

Sistema de Arquivo ZSF



Introdução ZFS

- Sistema de arquivos avançado com gerenciamento de volumes integrado
- Projetado para alta confiabilidade, segurança e escalabilidade
- Usado em servidores, datacenters e sistemas críticos
- Capaz de gerenciar até 256 quatrilhões de zettabytes
 - 1 zettabyte = 1 bilhão de TB
- Garante integridade dos dados com verificação automática de erros



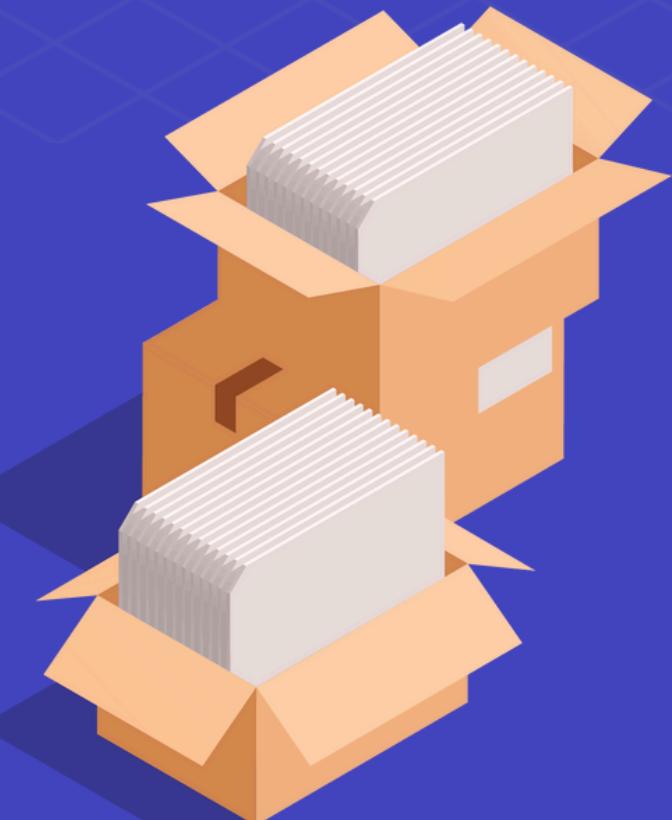
História e Objetivos do ZFS

História:

- Criado pela Sun Microsystems em 2005
- Oracle tornou-se proprietária
- Versão aberta continua como OpenZFS

Objetivos:

- Evitar perda de dados
- Garantir integridade com verificação automática
- Facilitar a administração de armazenamento



O que Torna o ZFS Diferente?

- **Uma Plataforma Unificada:** O ZFS não é apenas um sistema de arquivos. Ele combina as funções de sistema de arquivos e gerenciador de volumes lógicos.
- **Controle Total:** Essa abordagem integrada confere ao ZFS controle completo sobre o layout dos dados nos discos físicos.
- **Pilares da Arquitetura:** Para entender seu poder, vamos explorar três componentes essenciais:
 - Pool de Armazenamento (ZPOOL)
 - Modelo Copy-on-Write (CoW)
 - Estrutura de Dados (Datasets e Metadados)

O Coração do ZFS - Pool de Armazenamento (ZPOOL)

- **A Base de Tudo:** O zpool é a base de qualquer sistema ZFS. Todos os seus discos (HDs, SSDs) são *fundidos* em um único e gigante pool de armazenamento.
- **Analogia do Tanque de Água:**

Pense no zpool como um grande tanque de água.
Você adiciona água de várias fontes (discos) e o **sistema apresenta um volume total disponível.**



- Organização com VDEVs: Dentro do pool, os discos são organizados em VDEVs (Virtual Devices), que definem o nível de redundância. Os tipos comuns de VDEVs incluem:
 - Stripe (**RAID 0**): Os dados são distribuídos entre os discos sem paridade. Oferece máximo desempenho e capacidade, mas nenhuma redundância. A falha de um único disco resulta na perda de todo o pool.
 - Mirror (**RAID 1**): Os dados são espelhados em dois ou mais discos. Cada disco no VDEV é uma cópia exata do outro. Oferece alta redundância e excelente desempenho de leitura.
 - RAIDZ (**RAID 5/6 Aprimorado**): Distribui dados e paridade de forma mais segura, evitando o problema do "write hole".



- RAIDZ: Uma implementação mais robusta que o RAID 5/6 tradicional.
 - RAIDZ-1 (similar ao RAID 5): Requer no mínimo 3 discos e permite a falha de um disco.
 - RAIDZ-2 (similar ao RAID 6): Requer no mínimo 4 discos e permite a falha de até dois discos.
 - RAIDZ-3: Requer no mínimo 5 discos e permite a falha de até três discos.

O RAIDZ soluciona um problema inerente ao RAID 5 chamado "write hole", onde uma queda de energia durante uma escrita pode deixar a paridade e os dados em um estado inconsistente. Graças ao mecanismo de copy-on-write, isso não ocorre no ZFS.

A Garantia de Consistência - Modelo Copy-on-Write (CoW)

- **ZFS Nunca Sobrescreve Dados:** Diferente de sistemas tradicionais, o ZFS nunca sobrescreve dados ativos no mesmo local.

Como Funciona:

- Quando um dado é modificado, o ZFS escreve *a nova versão em uma área livre do disco*.
- Somente após a confirmação da escrita, os metadados ("ponteiros") são atualizados para apontar para o novo local.

Benefício Principal: Consistência Absoluta.

- O sistema de arquivos está sempre em um estado consistente.
- Quedas de energia não corrompem dados, pois a transação ou é concluída ou descartada.

Organização Inteligente - Estrutura de Dados

Datasets: Partições Flexíveis

Dentro de um zpool, você cria Datasets, que são sistemas de arquivos individuais (como /home ou /documentos).

Eles compartilham o espaço do pool, mas cada um pode ter suas próprias regras:

- Ativar compressão para documentos.
- Definir cotas para usuários.
- Ajustar o tamanho do bloco para bancos de dados.

Metadados Transacionais: A Confirmação Final

1. Cada operação de escrita no ZFS é tratada como uma transação.
2. A atualização dos metadados (ponteiros) é uma operação atômica que só ocorre após a escrita dos dados ser confirmada.
3. Isso garante que o estado do sistema de arquivos nunca fique inconsistente.

Resumo da Arquitetura

- ZPOOL: Unifica discos físicos em um único pool de armazenamento flexível e redundante.
- Copy-on-Write (CoW): Garante a consistência dos dados ao nunca sobrescrever informações, tornando o sistema imune a escritas parciais.
- Estrutura de Dados (Datasets): Permite o gerenciamento granular e a aplicação de políticas específicas para diferentes tipos de dados.
- Resultado: A combinação desses componentes resulta em uma plataforma de armazenamento com uma integridade de dados, consistência e flexibilidade de gerenciamento inigualáveis.

Mecanismos de Proteção — Integridade dos Dados

Integridade dos Dados:

- Checksums:
 - Cada bloco de dados tem um código de verificação (checksum) armazenado separadamente. Isso permite detectar erros silenciosos.
 - Auto-cura (Self-healing):
 - Se um erro for detectado e o ZFS tiver cópias redundantes, ele corrige automaticamente os dados com base em cópias válidas
- RAID-Z:
 - Variação do RAID nativa do ZFS (sem necessidade de hardware)
 - Evita o problema do write hole
 - Suporte a paridade, espelhamento e tolerância a falhas (RAID-Z1, Z2, Z3)

Otimizações de Armazenamento

Deduplicação:

- Elimina blocos duplicados de dados automaticamente
- Economiza espaço, especialmente em backups e arquivos repetidos

Compressão Transparente:

- Compacta os dados automaticamente ao serem gravados
- A compressão é feita de forma transparente para o usuário
- Reduz uso de disco e pode aumentar o desempenho (menos I/O)

Casos de Uso Ideais:

- Data Centers: Armazenamento confiável para grandes volumes.
- NAS/SAN: Sistemas de backup e arquivamento.
- Ambientes Virtualizados: Discos virtuais com snapshots eficientes.



Vantagens e Desafios

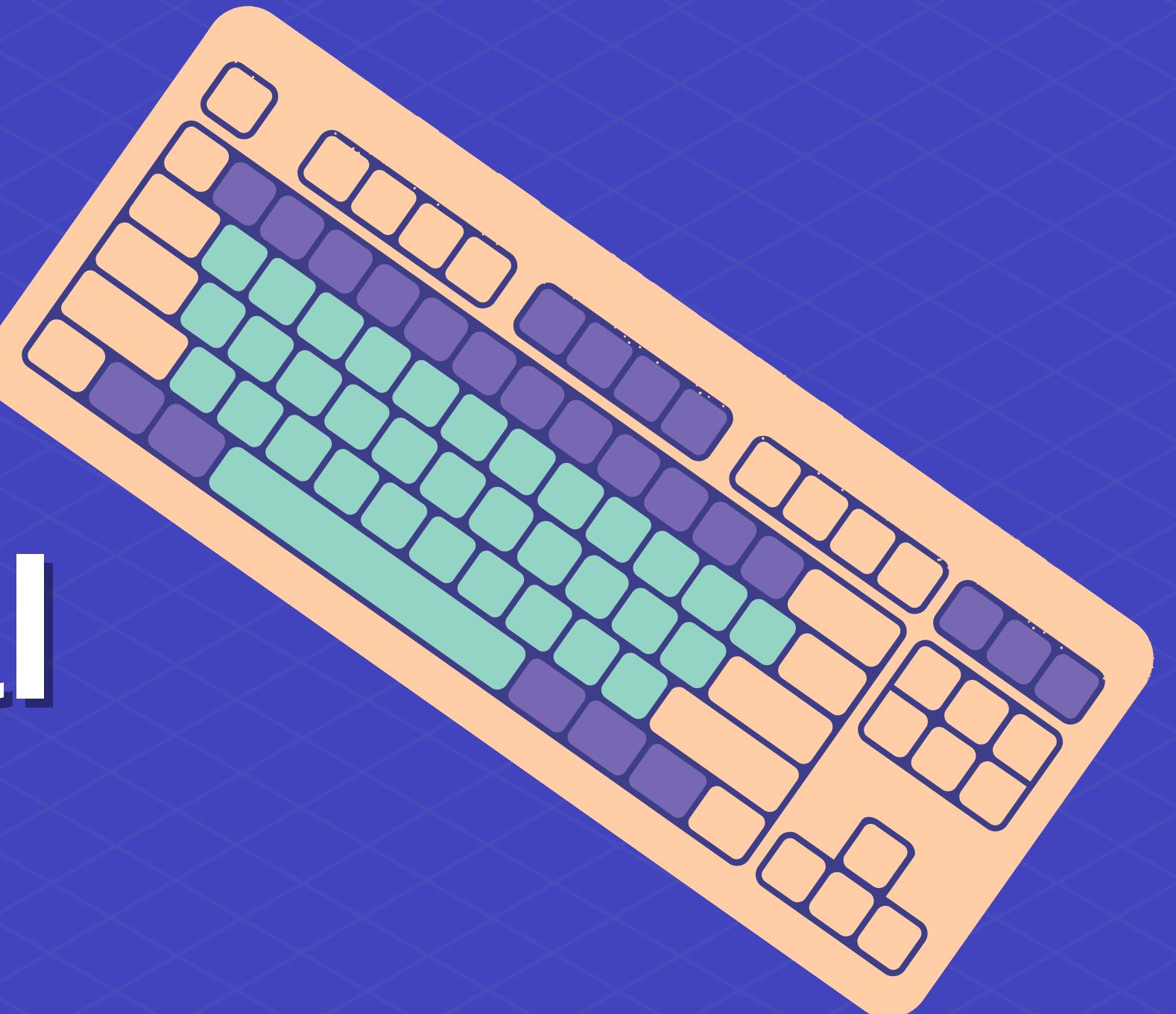
- **VANTAGENS**

- Escalabilidade extrema: Número praticamente ilimitado de arquivos/diretórios.
- Snapshots e Clones: Cópias instantâneas de baixo custo para backup/recuperação.
- QoS via "Reservations" e "Quotas": Alocação garantida de recursos.

- **DESAFIOS**

- Recursos de Hardware: ARC consome RAM significativa
- Complexidade: Configuração de ZPOOL, RAID-Z e datasets requer expertise.
- Licenciamento: CDDL (OpenZFS) pode ter implicações em integração com o kernel Linux.

Projeto: Teclado Virtual



Estrutura

- Front-end
 - Python
 - Gera a GUI
 - É usado pelo back-end para interagir com o usuário.
- Back-end
 - C
 - Gera os eventos a serem processados pelo Kernel Linux por meio da API uinput

Utilidade

- Teclado Virtual.
- Softwares de acesso remoto.
 - TeamView
 - AnyDesk
 - Quick Assist
- Automação da interface de usuário.
 - PyAutoGUI

Código

Abre o descriptor de arquivo

```
int fd = open("/dev/uinput", O_WRONLY | O_NONBLOCK);
```

Define as teclas usadas

```
ioctl(fd, UI_SET_EVBIT, EV_KEY);
ioctl(fd, UI_SET_KEYBIT, KEY_SPACE); // Para a tecla espaço
ioctl(fd, UI_SET_KEYBIT, KEY_A);    // Para a tecla A
```

Configura o dispositivo virtual

```
memset(&usetup, 0, sizeof(usetup));
usetup.id.bustype = BUS_USB;
usetup.id.vendor = 0x1234; /* sample vendor */
usetup.id.product = 0x5678; /* sample product */
strcpy(usetup.name, "Teclado Virtual");
```

Cria o dispositivo virtual

```
ioctl(fd, UI_DEV_SETUP, &usetup);
ioctl(fd, UI_DEV_CREATE);
```

Código

Inicia a GUI e abre um pipe

```
FILE *fp = popen("python3 window.py", "r");
char buffer[128];
```

Inicia o loop de leitura

```
while (fgets(buffer, sizeof(buffer), fp) != NULL) {
    click(fd, buffer[0]);
}
```

Código

GUI desenha os botões e define os valore
ASCII

```
root = tk.Tk()
root.overrideredirect(True)
root.title("Teclado Virtual Linux")

for i in range(2):
    for j in range(13):
        num = 13 * i + j
        key = chr(ord('A') + num)
        button = tk.Button(root, text=f"{key}", command=lambda x=key: keyPress(x), width=5)
        button.grid(row=i, column=j)

tk.Button(root, text="SAIR", command=quit, width=10).grid(row=2, column=13)
tk.Button(root, text="ENTER", command=lambda: keyPress("1"), width=10).grid(row=1, column=13)
tk.Button(root, text="ESPAÇO", command=lambda: keyPress("0")).grid(row=2, column=0, columnspan=13, sticky="nsew")
tk.Button(root, text="APAGAR", command=lambda: keyPress("2"), width=10).grid(row=0, column=13)
```