Contents

disko-zfs: Declaratively Managing ZFS Datasets

Published on Jan 23, 2026

The work on disko-zfs was carried out under contract with Numtide. Thank you Numtide for making this project a reality!

If you're at least somewhat like me, then you use ZFS almost religiously. Every server, every laptop, every appliance (excluding those that really could not run ZFS due to memory constraints or fragile flash storage) runs ZFS. You run ZFS even if you don't use mirrors, stripes, or any kind of special vdevs. You run ZFS because you want to, because having snapshots, datasets and compression makes the device feel familiar and welcoming. If that's at least somewhat you, you will appreciate the tool I have for you in store today.

Too Many Datasets

Given a situation where a ZFS pool has just too many datasets for you to comfortably manage, or perhaps you have a few datasets, but you just learned of a property that you really *should* have set from the start, what do you do? Well, I don't know what *you* do, I would love to hear about that, so please do reach out to me, over Matrix preferably.

In any case, what I came up with is disko-zfs. A simple Rust program that will declaratively manage datasets on a zpool. It does this based on a JSON specification, which lists the datasets, their properties and a few pieces of extra information.

▶ disko-zfs is safe

The Schema

Listing 1: A production specification from one of Numtide's machines

```
{
 "datasets": {
  "zroot": {
   "properties": {
    "atime": "off",
    "com.sun:auto-snapshot": "false",
    "compression": "zstd-2",
    "dnodesize": "auto",
    "mountpoint": "none",
    "recordsize": "128K",
    "xattr": "on"
   }
  },
```

```
    ...
    "zroot/ds1/persist/var/lib/forgejo": {
      "properties": {
        "mountpoint": "legacy"
      }
    },
    "zroot/ds1/persist/var/lib/postgresql": {
      "properties": {
        "mountpoint": "legacy",
        "recordsize": "8k"
      }
    },
    ...
  },
  "ignoredDatasets": [
    "zroot/ds1/root/*"
  ],
  "ignoredProperties": [
    "nixos:shutdown-time",
    ":generation",
    "com.sun:auto-snapshot"
  ],
  "logLevel": "info"
}
```

As you can see, it's relatively self explanatory. The information that this JSON format carries is:

1. an attribute set of datasets and their properties
2. a list of ignored datasets that `disko-zfs` will never create, modify, or destroy
3. a list of ignored properties that `disko-zfs` will never create, modify, or delete
4. the loglevel at which `disko-zfs` logs

With the schema in hand, we can execute `disko-zfs apply --file spec.json` on a live machine, where we get the following output:

```
# !! Destructive Commands !!
> zfs destroy zroot/ds1/persist/var/lib/gitea
# Applying...
+ zfs set mountpoint=legacy zroot/ds1/persist/var/lib/forgejo
+ zfs create -orecordsize=8k -omountpoint=legacy zroot/ds1/persist/var/lib/forgejo
# Done!
```

It tells us a few things:

1. that `zfs destroy zroot/ds1/persist/var/lib/gitea` would be executed, but it wasn't
2. that the following two commands have been executed:
   - `zfs set mountpoint=legacy zroot/ds1/persist/var/lib/forgejo`
   - `zfs create -orecordsize=8k -omountpoint=legacy zroot/ds1/persist/var/lib/forgejo`

If we run `disko-zfs apply --file spec.json` again on the same machine, `disko-zfs` will correctly report that there is nothing to do.

Instead of directly applying the configuration we could have used the `plan` subcommand to tell `disko-zfs` just tell us what it would do, if we executed `apply`, even for non-destructive commands. This is a good way of verifying that `disko-zfs` won't do anything unexpected.

NixOS Integration

If you also happen to be a NixOS user, then this section will be of interest to you. `disko-zfs` supports NixOs natively; it exposes a NixOS module, which adds the `disko.zfs` option group. It importantly adds the `disko.zfs.settings` option, whose structure is isomorphic to the structure used by the `disko-zfs` program itself; as such we can directly translate the previously shown JSON to Nix and everything will work as expected, neat!

```
{
  disko.zfs = {
    enable = true;
    settings = {
      datasets = {
        "zroot" = {
          properties = {
            atime = "off";
            "com.sun:auto-snapshot" = "false";
            compression = "zstd-2";
            dnodesize = "auto";
            mountpoint = "none";
            recordsize = "128K";
            xattr = "on";
          };
        };
        ...
        "zroot/ds1/persist/var/lib/forgejo" = {
          properties = {
            mountpoint = "legacy";
          };
        };
        "zroot/ds1/persist/var/lib/postgresql" = {
          properties = {
            mountpoint = "legacy";
            recordsize = "8k";
          };
        };
        ...
      };
      ignoredDatasets = [
        "zroot/ds1/root/*"
      ];
      ignoredProperties = [
        "nixos:shutdown-time"
        ":generation"
        "com.sun:auto-snapshot"
      ];
      logLevel = "info";
    };
  };
}
```

Given the above NixOS configuration, next time you run `nixos-rebuild switch --flake .#your-server`, `disko-zfs` will make any changes necessary to bring your zpool into shape. You can also execute `nixos-rebuild dry-activate --flake .#your-server` instead, which will cause `disko-zfs` to run in `plan` mode and merely print out the changes it would make, nifty way to test your changes.

Disko Integration

And if you use ZFS, NixOS, **and** `disko` then I have another thing in store for you. `disko-zfs` integrates with, well, `disko`, who would have expected that?

If the `disko-zfs` module detects that `disko` is also imported, it will translate any zpool and datasets declared through disko into the format used by `disko-zfs` and automatically apply them to the `disko.zfs.settings.datasets` option. That means that you can still declare you datasets through `disko` and let `disko` do the thing it's great at, installation. Once however you need to create more datasets, or perhaps delete a dataset, that's where `disko-zfs` kicks in and realizes your wishes. With `disko-zfs`, `disko` gains the power to keep your datasets up to date even after installation.

Built with Emacs 30.2, OrgMode 9.7.11, `org-thtml 0.3`
Source code here