del espacio en disco y su expansión sencilla y dinámica mediante pools.

Algunos ejemplos de uso son:

- **TrueNAS:** Antes llamado FreeNAS. Se trata de un sistema operativo open-source basado en BSD que emplea ZFS para ofrecer almacenamiento confiable con una interfaz de fácil instalación y administración.
- **Proxmox VE:** Consiste en un entorno de virtualización open-source basado en Debian que usa ZFS como backend para almacenamiento de máquinas virtuales y contenedores, permitiendo snapshots y replicación en entornos empresariales.

2.- Copias de seguridad y archivo de datos.

ZFS es ideal para backups y archivado gracias, principalmente, a su diseño Copy-on-Write (CoW), pero también a su eficiente y rápido manejo de snapshots, detección y corrección de corrupción mediante checksums, etc.

Tras crear un snapshot con el comando antes mencionado `zfs snapshot pool/dataset@backup1`, es posible enviarlo a otro servidor con:

```
zfs send pool/dataset@backup1 | ssh backupserver zfs receive backup/dataset
```

De esta manera, se pueden replicar datos de manera eficiente sin copiar archivos individualmente.

Así, ZFS es el sistema de archivos indicado para empresas que necesiten backups rápidos y confiables sin provocar interrupciones en el sistema, CPDs con retención de datos a largo plazo, o incluso usuarios domésticos que quieran hacer copias en discos o servidores.

3.- Sistemas empresariales y bases de datos.

Además de entornos empresariales, ZFS es especialmente empleado en bases de datos y sistemas críticos por su alto rendimiento, determinado por su eficiencia; y su gran protección de datos, gracias a sus snapshots, clones, CoW y mecanismos como RAID-Z que protegen contra fallos de discos sin necesidad de hardware RAID.

Como se ha mencionado, es común su uso en bases de datos como PostgreSQL, MySQL, entre otras. Esto se debe a que la base de datos puede ser configurada en un dataset ZFS con compresión para mejorar el rendimiento.

Otro ejemplo de uso en este aspecto son, de nuevo, las pruebas de desarrollo, pues es también posible la creación de snapshots antes de actualizar datos y revertir cambios, operaciones muy comunes y habituales en bases de datos que, si mal realizadas, conducen en muchas ocasiones a estados irreversibles.

En general, cualquier base de datos que requiera alta disponibilidad y seguridad, cualquier sistema de registro donde no se puedan perder ni corromper datos o cualquier CPD con necesidad de flexibilidad y eficiencia en almacenamiento, son situaciones ideales para el empleo de ZFS.

Desventajas de ZFS

A pesar de sus numerosas ventajas, ZFS no es una solución perfecta y presenta algunas desventajas y limitaciones que pueden hacer que no sea la mejor opción en ciertos escenarios.

1.- Alto consumo de RAM (1GB de RAM por TB de almacenamiento).

ZFS está diseñado para ofrecer alta integridad y rendimiento, pero a costa de un uso intensivo de memoria RAM. Este alto consumo tiene varios motivos que lo explican (y que imposibilitan que sea de otra manera).

En primer lugar, y una de las principales causas de este consumo, se encuentra el uso que hace ZFS de un sistema de caché avanzado llamado ARC (Adaptive Replacement Cache). ARC implementa un algoritmo adaptativo que diferencia entre datos frecuentemente accedidos y datos recientemente accedidos, y, a pesar de que esta caché mejora notablemente el rendimiento gracias a evitar accesos innecesarios al disco y reducir la latencia en lecturas repetitivas, también es la culpable de ocupar hasta el 50% de la memoria RAM total.

Sin embargo, es posible limitar el tamaño de ARC reduciendo el parámetro `zfs_arc_max`, muy recomendable en sistemas con poca RAM.

Por otro lado, los checksums y metadatos que ZFS emplea para conseguir la integridad de datos también perjudican al consumo de memoria RAM. La explicación es lógica, y es que cada operación de lectura/escritura implica cálculos adicionales para validar la integridad de los datos. Además, los metadatos de ZFS (snapshots, registros de transacciones...) ocupan espacio en la memoria RAM. En discos grandes con millones de archivos, la gestión de estos metadatos pueden consumir una cantidad considerable de memoria.

Ante esto, en sistemas con poca RAM, se recomienda desactivar la deduplicación (se tratará a continuación) y reducir el uso de snapshots.

Por último, y relacionado con el motivo anterior, se debe mencionar que, en caso de activarla, la deduplicación puede requerir de gran espacio.

La deduplicación permite que ZFS almacene bloques de datos idénticos una sola vez, lo que ahorra espacio en disco. Sin embargo, esto perjudica el espacio en RAM, pues cada bloque necesita almacenar una huella digital (hash) para verificar si ya existe en el sistema, lo que puede disparar el consumo de RAM hasta 5GB por cada 1TB de datos.

La solución es sencilla, no activar la deduplicación a menos que realmente se necesite y se disponga de la memoria suficiente.

Por todo esto, ZFS puede ser una mala elección en sistemas con menos de 4GB de RAM sin ajustes adecuados en los que será más problemático que beneficioso.

2.- Problemas de compatibilidad con Linux (licencias CDDL vs GPL).

ZFS fue desarrollado originalmente por Sun Microsystems y lanzado bajo la licencia CDDL (Common Development and Distribution License). Esta licencia permite modificar y distribuir código, pero exige que cualquier modificación en archivos bajo CDDL se publique también bajo CDDL.

Sin embargo, Linux usa la licencia GPL (General Public License), lo que ha generado conflictos legales. De igual manera, GPL requiere que todo el código que se vincule con el kernel de Linux sea también GPL o compatible con este.

A pesar de que ambas licencias buscan garantizar la libertad del software, imponen restricciones sobre cómo se pueden mezclar con otras licencias.

Por tanto, si un desarrollador incluyera directamente ZFS en el kernel de Linux, tendría que elegir entre violar la GPL al distribuir el kernel con código bajo CDDL, o violar la CDDL al redistribuir ZFS bajo la GPL.

Como conclusión: no es posible fusionarlas legalmente en el mismo proyecto.

La única forma de poder usar ZFS en Linux sin infringir la GPL es usándolo como módulo externo. Es decir, compilar ZFS como un módulo del kernel, pero no se distribuye junto con él. El usuario lo instalaría a parte con `zfs-dkms`, en el caso de Debian/Ubuntu.

De esta manera se considera legal porque el usuario lo carga después, no como parte del kernel oficial.

Aún así, queda una pregunta por responder: ¿por qué FreeBSD no tiene este problema?

FreeBSD y otros sistemas BSD pueden incluir ZFS sin problemas porque su licencia BSD, es mucho más permisiva y compatible con la CDDL. Es por esto que ZFS está integrado directamente en el sistema operativo sin necesidad de módulos externos.

6. Conclusión y Preguntas

ZFS es, sin duda, uno de los sistemas de archivos más avanzados que existen. Su enfoque en la integridad de los datos, la eficiencia en la gestión del almacenamiento y la flexibilidad en la administración lo convierten en una opción ideal para servidores, NAS y sistemas empresariales donde la fiabilidad es una prioridad. Su arquitectura basada en pools de almacenamiento en lugar de volúmenes tradicionales permite una mejor gestión de los discos y facilita la expansión del almacenamiento sin interrupciones.

Uno de sus mayores puntos fuertes es el uso de Copy-on-Write (CoW), que evita la corrupción de datos al escribir siempre en un nuevo bloque en lugar de sobrescribir los datos originales. Gracias a esto, los snapshots y clones son increíblemente eficientes, permitiendo mantener versiones históricas de los datos sin apenas impacto en el rendimiento ni en el espacio de almacenamiento. Además, la combinación de checksums en cada bloque y la capacidad de detectar y corregir errores garantiza que los datos almacenados sean siempre fiables, algo esencial en sistemas críticos.

En cuanto a los backups, ZFS ofrece una solución integrada con `zfs send` y `zfs receive`, que permite replicar datos entre sistemas de manera eficiente, sin necesidad de herramientas externas. Esto, sumado a su compatibilidad con RAID-Z en distintos niveles (RAID-Z1, RAID-Z2 y RAID-Z3), lo hace ideal para entornos donde la seguridad y redundancia de datos son fundamentales.

Sin embargo, ZFS no es perfecto. Su principal desventaja es el alto consumo de RAM, ya que su caché avanzada ARC necesita grandes cantidades de memoria para funcionar de manera óptima. Esto hace que no sea recomendable en equipos con pocos recursos. Además, su historial de problemas con Linux debido a la incompatibilidad de licencias entre CDDL y GPL ha limitado su adopción en algunas distribuciones, aunque proyectos como OpenZFS han ayudado a mitigar este problema.

Entonces, si es tan potente, ¿por qué no está en todas partes? La respuesta es que no todos los sistemas necesitan las características avanzadas de ZFS. Para usuarios domésticos o servidores simples, sistemas como ext4 o XFS pueden ser más que suficientes sin el sobrecoste de memoria y complejidad. Pero en entornos donde la integridad y seguridad de los datos son prioritarias, como servidores de almacenamiento, bases de datos y backups, ZFS

no solo es una buena opción, sino que puede marcar la diferencia entre perder o conservar datos valiosos.

En definitiva, ZFS no es solo un sistema de archivos: es una solución completa para el almacenamiento moderno. Aunque tiene requisitos elevados, las ventajas que ofrece justifican su uso en entornos donde la fiabilidad es clave. No es la mejor opción para todos, pero donde encaja, lo hace como ningún otro.

7. Bibliografía

<https://www.open-e.com/blog/copy-on-write-snapshots>

<https://docs.oracle.com/en/operating-systems/solaris/oracle-solaris/11.4/manage-zfs/raid-z-storage-pool-configuration.html>

<https://www.raidz-calculator.com/raidz-types-reference.aspx>

https://docs.oracle.com/cd/E18752_01/html/819-5461/gbchx.html

<https://docs.oracle.com/cd/E19253-01/819-5461/gbinw/index.html>

<https://openzfs.github.io/openzfs-docs/Basic%20Concepts/Checksums.html>

<https://docs.freebsd.org/en/books/handbook/zfs/>

[https://es.wikipedia.org/wiki/ZFS_\(sistema_de_archivos\)](https://es.wikipedia.org/wiki/ZFS_(sistema_de_archivos))

<https://www.redeszone.net/tutoriales/servidores/sistema-archivos-zfs-servidores/>

<https://arstechnica.com/information-technology/2020/05/zfs-101-understanding-zfs-storage-and-performance/>

<https://klarasystems.com/articles/openzfs-understanding-zfs-vdev-types/>

<https://openai.com/es-ES/index/chatgpt/>