ZFSBootMenu

Release 2.1.0

ZFSBootMenu Team

2023-02-07

CONTENTS

CHAPTER

ONE

ZFSBOOTMENU V2.1.0 (2022-12-19)

1.1 Deprecated features

- `syslinux` support as a core part of `generate-zbm` will be removed in the next release. contrib/syslinux-update.sh should be used to create `syslinux.cfg` moving forward. Refer to the script for usage documentation.
- Awareness of platform endianness when writing `/etc/hostid` was rendered moot when support for `skim` was removed because `fzf` is only supported on little-endian systems. All hostid writes assume little-endian byte order by default.

1.2 New features

- The command line option `zbm.prefer` has been extended with a `!!` marker to import exactly one pool when multiple are available on a system. Refer to docs/man/zfsbootmenu.7.rst for more details.
- Users of the binary releases are now able to use their own custom hooks with the new `zbm.hookroot` command line option. Using this, a partition and directory specification can be provided which allows for additional scripts to be loaded at runtime. Refer to docs/man/zfsbootmenu.7.rst for more details.
- contrib/remote-ssh-build.sh has been provided by a new contributor. This script helps ease the creation of a custom EFI file with an embedded SSH server and keys.
- A global application header has been added to highlight the pages that are available and the currently selected page.
- Unless explicitly configured, `generate-zbm` will now default to `dracut` but fall back to `mkinitcpio` when it cannot find `dracut` in the path.
- The build container `ghcr.io/zbm-dev/zbm-builder` has been substantially improved, making it easier to manage custom images built in a controlled, compatible environment using podman or docker.

1.3 Fixes

- On some systems, Dracut was incorrectly using `dash` where `bash` should be used. Forbidding the inclusion of `dash` with Dracut resolves this issue.
- The `drm` Dracut module has been blacklisted. ZFSBootMenu should never attempt to load firmware for video cards.
- When using ZFSBootMenu to pin a kernel, ensure that an anchor is attached to the end of the pin. This resolves incorrect matches on systems that have unversioned kernels.

- To avoid a potential conflict between ZFS pool names and files/directories created by ZFSBootMenu, all detected boot environment mountpoints have been moved to a dedicated `environments/` sub-directory.

1.4 Significant commits in this release

- 8e6ca4a - set_default_kernel: properly clear default when no kernel is specified (Andrew J. Hesford)
- 0b40f44 - interface: enable left/right arrow key navigation (Zach Dykstra)
- 235eb17 - Stop installing zpool.cache (Andrew J. Hesford)
- 5427883 - Improve containerized builds (Andrew J. Hesford)
- c023517 - Automatically select initramfs generator when not forced (Andrew J. Hesford)
- 6223804 - Update artifact uploader to latest version (Zach Dykstra)
- 00ef434 - Add/use border label feature flag (Zach Dykstra)
- 562b14a - Fix booting test VMs on some hardware (Zach Dykstra)
- 8d3bbff - Fix kernel selection after introducing $BASE/environments (Alexander Lobakin)
- b743893 - contrib/README.md: document contrib scripts (Andrew J. Hesford)
- a3f15ed - generate-zbm: deprecate syslinux support (Andrew J. Hesford)
- 9f85003 - Include the OpenSSH client in recovery images (Zach Dykstra)
- 91900d3 - Explicitly require bash, blacklist dash (Andrew J. Hesford)
- 3399f4b - Move dataset mountpoint to $BASE/environments/ (Zach Dykstra)
- 7eb7780 - Allow imports of hooks from external sources at runtime (Andrew J. Hesford)
- 57b46a2 - Extend capabilities of zbm.prefer (Zach Dykstra)
- af1c74b - Remove diff from early days of playing around (Zach Dykstra)
- 3da5547 - Center key bind text on horizontal layout fzf screens (Zach Dykstra)
- df8ab05 - Support busybox as /bin/sh in chroot (Zach Dykstra)
- 2fa207e - fix: ensure $uefi_stub is defined before checking if it is a file (Zach Dykstra)
- 906ce3b - Add anchor to end of string when pinning a kernel (Zach Dykstra)
- 9c91afa - Make default efi path distro agnostic (gardar)
- 333b6d8 - Wrapper Build Script for ZBM with SSH access (Gerhard Roethlin)
- 05dbf11 - Show /etc/zbm-commit-hash in zreport (Zach Dykstra)
- d4b35a0 - zbm-build.sh: don't upgrade packages when installing custom software (Andrew J. Hesford)
- 10e3624 - Exploit common configurations for recovery/release images (Andrew J. Hesford)
- 1be8ed0 - Drop drm module from standard dracut config (Andrew J. Hesford)
- 9b57d8a - testing: move Ubuntu to 22.04 LTS, make column available to Debian/Ubuntu (Andrew J. Hesford)
- feb6a26 - documentation: provide a link to the wiki (Zach Dykstra)
- f2d3336 - Add a contrib script to snapshot the BE prior to boot (Zach Dykstra)

CHAPTER

TWO

ZFSBOOTMENU V2.0.0 (2022-06-28)

ZFSBootMenu 2.0.0 introduces a major internal reorganization that allows images to be built with initramfs generators other than dracut and includes some helpful command-line utilities. This release is based on Linux 5.10.125 and ZFS 2.1.5.

2.1 New features

- Dracut is now optional; ZFSBootMenu images may currently be built with mkinitcpio
- Generalizations to support mkinitcpio should also apply to, *e.g.*, initramfs-tools, although installation hooks for initramfs-tools have not (yet) been written
- A new utility, `zbm-efi-kcl`, provides the ability to edit the kernel command-line embedded in a ZBM EFI bundle rather than requiring regeneration of the bundle
- A new utility, `zbm-builder.sh`, provides a simple interface for creating custom, local images using the official ZBM build container; it is now possible to build local images without installing ZBM or its Perl dependencies

2.2 Fixes

- Video drivers are now omitted from images by default to avoid GPU initialization issues in the final boot environment
- General fixes to shell functions were made in support of using busybox in mkinitcpio images
- The online command-line editor now provides an option to revert to the default

2.3 Significant commits in this release

- 8933a57 - Basic EFI KCL editor (Zach Dykstra)
- ada36c8 - Include cryptsetup in recovery images (Zach Dykstra)
- f1b0270 - Include cryptsetup in containarized builds (Grzegorz Uriasz)
- fd52d7c - Omit video drivers by default (Zach Dykstra)
- 65dd0cd - Add option to revert to default KCL via ctrl-t in draw_be (Zach Dykstra)
- 632b24f - Add special escape sequences to PS1 (Zach Dykstra)
- 3d9f57b - Build recovery images in CI/CD (Zach Dykstra)

- 9368171 - Accept any root= value as valid (Zach Dykstra)
- 7982971 - zbm-builder.sh: support custom build/source paths and hooks (James R. Todd)
- fc83894 - Document scrolling keys in help viewer (Zach Dykstra)
- 6f09d39 - Fix broken links (Jip de Beer)
- 64936f3 - Add minimal documentation to ESP sync hook (Zach Dykstra)
- d559458 - Add BUILD.md, a quick-start containerized build guide (James Todd)
- 1aca5d1 - Bug fixes and use local repository option for zbm-builder.sh (James Todd)
- 8b1aade - Make the emergency shell slightly fancy (Zach Dykstra)
- 5a5ab0b - Write data to console to recalculate size (Zach Dykstra)
- eac0547 - install-helpers.sh: explicitly require setsid (Andrew J. Hesford)
- dbd9642 - Replace 'tr' with bash string manipulation (Zach Dykstra)
- 8916a10 - Update luks-unlock.sh to work with current libraries (Zach Dykstra)
- 33883e5 - zfsbootmenu-core.sh: drop preload_be_cmdline, improve KCL caching (Andrew J. Hesford)
- 7548af5 - zbm-kcl: remove dracut from ZBM_MODULEDIR warning (Andrew J. Hesford)
- 306f36b - Fix install location for common module files (Zach Dykstra)
- 644b0f1 - Use an early-setup hook to force console init in release images (Andrew J. Hesford)
- ec89a7c - Use column to layout footer (Zach Dykstra)
- 2bd6757 - generate-zbm: remove INI migration support (Zach Dykstra)
- 65167e6 - Define all ZBM requirements in install-helpers.sh (Andrew J. Hesford)
- 812f9b4 - Remove support for deprecated KCL sources (Andrew J. Hesford)
- 3dcaa88 - Improve container-based builds and support mkinitcpio images (Andrew J. Hesford)
- f93687f - Annotate mkinitcpio.conf to explain ZBM specifics (Andrew J. Hesford)
- 91b4b3b - De-prioritize Dracut as the primary generator (Zach Dykstra)
- af34fa9 - Support building mkinitcpio images in generate-zbm (Andrew J. Hesford)
- 50a2da4 - General fixes to support busybox in mkinitcpio images (Andrew J. Hesford)
- 1fe8c4c - Add support for mkinitcpio (Andrew J. Hesford)
- e1bd708 - zbm-builder.sh: add script to build custom images using GHCR container (Andrew J. Hesford)
- 20a8ac7 - Add manual page for zbm-kcl (Andrew J. Hesford)

CHAPTER

THREE

ZFSBOOTMENU V1.12.0 (2022-01-25)

This release brings multiple changes to how ZFSBootMenu works at run-time. These changes were introduced in an effort to:

1. Reduce the ZFSBootMenu startup time,
2. Remove dependencies on Dracut-specific helper functions, and
3. Make ZFSBootMenu more modular and more easily maintained.

Prior to this release, the only Dracut helper functions in use were those that retrieve command-line arguments to dynamically configure ZFSBootMenu. These have all been replaced by internal functions that are both more correct and roughly an order of magnitude faster.

With the help of the flamegraph visualization tool, multiple unnecessary Dracut modules have been pruned from the binary releases. The net effect of these changes is a large decrease in the time spent booting to either the menu or directly to a boot environment.

ZFSBootMenu v1.12 is expected to be the last release series before ZFSBootMenu v2.0. The internal changes and dramatically reduced dependence on Dracut will allow ZFSBootMenu v2.0 images to be built using either dracut or mkinitcpio.

3.1 Deprecated features

- Support for `skim` has been removed. ZFSBootMenu now requires `fzf` for all menu functionality.
- Support for using `/etc/default/grub` and `/etc/default/zfsbootmenu` for boot environment kernel command line parameters will be removed in the next release.

3.2 Fixes

- Use `kexec --kexec-syscall-auto` to try multiple different ways to load a kernel
- Change the release files names to better indicate to rEFInd that they are Linux kernels
- `root=` is always removed/suppressed when passing a commandline into the boot environment
- More kernel commandline parameters are validated for correctness

3.3 New feaures

- Add the `zbm-kcl` userland tool to view and edit boot environment kernel command lines
- `generate-zbm` can now execute user hooks before and after an initramfs or EFI bundle has been created
- Teardown hooks now have access to variables indicating which boot environment, kernel and initramfs were selected
- We now provide `recovery` builds that include networking and a few disk-related tools
- Buildah is now used to create build images
- Countdown timers now have colored text by default

3.4 Significant commits in this release

- bd736f1 - Add a shell utility to modify and review KCL properties (Andrew J. Hesford)
- 466e4d2 - Allow rollbacks from the snapshot menu (Andrew J. Hesford)
- b9c4ed9 - Add Alpine to the ZBM test suite (Andrew J. Hesford)
- 2c8eacc - Add support for early/late hooks in generate-zbm (Zach Dykstra)
- 1ca3cf4 - Define BE selection variables in teardown hook environment (Andrew J. Hesford)
- 89b6273 - Enable profiling framework in core ZFSBootMenu tools (Zach Dykstra)
- 46b0eb9 - Completely rewrite KCL handling (Andrew J. Hesford, Zach Dykstra)
- 4b836f3 - Generate release and recovery image builds (Zach Dykstra)
- 6789ee2 - Provide, and use, a buildah script to construct zbm-builder images (Andrew J. Hesford)
- 70bad38 - Add initial boot-environment guide (Andrew J. Hesford)
- 4b836f3 - Generate release and recovery image builds (Zach Dykstra)
- 044df51 - Separate core library routines from main UI routines (Andrew J. Hesford)

CHAPTER

FOUR

ZFSBOOTMENU V1.11.0 (2021-10-31)

4.1 Updated defaults

- The chroot hot-key has been changed to `MOD+J` to avoid a key conflict with the usage of `CTRL+I`.

4.2 Deprecated features

- Support for reading the KCL from /etc/default/zfsbootmenu or /etc/default/grub will be removed in a future release. During the transition period, ZFSBootMenu will attempt to automatically convert a deprecated KCL configuration to the org.zfsbootmenu:commandline property.

4.3 Fixes

- The Makefile now correctly installs all files (this fixes the missing help pages)
- Make the width of the count down menu consistent as digits are dropped from the timer
- Remove a spurious function call in the preview renderer
- Where possible, prefer built-in Bash regex support over forking to grep
- Remove uses of basename, prefering Bash string manipulation
- Properly find the zpool executable when creating an initramfs
- Set a consistent environment for SSH and recovery shells
- Set default columns and rows for serial consoles when none are provided
- Improve the reliability of detecting which libgcc_s.so to install
- Set keylocation and a corrected encryption root for full-copy clones

4.4 New features

- Allow taking over a running instance of ZFSBootMenu via `zbm`
- Create a docker/podman image, based on Void Linux, that can be used to create ZFSBootMenu images
- Allow diffing a snapshot with another snapshot, instead of just with the current state of the filesystem
- Add early hooks, just after kernel modules have been loaded but before any pools have been imported
- org.zfsbootmenu:keysource can now be an arbitrary ZFS filesystem instead of another boot environment
- Snapshots can now be created, providing a starting point for a new boot environment
- `zreport` is available in the recovery shell, which can provide details-at-a-glance for bug reports
- The `testing/` infrastructure has been updated to allow the installation of multiple distributions in one command
- The Debian testing environment has been updated to Bullseye
- EFI bundles can be built and tested under `testing/`

4.5 Significant commits in this release

- 32f138f - Try to inherit key (encryptionroot) in duplicate_snapshot (Andrew J. Hesford)
- 357aa5e - Make keylocation of target match that of source in duplicate_snapshot (Zach Dykstra)
- c048ead - testing: move debian to bullseye (Andrew J. Hesford)
- 7bb154d - testing: allow installation of multiple distros in one setup run (Andrew J. Hesford)
- 8fab2b5 - Rework GitHub actions to also build assets on push (Zach Dykstra)
- 8922f87 - Add zreport function (Zach Dykstra)
- f0d402c - Support creating EFI bundles in run.sh, along with booting them. (Zach Dykstra)
- 2c58479 - Improve search for libgcc_s in module-setup.sh (Andrew J. Hesford)
- 726894f - Default stty cols/rows values for serial consoles (Zach Dykstra)
- 78656b0 - Small fixes for SSH usage (Zach Dykstra)
- 0e14c7b - Update zfsbootmenu.7 to reflect new org.zfsbootmenu:keysource behavior (Andrew J. Hesford)
- 891e44a - Allow snapshot creation / move logic to function (Zach Dykstra)
- d208103 - Make cache_key aware of mountpoints on key sources (Andrew J. Hesford)
- 2d03ff5 - Remove explicit path to zpool ldd check (Zach Dykstra)
- 29a1049 - Deprecate KCL in /etc/default/{zfsbootmenu,grub} (Andrew J. Hesford)
- c2a1bee - Support early hooks, add example luks-unlock.sh (Zach Dykstra)
- 62f7315 - Enable snapshot <> snapshot diffing (Zach Dykstra)
- c920c76 - Replace mod+i with mod+j (Zach Dykstra)
- a79b9d0 - zkexec: recovery shell arbitrary kexec wrapper (Zach Dykstra)
- 8d4d776 - Makefile: recursively install dracut module (Andrew J. Hesford)
- c1feb56 - recovery shell quality of life improvements (Zach Dykstra)
- 5876ca9 - Create a global debug/trace log viewer (Zach Dykstra)

- 83b1766 - Overhaul, generalize containerized build scripts (Andrew J. Hesford)
- cfa0455 - releng/make-binary.sh: use containerized image builds (Andrew J. Hesford)
- 4166967 - Add Dockerfile and Compose service to build ZBM images (Thomas Oster)
- 687295a - Allow zfsbootmenu to takeover a running instance (Zach Dykstra)
- 9000687 - Start adding aliases for missing niceties (Zach Dykstra)
- 804ef02 - Make sure feature flags are available everywhere (Zach Dykstra)
- 739540a - Error out in testing if machine type isn't known to run.sh. (Érico Nogueira Rolim)

CHAPTER

FIVE

ZFSBOOTMENU V1.10.1 (2021-07-04)

ZFSBootMenu 1.10.1 brings a pair of fixes to issues seen in the wild.

5.1 Fixes

- Remove a spurious warning when `generate-zbm` incorrectly detected a mismatch between `/etc/hostid` and the run-time `spl_hostid` SPL module parameter.
- Fix a race condition between ZFSBootMenu and udev when attempting to load ZFS/SPL kernel modules, resulting in an incorrect drop to a recovery shell.

5.2 Significant commits in this release

- 2a9a6ab - Relax insmod spl failures, try to load zfs.ko (Zach Dykstra)
- f657717 - generate-zbm: fix hostid comparison (Andrew J. Hesford)

CHAPTER

SIX

ZFSBOOTMENU V1.10.0 (2021-06-27)

ZFSBootMenu 1.10.0 brings some minor new features and some behavior changes that should improve the booting and configuration experience. Notably, some default behaviors have changed in this release. Read on for details about how this may impact your configuration.

6.1 Updated defaults

Previous releases have had `zbm.import_policy=strict` and `zbm.set_hostid=0` set by default. Starting with this release, the default values are `zbm.import_policy=hostid` and `zbm.set_hostid=1`. `zbm.import_policy=hostid` can help ZFSBootMenu automatically and safely import a pool when the wrong hostid is provided. `zbm.set_hostid=1` passes the hostid used to import the pool to the boot environment, ensuring that it can also correctly import the pool.

Please refer to Command-Line Parameters for a compete description of both of these feature flags.

6.2 Fixes

- When no hostid is provided or discoverable, use `0x00bab10c` instead of `0x0`.
- Force SPL to use `/etc/hostid` and always ensure that a valid hostid is stored in the file.
- Blacklist Plymouth; the splash screens it draws interferes with the ZBM interface.
- Persist runtime configuration to a file that is used when the interface is launched normally or through SSH.

6.3 New features

- When exiting the diff browser, exit back to the snapshot list with the snapshot selection preserved.
- Set a hostname in ZFSBootMenu so that it shows up in the pool history for read/write operations.
- When generating an initramfs, warn if `/etc/hostid` doesn't match `spl_hostid` provided as a module parameter to SPL.
- Support `console=` kernel parameters with a `,speed` suffix. This is normally used when setting a serial port as the machine console.
- Add a shortcut key to remove the pinned kernel value for a boot environment.

6.3.1 Allow references to parent properties in org.zfsbootmenu:commandline

Any reference to `%{parent}` in `org.zfsbootmenu:commandline` will be replaced with the value of the same property on the parent filesystem (with parent references above recursively expanded), allowing easy specification of common options at a mutual parent of two BEs and overrides or additions of individual options per-BE. The value of `%{parent}` is always an empty string on a root filesystem.

This is not intended to be sophisticated, and `%{parent}` appearing within other words will be replaced regardless. The assumption is that `%{parent}` is unique enough and will not conflict with real KCL options, so dumb global replacement is sufficient.

6.3.2 zpool import process improvements

The existing `zbm.prefer` option has been extended to support defining a mandatory pool. Append `!` to the pool name to indicate that the specific pool MUST be imported before any other pool imports will be attempted.

- `zbm.prefer=zroot!` will require that `zroot` be imported on boot.

Between pool import attempts, `zbm.import_delay` (default of 5 seconds) controls how long to pause. During this delay window, the escape key can be used to access a full recovery shell.

Either one of `spl_hostid` or `spl.spl_hostid` can be provided on the ZFSBootMenu kernel command line, in either hex or decimal format. The parameter value is checked to ensure that it's either valid hex or decimal, and then normalized to an 8 digit hex value.

Additional steps have been taken to ensure that SPL can be loaded if an invalid `spl.spl_hostid` value is provided on the kernel command line. A more strict test is now used to determine if the ZFS kernel module has been loaded and drop to a recovery shell if not.

6.4 Significant commits in this release

- c759df9 - For musl/non-dracut compat, change default hostid to non-zero value (Zach Dykstra)
- 8886db3 - Force spl.spl_hostid=0 when matching hostid (Andrew J. Hesford)
- 24e6e6a - Blacklist plymouth; it directly conflicts with how we manage the tty (Zach Dykstra)
- 9875115 - Store options from KCL in a file for easy sourcing (Andrew J. Hesford)
- 98f3164 - Use fzf's execute[] function to render draw_diff (Zach Dykstra)
- 64bd359 - Set a hostname in ZBM, check if spl.spl_hostid matches /etc/hostid (Zach Dykstra)
- 40dd0e3 - Small quality of life improvements (Zach Dykstra)
- 635d140 - Add keyboard shortcut to remove pinned kernel (Zach Dykstra)
- 04f5b87 - Allow references to parent properties in org.zfsbootmenu:commandline (Andrew J. Hesford)
- 5142aa1 - Support pool import retries (Andrew J. Hesford)
- e2caa81 - Improved pool imports (Andrew J. Hesford)
- 31cc1b3 - Validate spl_hostid, control loading spl.ko (Zach Dykstra)
- ccfc92c - Change ZBM hostid handling defaults (Zach Dykstra)

CHAPTER

SEVEN

ZFSBOOTMENU V1.9.0 (2021-03-29)

This release is dedicated to the late Jürgen Buchmüller (@pullmoll), a major contributor to the Void Linux project. Although ZFSBootMenu strives to support as many Linux distributions as possible, Void Linux is the distribution of choice for the entire ZFSBootMenu team. We have benefited greatly from pullmoll's enduring commitment to Void Linux. He will be missed.

7.1 Fixes

- Snapshot duplication now carries over ZFS properties from the source (cloning has always copied properties)
- When forcing a pool read-write via MOD+W, the screen is now cleared before a password prompt
- Prompts in Arch/Ubuntu/Debian chroots are now correctly set
- Build-time checks for version-specific flags in fzf/sk/dmesg improve compatibility with older versions found on Debian and Ubuntu
- Add `stty` to the list of required binaries
- When `generate-zbm` is executed with the `--debug` flag, `-q` is no longer passed in to Dracut
- When possible, try to let Dracut determine the path to the EFI stub file. The path can be explicitly specified by setting `EFI.Stub` in `config.yaml`
- Update the list of allowed characters in a ZFS filesystem name; removing `,` and adding `:`
- Use the Dracut built-in `inst_rules` to install udev rules, instead of hard-coding a file path
- Correctly set a `root=` prefix for Gentoo systems
- Improve kernel version detection in unversioned file names
- Update documentation to clarify driver exclusions and teardown hooks

7.2 Major New features

7.2.1 `hostid` configuration assistance

A pair of new features have been developed to combat the delicate dance sometimes required to synchronize `hostid` in the boot environment (BE), the initramfs for the BE, and the initramfs for ZFSBootMenu. Both of these are controlled by ZFSBootMenu kernel command line options.

Set `zbm.import_policy=hostid` to allow run-time reconfiguration of the SPL hostid. If a pool is preferred via `zbm.prefer` and the pool can not be imported with a preconfigured hostid, the system will attempt to adopt the hostid of

the system that last imported the pool. If a preferred pool is not set and no pools can be imported using a preconfigured hostid, the system will adopt the hostid of the first otherwise-importable pool. After adopting a detected hostid, ZFSBootMenu will subsequently attempt to import as many pools as possible.

Setting `zbm.set_hostid` will cause ZFSBootMenu to set the `spl.spl_hostid` command-line parameter for the selected boot environment to the hostid used to import its pool. The SPL kernel module will use this value as the hostid of the booted environment regardless of the contents of `/etc/hostid`. As a special case, if the hostid to be set is zero, ZFSBootMenu will instead set `spl_hostid=00000000`, which should be used by dracut-based initramfs images to write an all-zero /etc/hostid in the initramfs prior to importing the boot pool.

7.3 Minor new features

7.3.1 Boot environment / snapshot sorting

The main boot environment screen and snapshot list screen can now be sorted by multiple criteria. By default, these listings are sorted by name. One of `zbm.sort_key=name`, `zbm.sort_key=creation`, or `zbm.sort_key=used` can be set on the command line to change the default used. `MOD+O` can be pressed at run time to change the sort order to the next in the list.

7.3.2 Helper functions in the recovery shell

The internal `zfsbootmenu-lib.sh` library is now sourced by default in the recovery shell. This makes a number of helper functions available for general use.

7.3.3 Combined command line printer

Since `Dracut` can find command line options from multiple places, it can be difficult to determine which option was used to control bootloader behavior. The command `zbmcmdline` will show the command line as seen by Dracut.

7.3.4 Shorcut key layout improvements

Instead of flowing the helper key text to the width of the screen, shortcut key text is now arranged in one or more columns. Low resolution screens or terminals with small dimensions will use a single column to show the text. As the terminal dimensions increase, text will be laid out in up to a maximum of three left-aligned columns.

7.3.5 Logging system

ZFSBootMenu has an internal logging system backed by `/dev/kmsg` and `dmesg`. Logging is now enabled throughout the entire system, tracking both `debug` level messages as well as `warnings` and `errors`. When a warning or error condition has occurred, a yellow `[!]` or red `[!]` notification will appear in the upper left of the screen. Pressing `MOD+L` will access this logging system, allowing you to easily see these messages.

To aid in debugging issues, debug logging has been added in all of the internal library functions. These messages can be enabled by setting `loglevel=7` on the ZFSBootMenu command line.

7.4 Binary release for x86_64

Binary releases in the form of a standalone EFI file and a kernel/initramfs pair for `x86_64` will now be made available with each future tagged release. To allow for the most compatiblity with the myriad system configurations out there, a few features will be embedded in the builds:

- `zbm.import_policy=hostid`
- `zbm.set_hostid`
- xhci-teardown.sh

The EFI binary can be used as a recovery tool by naming it `BOOTX86.EFI` and adding it to an `EF00` partition on a USB drive. It can also be used as a drop-in bootloader for your system without needing to locally build a copy. See UEFI Booting for example `efibootmgr` commands.

7.5 Other changes

Some command line options have now been deprecated with the inclusion of `zbm.import_policy`. See zfsbootmenu(7).

7.6 Testing improvements

- Ability to install Arch / Ubuntu / Debian / Void / Void-musl on ZFS, in a VM, with a single command
  - This is huge.
- Verify that ZBM builds on Arch / Ubuntu / Debian / Void / Void-musl
- Verify that ZBM built on any distro boots any other distro
- Improvments to `run.sh` and `setup.sh` testing scripts to simplify usage

7.7 Significant commits in this release

- fadfd0f - Use --props and --raw when send | recv duplicating (Andrew J. Hesford)
- 0a4b1f7 - Use signify to sign binary assets (Andrew J. Hesford)
- 98cfc69 - Show if BE is encrypted, add path to chroot prompt (#157) (Zach Dykstra)
- 848bfc9 - Do not drop to emergency shell if zbm.preferred cannot be imported (Andrew J. Hesford)
- fd94635 - Fix PS1 in arch/debian/ubuntu, set custom prompt in emergency shell (#151) (Zach Dykstra)
- dee6228 - Support older versions of fzf/sk/dmesg (Zach Dykstra)
- 6e28f42 - Non-zero return from chroot should be debug info, not an error (Andrew J. Hesford)
- 82ab9a7 - Add `stty` to required exectables in module-setup.sh (Andrew J. Hesford)
- f962f50 - Use configurable destinations for all paths in Makefile (Andrew J. Hesford)
- 03a38b5 - Column view for help key footers (Andrew J. Hesford)
- 951111e - Add support to discover and assume a hostid, as well as fix arguments passed to a BE (#147) (Zach Dykstra)

- 29cf15c - Logging lifecycle (#146) (Zach Dykstra)
- 629346f - Automagically source zfsbootmenu-lib in emergency shell (Zach Dykstra)
- 73dc5c2 - Move more helper functions to -lib, cleanup scripts (#144) (Zach Dykstra)
- 44ed892 - Allow BE and snapshot sorting by different criteria (#143) (Zach Dykstra)
- 9afa8b4 - comma is invalid, colon is valid (Zach Dykstra)
- a99a8de - Update README.md to explain driver exclusions and teardown hooks (RoundDuckKira)
- 5ac481f - Use inst_rules to install rule programs (Witaut Bajaryn)
- 6cf753f - Use genkernel root prefix by default on Gentoo (#139) (Witaut Bajaryn)
- a6e02c0 - Only specify `--uefi-stub` when EFI.Stub is configured (Andrew J. Hesford)
- 893cbe9 - Improve kernel version detection. (Andrew J. Hesford)
- 8db6d8f - README: Describe more Perl dependencies (#134) (Witaut Bajaryn)

CHAPTER

EIGHT

ZFSBOOTMENU V1.8.1 (2021-01-12)

Happy New Year! ZFSBootMenu 1.8.1 provides a few minor enhancements and bug fixes.

8.1 Fixes

- Properly handle encryption keys as raw devices rather than normal files. (#127)
- Improve detection of latest kernels when `/boot` contains some unversioned kernel files. (#128)
- Accept `Ctrl` and `Ctrl-Alt` as hotkey modifiers in addition to `Alt` to fix issues with some non-US keymaps. (#124)
- Everywhere a chroot hotkey is offered, use the same hotkey.

8.2 New features

- Add hard wraps to the hotkey menus to improve clarity; ignored for small screens.
- In the snapshot list, add an option to jump into a read-only chroot for that snapshot.
- The `force_import` and `timeout` kernel command-line options are now expected to be `zbm.force_import` and `zbm.timeout`; the old forms are deprecated but will continue to work for the forseeable future.
- New `zbm.show` and `zbm.skip` command-line options force the menu to appear or be skipped if `zbm.timeout` is not also set.

8.3 Significant commits in this release

- ab95346 - Expect names of the form - when finding latest kernel (Andrew J. Hesford)
- 798fce8 - Normalize tests for paths. (Andrew J. Hesford)
- d02865e - Change chroot keys, add help text (#126) (Zach Dykstra)
- 83c58b2 - Support Ctrl and Ctrl-Alt in addition to Alt as keybind modifiers (Andrew J. Hesford)
- 70c3d70 - Namespace our KCL args, organize parsing (#120) (Zach Dykstra)
- c1d6ce5 - Support chroot'ing into a snapshot (Zach Dykstra)
- 7a63beb - Support for hard wrap points in header_wrap (Andrew J. Hesford)

CHAPTER

NINE

ZFSBOOTMENU V1.8.0 (2020-12-15)

ZFSBootMenu 1.8.0 offers a significant list of new features, fixes and general improvements.

9.1 Fixes

- When duplicating snapshots, the process can now be interrupted with SIGINT (Ctrl-C).
- Availability of sufficient free space is confirmed before attempting snapshot duplication.
- The `generate-zbm` command now ensures that the target directory for boot images has sufficient space, rather than copying partial and generally broken files.
- Because ZFSBootMenu never modifies filesystem contents, ZFS filesystems are always mounted read-only, even if the pool is writable.
- Changes to the handling of encryption keys correctly handle some corner cases, such as duplicating a snapshot of a filesystem with a different encryptionroot than its parent.

9.2 New features

- Colored text and timed messages are used to bring emphasis to important messages presented by ZFSBootMenu.
- Extensive logging to the kernel ring-buffer has been enabled throughout ZFSBootMenu, with verbosity controlled by the `loglevel` kernel-command-line argument.
- Much of the core menu functionality has been separated into a standalone `zfsbootmenu` program on the initramfs, allowing the menu to be accessed over SSH using something like the dracut-crypt-ssh module.
- Boot images can be configured to include tmux, allowing the boot menu to be presented in a multi-pane, detachable view. This is primarily aimed at development/debugging efforts. See descriptions of the `zbm.tmux` command-line option and the `zfsbootmenu_tmux` dracut option in the zfsbootmenu(7) manual page.
- ZFSBootMenu can now execute arbitrary, user-supplied "setup" hooks before the menu is displayed and "teardown" hooks immediately before jumping into a selected boot environment. See descriptions of the `zfsbootmenu_setup` and `zfsbootmenu_teardown` dracut options in the zfsbootmenu(7) manual page.
- Encryption keys can now be cached by pointing the `org.zfsbootmenu:keysource` property of a ZFS encryptionroot to a specific filesystem. When ZFSBootMenu attempts to load a key from a `file://` location, it will first attempt to load the key at that location relative to the filesystem specified by `org.zfsbootmenu:keysource`. If this succeeds, ZFSBootMenu will retain a copy of the key in the initramfs so that subsequent need for the key (for example, when re-importing a pool read-write to set default boot options or duplicate a snapshot) will not require re-entry of the passphrase. See descriptions of the `org.zfsbootmenu:keysource` ZFS property in the zfsbootmenu(7) manual page.

- The `Alt+C` hot key provides the means to chroot into a selected boot environment. If the pool is mounted read-write, the chroot will be writable, allowing recovery operations directly from ZFSBootMenu.

9.3 Significant commits in this release

- 8ffe139 - Add a keybind for zfs-chroot, rework script with -lib in mind (Zach Dykstra)
- 2c3e9b5 - Support configurable, opt-in caching of key files (Andrew J. Hesford)
- 45d0066 - Add zlog() logging helper, along with debug-level logging (Zach Dykstra)
- 47aa434 - Add support for richer setup and teardown hooks (Andrew J. Hesford)
- 88818a9 - Support optional "teardown" script to run before kexec (Andrew J. Hesford)
- 2045315 - Optionally launch under tmux (Zach Dykstra)
- 7f71d4f - Mount ZFS filesystems readonly (Andrew J. Hesford)
- d559b96 - Improve key handling (Andrew J. Hesford)
- ee468aa - Initial split of core menu logic from initialization (Andrew J. Hesford)
- 25baa48 - Log dracut command, add man page documentation (Zach Dykstra)
- 3f02b8f - Prevent copying of partial files when target volume is full (Andrew J. Hesford)
- 258d18a - Merge fuctionality of createInitramfs and unifiedEFI (Andrew J. Hesford)
- 4b6e9ae - Add subroutine for debug logging (Zach Dykstra)
- 82bbdc4 - Rely on local IFS override instead of global changes (Zach Dykstra)
- 402474b - Make alt-w a toggle between R/O and R/W imports (Andrew J. Hesford)
- 953c724 - Richer color handling in timed_prompt (formerly warning_prompt) (Andrew J. Hesford)
- eb39ea6 - Show countdown in warning_prompt, use to display auto-boot countdown (Zach Dykstra)
- f10e7e8 - Rough avail space validation in duplicate_snapshot (Zach Dykstra)
- bc0e117 - Move duplicate to sub shell, exit on sigint (Zach Dykstra)

CHAPTER

TEN

ZFSBOOTMENU V1.7.1 (2020-11-18)

This is a minor bug-fix release.

10.1 Fixes

- When ZFSBootMenu fails to import any usable pools on startup and drops to an emergency shell, the user can now manually import a pool if possible and exit the shell to attempt to continue the boot process. Previously, a reboot was required to retry the boot process.
- An oversight in the loading of encryption keys caused a harmless error message to be displayed above the password prompt. This oversight has been fixed and the error no longer appears.
- Changes made to the handling of `/etc/hostid` in the ZFSBootMenu dracut module in anticipation of similar changes in the upcoming OpenZFS 2.0.0 release. This caused inconsistent behavior on systems using the musl C library with current versions of ZFS on Linux, resulting in potentially unbootable systems without forcing the `spl_hostid` command-line parameter. Now, the ZFSBootMenu dracut module attempts to discover the installed version of ZFS and behave consistently.

10.2 Significant commits in this release

- 940cd4c - Fall back to legacy hostid creation for ZFS < 2.0 (Andrew J. Hesford)
- 6cc0076 - Fix key_wrapper calls with out CLEAR_SCREEN defined (Zach Dykstra)
- 65a1a33 - Loop the emergency shell when initial pool imports fail (Andrew J. Hesford)

CHAPTER

ELEVEN

ZFSBOOTMENU V1.7.0 (2020-11-15)

In addition to a bug fixes, this release targets refinements that improve usability and offer contextual help within the menus.

11.1 Fixes

- ZFSBootMenu now respects the `console` kernel command-line option and should behave as expected over a serial console.
- Command lists at the bottom of each menu are now sensibly wrapped to the terminal width, with extra coloring to highlight key combinations.
- Rather than rely solely on the `hostid(1)` command to populate the default hostid in the ZFSBootMenu initramfs, the dracut module will prefer to copy the `/etc/hostid` from the host, which should produce more consistent behavior on musl systems.
- Boot environments are now explicitly sorted, with the default boot environment appearing at the top of the list and selected by default.

11.2 New features

- An online help system, accessible from `alt-h` within any menu, provides descriptions of functionality provided by ZFSBootMenu.
- The description at the top of the menu now indicates whether the selected boot environment is on a pool currently imported readonly or writable.
- New command-line arguments `zbm.lines` and `zbm.columns` allow the size of the terminal at boot time.
- When `generate-zbm` fails to parse the YAML configuration, more detailed messages pinpoint parsing errors.

11.3 Significant commits in this release

- dbe91a1 - Fix console handling when attached to a serial line (Andrew J. Hesford)
- 9959d10 - Respect ZFS hostid behavior on musl (Andrew J. Hesford)
- f4a60e6 - Capture and print config.yaml eval failure (Zach Dykstra)
- 2ebad45 - Sort environments, fix preview (Zach Dykstra)
- cc6e27c - Control size/target of ZFSBootMenu output (Zach Dykstra)

- bb294b3 - Enable dynamic line wrapping for header (Zach Dykstra)
- 8993591 - Enable global help system (Zach Dykstra)
- 7879876 - Read-only helpers (Zach Dykstra)

CHAPTER

TWELVE

ZFSBOOTMENU V1.6.1 (2020-10-27)

Revert omitting `rootfs-block` by default from the ZFSBootMenu initramfs. `rootfs-block` is a hard requirement of `crypt`, which is used to setup LUKS beneath ZFS.

CHAPTER

THIRTEEN

ZFSBOOTMENU V1.6.0 (2020-10-24)

This release brings significant improvements to the pool import process. Previously, all available pools were discovered and then their health was scraped to confirm that they were in an 'ONLINE' state before importing them. This process had a few subtle shortcomings that were highlighted by the pending release of OpenZFS 2.0.0. In particular, if a zpool has been upgraded via `zpool upgrade` to enable OpenZFS 2.0.0 feature flags, but the ZFSBootMenu initramfs contains an older version of OpenZFS, the pool was not able to be automatically imported in ZFSBootMenu. The import process now relies on `zpool import -N -a -o readonly=on` to attempt to import all available and otherwise healthy pools in read-only mode. By using zpool itself to determine all of the pools that can/should be imported, ZFSBootMenu now avoids the fragile process of scraping and interpreting the human-friendly text output of `zpool import`.

13.1 Fixes

- Improve the reliability of returning the correct kernel command line at all times. A helper function can now return corrected arguments in cases where `resume` is on the command line, but the pool has been imported read-write. This function is now used when generating a preview, making the modification more transparent to user inspection.
- Omit systemd and other modules from ZFSBootMenu by default. These install hooks that interfere with the correct operation of ZFSBootMenu.
- The testing/virtualiztion frame work has received a lot of attention during this development cycle. This has allowed us to create a variety of pool configurations that would be otherwise difficult to accomplish with physical hardware.

13.2 New features

- When accepting user input (new filesystem name, resume protections), allow CTRL-C to be used to cancel the process.

13.3 Significant commits in this release

- 219a632 - Rewrite pool import ahead of OpenZFS 2.0.0 (Zach Dykstra, Andrew J. Hesford)
- fb866c8 - Use zfsbootmenu-input in noresume prompt (Andrew J. Hesford)
- 1bb4f41 - Allow ctrl-c to be used during user input (Zach Dykstra)
- ca8eda8 - use -F to display the file type for more filtering options (Zach Dykstra)
- 08f22e2 - Simplify handling of BE command lines, use in previews (Andrew J. Hesford)
- cd8f889 - omit rootfs-block; it will never generate a correct KCL (Zach Dykstra)
- c96165c - Omit systemd related modules (Zach Dykstra)

CHAPTER

FOURTEEN

ZFSBOOTMENU V1.5.0 (2020-09-16)

14.1 Fixes

- The required binaries were audited, cleaning up tools that were no longer used.
- The installation of required binaries, core scripts and other files is checked during initramfs creation. If any of these critical files can not be installed, the image is not created.
- Parameters are passed to the pool import function to control how a pool is imported. This allows for more than just readonly changes when importing.
- `config.yaml` defaults were adjusted to more closely match normal use cases. This works in tandem with the `--enable` toggle in `generate-zbm` to provide a better out-of-the-box experience.

14.2 New features

- When importing a pool as R/W, and a resume image is found, the kernel command line is modified to remove `resume=` and to subsequently append `noresume`.
- Support `skim` in place of `fzf` for greater platform availability. `fzf` is preferred when creating an initramfs.
- The health of discovered pools can now be viewed. Optionally, a checkpoint rewind can be performed if one has been set. Use caution, this action can NOT be undone.
- Global image creation can be toggled via `generate-zbm --enable` or `generate-zbm --disable`

14.3 Significant commits in this release

- cfec416 - Support automatic "noresume" when importing pools R/W (Andrew J. Hesford)
- a6a36bf - Add support for pool status and checkpoints (Zach Dykstra)
- 79bcca3 - Fix logic inversion in import_pool handling of $read_write (Andrew J. Hesford)
- 14b88b9 - generate-zbm: enable components in config.yaml (Andrew J. Hesford)
- 71cd075 - Add --enable/--disable (Zach Dykstra)
- 999e6c2 - Drop import_args variable and let import_pool build its own arguments (Andrew J. Hesford)
- b85d836 - support sk, an fzf workalike, for menu presentation (Zach Dykstra)
- 2225fea - module-setup.sh: catch installation failures, warn or fail as appropriate (#72) (Andrew J. Hesford)
- 5801e00 - Prune everything not explicitly needed (Zach Dykstra)

CHAPTER

FIFTEEN

ZFSBOOTMENU V1.4.1 (2020-08-19)

ZFSBootMenu 1.4.1 is a minor update, updating the provided Makefile for packagers.

CHAPTER

SIXTEEN

ZFSBOOTMENU V1.4 (2020-08-19)

ZFSBootMenu 1.4 includes significant internal changes and some user-visible functional changes in the `generate-zbm` script.

16.1 Fixes

- Correct an issue that required two attempts to set default boot environments.
- Internal improvements to `generate-zbm` to improve consistency and facilitate future development.
- Management of versioned image retention should now be more consistent with expectation. Versioned ZBM images now increment a revision number when existing images with the same version already exist, and the retention policy preserves a configurable number of revisions for the current version alongside the latest revision of each of the same number of prior versions.
- Improved error handling should make failures in `generate-zbm` easier to understand.

16.2 New features

- Provide man pages (generate-zbm.5, generate-zbm.8 and zfsbootmenu.7) to document the creation and use of ZFSBootMenu images.
- Move from an INI configuration format to YAML, which should provide more flexibility for future enhancements.
- Provide a `--migrate` command-line option to convert existing INI configurations to the new format.
- Add configuration options to change default behavior for `--kernel`, `--kver` and `--prefix` to make `generate-zbm` easier to incorporate on non-Void systems.
- Add a configuration option to change the default behavior for `--version` to allow customized output versioning of images.
- Support string interpolation of `%current` or `%{current}` tags in `--kver` and `--version` values.
- Add a `--cmdline` command-line option to override the configured `CommandLine` value without editing the configuration file.

16.3 Significant commits in this release

- 0979051 - Add documentation for generate-zbm, its config and initramfs options (Zach Dykstra, et al.)
- ee1d9d8 - Unmask `import_args` in functions calling `import_pool` (Zach Dykstra)
- 3b2b2f0 - Add explicit `--migrate` option to generate-zbm (Andrew J. Hesford)
- 3cd3a8e - Improve error handling and automatic config conversion (Andrew J. Hesford)
- 80e0c30 - Switch syslinux entry to heredoc, fix syslinux.cfg file copy (Zach Dykstra)
- 6351226 - Move to YAML configuration, improve version handling (Andrew J. Hesford, Zach Dykstra)
- 5fdb872 - Add configuration options for kernel, version and prefix (Andrew J. Hesford)
- 79295ec - Add an optional parameter to safeCopy: (Zach Dykstra)
- 8aa133f - Clean up control flow in generate-zbm (Andrew J. Hesford)

CHAPTER

SEVENTEEN

ZFSBOOTMENU V1.4RC1 (2020-08-11)

Except for the addition of man pages and the fix in commit ee1d9d8, the new features and fixes in this release are fully described in the final v1.4 release notes.

CHAPTER

EIGHTEEN

ZFSBOOTMENU V1.3.1 (2020-07-14)

This release fixes an issue found minutes after v1.3 was tagged and released - such is life. After timing out on the countdown menu, the screen is now cleared before displaying a prompt for the pool password.

CHAPTER

NINETEEN

ZFSBOOTMENU V1.3 (2020-07-14)

This release features several fixes and new features.

19.1 Fixes

- When creating a boot image, `generate-zbm` will fail if the EFI System Partition (`BootMountPoint` in the configuration) is not and cannot be mounted.
- When `generate-zbm` creates backup kernels, initramfs images or UEFI bundles, timestamps of the original files will be preserved when possible, which may help boot loaders like rEFInd properly order the images.
- Some display issues in the boot menu have been fixed.

19.2 New features

The following new features should allow ZFSBootMenu to work with distributions such as Arch or Ubuntu:

- The Dracut module now searches for a much broader range of kernel/initramfs pairs in boot environments, including unversioned kernels with names like `linux` or `vmlinuz`.
- `generate-zbm` now has a `--kver` argument that can specify a version number when one cannot be correctly determined from the name of a kernel file, allowing creation of ZFSBootMenu images on systems like Arch that do not encode version information in kernel names.
- When starting a boot environment, the `root=` command-line argument is now set with a prefix (*e.g.*, the `zfs:` part of `root=zfs:pool/ROOT/void`) that is chosen based on distribution ID in `/etc/os-release` or `/usr/lib/os-release`, if available; the default selection can be overridden by setting the `org.zfsbootmenu:rootprefix` property.

In addition:

- In the ZFSBootMenu snapshot browser, an option to view a `zfs diff` between the live boot environment and a selected snapshot allow convenient review of the changes since the snapshot was taken.
- ZFSBootMenu now attempts to detect an active suspend-to-disk image and prevent any operations on ZFS pools that could lead to corruption on resume.
- The currently selected boot environment and kernel are displayed throughout submenus.
- In addition to identifying boot environments by the property `mountpoint=/`, ZFSBootMenu will also identify boot environments with the properties `mountpoint=legacy` and `org.zfsbootmenu:active=on`.
- Boot environments with `mountpoint=/` can be hidden from ZFSBootMenu by setting the property `org.zfsbootmenu:active=off`.

19.3 Significant commits in this release

- 7122be9 - Use mountpoint to check for ESP (Zach Dykstra)
- 315e326 - Check return of mount operation (Zach Dykstra)
- 8e434b1 - Allow root prefix to be customized for other distributions (Andrew J. Hesford)
- fcaba86 - Support unversioned kernel naming in generate-zbm (Andrew J. Hesford)
- 294a84d - Broaden search for kernels and initramfs images (Andrew J. Hesford)
- 2263dbe - Handle kernels with multi-part versions (Zach Dykstra)
- 95f65a6 - Initial support for org.zfsbootmenu:active visibility (Andrew J. Hesford)
- 69c3d63 - Draw the preview header on kernel, snapshot and diff screens (Zach Dykstra)
- 83b2cbb - Initial support for resume guard (Andrew J. Hesford)
- 6828550 - Initial snapshot diff browser (Zach Dykstra)
- 4c0a968 - Report source size when cloning/duplicating a snapshot (Zach Dykstra)

CHAPTER

TWENTY

ZFSBOOTMENU V1.3RC2 (2020-07-09)

This release contains all of the fixes and new features in v1.3rc1, as well as some fixes to command-line generation that should allow ZFSBootMenu to properly boot Arch Linux systems.

CHAPTER

TWENTYONE

ZFSBOOTMENU V1.3RC1 (2020-07-07)

The new features and fixes in this release are fully described in the final v1.3 release notes.

CHAPTER

TWENTYTWO

ZFSBOOTMENU V1.2 (2020-06-22)

This release features substantial code and idea contributions from @ahesford . Thank you for all of your help writing features, debugging code and improving documentation.

22.1 Snapshot overhaul

Previously, snapshots could be cloned to a boot environment with a pre-generated, and often long, BE name. Substantial quality of life improvements were made here, including:

- Add the ability to do a full zfs send | zfs recv clone from a local snapshot. This lets you establish a new boot environment that is not dependent on any other environments or snapshots.
- Add the ability to clone and promote a snapshot, or simply clone it.
- When cloning a snapshot, local ZFS properties of the parent filesystem are now transferred to the clone.
- For all snapshot operations, you can now directly enter a boot environment name. This name is checked for character validity, and to confirm that it is not already taken.

22.2 Always up-to-date menu system

Any time you transition from one menu to another (Snapshots, Kernels, recovery shell), the list of boot environments and kernels is completely regenerated. This helps remove potential disconnects between the state of your pool and boot environments and the ZFSBootMenu interface.

22.3 UEFI bundle improvements

If you create unversioned UEFI bundles for static boot entries, `generate-zbm` will now create a `-backup` file for you on upgrade. This will allow you a recovery option if the active UEFI bundle has a problem.

22.4 Protect against missing kernel modules

When creating a new initramfs, the `zfsbootmenu` Dracut module needs to install a number of ZFS-related kernel modules. Previously, the modules were installed through a Dracut helper function that did not verify if the copy succeeded. This process has been reworked to ensure that all of the required kernel modules are installed in the initramfs, or the creation of the image is marked as a failure. If it is marked as a failure, existing/current images are not deleted or otherwise replaced.

22.5 Set default kernel

Much like setting a default boot environment, you can now set a default kernel for a specific boot environment. You no longer need to manually set a kernel version in your booted OS, you can simply do it from the menu!

22.6 Shellcheck

On commit, the shell scripts that power the boot menu and Dracut setup are run through a validator to check for common errors and pitfalls. This can help reduce some classes of bugs.

CHAPTER

TWENTYTHREE

ZFSBOOTMENU V1.1 (2020-06-11)

This release includes a number of small fixes and improvements.

CHAPTER

TWENTYFOUR

FIXES

- Correctly handle exiting the recovery shell, fixing an infinite loop. A reboot is no longer required to recover from the recovery shell. Thanks, @ahesford.
- Check that the EFI stub file is present on disk at the specified location. If the file is missing and an EFI bundle is requested, exit with an error.
- Minor documentation fixes.

CHAPTER

TWENTYFIVE

FEATURES

- Handle console fonts defined on the kernel command line. This is useful for systems with a 4k display. You should now be able to read the screen without a magnifying glass. Thanks to @ahesford for the significant time spent on tracking this down.
- Instead of calling `objcopy` directly, `dracut` is now used to generate a bundled EFI file. This means that the bundled EFI file can now be signed by Secure Boot keys! Thanks to @ericonr for this feature - and learning a bit of Perl to do it!

No configuration file changes are needed to use this release. Enjoy!

CHAPTER

TWENTYSIX

ZFSBOOTMENU V1.0 (2020-05-15)

We're jumping straight up to v1.0!

CHAPTER

TWENTYSEVEN

SMALL CHANGES

- Set the kernel log level to 0 when in the menu system, then restore it to the original value on boot
- Reverse sort the kernel list, so the most recent is always first
- Add an initial chroot helper script, `zfs-chroot` for the recovery shell. It can be invoked as `zfs-chroot pool/ROOT/BE`.
- Set the default config path in `generate-zbm` to `/etc/zfsbootmenu/conf.d`
- Allow the EFI stub file to be defined in config.ini
- Optionally read the kernel command line from `/etc/default/zfsbootmenu`

CHAPTER

TWENTYEIGHT

LARGE CHANGES

- Clean up or whitelist all issues in `zfsbootmenu-lib.sh`, `zfsbootmenu-preview.sh` and `zfsbootmenu.sh` noted by shellcheck.
- Support entering a custom kernel command line via `alt-c` on the main menu. The input line is pre-filled with the command line that would have been used on the next boot, for that environment. This command line is NOT persisted between reboots, it's simply here to let you recover an unbootable system.
- Add support for reading the kernel commandline from the ZFS property `org.zfsbootmenu:commandline`. This property is now considered the default/primary source of truth for the kernel command line - it takes precedence over `/etc/default/zfsbootmenu` and `/etc/default/grub`.

A big thank you to @ahesford for his code contributions and testing leading up to this release!

CHAPTER

TWENTYNINE

ZFSBOOTMENU V1.0RC2 (2020-05-12)

This release contains all of the fixes and features from 1.0rc1, as well as the following:

- Add support for reading the kernel commandline from the ZFS property `org.zfsbootmenu:commandline`. This property takes precedence over `/etc/default/grub` and `/etc/default/zfsbootmenu`
- Clean up or whitelist all issues in `zfsbootmenu-lib.sh`, `zfsbootmenu-preview.sh` and `zfsbootmenu.sh` noted by shellcheck.

A big thank you to @ahesford for his code contributions and testing leading up to this release candidate. It is very much appreciated!

CHAPTER

THIRTY

ZFSBOOTMENU 1.0RC1 (2020-05-11)

This release has been a long time coming. In no particular order, it contains the following:

- Set the kernel log level to 0 when in the menu system, then restore it to the original value on boot
- Reverse sort the kernel list, so the most recent is always first
- Add an initial chroot helper script for the recovery shell
- Set the default config path in `generate-zbm` to `/etc/zfsbootmenu/conf.d`
- Allow the EFI stub file to be defined in config.ini
- Support entering a custom kernel command line via `alt-c` on the main menu. The input line is pre-filled with the command line that would have been used on the next boot, for that environment. This command line is NOT persisted between reboots, it's simply here to let you recover an unbootable system.

CHAPTER

THIRTYONE

ZFSBOOTMENU 0.8.1 (2020-01-19)

This release adds a few improvements to generate-zbm.

- When finding kernels in /boot to use as the base for zfsbootmenu, the list is properly sorted so something like 5.4.11 is now a higher version than 5.3.18.
- Output files now have `_1` appended to them (both kernel and initramfs) to workaround a gotcha with Petitboots' syslinux.cfg parser ignoring files ending in `.0`.
- If the bootloader partition is defined in config.ini, generate-zbm will now mount it and unmount it for you. If it's already mounted, it's left alone.
- Syslinux mode now builds off of [Components] mode, instead of adding yet another code path for generating the initramfs. This makes syslinux mode simply a syslinux.cfg renderer.

CHAPTER

THIRTYTWO

ZFSBOOTMENU 0.8.0 (2020-01-12)

This release adds a preview function to the primary menu screen. Two lines are shown at the top of the display.

Line 1: Selected boot environment (if it's the bootfs) - and the default kernel Line 2: The discovered kernel arguments for the kernel in that boot environment

Additional features added in include:

- Set bootfs for a pool via `alt+d` on the main menu screen
- Ability to clone the same snapshot up to 1000 times. After cloning a snapshot, it can be set as the bootfs in the menu.
- Reduce the helper text to one line on all screens, to make it not as visually cluttered
- Finding kernel arguments has now been moved to a function, which can be extended to support multiple OS's as the need arises

CHAPTER

THIRTYTHREE

ZFSBOOTMENU 0.7.7 (2020-01-08)

This release adds support for generating a syslinux/extlinux-compatible configuration file. This can be used both with Petitboot on POWER hardware and with extlinux on x86_64.

CHAPTER

THIRTYFOUR

ZFSBOOTMENU 0.7.6 (2019-12-31)

This is a fairly minor release, with the removal of an external VERSION file and the inclusion of a basic Makefile being the major impacting changes.

CHAPTER

THIRTYFIVE

ZFSBOOTMENU 0.7.5 (2019-12-24)

A previous release defaulted pool imports to read-only mode. Because of this, cloning a snapshot would fail. This release fixes that by detecting if a pool is imported as read-only, exporting it and re-importing it read-write. If the snapshot being cloned is encrypted, keys will be loaded again (from file or via prompt).

To control read-write behavior, the command line argument `read_write` has been added. If unset, this option defaults to 0, importing the pool in read-only mode. Set to 1 to enable read-write on imported pools by default.

CHAPTER

THIRTYSIX

ZFSBOOTMENU 0.7.4.1 (2019-12-20)

Point release to fix issues on a fresh installation with missing target paths.

CHAPTER

THIRTYSEVEN

ZFSBOOTMENU 0.7.4 (2019-12-20)

This release fixes a glaring integration issue with rEFInd. Unified EFI files now use the platform kernel base (vmlinux, vmlinuz, etc), with the version and EFI appended. This allows them to integrate nicely with rEFInd's boot options and kernel roll-up features.

CHAPTER

THIRTYEIGHT

ZFSBOOTMENU 0.7.3 (2019-12-20)

This release adds the helper bin/generate-zbm which can help control the lifecycle of a the components needed to boot a system. It can generate a versioned kernel and initramfs, and/or a combined kernel/initramfs/commandline EFI executable. The helper script is able to able to do un-versioned files as well, creating initramfs-zfsbootmenu.img and then rotating that into initramfs-zfsbootmenu-backup.img when a new initramfs is created.

To better integrate with boot loaders like rEFInd, the option 'timeout' can be set on the command line when booting into the bootmenu.

A value of 0 will bypass the countdown timer and menu screens, attempting to boot the environment set by bootfs. A value of -1 will force you into the menu system, where you can pick a boot environment, kernel, snapshot etc. Any positive integer value will enable the countdown timer, where the system will then attempt to boot the environment set by bootfs.

The default timeout value remains at 10 seconds, consistent with previous behavior.

CHAPTER

THIRTYNINE

ZFSBOOTMENU 0.7.2 (2019-12-10)

Default the creation of the initramfs to hostonly mode, to substantially reduce the size of the generated file.

CHAPTER

FORTY

ZFSBOOTMENU 0.7.1 (2019-12-10)

Minor bug fix release. Correctly handle a zpool value on the command line pointing to a non-existent pool. Additionally, when trying to read a pinned kernel version, read it from the selected BE, and not the BE pointed to by `bootfs`

CHAPTER

FORTYONE

ZFSBOOTMENU 0.7.0 (2019-12-09)

This release includes the following:

- Include documentation on how to use ZFSBootMenu on a system with native encryption.
- Add the ability to prefer a specific kernel version via `org.zfsbootmenu:kernel` set on a filesystem. Refer to 17302b7d for additional details.
- Split out functions into `zfsbootmenu-lib.sh`, with some degree of documentation
- Modify clone snapshot functionality to clone the snapshot and then discover all kernels present in it. After cloning a specific snapshot, you can now access it from the main BE menu and boot any kernel, or the one set by `org.zfsbootmenu:kernel` at the time the snapshot was taken.
- Fix a small syntax error when handling no pools being imported
- Add supporting files for xbps packaging
- Switch pools to readonly on import, for additional safety
- No longer export a pool before kexec - saving us some number of seconds on boot.
- Do not sanity check memory in kexec for faster booting

CHAPTER

FORTYTWO

ZFSBOOTMENU 0.6.5 (2019-11-16)

This is largely a bug-fix release, built on top of Linux 5.3.10 and ZFS 0.8.2 for x86_64 and ppc64le (POWER8+).

- Properly sort the list of kernels by version, so that 5.x.10 is considered a higher version than 5.x.9
- Correctly set a return value for all zfs load-key operations
- Greatly simplify when a BE is mounted when trying to find kernels in /boot
- Only add a BE to the environment list if one or more kernel/initramfs pairs were found

CHAPTER

FORTYTHREE

ZFSBOOTMENU 0.6 (2019-11-06)

This release brings support for native ZoL encryption! It supports encryption on the entire pool, or enabled for a specific boot environment!

Prompting for the passphrase happens if the key needs to be loaded to boot the environment set in `bootfs` or if you escape the auto-boot screen to enter the environment/snapshot/kernel browser.

A patch is provided for 90zfs/mount-zfs.sh which detects if the keylocation is a file, and then attempts to load it from disk. If the key file is not present, and the type is passphrase, it will prompt.

The default auto-boot screen now attempts to center itself in your tty, for a slightly easier to read output.

Booting from a snapshot has been fixed - the snapshot is now correctly unmounted after a kernel/initramfs pair is located in /boot from the snapshot.

CHAPTER

FORTYFOUR

ZFSBOOTMENU 0.5 (2019-10-22)

Initial release!

The dracut module has been built into an initramfs for both x86_64 and ppc64le (POWER8+) - with Linux 5.3.7. A sample grub.cfg is provided, demonstrating how to enter the boot menu. Update your pool name, and set spl_hostid based on the output of 'hostid' on your machine.

CHAPTER

FORTYFIVE

ZFSBOOTMENU

45.1 SYNOPSIS

ZFSBootMenu behavior is controlled through ZFS filesystem properties and command-line options provided to the ZFSBootMenu kernel.

45.2 Command-Line Parameters

These options are set on the kernel command line when booting the initramfs or UEFI bundle. Default options were chosen to allow general systems to boot without setting any values.

spl_hostid=<hostid>

When creating an initramfs or UEFI bundle, the */etc/hostid* from the system is copied into the target. If this image will be used on another system with a different hostid, replace **<hostid>** with the desired hostid, as an eight-digit hexadecimal number, to override the value contained within the image.

zbm.prefer

ZFSBootMenu will attempt to import as many pools as possible to identify boot environments and will, by default, look for the *bootfs* property on the first imported pool (sorted alphabetically) to select the default boot environment. This option controls this behavior.

zbm.prefer=<pool>

The simplest form attempts to import **<pool>** before any other pool. The *bootfs* value from this pool will control the default boot environment.

zbm.prefer=<pool>!

If a literal *!* has been appended to the pool name, ZFSBootMenu will insist on successfully importing the named pool before attempting to import any others.

zbm.prefer=<pool>!!

If a literal *!!* has been appended to the pool name, ZFSBootMenu will insist on successfully importing the named pool and no others.

zbm.import_delay=<time>

Should ZFSBootMenu fail to successfully import any pool, it will repeat import attempts indefinitely until at least one pool can be imported or the user chooses to drop to a recovery shell. Each subsequent attempt will proceed after a delay of **<time>** seconds. When **<time>** is unspecified or is anything other than a positive integer, a default value of 5 seconds will be used.

zbm.import_policy

This option controls how the pool import process should take place.

zbm.import_policy=hostid

Set this option to allow run-time reconfiguration of the SPL hostid. If a pool is preferred via **zbm.prefer** and the pool can not be imported with a preconfigured hostid, the system will attempt to adopt the hostid of the system that last imported the pool. If a preferred pool is not set and no pools can be imported using a preconfigured hostid, the system will adopt the hostid of the first otherwise-importable pool. After adopting a detected hostid, ZFSBootMenu will subsequently attempt to import as many pools as possible. This is the default import policy.

zbm.import_policy=strict

Set this option to only import pools that match the SPL hostid configured in ZFSBootMenu. If none can be imported, an emergency shell will be invoked. The *strict* policy is consistent with the behavior of earlier versions of ZFSBootMenu.

zbm.import_policy=force

Set this option to attempt to force pool imports. When set, this invokes *zpool import -f* in place of the regular *zpool import* command, which will attempt to import a pool that's potentially in use on another system. Use this option with caution!

zbm.set_hostid

Setting this option will cause ZFSBootMenu to set the *spl.spl_hostid* command-line parameter for the selected boot environment to the hostid used to import its pool. The SPL kernel module will use this value as the hostid of the booted environment regardless of the contents of */etc/hostid*. As a special case, if the hostid to be set is zero, ZFSBootMenu will instead set *spl_hostid=00000000*, which should be used by dracut-based initramfs images to write an all-zero */etc/hostid* in the initramfs prior to importing the boot pool. This option is on by default.

Note: Setting *spl.spl_hostid* to a non-zero value on the kernel commandline will make the ZFS kernel modules **ignore** any value set in */etc/hostid*. To restore standard ZFS behavior on a running system, execute

```
echo 0 > /sys/module/spl/paramters/spl_hostid
```

---

zbm.sort_key

This option accepts a ZFS property name by which the boot environment and snapshot lists will be sorted.

zbm.sort_key=name

Sort the lists by *name*. This is the default sorting method.

zbm.sort_key=creation

Sort the lists by *creation* date.

zbm.sort_key=used

Sort the lists by size *used*.

zbm.timeout

This option accepts numeric values that control whether and when the boot-environment menu should be displayed.

zbm.timeout=0 | zbm.skip

When possible, bypass the menu and immediately boot a configured *bootfs* pool property.

zbm.timeout=-1 | zbm.show

Rather than present a countdown timer for automatic selection, immediately display the boot-environment menu.

zbm.timeout=<positive integer>

Display a countdown timer for the specified number of seconds before booting the configured *bootfs* boot environment.

zbm.hookroot=<hookspec>

Tell ZFSBootMenu to attempt to read any early-setup, setup or teardown hooks from the path specified by *hookspec* in addition to any included directly in the image.

The *hookspec* parameter takes the form

```
device//path
```

where *device* is either a regular device node (e.g., */dev/sda*) or other partition identifier recognized by *mount(8)* (e.g., *LABEL=<label>* o *UUID=<uuid>*). The *path* component following *//* represents the location of a directory with respect to the root of the filesystem on *device*. For example, if a partition with a UUID of *DEAD-BEEF* is mounted at */boot/efi* on a running system and the hook root should refer to the path

```
/boot/efi/EFI/zfsbootmenu/hooks,
```

the corresponding hook specification should be

```
zbm.hookroot=UUID=DEAD-BEEF//EFI/zfsbootmenu/hooks
```

on the ZFSBootMenu command line. Note that any kernel modules necessary to mount the specified filesystem must be present in the ZFSBootMenu image. (For example, mounting a FAT32 filesystem may require that *vfat.ko*, *fat.ko*, *nls_cp437.ko* and *nls_iso8859_1.ko* be added to the image.)

Within the hook root, create subdirectories *early-setup.d*, *setup.d* or *teardown.d* to hold hooks for the respective stages of hook execution (early-setup, setup and teardown). ZFSBootMenu will mount the device named by the hook specification, look for the individual hook directories, and copy any files found therein into its own memory-backed root filesystem. The copy is not recursive and further subdirectorie are ignored. Note that, because ZFSBootMenu copies these scripts into its standard hook paths at each boot, it is possible to "mask" a script explicitly included in the ZFSBootMenu image by including an external hook script with the same name in the appropriate directory.

45.3 Deprecated Command-Line Parameters

timeout

Deprecated; use **zbm.timeout**.

root=zfsbootmenu:POOL=<pool>

Deprecated; use **zbm.prefer**.

force_import=1

Deprecated; use **zbm.import_policy=force**.

zbm.force_import=1

Deprecated; use **zbm.import_policy=force**.

45.4 ZFS Properties

The following properties can be set at any level of the boot-environment hierarchy to control boot behavior.

org.zfsbootmenu:kernel

An identifier used to select which kernel to boot among all kernels found in the */boot* directory of the selected boot environment. This can be a partial kernel name (e.g., *5.4*) or a full filename (e.g., *vmlinuz-5.7.11_1*).

If the identifier does not match any kernels, the latest kernel will be chosen as a fallback.

org.zfsbootmenu:commandline

A list of command-line arguments passed to the kernel selected by ZFSBootMenu for final boot. The special keyword *%{parent}* will be recursively expanded to the value of **org.zfsbootmenu:commandline** at the parent of the boot environment. Thus, for example,

```
zfs set org.zfsbootmenu:commandline="zfs.zfs_arc_max=8589934592" zroot
zfs set org.zfsbootmenu:commandline="%{parent} elevator=noop" zroot/ROOT
zfs set org.zfsbootmenu:commandline="loglevel=7 %{parent}" zroot/ROOT/be
```

will cause ZFSBootMenu to interpret the kernel command-line for *zroot/ROOT/be* as

```
loglevel=7 zfs.zfs_arc_max=8589934592 elevator=noop
```

Never set the *root=* argument; ZFSBootMenu always sets this option based on the selected boot environment.

org.zfsbootmenu:active

This controls whether boot environments appear in or are hidden from ZFSBootMenu.

off

For boot environments with *mountpoint=/*, set **org.zfsbootmenu:active=off** to **HIDE** the environment.

on

For boot environments with *mountpoint=legacy*, set **org.zfsbootmenu:active=on** to **SHOW** the environment.

By default, ZFSBootMenu only shows boot environments with the property *mountpoint=/*.

org.zfsbootmenu:rootprefix

This specifies the prefix added to the ZFS filesystem provided as the root filesystem on the kernel command line. For example, the command-line argument *root=zfs:zroot/ROOT/void* has root prefix *root=zfs:*.

The default prefix is *root=zfs:* for most boot environments. Environments that appear to be Arch Linux will use *zfs=* by default, while those that appear to be Gentoo or Alpine will use a default of *root=ZFS=*. The root prefix is generally determined by the initramfs generator, and the default is selected to match the expectation of the preferred initramfs generator on each distribution.

Set this property to override the value determined from inspecting the boot environment.

org.zfsbootmenu:keysource=<filesystem>

If specified, this provides the name of the ZFS filesystem from which keys for a particular boot environment will be sourced.

Normally, when ZFSBootMenu attempts to load encryption keys for a boot environment, it will attempt to look for a key file at the path specified by the *keylocation* property on the *encryptionroot* for that boot environment. If that file does not exist, and *keyformat=passphrase* is set for the *encryptionroot* (or *keylocation=prompt*), ZFSBootMenu will prompt for a passphrase to unlock the boot environment. These passphrases entered are not cached by default.

When **org.zfsbootmenu:keysource** is a mountable ZFS filesystem, before prompting for a passphrase when *keylocation* is not set to *prompt*, ZFSBootMenu will attempt to mount **<filesystem>** (unlocking that, if necessary) and search for the key file within **<filesystem>**. When **<filesystem>** specifies a *mountpoint* property that is not *none* or *legacy*, the specified mount point will be stripped (if possible) from the beginning of any *keylocation* property to attempt to identify a key at the point where it would normally be mounted. If no file exists at the stripped path (or the *mountpoint* specifies *none* or *legacy*), keys will be sought at the full path of *keylocation* relative to **<filesystem>**. If a key is found at either location, it will be copied to the initramfs. The copy in the initramfs will be used to decrypt the original boot environment. Copied keys are retained until ZFSBootMenu boots an environment, so a single password prompt can be sufficient to unlock several pools with the same *keysource* or prevent prompts from reappearing when the pool must be exported and reimported (for example, to alter boot parameters from within ZFSBootMenu).

45.5 Options for dracut

In addition to standard dracut configuration options, the ZFSBootMenu dracut module supports addtional options to customize boot behavior.

zfsbootmenu_early_setup=<executable-list>

An optional variable specifying a space-separated list of paths to setup hooks that will be installed in the ZFSBootMenu initramfs. Any path in the list **<executable-list>** that exists and is executable will be installed.

Any installed early hooks are run after SPL and ZFS kernel modules are loaded and a hostid is configured in */etc/hostid*, but before any zpools have been imported.

zfsbootmenu_setup=<executable-list>

An optional variable specifying a space-separated list of paths to setup hooks that will be installed in the ZFSBootMenu initramfs. Any path in the list **<executable-list>** that exists and is executable will be installed.

Any installed hooks are run right before the ZFSBootMenu menu will be presented; ZFS pools will generally have been imported and the default boot environment will be available in the *BOOTFS* environment variable. Hooks will not be run if the countdown timer expires (or was set to zero) and the default boot environment is automatically selected. **Note:** The hooks may be run multiple times if the menu is invoked multiple times, e.g., by dropping to an emergency shell and then returning to the menu. If a script should only run once, the script is responsible for keeping track of this.

zfsbootmenu_teardown=<executable-list>

An optional variable specifying a space-separated list of paths to teardown hooks that will be installed in the ZFSBootMenu initramfs. Any path in the list **<executable-list>** that exists and is executable will be installed.

Some hardware initialized by the kernel used to boot ZFSBootMenu may not be properly reinitialized when a boot environment is launched. Any teardown hooks installed into the ZFSBootMenu initramfs will be run immediately before **kexec** is invoked to jump into the selected kernel. This script can be used, for example, to unbind drivers from hardware or remove kernel modules.

Teardown hooks have access to three environment variables that describe the boot environment that is about to be launched:

ZBM_SELECTED_BE

The ZFS filesystem containing the boot environment that is about to be launched.

ZBM_SELECTED_KERNEL

The path to the kernel that will be booted, relative to the root of **ZBM_SELECTED_BE**.

ZBM_SELECTED_INITRAMFS

The path to the initramfs corresponding to the selected kernel, again relative to the root of **ZBM_SELECTED_BE**.

The hook *must not* assume that the filesystem **ZBM_SELECTED_BE** is currently mounted or that the pool on which it resides is currently imported. However, a teardown hook has the freedom to import a pool (preferably read-only) and mount the boot environment to inject additional processing before boot. To abort a pending boot, invoking

```
kexec --unload
```

should be sufficient to return to the main menu. Likewise, the hook may construct and execute its own *kexec* command to alter boot-time parameters. This may be useful, for example, to allow ZFSBootMenu to select a boot environment and then restructure the boot process to launch a Xen kernel with the selected environment configured as dom0.

45.6 Options for mkinitcpio

The **dracut** options specified above may also be specified in a mkinitcpio configuration file when **generate-zbm** is configured to create images using **mkinitcpio**. However, whereas the **<executable-list>** values in the dracut configuration should be specified as a single, space-separated string; in the mkinitcpio configuration, each **<executable-list>** value must be specified as a Bash array like the standard mkinitcpio arguments.

The following additional arguments may be provided in the mkinitcpio configuration file to further control the creation of ZFSBootMenu images:

zfsbootmenu_module_root=<path>

Set this variable to override the default **<path>** where the mkinitcpio hook looks for the components of ZFSBootMenu that must be installed in the created image.

zfsbootmenu_miser=yes

By default, **mkinitcpio** uses busybox to populate initramfs images. However, the *zfsbootmenu* hook will install system versions of several utilities that it requires to operate. On most systems, these versions will be provided by util-linux rather than busybox. To prefer busybox for these utilities when possible, set **zfsbootmenu_miser=yes**. Synonyms for *yes* are *1*, *y* or *on*, without regard to letter case.

45.7 SEE ALSO

generate-zbm(5) *generate-zbm(8)* `dracut.conf(5)` `mkinitcpio.conf(5)`

CHAPTER

FORTYSIX

GENERATE-ZBM

46.1 SYNOPSIS

generate-zbm [options]

46.2 OPTIONS

Where noted, command-line options supersede options in the *generate-zbm(5)* configuration file.

--version|-v *zbm-version*

Override the ZFSBootMenu version in output files; supersedes *Global.Version*

--kernel|-k *kernel-path*

Manually specify a specific kernel; supersedes *Kernel.Path*

--kver|-K *kernel-version*

Manually specify a specific kernel version; supersedes *Kernel.Version*

--prefix|-p *image-prefix*

Manually specify the output image prefix; supersedes *Kernel.Prefix*

--initcpio|-i

Force the use of mkinitcpio instead of dracut.

--no-initcpio|-i

Force the use of dracut instead of mkinitcpio.

--confd|-C *config-path*

Specify initramfs configuration path

- For dracut: supersedes *Global.DracutConfDir*
- For mkinitcpio: supersedes *Global.InitCPIOConfig*

--hookd|-H *hookd-path*

Specify mkinitcpio hook directory; supersedes *Global.InitCPIOHookDirs*

May be specified more than once. Ignored when using dracut.

--cmdline|-l *options*

Override the kernel command line; supersedes *Kernel.CommandLine*

--bootdir|-b *boot-path*

Specify the path to search for kernel files; default: */boot*

--config|-c *conf-file*

Specify the path to a configuration file; default: */etc/zfsbootmenu/config.yaml*

--enable

Set the *Global.ManageImages* option to true, enabling image generation.

--disable

Set the *Global.ManageImages* option to false, disabling image generation.

--debug|-d

Enable debug output

--showver|-V

Print ZFSBootMenu version and quit.

46.3 SEE ALSO

generate-zbm(5) *zfsbootmenu(7)*

CHAPTER

FORTYSEVEN

ZBM-KCL

47.1 SYNOPSIS

zbm-kcl [OPTION]... [FILESYSTEM|EFI_EXECUTABLE]

47.2 DESCRIPTION

The **zbm-kcl** utility allows review and manipulation of the *org.zfsbootmenu:commandline* property on ZFS filesystems or the *.cmdline* section encoded within ZFSBootMenu EFI executables. ZFSBootMenu reads the property *org.zfsbootmenu:commandline*, as set or inherited on each environment that it recognizes, to set the command line for the kernel that it boots. The ZFSBootMenu EFI executable reads its own *.cmdline* section to parse options that control the behavior of ZFSBootMenu itself.

The final argument is treated as a ZFS filesystem as long as one exists with the specified name. If a matching filesystem cannot be found, the argument is treated as an EFI executable. To force **zbm-kcl** to treat the final argument as a relative path to an EFI executable even when a ZFS filesystem exists with the same name, prefix the path with *./*.

When neither a filesystem nor an EFI executable is specified, **zbm-kcl** will attempt to determine the root filesystem and operate on that.

If an EFI executable of - is specified, *stdin* will be read as an EFI executable.

With no options specified, **zbm-kcl** will print the current value of *org.zfsbootmenu:commandline* of the selected filesystem or the *.cmdline* section of the named EFI executable and exit.

47.3 OPTIONS

-a *argument*

Append the value of *argument* to the kernel command line. The value of *argument* can be a simple variable name for Boolean arguments or may take the form *var*=*value* to provide a non-Boolean value. Multiple command-line arguments may be accumulated into a single *argument*. If the value of any variable value contains spaces, it should be surrounded by double quotes. In that case, surround the entire argument in single quotes to ensure that the double quotes are recorded in the property:

```
zbm-kcl -a 'variable="some argument with spaces"'
```

This argument may be repeated any number of times.

-r *argument*

Remove *argument* from the kernel command line. The value of *argument* can be a simple variable name, in which case all arguments of the form *argument* or *argument*=<*arbitrary-value*> will be stripped. Alternatively, a specific argument may be selected by specifying *argument*=<*specific-value*>.

This argument may be repeated any number of times.

Note: All removal options are processed *before* any append options are processed, making it possible to replace an existing argument by combining removal and append options into a single invocation of **zbm-kcl**.

-e

Open the contents of the command-line in an interactive editor. If the environment defines *$EDITOR*, that will be used; otherwise, **vi** will be used by default. After making changes as desired, overwrite the (temporary) file that was opened and quit the editor. The contents of the saved file will be written by **zbm-kcl** as the new command line.

-d

Delete the command-line property.

For a ZFS filesystem, this is accomplished by calling

```
zfs inherit org.zfsbootmenu:commandline <filesystem>
```

to allow the boot environment to inherit any command-line property that may be defined by some parent.

For a ZFSBootMenu EFI executable, the *.cmdline* section will be stripped.

-o *destination*

Save the modified command line to *destination* rather than back to the original source. When the source is a ZFS filesystem, the destination must also be a valid ZFS filesystem. When the source is an EFI executable, the destination will be treated as a file; a special EFI *destination* of - will cause the file to be written to *stdout*.

47.4 EXAMPLES

Change the *loglevel* value on the currently booted environment by removing any existing value from the command line and appending the desired argument:

```
zbm-kcl -a loglevel=7 -r loglevel
```

Delete the entire command line from the *zroot/ROOT/void* boot environment, allowing it to inherit a command line set at *zroot* or *zroot/ROOT* if either of these defines a value:

```
zbm-kcl -d zroot/ROOT/void
```

Allow interactive editing of the command line on the *zroot/ROOT* filesystem, but save the resulting changes to *zroot/ROOT/void* rather than back to *zroot/ROOT*:

```
zbm-kcl -e -o zroot/ROOT/void zroot/ROOT
```

Review the current command line embedded in the EFI file */boot/efi/EFI/zfsbootmenu/zfsbootmenu.EFI*:

```
zbm-kcl /boot/efi/EFI/zfsbootmenu/zfsbootmenu.EFI
```

Fetch the official ZFSBootMenu release EFI executable, customizing the menu timeout and saving the result to *zfsbootmenu-custom.EFI*:

```
curl -L https://get.zfsbootmenu.org/efi | \
  zbm-kcl -a zbm.timeout=15 -r zbm.timeout -o zfsbootmenu-slow.EFI -
```

47.5 SEE ALSO

zfsbootmenu(7)

CHAPTER

FORTYEIGHT

GENERAL

48.1 Boot Environments and You: A Primer

ZFSBootMenu adapts to a wide range of system configurations by making as few assumptions about filesystem layout as possible. When looking for Linux kernels to boot, the *only* requirements are that:

1. At least one ZFS pool be importable,
2. At least one ZFS filesystem on any importable pool have *either* the properties
   - `mountpoint=/` and **not** `org.zfsbootmenu:active=off`, or
   - `mountpoint=legacy` and `org.zfsbootmenu:active=on`

   For filesystems with `mountpoint=/`, the property `org.zfsbootmenu:active` provides a means to opt **out** of scanning this filesystem for kernels. For filesystems with `mountpoint=legacy`, `org.zfsbootmenu:active` provides a means to opt **in** to scanning this filesystem for kernels. Filesystems that do not satisfy these conditions are *never* touched by ZFSBootMenu.
3. At least one of the scanned ZFS filesystems contains a `/boot` subdirectory that contains at least one paired kernel and initramfs.

ZFSBootMenu will present a list of all ZFS filesystems that satisfy these constraints. Additionally, if any filesystem provides more than one paired kernel and initramfs, it is possible to choose which kernel will be loaded should that filesystem be selected for booting. (It is, of course, possible to automate the selection of a filesystem and kernel so that ZFSBootMenu can boot a system without user intervention.)

48.1.1 Finding Kernels

Although it may be possible to compile a kernel with built-in ZFS support that would be capable of booting from a ZFS root without an initramfs, this is not standard practice and would require considerable expertise. Consequently, ZFSBootMenu requires that a kernel be matched with an initramfs image before it will attempt to boot the kernel. ZFSBootMenu tries hard to identify matched pairs of kernels and initramfs images as installed by a wide range of Linux distributions. As noted above, kernel and initramfs pairs are required to reside in a `/boot` subdirectory of ZFS filesystem scanned by ZFSBootMenu. The kernel must begin with one of the following prefixes:

- vmlinuz
- vmlinux
- linux
- linuz
- kernel

After the prefix, the name of a kernel may be optionally followed by a hyphen (`-`) and an arbitrary string, which ZFSBootMenu considers a *version* identifier.

ZFSBootMenu attempts to match one of several possible initramfs names for each kernel it identifies. Broadly, an initramfs is paired with a kernel when its name matches one of four forms:

- `initramfs-${label}${extension}`
- `initramfs${extension}-${label}`
- `initrd-${label}${extension}`
- `initrd${extension}-${label}`

The value of `${extension}` may be empty or the text `.img` and may additionally be followed by one of several common compression suffixes: `gz`, `bz2`, `xz`, `lzma`, `lz4`, `lzo`, or `zstd`. The value of `${label}` is either:

- The full name of the kernel file with path components removed, *e.g.*, `vmlinuz-5.15.9_1` or `linux-lts`; or
- The version part of a kernel file (if the kernel contains a version part):
  - For `vmlinuz-5.15.9_1`, this is `5.15.9_1`;
  - For `linux-lts`, this is `lts`.

ZFSBootMenu prefers the more specific label (the full kernel name) when it exists.

48.1.2 Boot Environments

Internally, ZFSBootMenu does not understand the concept of a boot environment. When it finds a suitable kernel and initramfs pair, it will load them and invoke `kexec` to jump into the chosen kernel. In fact, ZFSBootMenu doesn't even require that a "root" filesystem be the real root that a kernel and initramfs will mount. It would be possible, for example, to mount a ZFS filesystem at `/kernels` and install kernels and matching initramfs images to the `/kernels/boot` subdirectory. As long as the `/kernels` filesystem has a `mountpoint` property (along with `org.zfsbootmenu:active` if needed), ZFSBootMenu will identify the kernels even if the filesystem at `/kernels` contains nothing besides the `boot` subdirectory.

Although ZFSBootMenu ensures maximum flexibility by imposing minimal assumptions on filesystem layout, not all layouts are equally sensible. For straightforward maintenance and administration, it is recommended that each Linux operating system that you wish to boot be stored as a self-contained boot environment. Conceptually, the ZFSBootMenu team recommends that a *boot environment* consist of a **single** ZFS filesystem that contains all of the *coupled* system state for that environment. Coupled system state embodies the executables, configuration and other files that are critical to proper system operation and must generally be kept consistent at all times. In most systems, coupled system state tends to be maintained by a package manager. The package manager might install programs in `/usr/bin`, configuration in `/etc` and other files throughout the filesystem. The package manager itself probably maintains a database of installed packages somewhere in `/var`.

ZFSBootMenu is certainly capable of booting an environment that mounts separate filesystems at `/` and other paths like `/etc`, `/usr` or `/var`. ZFSBootMenu never needs to understand these details because either the initramfs or root filesystem will assume responsibility for mounting all filesystems it needs. However, a key benefit of boot environments is *atomicity*. In general, it is bad to allow the contents of `/usr` to become inconsistent with the package manager database on `/var`. Configuration files in `/etc` can often be tied to specific versions of software, so they should be kept consistent as well. When these directories live on different filesystems, ensuring consistency becomes much more challenging.

For example, suppose that a software update has gone wrong and a program has been overwritten by a corrupt or buggy version. With ZFS snapshots, `zfs rollback` is sufficient to restore functionality. However, when `/usr` and `/var` reside on different filesystems, both must be rolled back to the same point in time. When the filesystems are on different snapshot schedules (or there is some delay between snapshotting one after the other), deciding which snapshots represent consistent state may not be a trivial task.

To some extent, this could be remedied with a recursive snapshot scheme that provides uniform nomenclature for consistent snapshots across multiple filesystems. However, ZFSBootMenu strives to provide simple management and recovery interfaces for all boot environments on a disk, and

1. Providing a convenient interface for rollback of a boot environment becomes substantially harder if ZFSBootMenu has to identify snapshots across multiple filesystems that might compose an environment,
2. Even identifying which filesystems should be considered part of an environment is not always a trivial task, and
3. The problem gets significantly more complex when a system holds multiple boot environments that might each have multiple sub-mounts.

Keeping the entire operating system contents on a single filesystem avoids these issues entirely. For the purposes of rolling back snapshots or cloning one boot environment to another, ZFSBootMenu expects that the environment consists of exactly one filesystem, so that a snapshot of the filesystem always presents a consistent view of system state, and rollbacks or clones behave as expected without the need for manual correlation. If you wish to maintain more complicated setups, you can always manually manage snapshot rollbacks or clone operations from the recovery shell that ZFSBootMenu provides.

Note that "coupled system state" does not include "user data" that should generally survive things like snapshot rollbacks. Recovering from a bad system update is generally not expected to discard user email or recent database transactions. For this reason, directories like `/home`, `/var/mail` and others that hold important data *not* managed by the system **should** reside on separate filesystems.

48.2 ZFSBootMenu Build Containers

48.2.1 A Simple Host-Specific Container Build

`zbm-builder.sh` mounts a build directory (by default, the current working directory) into the container to provide a path to inject custom configuration into the container. If the system will manage ZFSBootMenu images exclusively via a build container, an obvious location for the build directory is `/etc/zfsbootmenu`. Start by creating this directory and populating a simple `config.yaml` for container builds:

```
mkdir -p /etc/zfsbootmenu

cat > /etc/zfsbootmenu/config.yaml <<EOF
Global:
  InitCPIO: true
Components:
  Enabled: false
EFI:
  Enabled: true
  Versions: false
Kernel:
  Prefix: zfsbootmenu
  CommandLine: zfsbootmenu ro quiet loglevel=4 nomodeset
EOF

curl -L -O /etc/zfsbootmenu/zbm-builder.sh https://raw.githubusercontent.com/zbm-dev/
↪zfsbootmenu/master/zbm-builder.sh
chmod 755 /etc/zfsbootmenu/zbm-builder.sh
```

In this configuration, `mkinitcpio` will be used instead of `dracut`. Component generation is disabled, so `generate-zbm` will produce only a UEFI bundle. That bundle has numeric versioning disabled, so `generate-zbm` will produce an unversioned `zfsbootmenu.EFI`; if the generator detects an existing `zfsbootmenu.EFI` in the output

directory, it will make a single backup of that file as `zfsbootmenu-backup.EFI` before overwriting it. A simple kernel command-line is specified and may be overridden as necessary.

For some systems, it is necessary to tear down USB devices before ZFSBootMenu launches a boot environment. Even when this is not needed, it is generally harmless. The ZFSBootMenu repository offers a teardown hook for this purpose, and it is possible to instruct `mkinitcpio` to include this teardown hook straight from the version of ZFSBootMenu inside the container:

```
mkdir -p /etc/zfsbootmenu/mkinitcpio.conf.d
echo "zfsbootmenu_teardown=( /zbm/contrib/xhci-teardown.sh )" \
    > /etc/zfsbootmenu/mkinitcpio.conf.d/teardown.conf
```

The default `mkinitcpio.conf` in the container, which should generally not be overridden, will source all files in `/etc/zfsbootmenu/mkinitcpio.conf.d`.

Custom Font

On high-resolution screens, the Linux kernel does not always do a good job choosing a console font. A nice font can be explicitly specified in the ZFSBootMenu configuration for `mkinitcpio`. The container entrypoint must be told to install the desired font and the `mkinitcpio` configuration should include the necessary module and executable to set the font:

```
echo "BUILD_ARGS+=( -p terminus-font )" >> /etc/zfsbootmenu/zbm-builder.conf

cat > /etc/zfsbootmenu/mkinitcpio.conf.d/consolefont.conf <<EOF
BINARIES+=(setfont)
HOOKS+=(consolefont)
EOF
```

This approach uses the configuration file capability of `zbm-builder.sh` to specify build options without requiring that they be included on the command line.

As configured, `mkinitcpio` will not see a configured console font and will omit the font from generated images. To make `mkinitcpio` aware of the desired font, it must be specified in `/etc/rc.conf` within the container. The "terraform" capabilities of the container entrypoint can be used to accomplish this:

```
mkdir -p /etc/zfsbootmenu/rc.d

cat > /etc/zfsbootmenu/rc.d/consolefont <<EOF
#!/bin/sh
sed -e '/FONT=/a FONT="ter-132n"' -i /etc/rc.conf
EOF

chmod 755 /etc/zfsbootmenu/rc.d/consolefont
```

When the container entrypoint finds an `rc.d` subdirectory in the build root, it will run each executable file therein before generating a ZFSBootMenu image. If any of these executable should fail, image generation is aborted.

Host-Specific Files

By default, `zbm-builder.sh` will copy the file `/etc/hostid` from the host to the build directory so that the hostid of the generated ZFSBootMenu image will match that of your host. This is often desirable for customized builds, but it would be undesirable for copies of these files in `/etc/zfsbootmenu` to fall out of synchronization with the host versions. To avoid this issue, tell `zbm-builder.sh` to remove any copies in `/etc/zfsbootmenu` before determining whether the host versions should be copied in for image creation:

```
echo "REMOVE_HOST_FILES=yes" >> /etc/zfsbootmenu/zbm-builder.conf
```

If you would rather not see those files at all, it is possible to instruct `generate-zbm` to remove them after they are used. Edit the configuration at `/etc/zfsbootmenu/config.yaml` and add the following key:

```
Global:
  PostHooksDir: /build/cleanup.d
```

Alternatively, tell the build container to add this option dynamically:

```
echo "BUILD_ARGS+=( -e '.Global.PostHooksDir=\"/build/cleanup.d\"' )" \
    >> /etc/zfsbootmenu/zbm-builder.conf
```

Next, create a post-generation hook to remove the files:

```
mkdir -p /etc/zfsbootmenu/cleanup.d

cat > /etc/zfsbootmenu/cleanup.d/hostfiles <<EOF
#!/bin/sh
rm -f /build/zpool.cache /build/hostid
EOF

chmod 755 /etc/zfsbootmenu/cleanup.d/hostfiles
```

The Output Directory

At this point, it should be possible to generate images by running

```
cd /etc/zfsbootmenu && ./zbm-builder.sh
```

However, these images will reside in `/etc/zfsbootmenu/build` and will require manual management. A better alternative is to let `generate-zbm` manage the ZFSBootMenu output directory directly. Assuming that ZFSBootMenu images should be installed in `/boot/efi/EFI/zfsbootmenu`, tell `zbm-builder.sh` to mount the directory inside the container, and tell the container that it should write its images to the mounted directory:

```
cat >> /etc/zfsbootmenu/zbm-builder.conf <<EOF
RUNTIME_ARGS+=( -v /boot/efi/EFI/zfsbootmenu:/output )
BUILD_ARGS+=( -o /output )
EOF
```

Now, running

```
cd /etc/zfsbootmenu && ./zbm-builder.sh
```

should create images directly in `/boot/efi/EFI/zfsbootmenu` and create a backup of any existing `zfsbootmenu.EFI`.

Networking in Rootfull Containers

Manipulating files in `/etc/zfsbootmenu` and `/boot/efi/EFI/zfsbootmenu` may require root privileges, which means that `zbm-builder.sh` and the build container will need to run as root. In some configurations, `podman` may not provide working networking for rootfull containers by default. A simple fix is to allow the containers to use the host network stack, which can be accomplished by running

```
echo "RUTNIME_ARGS+=( --net=host )" >> /etc/zfsbootmenu/zbm-builder.conf
```

Adding Remote Access Capabilities

The process for including `dropbear` for remote access to container-built ZFSBootMenu images is largely the same as the *process for host-built images*, but care must be taken to ensure that all necessary components are available within the build directory.

- The *core configuration changes* should be **ignored**. They are unnecessary with the container configuration described above.
- The *basic network access* and *dropbear* instructions are generally applicable, except **no changes should be made to** `/etc/zfsbootmenu/mkinitcpio.conf` and **all references to paths in** `/etc/dropbear` **should be replaced with corresponding references to paths in** `/etc/zfsbootmenu/dropbear`.

Specific alterations are noted below.

Configuring Basic Network Access

Commands to fetch and unpack the `mkinitcpio-rclocal` module and create an `/etc/zfsbootmenu/initcpio/rc.local` script still apply as described to containerized builds. Subsequent `sed` and `echo` commands that write to `/etc/zfsbootmenu/mkinitcpio.conf` should be ignored because this file should not exist. Instead, create a configuration snippet that will add network configuration to the ZFSBootMenu image:

```
cat > /etc/zfsbootmenu/mkinitcpio.conf.d/network.conf <<EOF
BINARIES+=(ip dhclient dhclient-script)
HOOKS+=(rclocal)
rclocal_hook="/build/initcpio/rc.local"
EOF
```

Note: If a static IP address will be configured, it is acceptable to leave `dhclient` and `dhclient-script` out of the `BINARIES` array.

Next, edit `/etc/zfsbootmenu/config.yaml` to add a hook directory configuration telling *mkinitcpio* where to find custom modules:

```
General:
  InitCPIOHookDirs:
    - /build/initcpio
    - /usr/lib/initcpio
```

Configuring Dropbear

The commands to fetch and unpack the `mkinitcpio-dropbear` module still apply to containerized builds. Instead of adding `dropbear` to the non-existent configuration `/etc/zfsbootmenu/mkinitcpio.conf`, create a snippet:

```
cat > /etc/zfsbootmenu/mkinitcpio.conf.d/dropbear.conf <<EOF
HOOKS+=(dropbear)
EOF
```

Rather than creating keys (and optional configuration) in `/etc/dropbear`, create the keys and configuration in `/etc/zfsbootmenu/dropbear`:

```
mkdir -p /etc/zfsbootmenu/dropbear

## Not strictly required; see note below
for keytype in rsa ecdsa ed25519; do
    dropbearkey -t "${keytype}" -f "/etc/dropbear/dropbear_${keytype}_host_key"
done

## If desired
echo 'dropbear_listen=2222' > /etc/zfsbootmenu/dropbear/dropbear.conf
```

Note: Generating keys is not strictly necessary and can be skipped if `dropbearkey` is not available on the host. The build container will generally lack SSH host keys, so the `mkinitcpio-dropbear` module will default to creating new, random keys in the build directory. These keys will persist for subsequent use.

The file `/etc/zfsbootmenu/dropbear/root_key` is required to provide a list of authorized keys in the ZFSBootMenu image. Unlike with host builds, this may not be a symlink to a user's `authorized_keys` file because that path will be unavailble in the container. Instead, simply copy a desired `authorized_keys` file to `/etc/zfsbootmenu/dropbear/root_key`. Alternatively, dynamism can be preserved by relying on bind-mounting a specific `authorized_keys` file into the build container:

```
echo "RUNTIME_ARGS+=( -v /home/${dropbear_user}/.ssh/authorized_keys:/authorized_keys:ro␣
↪) >> /etc/zfsbootmenu/zbm-builder.conf
ln -s /authorized_keys /etc/zfsbootmenu/dropbear/root_key
```

Replace `${dropbear_user}` with the desired user whose `authorized_keys` file should govern access to ZFSBootMenu.

Make sure that the build container installs the packages necessary to provide `dropbear`:

```
echo "BUILD_ARGS+=( -p dropbear -p psmisc )" >> /etc/zfsbootmenu/zbm-builder.conf
```

Finally, add a "terraform" script to link the expected `/etc/dropbear` directory to that in the build directory:

```
cat > /etc/zfsbootmenu/rc.d/dropbear <<EOF
#!/bin/sh

[ -d /build/dropbear ] || exit 0

if [ -d /etc/dropbear ] && [ ! -L /etc/dropbear ]; then
    if ! rmdir /etc/dropbear; then
        echo "ERROR: failed to remove existing /etc/dropbear directory"
```

(continues on next page)

(continued from previous page)

```
        exit 1
    fi
fi

if ! ln -Tsf /build/dropbear /etc/dropbear; then
    echo "ERROR: failed to make /etc/dropbear symlink"
    exit 1
fi
EOF

chmod 755 /etc/zfsbootmenu/rc.d/dropbear
```

Contents

48.2.2 Introduction

Official ZFSBootMenu release assets are built within OCI containers based on the zbm-builder image. The image is built atop Void Linux and provides a predictable environment without the need to install ZFSBootMenu or its dependencies on the host system. While ZFSBootMenu is officially packaged for Void Linux and is guaranteed to work well with the tools provided therein, the experience is not always as smooth for users of other distributions. System packages for ZFSBootMenu or its requirements may be missing entirely. Tooling may be outdated and missing features that ZFSBootMenu uses to provide an enhanced user experience. (Where possible, ZFSBootMenu will test for features and work around their absence.) The `zbm-builder` container image provides a means to work around the limitations of particular distributions and provides all users with first-class ZFSBootMenu support.

The `zbm-builder.sh` script provides a front-end for integrating custom ZFSBootMenu configurations into the build container without the complexity of directly controlling the container runtime.

Users wishing to build custom ZFSBootMenu images should be familiar with the core concepts of ZFSBootMenu as outlined in the project README. For those interested, the container README provides more details on the operation of the ZFSBootMenu build container. However, the `zbm-builder.sh` build helper provides a front-end for integrating custom ZFSBootMenu configurations into the build container and abstracts away many of the complex details discussed in that document.

48.2.3 Dependencies

To build ZFSBootMenu images from a build container, either podman or docker is required. The development team prefers `podman`, but `docker` may generally be substituted without consequence.

If a custom build container is desired, buildah and `podman` are generally required. A Dockerfile is provided for convenience, but feature parity with the `buildah` script is not guaranteed. The container README provides more information about the process of creating a custom build image.

Podman

Install `podman` and `buildah` (if desired) using the package manager in your distribution:

- On Void, `xbps-install podman buildah`
- On Arch or its derivatives, `pacman -S podman buildah`
- On Debian or its derivatives, `apt-get install podman buildah`

It is possible to configure `podman` for rootless container deployment. Consult the tutorial for details.

Docker

Install `docker` using the package manager in your distribution:

- On Void, `xbps-install docker`
- On Arch or its derivatives, `pacman -S docker`
- On Debian or its derivatives, `apt-get install docker`

Non-root users that should be permitted to work with Docker images and containers should belong to the `docker` groups. For example:

```
usermod -a -G docker zbmuser
```

will add `zbmuser` to the `docker` group on systems that provide the `usermod` program.

48.2.4 The Build Container and Its Helper

The `zbm-builder` container is based on a Void Linux image that uses an LTS kernel and a relatively recent version of ZFS. When run, the container entrypoint will:

1. Fetch a specified or default version of the ZFSBootMenu source repository (or use a local copy that is bind-mounted into the container);
2. Perform an "installation" of this repository into the container instance to ensure that ZFSBootMenu is usable within;
3. Optionally run some scripts to customize the container instance;
4. Merge default and build-specific ZFSBootMenu configurations; and
5. Produce a ZFSBootMenu image.

To facilitate interaction with the host, the container should be run with a build directory (along with an output directory, if it is not already a child of the build directory) bind-mounted into the container.

Note: Advanced users may wish to build images from a local copy of the ZFSBootMenu source tree. To make this possible, either fetch and unpack a source tarball or clone the git repository locally. The local repository should be bind-mounted to the `/zbm` directory within the container.

The `zbm-builder.sh` helper script requires nothing more than functional installations of `bash` and one of `podman` or `docker`. Simply download a copy of the script to a convenient directory. The helper coordinates volume mounts necessary to read configurations from and write output to the host and ensures that the system's hostid file is passed through. The script also supports a simple configuration file that allows options to be recorded for repeated use.

48.2.5 Building a ZFSBootMenu Image

To build a default image, invoke `zbm-builder.sh` with no arguments. For example, from the directory that contains the script, run `./zbm-builder.sh` to produce a default kernel/initramfs pair in the `./build` subdirectory.

The default behavior of `zbm-builder.sh` will:

1. Pull the default builder image, `ghcr.io/zbm-dev/zbm-builder:latest`.
2. If `./hostid` does not exist, copy `/etc/hostid` (if it exists) to `./hostid`.
3. Spawn an ephemeral container from the builder image and run its build process:

1. Bind-mount the working directory into the container to expose local configurations to the builder
2. If `./config.yaml` exists, inform the builder to use that custom configuration instead of the default
3. Run the internal build script to produce output in the `./build` subdirectory

Custom ZFSBootMenu Hooks

ZFSBootMenu supports *custom hooks* in three stages:

1. `early_setup` hooks run after the `zfs` kernel driver has been loaded, but before ZFSBootMenu attempts to import any pools.
2. `setup` hooks run after pools are imported, right before ZFSBootMenu will either boot a default environment or present a menu.
3. `teardown` hooks run immediately before ZFSBootMenu will `kexec` the kernel for the selected environment.

When `zbm-builder.sh` runs, it will identify custom hooks as executable files in the respective subdirectories of its build directory:

1. `hooks.early_setup.d`
2. `hooks.setup.d`
3. `hooks.teardown.d`

For each hook directory that contains at least one executable file, `zbm-builder.sh` will write custom configuration snippets for `dracut` and `mkinitcpio` that will include these files in the output images.

Fully Customizing Images

The entrypoint for the ZFSBootMenu implements a tiered configuration approach that allows default configurations to be augmented or replaced with local configurations in the build directory. A custom `config.yaml` may be provided in the working directory to override the default ZFSBootMenu configuration; configuration snippets for `dracut` or `mkinitcpio` can be placed in the `dracut.conf.d` and `mkinitcpio.conf.d` subdirectories, respectively. For `mkinitcpio` configurations, a complete `mkinitcpio.conf` can be placed in the working directory to override the standard configuration.

Note: The `mkinitcpio` configuration prepared by `zbm-builder.sh` may include custom snippets installed in a `mkinitcpio.d` subdirectory of the build directory. The default mkinitcpio configuration includes a loop to source these snippets. Should you prefer to overide the default `mkinitcpio.conf` in your build, any files in the `mkinitcpio.d` subdirectory will need to be sourced within your custom configuration. In general, it is better to leave the default `mkinitcpio.conf` and store all custom configurations in the `mkinitcpio.d` subdirectory.

The build container runs its build script from the working directory on the host. In general, relative paths in custom configuration files are generally acceptable and refer to locations relative to the build directory. If absolute paths are preferred or required for some configurations, note that the build directory will be mounted as `/build` in the container.

The internal build script **always** overrides the output paths for ZFSBootMenu components and UEFI executables to ensure that the images will reside in a specified output directory (or, by default, a `build` subdirectory of build directory) upon completion. Relative paths are primarily useful for specifying local `dracut` or `mkinitcpio` configuration paths.

More advanced users may wish to alter the build process itself. Some control over the build process is exposed through command-line options that are described in the output of `zbm-builder.sh -h`.

Before adjusting these command-line options, seek a thorough understanding of the image build process and the command sequence of `zbm-builder.sh` itself.

48.3 Native Encryption

ZFSBootMenu can import pools or filesystems with native encryption enabled. If your boot environments are not encrypted but, for example, `/home` is, you will not receive a decryption prompt during boot. To ensure that you can decrypt your pool to load the kernel and initramfs, you'll need to you have the filesystem parameters configured correctly.

It's critical that `keyformat` is set to `passphrase`, otherwise you'll be unable to enter the correct value in the bootloader. OpenZFS currently supports only one key, but in a way which ZFSBootMenu can exploit: if you configure the `keylocation` value to a file on disk, put your passphrase in that, and then include that file into the FINAL initramfs (the one in the `/boot` subdirectory of your encrypted root), you won't receive a second password prompt on boot. When ZFSBootMenu attempts to unlock root filesystems, it will override any `file://` URI it encounters as a `keylocation` if that file is not accessible from within the bootloader image. This allows ZFSBootMenu to prompt for passphrases when necessary.

Note: **Never** place encryption keys inside a custom ZFSBootMenu image! The ZFSBootMenu image will typically be installed on an unencrypted partition with minimal or no access restrictions. If an encryption key is placed in such a location, anybody with access to the system will be able to read your passphrase.

As an example, Consider a filesystem layout such as:

```
zfs get all zroot | egrep '(encryption|keylocation|keyformat)'
zroot  encryption            aes-256-gcm               -
zroot  keylocation           file:///etc/zfs/zroot.key  local
```

(continues on next page)

(continued from previous page)

```
zroot  keyformat            passphrase              -
zroot  encryptionroot       zroot                   -
```

On systems that use `dracut`, the key for `zroot` can be added to initramfs images by running:

```
echo 'install_items+=" /etc/zfs/zroot.key "' > /etc/dracut.conf.d/zfs-keys.conf
```

For `mkinitcpio`, add the key to the `FILES` array in `mkinitcpio.conf`:

```
echo 'FILES+=(/etc/zfs/zroot.key)' >> /etc/mkinitcpio.conf
```

Note: When adding encryption keys to initramfs images, **always ensure** that the resulting images are not readable by any user other than root. Recent versions of `dracut` and `mkinitcpio` ensure this by default with umask of `0077`. Users with read access to your initramfs image will be able to read your ZFS key file even if it has mode `000` in the image; always confirm for your self that the initramfs is protected!

For convenience, ZFSBootMenu recognizes the ZFS property `org.zfsbootmenu:keysource` as the name of a filesystem that should be searched for ZFS key files. When a boot environment specifies a `file://` URI as its `keylocation`, ZFSBootMenu will attempt to mount a filesystem indicated by the `org.zfsbootmenu:keysource` property (if it exists) and search for the named `keylocation` therein. If found, ZFSBootMenu will copy the key into a cache within the in-memory root filesystem so that subsequent operations that require reloading the key (for example, changing the default boot environment or cloning a snapshot) will not prompt the user for passphrases.

When searching for a `keylocation` relative to the filesystem named by `org.zfsbootmenu:keysource`, ZFSBootMenu will first try to strip the `mountpoint` of the keysource filesystem from any `keylocation` URI that references the keys to map the `keylocation` that would be observed on a running system to the proper location in the keysource. For example, if the running system is set up so that `zroot` is the `encryptionroot` for all filesystems on a pool, running the commands:

```
zfs create -o mountpoint=/etc/zfs/keys zroot/keystore
echo "MySecretPassphrase" > /etc/zfs/keys/zroot.key
chmod 000 /etc/zfs/keys/zroot.key
zfs set keylocation=file:///etc/zfs/keys/zroot.key zroot
zfs set org.zfsbootmenu:keysource=zroot/keystore zroot
echo install_optional_items+=" /etc/zfs/keys/zroot.key " >> /etc/dracut.conf.d/zol.conf
```

will cause ZFSBootMenu to attempt to cache the key `file:///etc/zfs/keys/zroot.key` from `zroot/keystore` when unlocking the `zroot` pool. Because `zroot/keystore` specifies `mountpoint=/etc/zfs/keys`, ZFSBootMenu will first try to strip `/etc/zfs/keys` from the `keylocation` URI, looking for the file `zroot.key` at the root of the filesystem `zroot/keystore`. If this fails, ZFSBootMenu will fall back to the full path, looking for `etc/zfs/keys/zroot.key` within the keysource filesystem. If either location is found, ZFSBootMenu will retain a cache of the key should it be needed to unlock the pool again.

48.4 UEFI Booting

Although ZFSBootMenu images can be booted on legacy BIOS systems or (on other platforms) alternative firmware, ZFSBootMenu integrates nicely with modern UEFI systems. ZFSBootMenu builds a custom initramfs image around a standard Linux kernel. Most distributions compile the Linux kernel with an EFI stub loader; the ZFSBootMenu kernel and initramfs pair can therefore be booted directly by most UEFI implementations or by EFI boot managers like rEFInd or gummiboot (systemd-boot).

When generating ZFSBootMenu images from a local host, it is possible to edit `/etc/zfsbootmenu/config.yaml` to copy the ZFSBootMenu kernel and initramfs directly to your EFI system partition. Suppose that the directory listing for your current `/boot` looks like:

```
# ls /boot
config-5.3.18_1
config-5.4.6_1
efi
initramfs-5.3.18_1.img
initramfs-5.4.6_1.img
System.map-5.3.18_1
System.map-5.4.6_1
vmlinuz-5.3.18_1
vmlinuz-5.4.6_1
```

Typically, EFI system partitions (ESP) are mounted at `/boot/efi`, as is shown above. An ESP may contain a number of sub-directories, including an `EFI` directory that often contains multiple independent EFI executables. In this example layout, `/boot/efi/EFI/zbm` may hold ZFSBootMenu kernels and initramfs images. After setting the `ImageDir` property of the `Components` section of `/etc/zfsbootmenu/config.yaml` to `/boot/efi/EFI/zbm`, running `generate-zbm` will cause ZFSBootMenu kernel and initramfs pairs to be installed in the desired location:

```
# lsblk -f /dev/sda
NAME   FSTYPE LABEL UUID                                 FSAVAIL FSUSE% MOUNTPOINT
sdg
├─sda1 vfat         AFC2-35EE                               7.9G     1% /boot/efi
└─sda2 swap         412401b6-4aec-4452-a6bd-6fc20fbdc2a5                [SWAP]

# ls /boot/efi/EFI/zbm/
initramfs-1.12.0_1.img
initramfs-1.12.0_2.img
vmlinuz-1.12.0_1
vmlinuz-1.12.0_2
```

After the kernel and initramfs pairs are made available on the ESP, you'll need a way to boot them on your system. This can be done directly via efibootmgr or via a third-party boot manager like rEFInd.

48.4.1 efibootmgr

```
efibootmgr --disk /dev/sda \
  --part 1 \
  --create \
  --label "ZFSBootMenu" \
  --loader '\EFI\zbm\vmlinuz-1.12.0_2' \
  --unicode 'zbm.prefer=zroot ro initrd=\EFI\zbm\initramfs-1.12.0_2.img quiet' \
  --verbose
```

Take note to adjust the arguments to `--disk` and `--part`, the path to the kernel in `--loader`, and the initramfs path (`initrd=`) and pool preference (`zbm.prefer=`) to match your system configuration.

Each time ZFSBootMenu is updated, a new EFI entry will need to be manually added, unless you disable versioning in the ZFSBootMenu configuration.

48.4.2 rEFInd

`rEFInd` is considerably easier to install and manage. Refer to your distribution's packages for installation. Once rEFInd has been installed, you can create `refind_linux.conf` in the directory holding the ZFSBootMenu files (`/boot/efi/EFI/zbm` in our example):

```
"Boot default"  "zbm.prefer=zroot ro quiet loglevel=0 zbm.skip"
"Boot to menu"  "zbm.prefer=zroot ro quiet loglevel=0 zbm.show"
```

As with the efibootmgr section, the `zbm.prefer=` option needs to be configured to match your environment.

This file will configure `rEFInd` to create two entries for each kernel and initramfs pair it finds. The first will directly boot into the environment set via the `bootfs` pool property. The second will force ZFSBootMenu to display its interactive user interface and allow you to boot alternate environments, kernels and snapshots.

48.4.3 Avoiding an Intermediate Boot Manager

On most UEFI systems, booting ZFSBootMenu without the use of an intermediate boot manager like rEFInd is possible. Linux kernels typically include an EFI stub and can be invoked as UEFI executables directly by the firmware. Unfortunately, while some UEFI implementations allow passing of command-line arguments to the UEFI kernel, others (from Dell, for example) seem to ignore all configured command-line arguments, making it impossible to specify needed options (such as the path to the ZFSBootMenu initramfs). Even those implementations that do respect configured arguments may provide no firmware interface to alter these arguments, which means booting a backup ZFSBootMenu image may not be possible if it wasn't configured in advance from a Linux installation.

These limitations are easily avoided if ZFSBootMenu is packaged as a *bundled UEFI executable* that encapsulates the Linux kernel, ZFSBootMenu initramfs and all needed command-line arguments. Dracut facilitates the creation of a bundled UEFI executable, and the `generate-zbm` script exposes this capability.

Creation of a Bundled UEFI Executable

The `EFI` section of the ZFSBootMenu *config.yaml* governs the creation of bundled UEFI executables. The default configuration disables this option; to enable it, set `EFI.Enabled: true`:

```
EFI:
  Enabled: true
```

The remaining keys in the `EFI` section allow control over where and how UEFI bundles are created:

- `ImageDir` is the location where the bundle will be written, and should generally be a subdirectory of the `EFI` subdirectory of your EFI system partition. The default, `/boot/efi/EFI/void`, is fine if the ESP is mounted at `/boot/efi` (and you are either running Void Linux or don't care if the directory name matches your distribution name).
- `Versions` controls whether UEFI bundles include a version and revision number in their name and, if so, how many prior versioned executables are retained. Because the firmware is not automatically reconfigured to boot the latest version after runs of `generate-zbm`, it is probably best to disabling `Versions` by setting its value to `false` or `0`. See the *description of this key in manual page* for more details about its behavior. Even when versioning is disabled, `generate-zbm` still makes a backup of your existing boot image by replacing its `.EFI` extension with `-backup.EFI` to provide a fallback.
- `Stub` specifies the location of the UEFI stub loader required when creating a bundled executable. Both `gummiboot` and its descendant `systemd-boot` provide stub loaders; `gummiboot`, for example, tends to store the loader at `/usr/lib/gummiboot/linuxx64.efi.stub`. If this key is omitted (as it is by default), `dracut` will attempt to find either the `systemd-boot` or `gummiboot` version at their expected locations. This key is useful when automatic detection fails.

In addition, two options in the `Kernel` section of the configuration file are used during bundle creation:

- `Prefix` provides the base name for the output bundle file. If this is omitted, the base name will be derived from the name of the kernel used to create the image; for example, the kernel `/boot/vmlinuz-<version>` will produce a bundle called `vmlinuz.EFI` in the configured `ImageDir`, while the kernel `/boot/vmlinuz-lts-<version>` will produce a bundle called `vmlinuz-lts.EFI`.
- `CommandLine` provides the command-line arguments that will be encoded in the bundle and passed to the kernel during boot. The `dracut` configuration option `kernel_cmdline` also provides a mechanism for encoding the kernel command-line; if the ZFSBootMenu configuration specifies `Kernel.CommandLine` and the `dracut` configuration for ZFSBootMenu specifies `kernel_cmdline`, the two values will be concatenated.

After adjusting the configuration options as desired, run `generate-zbm` and a bundled UEFI executable will be created in `EFI.ImageDir`.

Booting the Bundled Executable

The efibootmgr utility provides a means to configure your firmware to boot the bundled executable. For example:

```
efibootmgr -c -d /dev/sda -p 1 -L "ZFSBootMenu" -l \\EFI\\VOID\\VMLINUZ.EFI
```

will create a new entry that will boot the executable written to `/boot/efi/EFI/void/vmlinuz.EFI` if your EFI system partition is `/dev/sda1` and is mounted at `/boot/efi`. (Remember that the EFI system partition should be a FAT volume, so the path separators are backslashes and paths should be case-insensitive.) For good measure, create an alternative entry that points at the backup image:

```
efibootmgr -c -d /dev/sda -p 1 -L "ZFSBootMenu (Backup)" -l \\EFI\\VOID\\VMLINUZ-BACKUP.
↪EFI
```

The firmware should provide some means to select between these alternatives.

It is also generally possible to configure the boot sequence from your firmware setup interface. Simply find and select the path to the bundled EFI executable from this interface.

48.5 Remote Access to ZFSBootMenu

Contents

Having SSH access to ZFSBootMenu can be critical because it allows some measure of recovery over a remote connection. If your boot environments reside in encrypted filesystems, SSH access is necessary if you ever intend to reboot a machine when you are not physically present. Because ZFSBootMenu supports Dracut and mkinitcpio, any mechanism that can provide remote access to a Dracut or mkinitcpio initramfs will work.

48.5.1 Dracut

The dracut-crypt-ssh provides a straightforward approach to configuring and launching an SSH server in Dracut images. The module is packaged in Void and does not rely on `systemd` within the initramfs. If you run a distribution that does not package `dracut-crypt-ssh`, you will need to track down its dependencies. The `dracut-network` module and `dropbear` are required to provide network access and an SSH server, respectively; other prerequisites are probably already installed on your system.

Simplified Installation Instructions

The `dracut-crypt-ssh` package comes with a few helper utilities in the `module/60crypt-ssh/helper` directory that are designed to simplify providing passwords and snooping console output so that you can interact with unlock processes that are already running in the initramfs. These components are not required for ZFSBootMenu and do not provide a lot of value. If you have no problems installing the package as intended, it is OK to leave the helpers installed. If your distribution has trouble compiling the helpers, just copy the contents of the `60crypt-ssh` directory, less the `helper` directory and `Makefile`, to the modules directory for Dracut. This will most likely be `/usr/lib/dracut/modules.d/60crypt-ssh`.

If you do not install the contents of `helper`, you may wish to edit the `module-setup.sh` script provided by the package to remove references to installing the helper. At the time of writing, these references consist of the last four lines (five, if you count the harmless comment) of the `install()` functioned. Removing these lines should not be critical, as Dracut should happily continue the initramfs creation process even if those installation commands fail.

If you use Dracut to produce the initramfs images in your boot environment, you may wish to disable the `crypt-ssh` module in those images. Just add

```
omit_dracutmodules+=" crypt-ssh "
```

to a configuration file in `/etc/dracut.conf.d`. The configuration file must have a `.conf` extension to be recognized; see dracut.conf(5) for more information.

Configuring Dropbear in ZFSBootMenu

By default, `dracut-crypt-ssh` will generate random host keys for your ZFSBootMenu initramfs. This is undesirable because SSH will complain about unknown keys every time you reboot. If you wish, you can configure the module to copy your regular host keys into the image. However, there are two problems with this:

1. The ZFSBootMenu image will generally be installed on a filesystem with no access permissions, allowing anybody to read your private host keys; and
2. The `dropbearconvert` program may be incapable of converting modern OpenSSH host keys into the required dropbear format.

To create dedicated host keys in the proper format, decide on a location, for example `/etc/dropbear`, and create the new keys:

```
mkdir -p /etc/dropbear
ssh-keygen -t rsa -m PEM -f /etc/dropbear/ssh_host_rsa_key
ssh-keygen -t ecdsa -m PEM -f /etc/dropbear/ssh_host_ecdsa_key
```

The module expects to install RSA and ECDSA keys, so both types are created here.

Note: When prompted for a passphrase when creating each host key, leave it blank. A non-empty password will prevent dropbear from reading a key.

To inform `dracut-network` that it must bring up a network interface, pass the kernel command-line parameters `ip=dhcp` and `rd.neednet=1` to your ZFSBootMenu image. If you use another boot loader to start ZFSBootMenu, *e.g.* rEFInd or syslinux, this can be accomplished by configuring that loader. However, it may be more convenient to add these parameters directly to the ZFSBootMenu image:

```
mkdir -p /etc/cmdline.d
echo "ip=dhcp rd.neednet=1" > /etc/cmdline.d/dracut-network.conf
```

It is possible to specify a static IP configuration by replacing `dhcp` with a properly formatted configuration string. Consult the dracut documentation for details about static IP configuration.

There are methods besides writing to `/etc/cmdline.d` or configuring another boot loader to specify kernel command-line arguments that will configure networking in Dracut. However, Dracut uses the `/etc/cmdline.d` directory to store "fake" arguments, which it processes directly rather than handing to the kernel. In my tests, using other methods (like adding these arguments to the `kernel_cmdline` Dracut option for a UEFI bundle) can cause the `ip=dhcp` argument to appear more than once on the kernel command-line, which may cause `dracut-network` to fail catastrophically and refuse to boot. Writing a configuration file in `/etc/cmdline.d` is a reliable way to ensure that `ip=dhcp` appears exactly once to `dracut-network`.

With critical pieces in place, ZFSBootMenu can be configured to bundle `dracut-crypt-ssh` in its images. Create the Dracut configuration file `/etc/zfsbootmenu/dracut.conf.d/dropbear.conf` with the following contents:

```
# Enable dropbear ssh server and pull in network configuration args
add_dracutmodules+=" crypt-ssh "
install_optional_items+=" /etc/cmdline.d/dracut-network.conf "
# Copy system keys for consistent access
dropbear_rsa_key=/etc/dropbear/ssh_host_rsa_key
dropbear_ecdsa_key=/etc/dropbear/ssh_host_ecdsa_key
# User zbmuser is the authorized unlocker here
dropbear_acl=/home/zbmuser/.ssh/authorized_keys
```

The last line is optional and assumes the user `zbmuser` should provide an `authorized_keys` file that will determine remote access to the ZFSBootMenu image. The `dracut-crypt-ssh` module does not allow for password authentication over SSH; instead, key-based authentication is forced. By default, the list of authorized keys is taken from `/root/.ssh/authorized_keys` on the host. If you would prefer to use the `authorized_keys` file from another user on your system, copy the above example and replace `zbmuser` with the name of the user whose `authorized_keys` you wish to include.

Note: The default configuration will start dropbear on TCP port 222. This can be overridden with the `dropbear_port` configuration option. Generally, you do not want the server listening on the default port 22. Clients that expect to find your normal host keys when connecting to an SSH server on port 22 will refuse to connect when they find different keys provided by dropbear.

Unless you've taken steps not described here, the network-enabled ZFSBootMenu image will not advertise itself via dynamic DNS or mDNS. You will need to know the IP address of the ZFSBootMenu host to connect. Thus, you should either configure a static IP address in `/etc/cmdline.d/dracut-network.conf` or configure your DHCP server to reserve a known address for the MAC address of the network interface you configure for `dracut-crypt-ssh`.

48.5.2 mkinitcpio

ZFSBootMenu also supports the mkinitcpio initramfs generator used by Arch Linux.

ZFSBootMenu Configuration Changes

Since version 2.0.0, ZFSBootMenu will install a standard mkinitcpio.conf in the `/etc/zfsbootmenu` configuration directory. This file is generally the same as a standard `mkinitcpio.conf`, except some additional declarations may be added to control aspects of the `zfsbootmenu` mkinitcpio module. The configuration file includes extensive inline documentation in the form of comments; configuration options specific to ZFSBootMenu are also described in the *zfsbootmenu(7)* manual page.

ZFSBootMenu still expects to use dracut by default. To override this behavior and instead use mkinitcpio, edit `/etc/zfsbootmenu/config.yaml` and add the following options:

```
Global:
  InitCPIO: true
  ## NOTE: The following three lines are OPTIONAL
  InitCPIOHookDirs:
    - /etc/zfsbootmenu/initcpio
    - /usr/lib/initcpio
```

Note: In the examples below, a couple of mkinitcpio modules will be installed to `/etc/zfsbootmenu/initcpio` to keep them isolated from system-installed modules. To accommodate this non-standard installation,

`InitCPIOHookDirs` must be defined in `/etc/zfsbootmenu/config.yaml`. Furthermore, because overriding the hook directory causes mkinitcpio to ignore its default module path, the default `/usr/lib/initcpio` must be manually specified. If all hooks are installed in `/usr/lib/initcpio` or `/etc/initcpio`, the ZFSBootMenu configuration does **not** need to specify `InitCPIOHookDirs`.

Without further changes, running `generate-zbm` should now produce a ZBM image based on mkinitcpio rather than dracut, although it will lack networking and remote-access capabilities. (By default, `generate-zbm` instructs mkinitcpio to use the configuration at `/etc/zfsbootmenu/config.yaml`, although this can be changed in the `generate-zbm` configuration file.) For these features, some additional mkinitcpio modules and configuration changes are necessary.

Because further configuration will require additional mkinitcpio modules, and these must be run before the `zfsbootmenu` module in the initramfs, edit `/etc/zfsbootmenu/mkinitcpio.conf` and **remove** any `zfsbootmenu` entry in the `HOOKS` definition. As the standard configuration file notes, the `zfsbootmenu` module is required for ZFSBootMenu to function, but `generate-zbm` will forcefully at this at the end of the module list. Thus, the simplest way to ensure that additions to the `HOOKS` array occur *before* the `zfsbootmenu` module is to omit the latter from the configuration. The standard `HOOKS` line in `/etc/zfsbootmenu/mkinitcpio.conf` should therefore be something like:

```
HOOKS=(base udev autodetect modconf block filesystems keyboard)
```

Basic Network Access

Network access in a mkinitcpio image can be realized in one of several ways. In Arch Linux, for example, the mkinitcpio-nfs-utils package provides a net module that allows the initramfs to parse `ip=` directives from the kernel command line. When a static IP configuration is sufficient, the mkinitcpio-rclocal module allows user scripts to be injected at several points in the initramfs boot process and provides a simple mechanism for configuring a network interface.

When installing mkinitcpio modules that are not provided by a system package manager, it may be preferable to keep them isolated from the ordinary module tree. Because this module will only be required in ZBM images, placing extra modules in `/etc/zfsbootmenu/initcpio` is convenient:

```
curl -L https://github.com/ahesford/mkinitcpio-rclocal/archive/master.tar.gz | tar -zxvf␣
↪- -C /tmp
mkdir -p /etc/zfsbootmenu/initcpio/{install,hooks}
cp /tmp/mkinitcpio-rclocal-master/rclocal_hook /etc/zfsbootmenu/initcpio/hooks/rclocal
cp /tmp/mkinitcpio-rclocal-master/rclocal_install /etc/zfsbootmenu/initcpio/install/
↪rclocal
rm -r /tmp/mkinitcpio-rclocal-master
```

Next, create an `rc.local` script that can be run within the mkinitcpio image to configure the `eth0` interface:

```
cat > /etc/zfsbootmenu/initcpio/rc.local <<RCEOF
#!/bin/sh

# Don't attempt to configure an interface that does not exist
ip link show dev eth0 >/dev/null 2>&1 || exit

# Bring up the interface
ip link set dev eth0 up

# Configure a static address for this host
ip addr add 192.168.1.2/24 brd + dev eth0
```

(continues on next page)

(continued from previous page)

```
ip route add default via 192.168.1.1

# Add some name servers
cat > /etc/resolv.conf <<-EOF
  nameserver 1.1.1.1
  nameserver 8.8.8.8
  EOF
RCEOF
```

Note: If your Ethernet interface is called something other than `eth0` or your static IP configuration is different, adjust the script as needed.

To ensure that the `rclocal` module is installed and run in the ZBM image, either append `rclocal` to the array defined on the `HOOKS` line in `/etc/zfsbootmenu/mkinitcpio.conf` or run

```
sed -e '/HOOKS=/a HOOKS+=(rclocal)' -i /etc/zfsbootmenu/mkinitcpio.conf
```

The `rclocal` module should be told where it can find the `rc.local` script to install and run by running:

```
echo 'rclocal_hook=/etc/zfsbootmenu/initcpio/rc.local' >> /etc/zfsbootmenu/mkinitcpio.
↪conf
```

Finally, make sure to include the `ip` executable in your initramfs image by manually adding `ip` to the `BINARIES` array in `/etc/zfsbootmenu/mkinitcpio.conf` or by running

```
sed -e '/BINARIES=/a BINARIES+=(ip)' -i /etc/zfsbootmenu/mkinitcpio.conf
```

Dropbear

Arch Linux provides a mkinitcpio-dropbear package that provides a straightforward method for installing, configuring and running the dropbear SSH server inside a mkinitcpio image. This package is based on a project of the same name by an Arch Linux developer. A fork of the mkinitcpio-dropbear project contains a few minor improvements in runtime configuration and key management. If these improvements are not needed, using the upstream project is perfectly acceptable.

Once again, the mkinitcpio module must first be downloaded and installed:

```
curl -L https://github.com/ahesford/mkinitcpio-dropbear/archive/master.tar.gz | tar -
↪zxvf - -C /tmp
mkdir -p /etc/zfsbootmenu/initcpio/{install,hooks}
cp /tmp/mkinitcpio-dropbear-master/rclocal_hook /etc/zfsbootmenu/initcpio/hooks/dropbear
cp /tmp/mkinitcpio-dropbear-master/rclocal_install /etc/zfsbootmenu/initcpio/install/
↪dropbear
rm -r /tmp/mkinitcpio-dropbear-master
```

The upstream `dropbear` module will attempt to copy host OpenSSH keys into `/etc/dropbear` if possible; otherwise, it will generate random host keys. Both options are undesirable. Copying host keys will leave these protected files directly accessible to anybody able to read a ZFSBootMenu image, which is probably every user on the system. Generating unique keys with each run inhibits your ability to detect interlopers when you connect to your bootloader via SSH. My fork will, by default, respect any existing dropbear keys available as `/etc/dropbear/dropbear_*_host_key`. Therefore, make some new host keys for use in your ZFSBootMenu image:

```
mkdir -p /etc/dropbear
for keytype in rsa ecdsa ed25519; do
    dropbearkey -t "${keytype}" -f "/etc/dropbear/dropbear_${keytype}_host_key"
done
```

The module also requires, at `/etc/dropbear/root_key`, a set of authorized SSH keys that will be given access to the `root` account in the image. On a single-user system, it is sufficient to do:

```
ln -s ${HOME}/.ssh/authorized_keys /etc/dropbear/root_key
```

assuming that `${HOME}` points to the home directory of the user who should be given access to ZFSBootMenu.

Finally, enable the `dropbear` module in `/etc/zfsbootmenu/mkinitcpio.conf` by manually appending `dropbear` to the `HOOKS` array, or by running:

```
sed -e '/HOOKS.*rclocal/a HOOKS+=(dropbear)' -i /etc/zfsbootmenu/mkinitcpio.conf
```

Final Steps

With the above configuration complete, running `generate-zbm` should produce a ZFSBootMenu image that contains the necessary components to enable an SSH server in your bootloader. This can be verified with the `lsinitrd` tool provided by dracut or the `lsinitcpio` tool provided by mkinitcpio. (The `lsinitcpio` tool is not able to inspect UEFI bundles, but `lsinitrd` can.) In the file listing, you should see keys in `/etc/dropbear`, the `dropbear` and `ip` executables, and the file `root/.ssh/authorized_keys`.

After rebooting, ZFSBootMenu should configure the network interface, launch an SSH server and accept connections on TCP port 22 by default. If your SSH client complains because it finds ZFSBootMenu keys when it expects to find your normal host keys, you may wish to reconfigure dropbear to listen on a non-standard port. My fork of `mkinitcpio-dropbear` supports this by writing a `dropbear_listen` definition to `/etc/dropbear/dropbear.conf`:

```
echo 'dropbear_listen=2222' > /etc/dropbear/dropbear.conf
```

After writing this file (adjust `2222` to whatever port you prefer), re-run `generate-zbm`, reboot and confirm that dropbear listens where expected.

48.5.3 Accessing ZFSBootMenu Remotely

When you connect to ZFSBootMenu via SSH, you will be presented a simple shell prompt. Launch `zfsbootmenu` to start the menu interface over the remote connection:

```
zfsbootmenu
```

You may then use the menu as if you were connected locally.

Note: recent versions of ZFSBootMenu automatically set the `TERM` environment variable to `linux`. If you are running an older version, your SSH client may have provided a more specific terminal definition that will not be recognized by the restricted environment provided by ZFSBootMenu. Under these circumstances, you may need to run:

```
export TERM=linux
```

from the login shell to ensure that basic terminal functionality works as expected.

If you followed the *Void Linux ZFSBootMenu install guide* and configured rEFInd to launch ZFSBootMenu, you may need to remove the `zbm.skip` argument from the default menu entry if you would like remote access and you have no encrypted boot environments. Otherwise, rEFInd will attempt to bypass the ZFSBootMenu countdown and your default boot environment will be started immediately if possible. In this case, either set `zbm.timeout` to a suitably long delay (*e.g.*, 60 sec) to give yourself time to connect and launch ZFSBootMenu remotely before the automatic boot can proceed, or use `zbm.show` by default to prevent automatic boot and force the local instance to show the interactive menu immediately.

Note: To provide some safety against multi-user conflicts, only one ZFSBootMenu instance is allowed to run at any given time. If you have encrypted boot environments, this will generally not present an issue, because the local instance will always block awaiting passphrase entry before launching the menu instance. Otherwise, the later instance of ZFSBootMenu will wait patiently for the earlier instance to terminate before continuing. If you are *certain* that the currently running instance is not being actively used, you can interrupt the wait loop by pressing `[ESC]` and then run:

```
rm /zfsbootmenu/active
```

to eliminate the indicator of the other running instance. You may then run `zfsbootmenu` again to launch the menu.

48.6 Portable ZFSBootMenu

UEFI makes it easy to deploy ZFSBootMenu without a local installation. Most UEFI systems will search for and run an EFI executable at the path `/EFI/BOOT/BOOTX64.EFI` on a FAT-formatted EFI System Partition located on any disk that the firmware is told to boot. This executable can be a standard ZFSBootMenu release image or a custom, locally generated image. Almost any modern system can be made to launch a ZFSBootMenu instance just by inserting and booting from a minimally configured USB drive.

48.6.1 Procedure

1. On a USB drive, create a GPT header.
2. Create an EFI system partition on the drive. The partition should be at least 100 MB.
   - With gdisk, this is accomplished by setting the parition type to `EF00`.
   - With parted, this is accomplished by setting the `boot` flag on the partition.
3. Format the partition as FAT.
4. Fetch a copy of the ZFSBootMenu release image:

   ```
   curl -LJO https://get.zfsbootmenu.org/efi
   ```

5. Save the resulting download as `EFI/BOOT/BOOTX64.EFI` within the EFI system partiton.
6. Tell your system to boot from the USB drive.

CHAPTER

FORTYNINE

VOID LINUX

49.1 UEFI

Contents

This guide can be used to install Void onto a single disk with with or without ZFS encryption.

It assumes the following:

- Your system uses UEFI to boot
- Your system is x86_64
- You will use `glibc` as your system libc.
- You're mildly comfortable with ZFS, EFI and discovering system facts on your own (`lsblk`, `dmesg`, `gdisk`, ...)

ZFSBootMenu does not require `glibc` and is not restricted to `x86_64`. If you are comfortable installing Void Linux on other architectures or with the `musl` libc, you can adapt the instructions here to your desired configuration.

Download the latest hrmpf, write it to USB drive and boot your system in EFI mode.

Confirm EFI support:

```
# dmesg | grep -i efivars
[    0.301784] Registered efivars operations
```

49.1.1 Configure Live Environment

Source `/etc/os-release`

The file */etc/os-release* defines variables that describe the running distribution. In particular, the *$ID* variable defined within can be used as a short name for the filesystem that will hold this installation.

```
source /etc/os-release
export ID="$ID"
```

Generate `/etc/hostid`

```
zgenhostid -f 0x00bab10c
```

49.1.2 Define disk variables

For convenience and to reduce the likelihood of errors, set environment variables that refer to the devices that will be configured during the setup.

For many users, it is most convenient to place boot files (*i.e.*, ZFSBootMenu and any loader responsible for launching it) on the the same disk that will hold the ZFS pool. However, some users may wish to dedicate an entire disk to the ZFS pool or create a multi-disk pool. A USB flash drive provides a convenient location for the boot partition. Fortunately, this alternative configuration is easily realized by simply defining a few environment variables differently.

Verify your target disk devices with `lsblk`. `/dev/sda` and `/dev/sdb` used below are examples.

First, define variables that refer to the disk and partition number that will hold **boot files**:

Single Disk

```
export BOOT_DISK="/dev/sda"
export BOOT_PART="1"
export BOOT_DEVICE="${BOOT_DISK}${BOOT_PART}"
```

Separate Boot Device

```
export BOOT_DISK="/dev/sdb"
export BOOT_PART="1"
export BOOT_DEVICE="${BOOT_DISK}${BOOT_PART}"
```

Next, define variables that refer to the disk and partition number that will hold the **ZFS pool**:

Single Disk

```
export POOL_DISK="/dev/sda"
export POOL_PART="2"
export POOL_DEVICE="${POOL_DISK}${POOL_PART}"
```

Separate Boot Device

```
export POOL_DISK="/dev/sda"
export POOL_PART="1"
export POOL_DEVICE="${POOL_DISK}${POOL_PART}"
```

49.1.3 Disk preparation

Wipe partitions

```
wipefs -a "$POOL_DISK"
wipefs -a "$BOOT_DISK"

sgdisk --zap-all "$POOL_DISK"
sgdisk --zap-all "$BOOT_DISK"
```

Create EFI boot partition

```
sgdisk -n "${BOOT_PART}:1m:+512m" -t "${BOOT_PART}:ef00" "$BOOT_DISK"
```

Create zpool partition

```
sgdisk -n "${POOL_PART}:0:-10m" -t "${POOL_PART}:bf00" "$POOL_DISK"
```

49.1.4 ZFS pool creation

Create the zpool

Unencrypted

```
zpool create -f -o ashift=12 \
 -O compression=lz4 \
 -O acltype=posixacl \
 -O xattr=sa \
 -O relatime=on \
 -o autotrim=on \
 -m none zroot "$POOL_DEVICE"
```

Encrypted

```
echo 'SomeKeyphrase' > /etc/zfs/zroot.key
chmod 000 /etc/zfs/zroot.key

zpool create -f -o ashift=12 \
 -O compression=lz4 \
 -O acltype=posixacl \
 -O xattr=sa \
 -O relatime=on \
 -O encryption=aes-256-gcm \
 -O keylocation=file:///etc/zfs/zroot.key \
 -O keyformat=passphrase \
 -o autotrim=on \
 -m none zroot "$POOL_DEVICE"
```

Note: It's out of the scope of this guide to cover all of the pool creation options used - feel free to tailor them to suit your system. However, the following options need to be addressed:

- `encryption=aes-256-gcm` - You can adjust the algorithm as you see fit, but this will likely be the most performant on modern x86_64 hardware.
- `keylocation=file:///etc/zfs/zroot.key` - This sets our pool encryption passphrase to the file `/etc/zfs/zroot.key`, which we created in a previous step. This file will live inside your initramfs stored *on* the ZFS boot environment.
- `keyformat=passphrase` - By setting the format to `passphrase`, we can now force a prompt for this in `zfsbootmenu`. It's critical that your passphrase be something you can type on your keyboard, since you will need to type it in to unlock the pool on boot.

Enable zpool.cache

To more quickly discover and import pools on boot, we need to set a pool cachefile:

```
zpool set cachefile=/etc/zfs/zpool.cache zroot
```

Create initial file systems

```
zfs create -o mountpoint=none zroot/ROOT
zfs create -o mountpoint=/ -o canmount=noauto zroot/ROOT/${ID}
zfs create -o mountpoint=/home zroot/home

zpool set bootfs=zroot/ROOT/${ID} zroot
```

Note: It is important to set the property `canmount=noauto` on any file systems with `mountpoint=/` (that is, on any additional boot environments you create). Without this property, the OS will attempt to automount all ZFS file systems and fail when multiple file systems attempt to mount at `/`; this will prevent your system from booting. Automatic mounting of `/` is not required because the root file system is explicitly mounted in the boot process.

Also note that, unlike many ZFS properties, `canmount` is not inheritable. Therefore, setting `canmount=noauto` on `zroot/ROOT` is not sufficient, as any subsequent boot environments you create will default to `canmount=on`. It is necessary to explicitly set the `canmount=noauto` on every boot environment you create.

Export, then re-import with a temporary mountpoint of `/mnt`

Unencrypted

```
zpool export zroot
zpool import -N -R /mnt zroot
zfs mount zroot/ROOT/${ID}
zfs mount zroot/home
```

Encrypted

```
zpool export zroot
zpool import -N -R /mnt zroot
zfs load-key -L prompt zroot
```

```
zfs mount zroot/ROOT/${ID}
zfs mount zroot/home
```

Verify that everything is mounted correctly

```
# mount | grep mnt
zroot/ROOT/void on /mnt type zfs (rw,relatime,xattr,posixacl)
zroot/home on /mnt/home type zfs (rw,relatime,xattr,posixacl)
```

Update device symlinks

```
udevadm trigger
```

49.1.5 Install Void

Adjust the mirror, libc, and package selection as you see fit.

```
XBPS_ARCH=x86_64 xbps-install \
  -S -R https://mirrors.servercentral.com/voidlinux/current \
  -r /mnt base-system
```

Copy our files into the new install

Unencrypted

```
cp /etc/hostid /mnt/etc
mkdir /mnt/etc/zfs
cp /etc/zfs/zpool.cache /mnt/etc/zfs
```

Encrypted

```
cp /etc/hostid /mnt/etc
mkdir /mnt/etc/zfs
cp /etc/zfs/zpool.cache /mnt/etc/zfs
cp /etc/zfs/zroot.key /mnt/etc/zfs
```

Chroot into the new OS

```
xchroot /mnt
```

49.1.6 Basic Void configuration

Set the keymap, timezone and hardware clock

```
cat << EOF >> /etc/rc.conf
KEYMAP="us"
HARDWARECLOCK="UTC"
EOF
ln -sf /usr/share/zoneinfo/<timezone> /etc/localtime
```

Configure your glibc locale

Note: This does not need to be done on musl, as musl does not have system locale support.

```
cat << EOF >> /etc/default/libc-locales
en_US.UTF-8 UTF-8
en_US ISO-8859-1
EOF
xbps-reconfigure -f glibc-locales
```

Set a root password

```
passwd
```

49.1.7 ZFS Configuration

Configure Dracut to load ZFS support

Unencrypted

```
cat << EOF > /etc/dracut.conf.d/zol.conf
nofsck="yes"
add_dracutmodules+=" zfs "
omit_dracutmodules+=" btrfs "
EOF
```

Encrypted

```
cat << EOF > /etc/dracut.conf.d/zol.conf
nofsck="yes"
add_dracutmodules+=" zfs "
omit_dracutmodules+=" btrfs "
install_items+=" /etc/zfs/zroot.key "
EOF
```

Install ZFS

```
xbps-install -S zfs
```

49.1.8 Install and configure ZFSBootMenu

Set ZFSBootMenu properties on datasets

Unencrypted

Assign command-line arguments to be used when booting the final kernel. Because ZFS properties are inherited, assign the common properties to the ROOT dataset so all children will inherit common arguments by default.

```
zfs set org.zfsbootmenu:commandline="quiet loglevel=4" zroot/ROOT
```

Encrypted

Assign command-line arguments to be used when booting the final kernel. Because ZFS properties are inherited, assign the common properties to the ROOT dataset so all children will inherit common arguments by default.

```
zfs set org.zfsbootmenu:commandline="quiet loglevel=4" zroot/ROOT
```

Setup key caching in ZFSBootMenu.

```
zfs set org.zfsbootmenu:keysource="zroot/ROOT/${ID}" zroot
```

Create a vfat filesystem

```
mkfs.vfat -F32 "$BOOT_DEVICE"
```

Create an fstab entry and mount

```
cat << EOF >> /etc/fstab
$( blkid | grep "$BOOT_DEVICE" | cut -d ' ' -f 2 ) /boot/efi vfat defaults 0 0
EOF

mkdir -p /boot/efi
mount /boot/efi
```

Install ZFSBootMenu

Package

```
xbps-install -S zfsbootmenu gummiboot-efistub
```

Configure *generate-zbm(5)* by ensuring that the following keys appear in /etc/zfsbootmenu/config.yaml:

```
Global:
  ManageImages: true
  BootMountPoint: /boot/efi
Components:
   Enabled: false
EFI:
  ImageDir: /boot/efi/EFI/zbm
  Versions: false
```

(continues on next page)

(continued from previous page)

```
  Enabled: true
Kernel:
  CommandLine: quiet loglevel=0
```

Configure *generate-zbm(5)* by ensuring that the following keys appear in `/etc/zfsbootmenu/config.yaml`:

```
Global:
  ManageImages: true
  BootMountPoint: /boot/efi
Components:
   Enabled: false
EFI:
  ImageDir: /boot/efi/EFI/zbm
  Versions: false
  Enabled: true
Kernel:
  CommandLine: quiet loglevel=0
```

Create a ZFSBootMenu image:

```
generate-zbm
```

Prebuilt

```
xbps-install -S curl
```

Fetch a prebuilt ZFSBootMenu EFI executable, saving it to the EFI system partition:

```
mkdir -p /boot/efi/EFI/ZBM
curl -o /boot/efi/EFI/ZBM/VMLINUZ.EFI -L https://get.zfsbootmenu.org/efi
cp /boot/efi/EFI/ZBM/VMLINUZ.EFI /boot/efi/EFI/ZBM/VMLINUZ-BACKUP.EFI
```

Configure EFI boot entries

Direct

```
xbps-install efibootmgr
```

```
efibootmgr -c -d "$BOOT_DISK" -p "$BOOT_PART" \
  -L "ZFSBootMenu (Backup)" \
  -l \\EFI\\ZBM\\VMLINUZ-BACKUP.EFI

efibootmgr -c -d "$BOOT_DISK" -p "$BOOT_PART" \
  -L "ZFSBootMenu" \
  -l \\EFI\\ZBM\\VMLINUZ.EFI
```

rEFInd

```
xbps-install -S refind
```

```
refind-install
rm /boot/refind_linux.conf

cat << EOF > /boot/efi/EFI/zbm/refind_linux.conf
"Boot default"  "quiet loglevel=0 zbm.skip"
"Boot to menu"  "quiet loglevel=0 zbm.show"
EOF
```

See also:

Some systems can have issues with EFI boot entries. If you reboot and do not see the above entries in your EFI selection screen (usually accessible through an F key during POST), you might need to use a well-known EFI file name. See *Portable EFI* for help with this. Your existing ESP can be used, in place of an external USB drive.

Refer to *zbm-kcl.8* and *zfsbootmenu.7* for details on configuring the boot-time behavior of ZFSBootMenu.

49.1.9 Prepare for first boot

Exit the chroot, unmount everything

```
exit
```

```
umount -n -R /mnt
```

Export the zpool and reboot

```
zpool export zroot
reboot
```

49.2 SYSLINUX MBR

Contents

This guide can be used to install Void onto a single ZFS disk with or without ZFS encryption. It assumes the following:

- Your system uses BIOS to boot
- Your system is x86_64
- You will use `glibc` as your system libc.
- `/dev/sda` is the disk to be used for both ZFS and syslinux
- You're mildly comfortable with ZFS and discovering system facts on your own (`lsblk`, `dmesg`, `gdisk`, ...)

ZFSBootMenu does not require `glibc` and is not restricted to `x86_64`. If you are comfortable installing Void Linux on other architectures or with the `musl` libc, you can adapt the instructions here to your desired configuration.

Download the latest hrmpf, write it to USB drive and boot your system in BIOS mode.

49.2.1 Configure Live Environment

Source `/etc/os-release`

The file */etc/os-release* defines variables that describe the running distribution. In particular, the *$ID* variable defined within can be used as a short name for the filesystem that will hold this installation.

```
source /etc/os-release
export ID="$ID"
```

Generate `/etc/hostid`

```
zgenhostid -f 0x00bab10c
```

49.2.2 Define disk variables

For convenience and to reduce the likelihood of errors, set environment variables that refer to the devices that will be configured during the setup.

For many users, it is most convenient to place boot files (*i.e.*, ZFSBootMenu and any loader responsible for launching it) on the the same disk that will hold the ZFS pool. However, some users may wish to dedicate an entire disk to the ZFS pool or create a multi-disk pool. A USB flash drive provides a convenient location for the boot partition. Fortunately, this alternative configuration is easily realized by simply defining a few environment variables differently.

Verify your target disk devices with `lsblk`. `/dev/sda` and `/dev/sdb` used below are examples.

First, define variables that refer to the disk and partition number that will hold **boot files**:

Single Disk

```
export BOOT_DISK="/dev/sda"
export BOOT_PART="1"
export BOOT_DEVICE="${BOOT_DISK}${BOOT_PART}"
```

Separate Boot Device

```
export BOOT_DISK="/dev/sdb"
export BOOT_PART="1"
export BOOT_DEVICE="${BOOT_DISK}${BOOT_PART}"
```

Next, define variables that refer to the disk and partition number that will hold the **ZFS pool**:

Single Disk

```
export POOL_DISK="/dev/sda"
export POOL_PART="2"
export POOL_DEVICE="${POOL_DISK}${POOL_PART}"
```

Separate Boot Device

```
export POOL_DISK="/dev/sda"
export POOL_PART="1"
export POOL_DEVICE="${POOL_DISK}${POOL_PART}"
```

49.2.3 Disk preparation

The disk(s) that will hold the syslinux partition and ZFS pool should be labeled in MBR format and configured to provide partitions for boot files as well as the ZFSBootMenu.

Single Disk

```
cat <<EOF | sfdisk "${POOL_DISK}"
label: dos
start=1MiB, size=512MiB, type=83, bootable
start=513MiB, size=+, type=83
EOF
```

Separate Boot Device

```
cat <<EOF | sfdisk "${POOL_DISK}"
label: dos
start=1MiB, size=+, type=83, bootable
EOF

cat <<EOF | sfdisk "${BOOT_DISK}"
label: dos
start=1MiB, size=512MiB, type=83
EOF
```

49.2.4 ZFS pool creation

Create the zpool

Unencrypted

```
zpool create -f -o ashift=12 \
 -O compression=lz4 \
 -O acltype=posixacl \
 -O xattr=sa \
 -O relatime=on \
 -o autotrim=on \
 -m none zroot "$POOL_DEVICE"
```

Encrypted

```
echo 'SomeKeyphrase' > /etc/zfs/zroot.key
chmod 000 /etc/zfs/zroot.key

zpool create -f -o ashift=12 \
 -O compression=lz4 \
 -O acltype=posixacl \
 -O xattr=sa \
 -O relatime=on \
 -O encryption=aes-256-gcm \
 -O keylocation=file:///etc/zfs/zroot.key \
 -O keyformat=passphrase \
```

(continues on next page)

(continued from previous page)

```
-o autotrim=on \
-m none zroot "$POOL_DEVICE"
```

Note: It's out of the scope of this guide to cover all of the pool creation options used - feel free to tailor them to suit your system. However, the following options need to be addressed:

- `encryption=aes-256-gcm` - You can adjust the algorithm as you see fit, but this will likely be the most performant on modern x86_64 hardware.
- `keylocation=file:///etc/zfs/zroot.key` - This sets our pool encryption passphrase to the file `/etc/zfs/zroot.key`, which we created in a previous step. This file will live inside your initramfs stored *on* the ZFS boot environment.
- `keyformat=passphrase` - By setting the format to `passphrase`, we can now force a prompt for this in `zfsbootmenu`. It's critical that your passphrase be something you can type on your keyboard, since you will need to type it in to unlock the pool on boot.

Enable zpool.cache

To more quickly discover and import pools on boot, we need to set a pool cachefile:

```
zpool set cachefile=/etc/zfs/zpool.cache zroot
```

Create initial file systems

```
zfs create -o mountpoint=none zroot/ROOT
zfs create -o mountpoint=/ -o canmount=noauto zroot/ROOT/${ID}
zfs create -o mountpoint=/home zroot/home

zpool set bootfs=zroot/ROOT/${ID} zroot
```

Note: It is important to set the property `canmount=noauto` on any file systems with `mountpoint=/` (that is, on any additional boot environments you create). Without this property, the OS will attempt to automount all ZFS file systems and fail when multiple file systems attempt to mount at `/`; this will prevent your system from booting. Automatic mounting of `/` is not required because the root file system is explicitly mounted in the boot process.

Also note that, unlike many ZFS properties, `canmount` is not inheritable. Therefore, setting `canmount=noauto` on `zroot/ROOT` is not sufficient, as any subsequent boot environments you create will default to `canmount=on`. It is necessary to explicitly set the `canmount=noauto` on every boot environment you create.

Export, then re-import with a temporary mountpoint of `/mnt`

Unencrypted

```
zpool export zroot
zpool import -N -R /mnt zroot
zfs mount zroot/ROOT/${ID}
zfs mount zroot/home
```

Encrypted

```
zpool export zroot
zpool import -N -R /mnt zroot
zfs load-key -L prompt zroot
```

```
zfs mount zroot/ROOT/${ID}
zfs mount zroot/home
```

Verify that everything is mounted correctly

```
# mount | grep mnt
zroot/ROOT/void on /mnt type zfs (rw,relatime,xattr,posixacl)
zroot/home on /mnt/home type zfs (rw,relatime,xattr,posixacl)
```

Update device symlinks

```
udevadm trigger
```

49.2.5 Install Void

Adjust the mirror, libc, and package selection as you see fit.

```
XBPS_ARCH=x86_64 xbps-install \
  -S -R https://mirrors.servercentral.com/voidlinux/current \
  -r /mnt base-system
```

Copy our files into the new install

Unencrypted

```
cp /etc/hostid /mnt/etc
mkdir /mnt/etc/zfs
cp /etc/zfs/zpool.cache /mnt/etc/zfs
```

Encrypted

```
cp /etc/hostid /mnt/etc
mkdir /mnt/etc/zfs
cp /etc/zfs/zpool.cache /mnt/etc/zfs
cp /etc/zfs/zroot.key /mnt/etc/zfs
```

Chroot into the new OS

```
xchroot /mnt
```

49.2.6 Basic Void configuration

Set the keymap, timezone and hardware clock

```
cat << EOF >> /etc/rc.conf
KEYMAP="us"
HARDWARECLOCK="UTC"
EOF
ln -sf /usr/share/zoneinfo/<timezone> /etc/localtime
```

Configure your glibc locale

Note: This does not need to be done on musl, as musl does not have system locale support.

```
cat << EOF >> /etc/default/libc-locales
en_US.UTF-8 UTF-8
en_US ISO-8859-1
EOF
xbps-reconfigure -f glibc-locales
```

Set a root password

```
passwd
```

49.2.7 ZFS Configuration

Configure Dracut to load ZFS support

Unencrypted

```
cat << EOF > /etc/dracut.conf.d/zol.conf
nofsck="yes"
add_dracutmodules+=" zfs "
omit_dracutmodules+=" btrfs "
EOF
```

Encrypted

```
cat << EOF > /etc/dracut.conf.d/zol.conf
nofsck="yes"
add_dracutmodules+=" zfs "
```

(continues on next page)

(continued from previous page)

```
omit_dracutmodules+=" btrfs "
install_items+=" /etc/zfs/zroot.key "
EOF
```

Install ZFS

```
xbps-install -S zfs
```

49.2.8 Install and configure ZFSBootMenu

Create a ext4 boot filesystem

```
mkfs.ext4 -O '^64bit' "$BOOT_DEVICE"
```

Note: Under some circumstances, syslinux may fail to recognize files on an ext4 filesystem. The issue may be related to the `64bit` feature of the filesystem, which is explicitly disabled in the command above. If syslinux still fails to recognize files on the ext4 partition, try using ext3 or ext2 as a fallback.

Create an fstab entry and mount

```
cat << EOF >> /etc/fstab
$( blkid | grep "$BOOT_DEVICE" | cut -d ' ' -f 2 ) /boot/syslinux ext4 defaults 0 0
EOF

mkdir /boot/syslinux
mount /boot/syslinux
```

Install the syslinux package, copy modules

```
xbps-install -S syslinux
cp /usr/lib/syslinux/*.c32 /boot/syslinux
```

Install extlinux

```
extlinux --install /boot/syslinux
```

Install the syslinux MBR data

```
dd bs=440 count=1 conv=notrunc if=/usr/lib/syslinux/mbr.bin of="$BOOT_DISK"
```

49.2.9 Install and configure ZFSBootMenu

Set ZFSBootMenu properties on datasets

Unencrypted

Assign command-line arguments to be used when booting the final kernel. Because ZFS properties are inherited, assign the common properties to the ROOT dataset so all children will inherit common arguments by default.

```
zfs set org.zfsbootmenu:commandline="quiet loglevel=4" zroot/ROOT
```

Encrypted

Assign command-line arguments to be used when booting the final kernel. Because ZFS properties are inherited, assign the common properties to the ROOT dataset so all children will inherit common arguments by default.

```
zfs set org.zfsbootmenu:commandline="quiet loglevel=4" zroot/ROOT
```

Setup key caching in ZFSBootMenu.

```
zfs set org.zfsbootmenu:keysource="zroot/ROOT/${ID}" zroot
```

Install the ZFSBootMenu package

```
xbps-install -S zfsbootmenu
```

Enable zfsbootmenu image creation

Edit `/etc/zfsbootmenu/config.yaml` and make sure that the following parameters are set:

```
Global:
  ManageImages: true
  BootMountPoint: /boot/syslinux
Components:
  Enabled: true
  Versions: false
  ImageDir: /boot/syslinux/zfsbootmenu
```

See *generate-zbm(5)* for more details.

Configure syslinux

The `generate-zbm` image-creation utility includes now-deprecated support for managing a syslinux configuration. Because this capability is slated for removal and was not reliable in the first place, it is better to create a static syslinux configuration. The ZFSBootMenu configuration described above disables explicit image versioning, which means that each invocation of `generate-zbm` will produce two output files at a predictable location:

- `/boot/syslinux/zfsbootmenu/vmlinuz-bootmenu`
- `/boot/syslinux/zfsbootmenu/initramfs-bootmenu.img`

In addition, any existing copies of the ZFSBootMenu kernel and initramfs will be saved to a backup location:

- `/boot/syslinux/zfsbootmenu/vmlinuz-bootmenu-backup`
- `/boot/syslinux/zfsbootmenu/initramfs-bootmenu-backup.img`

The following syslinux configuration will provide a simple menu that provides a choice between the current and backup images:

```
cat > /boot/syslinux/syslinux.cfg <<EOF
UI menu.c32
PROMPT 0

MENU TITLE ZFSBootMenu
TIMEOUT 50

DEFAULT zfsbootmenu

LABEL zfsbootmenu
  MENU LABEL ZFSBootMenu
  KERNEL /zfsbootmenu/vmlinuz-bootmenu
  INITRD /zfsbootmenu/initramfs-bootmenu.img
  APPEND zfsbootmenu quiet loglevel=4

LABEL zfsbootmenu-backup
  MENU LABEL ZFSBootMenu (Backup)
  KERNEL /zfsbootmenu/vmlinuz-bootmenu-backup
  INITRD /zfsbootmenu/initramfs-bootmenu-backup.img
  APPEND zfsbootmenu quiet loglevel=4
EOF
```

Consult the syslinux documentation for more details on the contents of the `syslinux.cfg` configuration file. To alter the command-line arguments passed to the ZFSBootMenu image, adjust the contents of the `APPEND` lines in the configuration.

Generate the initial ZFSBootMenu initramfs

```
xbps-reconfigure -f zfsbootmenu
```

49.2.10 Prepare for first boot

Exit the chroot, unmount everything

```
exit
```

```
umount -n -R /mnt
```

Export the zpool and reboot

```
zpool export zroot
reboot
```

CHAPTER

FIFTY

ALPINE LINUX

50.1 UEFI

Contents

This guide can be used to install Alpine onto a single disk with with or without ZFS encryption.

It assumes the following:

- Your system uses UEFI to boot
- Your system is x86_64
- You're mildly comfortable with ZFS, EFI and discovering system facts on your own (`lsblk`, `dmesg`, `gdisk`, ...)

ZFSBootMenu does not require `glibc` and is not restricted to `x86_64`. If you are comfortable installing Void Linux on other architectures or with the `musl` libc, you can adapt the instructions here to your desired configuration.

Download the latest Alpine Extended ISO, write it to USB drive and boot your system in EFI mode.

Confirm EFI support:

```
# dmesg | grep -i efivars
[    0.301784] Registered efivars operations
```

50.1.1 Configure Live Environment

Source `/etc/os-release`

The file */etc/os-release* defines variables that describe the running distribution. In particular, the *$ID* variable defined within can be used as a short name for the filesystem that will hold this installation.

```
source /etc/os-release
export ID="$ID"
```

Add package repositories

```
cat <<EOF > /etc/apk/repositories
http://dl-cdn.alpinelinux.org/alpine/latest-stable/main/
https://dl-cdn.alpinelinux.org/alpine/latest-stable/main/
EOF
apk update
```

Setup additional tools

```
apk add zfs sgdisk wipefs eudev
modprobe zfs
setup-devd udev
```

Generate `/etc/hostid`

```
zgenhostid -f 0x00bab10c
```

50.1.2 Define disk variables

For convenience and to reduce the likelihood of errors, set environment variables that refer to the devices that will be configured during the setup.

For many users, it is most convenient to place boot files (*i.e.*, ZFSBootMenu and any loader responsible for launching it) on the the same disk that will hold the ZFS pool. However, some users may wish to dedicate an entire disk to the ZFS pool or create a multi-disk pool. A USB flash drive provides a convenient location for the boot partition. Fortunately, this alternative configuration is easily realized by simply defining a few environment variables differently.

Verify your target disk devices with `lsblk`. `/dev/sda` and `/dev/sdb` used below are examples.

First, define variables that refer to the disk and partition number that will hold **boot files**:

Single Disk

```
export BOOT_DISK="/dev/sda"
export BOOT_PART="1"
export BOOT_DEVICE="${BOOT_DISK}${BOOT_PART}"
```

Separate Boot Device

```
export BOOT_DISK="/dev/sdb"
export BOOT_PART="1"
export BOOT_DEVICE="${BOOT_DISK}${BOOT_PART}"
```

Next, define variables that refer to the disk and partition number that will hold the **ZFS pool**:

Single Disk

```
export POOL_DISK="/dev/sda"
export POOL_PART="2"
export POOL_DEVICE="${POOL_DISK}${POOL_PART}"
```

Separate Boot Device

```
export POOL_DISK="/dev/sda"
export POOL_PART="1"
export POOL_DEVICE="${POOL_DISK}${POOL_PART}"
```

50.1.3 Disk preparation

Wipe partitions

```
wipefs -a "$POOL_DISK"
wipefs -a "$BOOT_DISK"

sgdisk --zap-all "$POOL_DISK"
sgdisk --zap-all "$BOOT_DISK"
```

Create EFI boot partition

```
sgdisk -n "${BOOT_PART}:1m:+512m" -t "${BOOT_PART}:ef00" "$BOOT_DISK"
```

Create zpool partition

```
sgdisk -n "${POOL_PART}:0:-10m" -t "${POOL_PART}:bf00" "$POOL_DISK"
```

50.1.4 ZFS pool creation

Create the zpool

Unencrypted

```
zpool create -f -o ashift=12 \
 -O compression=lz4 \
 -O acltype=posixacl \
 -O xattr=sa \
 -O relatime=on \
 -o autotrim=on \
 -m none zroot "$POOL_DEVICE"
```

Encrypted

```
echo 'SomeKeyphrase' > /etc/zfs/zroot.key
chmod 000 /etc/zfs/zroot.key

zpool create -f -o ashift=12 \
 -O compression=lz4 \
 -O acltype=posixacl \
 -O xattr=sa \
```

(continues on next page)

(continued from previous page)

```
  -O relatime=on \
  -O encryption=aes-256-gcm \
  -O keylocation=file:///etc/zfs/zroot.key \
  -O keyformat=passphrase \
  -o autotrim=on \
  -m none zroot "$POOL_DEVICE"
```

Note: It's out of the scope of this guide to cover all of the pool creation options used - feel free to tailor them to suit your system. However, the following options need to be addressed:

- `encryption=aes-256-gcm` - You can adjust the algorithm as you see fit, but this will likely be the most performant on modern x86_64 hardware.
- `keylocation=file:///etc/zfs/zroot.key` - This sets our pool encryption passphrase to the file `/etc/zfs/zroot.key`, which we created in a previous step. This file will live inside your initramfs stored *on* the ZFS boot environment.
- `keyformat=passphrase` - By setting the format to `passphrase`, we can now force a prompt for this in `zfsbootmenu`. It's critical that your passphrase be something you can type on your keyboard, since you will need to type it in to unlock the pool on boot.

Enable zpool.cache

To more quickly discover and import pools on boot, we need to set a pool cachefile:

```
zpool set cachefile=/etc/zfs/zpool.cache zroot
```

Create initial file systems

```
zfs create -o mountpoint=none zroot/ROOT
zfs create -o mountpoint=/ -o canmount=noauto zroot/ROOT/${ID}
zfs create -o mountpoint=/home zroot/home

zpool set bootfs=zroot/ROOT/${ID} zroot
```

Note: It is important to set the property `canmount=noauto` on any file systems with `mountpoint=/` (that is, on any additional boot environments you create). Without this property, the OS will attempt to automount all ZFS file systems and fail when multiple file systems attempt to mount at `/`; this will prevent your system from booting. Automatic mounting of `/` is not required because the root file system is explicitly mounted in the boot process.

Also note that, unlike many ZFS properties, `canmount` is not inheritable. Therefore, setting `canmount=noauto` on `zroot/ROOT` is not sufficient, as any subsequent boot environments you create will default to `canmount=on`. It is necessary to explicitly set the `canmount=noauto` on every boot environment you create.

Export, then re-import with a temporary mountpoint of `/mnt`

Unencrypted

```
zpool export zroot
zpool import -N -R /mnt zroot
zfs mount zroot/ROOT/${ID}
zfs mount zroot/home
```

Encrypted

```
zpool export zroot
zpool import -N -R /mnt zroot
zfs load-key -L prompt zroot
```

```
zfs mount zroot/ROOT/${ID}
zfs mount zroot/home
```

Verify that everything is mounted correctly

```
# mount | grep mnt
zroot/ROOT/alpine on /mnt type zfs (rw,relatime,xattr,posixacl)
zroot/home on /mnt/home type zfs (rw,relatime,xattr,posixacl)
```

Update device symlinks

```
udevadm trigger
```

50.1.5 Install Alpine

```
apk --arch x86_64 -X http://dl-cdn.alpinelinux.org/alpine/latest-stable/main \
 -U --allow-untrusted --root /mnt --initdb add alpine-base
```

Copy our files into the new install

Unencrypted

```
cp /etc/hostid /mnt/etc
cp /etc/resolv.conf /mnt/etc
cp /etc/apk/repositories /mnt/etc/apk
cp /etc/network/interfaces /mnt/etc/network
mkdir /mnt/etc/zfs
cp /etc/zfs/zpool.cache /mnt/etc/zfs
```

Encrypted

```
cp /etc/hostid /mnt/etc
cp /etc/resolv.conf /mnt/etc
cp /etc/apk/repositories /mnt/etc/apk
cp /etc/network/interfaces /mnt/etc/network
mkdir /mnt/etc/zfs
cp /etc/zfs/zpool.cache /mnt/etc/zfs
cp /etc/zfs/zroot.key /mnt/etc/zfs
```

Chroot into the new OS

```
mount --rbind /dev /mnt/dev
mount --rbind /sys /mnt/sys
mount --rbind /proc /mnt/proc
chroot /mnt
```

Set a root password

```
passwd
```

Enable startup targets

```
rc-update add hwdrivers sysinit
rc-update add networking
rc-update add hostname
apk add udev
setup-devd udev
```

50.1.6 ZFS Configuration

Install ZFS

```
apk add zfs zfs-lts
rc-update add zfs-import sysinit
rc-update add zfs-mount sysinit
```

Configure mkinitfs to load ZFS support

Unencrypted

```
echo "/etc/hostid" >> /etc/mkinitfs/features.d/zfshost.files
echo "/etc/zfs/zpool.cache" >> /etc/mkinitfs/features.d/zfshost.files
echo 'features="ata base keymap kms mmc scsi usb virtio zfs zfshost"' > /etc/mkinitfs/
↪mkinitfs.conf
```

Encrypted

```
echo "/etc/hostid" >> /etc/mkinitfs/features.d/zfshost.files
echo "/etc/zfs/zpool.cache" >> /etc/mkinitfs/features.d/zfshost.files
echo "/etc/zfs/zroot.key" >> /etc/mkinitfs/features.d/zfshost.files
echo 'features="ata base keymap kms mmc scsi usb virtio zfs zfshost"' > /etc/mkinitfs/
↪mkinitfs.conf
```

Regenerate initramfs

```
mkinitfs -c /etc/mkinitfs/mkinitfs.conf "$(ls /lib/modules)"
```

50.1.7 Install and configure ZFSBootMenu

Set ZFSBootMenu properties on datasets

Unencrypted

Assign command-line arguments to be used when booting the final kernel. Because ZFS properties are inherited, assign the common properties to the ROOT dataset so all children will inherit common arguments by default.

```
zfs set org.zfsbootmenu:commandline="quiet loglevel=4" zroot/ROOT
```

Encrypted

Assign command-line arguments to be used when booting the final kernel. Because ZFS properties are inherited, assign the common properties to the ROOT dataset so all children will inherit common arguments by default.

```
zfs set org.zfsbootmenu:commandline="quiet loglevel=4" zroot/ROOT
```

Setup key caching in ZFSBootMenu.

```
zfs set org.zfsbootmenu:keysource="zroot/ROOT/${ID}" zroot
```

Create a `vfat` filesystem

```
mkfs.vfat -F32 "$BOOT_DEVICE"
```

Create an fstab entry and mount

```
cat << EOF >> /etc/fstab
$BOOT_DEVICE /boot/efi vfat defaults 0 0
EOF

mkdir -p /boot/efi
mount /boot/efi
```

Install ZFSBootMenu

Prebuilt

```
apk add curl
```

Fetch a prebuilt ZFSBootMenu EFI executable, saving it to the EFI system partition:

```
mkdir -p /boot/efi/EFI/ZBM
curl -o /boot/efi/EFI/ZBM/VMLINUZ.EFI -L https://get.zfsbootmenu.org/efi
cp /boot/efi/EFI/ZBM/VMLINUZ.EFI /boot/efi/EFI/ZBM/VMLINUZ-BACKUP.EFI
```

Configure EFI boot entries

Direct

```
apk add efibootmgr
```

```
efibootmgr -c -d "$BOOT_DISK" -p "$BOOT_PART" \
  -L "ZFSBootMenu (Backup)" \
  -l \\EFI\\ZBM\\VMLINUZ-BACKUP.EFI

efibootmgr -c -d "$BOOT_DISK" -p "$BOOT_PART" \
  -L "ZFSBootMenu" \
  -l \\EFI\\ZBM\\VMLINUZ.EFI
```

rEFInd

```
cat <<EOF >> /etc/apk/repositories
http://dl-cdn.alpinelinux.org/alpine/edge/testing
EOF

apk update
apk add refind
```

```
refind-install
rm /boot/refind_linux.conf

cat << EOF > /boot/efi/EFI/zbm/refind_linux.conf
"Boot default"  "quiet loglevel=0 zbm.skip"
"Boot to menu"  "quiet loglevel=0 zbm.show"
EOF
```

See also:

Some systems can have issues with EFI boot entries. If you reboot and do not see the above entries in your EFI selection screen (usually accessible through an F key during POST), you might need to use a well-known EFI file name. See *Portable EFI* for help with this. Your existing ESP can be used, in place of an external USB drive.

Refer to *zbm-kcl.8* and *zfsbootmenu.7* for details on configuring the boot-time behavior of ZFSBootMenu.

50.1.8 Prepare for first boot

Exit the chroot, unmount everything

```
exit
```

```
cut -f2 -d" " /proc/mounts | grep ^/mnt | tac | while read i; do umount -l $i; done
```

Export the zpool and reboot

```
zpool export zroot
reboot
```

CHAPTER

FIFTYONE

DEBIAN

51.1 Bullseye UEFI

Contents

This guide can be used to install Debian onto a single disk with or without ZFS encryption.

It assumes the following:

- Your system uses UEFI to boot
- Your system is x86_64
- You're mildly comfortable with ZFS, EFI and discovering system facts on your own (`lsblk`, `dmesg`, `gdisk`, ...)

Download the latest Debian Bullseye (11) Live image, write it to a USB drive and boot your system in EFI mode.

Confirm EFI support:

```
# dmesg | grep -i efivars
[    0.301784] Registered efivars operations
```

51.1.1 Configure Live Environment

Switch to a root shell

```
sudo -i
```

Source `/etc/os-release`

The file */etc/os-release* defines variables that describe the running distribution. In particular, the *$ID* variable defined within can be used as a short name for the filesystem that will hold this installation.

```
source /etc/os-release
export ID="$ID"
```

Configure and update APT

```
cat <<EOF > /etc/apt/sources.list
deb http://deb.debian.org/debian bullseye main contrib
deb-src http://deb.debian.org/debian bullseye main contrib
EOF
apt update
```

Note: You may see faster downloads replacing `deb.debian.org` with a local mirror. If you want to use HTTPS transport, make sure that the `ca-certificates` and `apt-transport-https` packages are installed and your mirror has a valid certificate; otherwise, apt will refuse to use the mirror.

Install helpers

```
apt install debootstrap gdisk dkms linux-headers-$(uname -r)
apt install zfsutils-linux
```

Generate `/etc/hostid`

```
zgenhostid -f 0x00bab10c
```

51.1.2 Define disk variables

For convenience and to reduce the likelihood of errors, set environment variables that refer to the devices that will be configured during the setup.

For many users, it is most convenient to place boot files (*i.e.*, ZFSBootMenu and any loader responsible for launching it) on the the same disk that will hold the ZFS pool. However, some users may wish to dedicate an entire disk to the ZFS pool or create a multi-disk pool. A USB flash drive provides a convenient location for the boot partition. Fortunately, this alternative configuration is easily realized by simply defining a few environment variables differently.

Verify your target disk devices with `lsblk`. `/dev/sda` and `/dev/sdb` used below are examples.

First, define variables that refer to the disk and partition number that will hold **boot files**:

Single Disk

```
export BOOT_DISK="/dev/sda"
export BOOT_PART="1"
export BOOT_DEVICE="${BOOT_DISK}${BOOT_PART}"
```

Separate Boot Device

```
export BOOT_DISK="/dev/sdb"
export BOOT_PART="1"
export BOOT_DEVICE="${BOOT_DISK}${BOOT_PART}"
```

Next, define variables that refer to the disk and partition number that will hold the **ZFS pool**:

Single Disk

```
export POOL_DISK="/dev/sda"
export POOL_PART="2"
export POOL_DEVICE="${POOL_DISK}${POOL_PART}"
```

Separate Boot Device

```
export POOL_DISK="/dev/sda"
export POOL_PART="1"
export POOL_DEVICE="${POOL_DISK}${POOL_PART}"
```

51.1.3 Disk preparation

Wipe partitions

```
wipefs -a "$POOL_DISK"
wipefs -a "$BOOT_DISK"

sgdisk --zap-all "$POOL_DISK"
sgdisk --zap-all "$BOOT_DISK"
```

Create EFI boot partition

```
sgdisk -n "${BOOT_PART}:1m:+512m" -t "${BOOT_PART}:ef00" "$BOOT_DISK"
```

Create zpool partition

```
sgdisk -n "${POOL_PART}:0:-10m" -t "${POOL_PART}:bf00" "$POOL_DISK"
```

51.1.4 ZFS pool creation

Create the zpool

Unencrypted

```
zpool create -f -o ashift=12 \
 -O compression=lz4 \
 -O acltype=posixacl \
 -O xattr=sa \
```

(continues on next page)

(continued from previous page)

```
 -O relatime=on \
 -o autotrim=on \
 -m none zroot "$POOL_DEVICE"
```

Encrypted

```
echo 'SomeKeyphrase' > /etc/zfs/zroot.key
chmod 000 /etc/zfs/zroot.key

zpool create -f -o ashift=12 \
 -O compression=lz4 \
 -O acltype=posixacl \
 -O xattr=sa \
 -O relatime=on \
 -O encryption=aes-256-gcm \
 -O keylocation=file:///etc/zfs/zroot.key \
 -O keyformat=passphrase \
 -o autotrim=on \
 -m none zroot "$POOL_DEVICE"
```

Note: It's out of the scope of this guide to cover all of the pool creation options used - feel free to tailor them to suit your system. However, the following options need to be addressed:

- `encryption=aes-256-gcm` - You can adjust the algorithm as you see fit, but this will likely be the most performant on modern x86_64 hardware.
- `keylocation=file:///etc/zfs/zroot.key` - This sets our pool encryption passphrase to the file `/etc/zfs/zroot.key`, which we created in a previous step. This file will live inside your initramfs stored *on* the ZFS boot environment.
- `keyformat=passphrase` - By setting the format to `passphrase`, we can now force a prompt for this in `zfsbootmenu`. It's critical that your passphrase be something you can type on your keyboard, since you will need to type it in to unlock the pool on boot.

Enable zpool.cache

To more quickly discover and import pools on boot, we need to set a pool cachefile:

```
zpool set cachefile=/etc/zfs/zpool.cache zroot
```

Create initial file systems

```
zfs create -o mountpoint=none zroot/ROOT
zfs create -o mountpoint=/ -o canmount=noauto zroot/ROOT/${ID}
zfs create -o mountpoint=/home zroot/home

zpool set bootfs=zroot/ROOT/${ID} zroot
```

Note: It is important to set the property `canmount=noauto` on any file systems with `mountpoint=/` (that is, on any additional boot environments you create). Without this property, the OS will attempt to automount all ZFS file systems

and fail when multiple file systems attempt to mount at /; this will prevent your system from booting. Automatic mounting of / is not required because the root file system is explicitly mounted in the boot process.

Also note that, unlike many ZFS properties, `canmount` is not inheritable. Therefore, setting `canmount=noauto` on `zroot/ROOT` is not sufficient, as any subsequent boot environments you create will default to `canmount=on`. It is necessary to explicitly set the `canmount=noauto` on every boot environment you create.

Export, then re-import with a temporary mountpoint of `/mnt`

Unencrypted

```
zpool export zroot
zpool import -N -R /mnt zroot
zfs mount zroot/ROOT/${ID}
zfs mount zroot/home
```

Encrypted

```
zpool export zroot
zpool import -N -R /mnt zroot
zfs load-key -L prompt zroot
```

```
zfs mount zroot/ROOT/${ID}
zfs mount zroot/home
```

Verify that everything is mounted correctly

```
# mount | grep mnt
zroot/ROOT/debian on /mnt type zfs (rw,relatime,xattr,posixacl)
zroot/home on /mnt/home type zfs (rw,relatime,xattr,posixacl)
```

Update device symlinks

```
udevadm trigger
```

51.1.5 Install Debian

```
debootstrap bullseye /mnt
```

Copy files into the new install

Unencrypted

```
cp /etc/hostid /mnt/etc
cp /etc/resolv.conf /mnt/etc
mkdir /mnt/etc/zfs
cp /etc/zfs/zpool.cache /mnt/etc/zfs
```

Encrypted

```
cp /etc/hostid /mnt/etc/hostid
cp /etc/resolv.conf /mnt/etc/
mkdir /mnt/etc/zfs
cp /etc/zfs/zpool.cache /mnt/etc/zfs
cp /etc/zfs/zroot.key /mnt/etc/zfs
```

Chroot into the new OS

```
mount -t proc proc /mnt/proc
mount -t sysfs sys /mnt/sys
mount -B /dev /mnt/dev
mount -t devpts pts /mnt/dev/pts
chroot /mnt /bin/bash
```

51.1.6 Basic Debian Configuration

Set a hostname

```
echo 'YOURHOSTNAME' > /etc/hostname
echo -e '127.0.1.1\tYOURHOSTNAME' >> /etc/hosts
```

Set a root password

```
passwd
```

Configure `apt`. Use other mirrors if you prefer.

```
cat <<EOF > /etc/apt/sources.list
deb http://deb.debian.org/debian bullseye main contrib
deb-src http://deb.debian.org/debian bullseye main contrib

deb http://deb.debian.org/debian-security bullseye-security main contrib
deb-src http://deb.debian.org/debian-security/ bullseye-security main contrib

deb http://deb.debian.org/debian bullseye-updates main contrib
deb-src http://deb.debian.org/debian bullseye-updates main contrib

deb http://deb.debian.org/debian bullseye-backports main contrib
```

(continues on next page)

(continued from previous page)

```
deb-src http://deb.debian.org/debian bullseye-backports main contrib
EOF
```

Update the repository cache

```
apt update
```

Install additional base packages

```
apt install locales keyboard-configuration console-setup
```

Configure packages to customize local and console properties

```
dpkg-reconfigure locales tzdata keyboard-configuration console-setup
```

Note: You should always enable the *en_US.UTF-8* locale because some programs require it.

51.1.7 ZFS Configuration

Install required packages

```
apt install linux-headers-amd64 linux-image-amd64 zfs-initramfs dosfstools
echo "REMAKE_INITRD=yes" > /etc/dkms/zfs.conf
```

Enable systemd ZFS services

```
systemctl enable zfs.target
systemctl enable zfs-import-cache
systemctl enable zfs-mount
systemctl enable zfs-import.target
```

Configure `initramfs-tools`

Unencrypted

No required steps

Encrypted

```
echo "UMASK=0077" > /etc/initramfs-tools/conf.d/umask.conf
```

Note: Because the encryption key is stored in `/etc/zfs` directory, it will automatically be copied into the system initramfs.

Rebuild the initramfs

```
update-initramfs -c -k all
```

51.1.8 Install and configure ZFSBootMenu

Set ZFSBootMenu properties on datasets

Unencrypted

Assign command-line arguments to be used when booting the final kernel. Because ZFS properties are inherited, assign the common properties to the `ROOT` dataset so all children will inherit common arguments by default.

```
zfs set org.zfsbootmenu:commandline="quiet loglevel=4" zroot/ROOT
```

Encrypted

Assign command-line arguments to be used when booting the final kernel. Because ZFS properties are inherited, assign the common properties to the `ROOT` dataset so all children will inherit common arguments by default.

```
zfs set org.zfsbootmenu:commandline="quiet loglevel=4" zroot/ROOT
```

Setup key caching in ZFSBootMenu.

```
zfs set org.zfsbootmenu:keysource="zroot/ROOT/${ID}" zroot
```

Create a `vfat` filesystem

```
mkfs.vfat -F32 "$BOOT_DEVICE"
```

Create an fstab entry and mount

```
cat << EOF >> /etc/fstab
$( blkid | grep "$BOOT_DEVICE" | cut -d ' ' -f 2 ) /boot/efi vfat defaults 0 0
EOF

mkdir -p /boot/efi
mount /boot/efi
```

Install ZFSBootMenu

Prebuilt

```
apt install curl
```

Fetch a prebuilt ZFSBootMenu EFI executable, saving it to the EFI system partition:

```
mkdir -p /boot/efi/EFI/ZBM
curl -o /boot/efi/EFI/ZBM/VMLINUZ.EFI -L https://get.zfsbootmenu.org/efi
cp /boot/efi/EFI/ZBM/VMLINUZ.EFI /boot/efi/EFI/ZBM/VMLINUZ-BACKUP.EFI
```

Source

Install all packages required to build a ZFSBootMenu image on Debian:

```
apt install \
  libsort-versions-perl \
  libboolean-perl \
  libyaml-pp-perl \
  git \
  fzf \
  mbuffer \
  kexec-tools \
  dracut-core \
  efibootmgr \
  systemd-boot-efi \
  bsdextrautils
```

Note: Choose 'No' when asked if kexec-tools should handle reboots.

Install all packages required to build a ZFSBootMenu image on Debian:

```
apt install \
  libsort-versions-perl \
  libboolean-perl \
  libyaml-pp-perl \
  git \
  fzf \
  mbuffer \
  kexec-tools \
  dracut-core \
  efibootmgr \
  systemd-boot-efi \
  bsdextrautils
```

Note: Choose 'No' when asked if kexec-tools should handle reboots.

Download the latest ZFSBootMenu source and install on the host:

```
mkdir -p /usr/local/src/zfsbootmenu
cd /usr/local/src/zfsbootmenu
```

(continues on next page)

(continued from previous page)

```
curl -L https://get.zfsbootmenu.org/source | tar -zxv --strip-components=1 -f -
make core dracut
```

Download the latest ZFSBootMenu source and install on the host:

```
mkdir -p /usr/local/src/zfsbootmenu
cd /usr/local/src/zfsbootmenu
curl -L https://get.zfsbootmenu.org/source | tar -zxv --strip-components=1 -f -
make core dracut
```

Configure *generate-zbm(5)* by ensuring that the following keys appear in `/etc/zfsbootmenu/config.yaml`:

```
Global:
  ManageImages: true
  BootMountPoint: /boot/efi
Components:
   Enabled: false
EFI:
  ImageDir: /boot/efi/EFI/zbm
  Versions: false
  Enabled: true
Kernel:
  CommandLine: quiet loglevel=0
```

Configure *generate-zbm(5)* by ensuring that the following keys appear in `/etc/zfsbootmenu/config.yaml`:

```
Global:
  ManageImages: true
  BootMountPoint: /boot/efi
Components:
   Enabled: false
EFI:
  ImageDir: /boot/efi/EFI/zbm
  Versions: false
  Enabled: true
Kernel:
  CommandLine: quiet loglevel=0
```

Create a ZFSBootMenu image:

```
generate-zbm
```

Configure EFI boot entries

```
mount -t efivarfs efivarfs /sys/firmware/efi/efivars
```

Direct

```
apt install efibootmgr
```

```
efibootmgr -c -d "$BOOT_DISK" -p "$BOOT_PART" \
  -L "ZFSBootMenu (Backup)" \
  -l \\EFI\\ZBM\\VMLINUZ-BACKUP.EFI

efibootmgr -c -d "$BOOT_DISK" -p "$BOOT_PART" \
  -L "ZFSBootMenu" \
  -l \\EFI\\ZBM\\VMLINUZ.EFI
```

rEFInd

```
apt install refind
```

```
refind-install
rm /boot/refind_linux.conf

cat << EOF > /boot/efi/EFI/zbm/refind_linux.conf
"Boot default"  "quiet loglevel=0 zbm.skip"
"Boot to menu"  "quiet loglevel=0 zbm.show"
EOF
```

See also:

Some systems can have issues with EFI boot entries. If you reboot and do not see the above entries in your EFI selection screen (usually accessible through an F key during POST), you might need to use a well-known EFI file name. See *Portable EFI* for help with this. Your existing ESP can be used, in place of an external USB drive.

Refer to *zbm-kcl.8* and *zfsbootmenu.7* for details on configuring the boot-time behavior of ZFSBootMenu.

51.1.9 Prepare for first boot

Exit the chroot, unmount everything

```
exit
```

```
umount -n -R /mnt
```

Export the zpool and reboot

```
zpool export zroot
reboot
```

CHAPTER

FIFTYTWO

UBUNTU

52.1 Jammy (22.04) UEFI

Contents

This guide can be used to install Ubuntu onto a single disk with or without ZFS encryption.

The end result will be a pristine Ubuntu install with no GUI or anything other than the base system. You'll be able to install *ubuntu-desktop*, *ubuntu-server-minimal* or whatever takes your fancy afterwards.

It assumes the following:

- Your system uses UEFI to boot
- Your system is x86_64
- You're mildly comfortable with ZFS, EFI and discovering system facts on your own (`lsblk`, `dmesg`, `gdisk`, ...)

Download the latest Ubuntu Desktop Jammy (22.04) Live image, write it to a USB drive and boot your system in EFI mode. You can use the server installation media if you want, although instructions are provided for installation using the desktop installer live session.

52.1.1 Configure Live Environment

Open a root shell

Open a terminal on the live installer session, then:

```
sudo -i
```

Confirm EFI support:

```
# dmesg | grep -i efivars
[    0.301784] Registered efivars operations
```

Source `/etc/os-release`

The file */etc/os-release* defines variables that describe the running distribution. In particular, the *$ID* variable defined within can be used as a short name for the filesystem that will hold this installation.

```
source /etc/os-release
export ID="$ID"
```

Install helpers

```
apt update
apt install debootstrap gdisk zfsutils-linux
```

Generate `/etc/hostid`

```
zgenhostid -f 0x00bab10c
```

52.1.2 Define disk variables

For convenience and to reduce the likelihood of errors, set environment variables that refer to the devices that will be configured during the setup.

For many users, it is most convenient to place boot files (*i.e.*, ZFSBootMenu and any loader responsible for launching it) on the the same disk that will hold the ZFS pool. However, some users may wish to dedicate an entire disk to the ZFS pool or create a multi-disk pool. A USB flash drive provides a convenient location for the boot partition. Fortunately, this alternative configuration is easily realized by simply defining a few environment variables differently.

Verify your target disk devices with `lsblk`. `/dev/sda` and `/dev/sdb` used below are examples.

First, define variables that refer to the disk and partition number that will hold **boot files**:

Single Disk

```
export BOOT_DISK="/dev/sda"
export BOOT_PART="1"
export BOOT_DEVICE="${BOOT_DISK}${BOOT_PART}"
```

Separate Boot Device

```
export BOOT_DISK="/dev/sdb"
export BOOT_PART="1"
export BOOT_DEVICE="${BOOT_DISK}${BOOT_PART}"
```

Next, define variables that refer to the disk and partition number that will hold the **ZFS pool**:

Single Disk

```
export POOL_DISK="/dev/sda"
export POOL_PART="2"
export POOL_DEVICE="${POOL_DISK}${POOL_PART}"
```

Separate Boot Device

```
export POOL_DISK="/dev/sda"
export POOL_PART="1"
export POOL_DEVICE="${POOL_DISK}${POOL_PART}"
```

52.1.3 Disk preparation

Wipe partitions

```
wipefs -a "$POOL_DISK"
wipefs -a "$BOOT_DISK"

sgdisk --zap-all "$POOL_DISK"
sgdisk --zap-all "$BOOT_DISK"
```

Create EFI boot partition

```
sgdisk -n "${BOOT_PART}:1m:+512m" -t "${BOOT_PART}:ef00" "$BOOT_DISK"
```

Create zpool partition

```
sgdisk -n "${POOL_PART}:0:-10m" -t "${POOL_PART}:bf00" "$POOL_DISK"
```

52.1.4 ZFS pool creation

Create the zpool

Unencrypted

```
zpool create -f -o ashift=12 \
 -O compression=lz4 \
 -O acltype=posixacl \
 -O xattr=sa \
 -O relatime=on \
 -o autotrim=on \
 -m none zroot "$POOL_DEVICE"
```

Encrypted

```
echo 'SomeKeyphrase' > /etc/zfs/zroot.key
chmod 000 /etc/zfs/zroot.key

zpool create -f -o ashift=12 \
 -O compression=lz4 \
 -O acltype=posixacl \
 -O xattr=sa \
 -O relatime=on \
 -O encryption=aes-256-gcm \
```

(continues on next page)

(continued from previous page)

```
 -O keylocation=file:///etc/zfs/zroot.key \
 -O keyformat=passphrase \
 -o autotrim=on \
 -m none zroot "$POOL_DEVICE"
```

Note: It's out of the scope of this guide to cover all of the pool creation options used - feel free to tailor them to suit your system. However, the following options need to be addressed:

- `encryption=aes-256-gcm` - You can adjust the algorithm as you see fit, but this will likely be the most performant on modern x86_64 hardware.
- `keylocation=file:///etc/zfs/zroot.key` - This sets our pool encryption passphrase to the file `/etc/zfs/zroot.key`, which we created in a previous step. This file will live inside your initramfs stored *on* the ZFS boot environment.
- `keyformat=passphrase` - By setting the format to `passphrase`, we can now force a prompt for this in `zfsbootmenu`. It's critical that your passphrase be something you can type on your keyboard, since you will need to type it in to unlock the pool on boot.

Enable zpool.cache

To more quickly discover and import pools on boot, we need to set a pool cachefile:

```
zpool set cachefile=/etc/zfs/zpool.cache zroot
```

Create initial file systems

```
zfs create -o mountpoint=none zroot/ROOT
zfs create -o mountpoint=/ -o canmount=noauto zroot/ROOT/${ID}
zfs create -o mountpoint=/home zroot/home

zpool set bootfs=zroot/ROOT/${ID} zroot
```

Note: It is important to set the property `canmount=noauto` on any file systems with `mountpoint=/` (that is, on any additional boot environments you create). Without this property, the OS will attempt to automount all ZFS file systems and fail when multiple file systems attempt to mount at `/`; this will prevent your system from booting. Automatic mounting of `/` is not required because the root file system is explicitly mounted in the boot process.

Also note that, unlike many ZFS properties, `canmount` is not inheritable. Therefore, setting `canmount=noauto` on `zroot/ROOT` is not sufficient, as any subsequent boot environments you create will default to `canmount=on`. It is necessary to explicitly set the `canmount=noauto` on every boot environment you create.

Export, then re-import with a temporary mountpoint of `/mnt`

Unencrypted

```
zpool export zroot
zpool import -N -R /mnt zroot
zfs mount zroot/ROOT/${ID}
zfs mount zroot/home
```

Encrypted

```
zpool export zroot
zpool import -N -R /mnt zroot
zfs load-key -L prompt zroot
```

```
zfs mount zroot/ROOT/${ID}
zfs mount zroot/home
```

Verify that everything is mounted correctly

```
# mount | grep mnt
zroot/ROOT/ubuntu on /mnt type zfs (rw,relatime,xattr,posixacl)
zroot/home on /mnt/home type zfs (rw,relatime,xattr,posixacl)
```

Update device symlinks

```
udevadm trigger
```

52.1.5 Install Ubuntu

```
debootstrap jammy /mnt
```

Copy files into the new install

Unencrypted

```
cp /etc/hostid /mnt/etc
cp /etc/resolv.conf /mnt/etc
mkdir /mnt/etc/zfs
cp /etc/zfs/zpool.cache /mnt/etc/zfs
```

Encrypted

```
cp /etc/hostid /mnt/etc/hostid
cp /etc/resolv.conf /mnt/etc/
mkdir /mnt/etc/zfs
cp /etc/zfs/zpool.cache /mnt/etc/zfs
cp /etc/zfs/zroot.key /mnt/etc/zfs
```

Chroot into the new OS

```
mount -t proc proc /mnt/proc
mount -t sysfs sys /mnt/sys
mount -B /dev /mnt/dev
mount -t devpts pts /mnt/dev/pts
chroot /mnt /bin/bash
```

52.1.6 Basic Ubuntu Configuration

Set a hostname

```
echo 'YOURHOSTNAME' > /etc/hostname
echo -e '127.0.1.1\tYOURHOSTNAME' >> /etc/hosts
```

Set a root password

```
passwd
```

Configure `apt`. Use other mirrors if you prefer.

```
cat <<EOF > /etc/apt/sources.list
# Uncomment the deb-src entries if you need source packages

deb http://archive.ubuntu.com/ubuntu/ jammy main restricted universe multiverse
# deb-src http://archive.ubuntu.com/ubuntu/ jammy main restricted universe multiverse

deb http://archive.ubuntu.com/ubuntu/ jammy-updates main restricted universe multiverse
# deb-src http://archive.ubuntu.com/ubuntu/ jammy-updates main restricted universe␣
↪multiverse

deb http://archive.ubuntu.com/ubuntu/ jammy-security main restricted universe multiverse
# deb-src http://archive.ubuntu.com/ubuntu/ jammy-security main restricted universe␣
↪multiverse

deb http://archive.ubuntu.com/ubuntu/ jammy-backports main restricted universe multiverse
# deb-src http://archive.ubuntu.com/ubuntu/ jammy-backports main restricted universe␣
↪multiverse

deb http://archive.canonical.com/ubuntu/ jammy partner
# deb-src http://archive.canonical.com/ubuntu/ jammy partner
EOF
```

Update the repository cache and system

```
apt update
apt upgrade
```

Install additional base packages

```
apt install --no-install-recommends linux-generic locales keyboard-configuration console-
↪setup
```

Note: The *--no-install-recommends* flag is used here to avoid installing recommended, but not strictly needed, packages (including *grub2*).

Configure packages to customize local and console properties

```
dpkg-reconfigure locales tzdata keyboard-configuration console-setup
```

Note: You should always enable the *en_US.UTF-8* locale because some programs require it.

52.1.7 ZFS Configuration

Install required packages

```
apt install dosfstools zfs-initramfs zfsutils-linux
```

Enable systemd ZFS services

```
systemctl enable zfs.target
systemctl enable zfs-import-cache
systemctl enable zfs-mount
systemctl enable zfs-import.target
```

Configure `initramfs-tools`

Unencrypted

No required steps

Encrypted

```
echo "UMASK=0077" > /etc/initramfs-tools/conf.d/umask.conf
```

Note: Because the encryption key is stored in `/etc/zfs` directory, it will automatically be copied into the system initramfs.

Rebuild the initramfs

```
update-initramfs -c -k all
```

52.1.8 Install and configure ZFSBootMenu

Set ZFSBootMenu properties on datasets

Unencrypted

Assign command-line arguments to be used when booting the final kernel. Because ZFS properties are inherited, assign the common properties to the `ROOT` dataset so all children will inherit common arguments by default.

```
zfs set org.zfsbootmenu:commandline="quiet loglevel=4" zroot/ROOT
```

Encrypted

Assign command-line arguments to be used when booting the final kernel. Because ZFS properties are inherited, assign the common properties to the `ROOT` dataset so all children will inherit common arguments by default.

```
zfs set org.zfsbootmenu:commandline="quiet loglevel=4" zroot/ROOT
```

Setup key caching in ZFSBootMenu.

```
zfs set org.zfsbootmenu:keysource="zroot/ROOT/${ID}" zroot
```

Create a `vfat` filesystem

```
mkfs.vfat -F32 "$BOOT_DEVICE"
```

Create an fstab entry and mount

```
cat << EOF >> /etc/fstab
$( blkid | grep "$BOOT_DEVICE" | cut -d ' ' -f 2 ) /boot/efi vfat defaults 0 0
EOF

mkdir -p /boot/efi
mount /boot/efi
```

Install ZFSBootMenu

Prebuilt

```
apt install curl
```

Fetch a prebuilt ZFSBootMenu EFI executable, saving it to the EFI system partition:

```
mkdir -p /boot/efi/EFI/ZBM
curl -o /boot/efi/EFI/ZBM/VMLINUZ.EFI -L https://get.zfsbootmenu.org/efi
cp /boot/efi/EFI/ZBM/VMLINUZ.EFI /boot/efi/EFI/ZBM/VMLINUZ-BACKUP.EFI
```

Source

Install all packages required to build a ZFSBootMenu image on Ubuntu:

```
apt install \
  libsort-versions-perl \
  libboolean-perl \
  libyaml-pp-perl \
  git \
  fzf \
  make \
  mbuffer \
  kexec-tools \
  dracut-core \
  efibootmgr \
  bsdextrautils
```

Note: Choose 'No' when asked if kexec-tools should handle reboots.

Install all packages required to build a ZFSBootMenu image on Ubuntu:

```
apt install \
  libsort-versions-perl \
  libboolean-perl \
  libyaml-pp-perl \
  git \
  fzf \
  make \
  mbuffer \
  kexec-tools \
  dracut-core \
  efibootmgr \
  bsdextrautils
```

Note: Choose 'No' when asked if kexec-tools should handle reboots.

Download the latest ZFSBootMenu source and install on the host:

```
mkdir -p /usr/local/src/zfsbootmenu
cd /usr/local/src/zfsbootmenu
```

(continues on next page)

(continued from previous page)

```
curl -L https://get.zfsbootmenu.org/source | tar -zxv --strip-components=1 -f -
make core dracut
```

Download the latest ZFSBootMenu source and install on the host:

```
mkdir -p /usr/local/src/zfsbootmenu
cd /usr/local/src/zfsbootmenu
curl -L https://get.zfsbootmenu.org/source | tar -zxv --strip-components=1 -f -
make core dracut
```

Configure *generate-zbm(5)* by ensuring that the following keys appear in `/etc/zfsbootmenu/config.yaml`:

```
Global:
  ManageImages: true
  BootMountPoint: /boot/efi
Components:
   Enabled: false
EFI:
  ImageDir: /boot/efi/EFI/zbm
  Versions: false
  Enabled: true
Kernel:
  CommandLine: quiet loglevel=0
```

Configure *generate-zbm(5)* by ensuring that the following keys appear in `/etc/zfsbootmenu/config.yaml`:

```
Global:
  ManageImages: true
  BootMountPoint: /boot/efi
Components:
   Enabled: false
EFI:
  ImageDir: /boot/efi/EFI/zbm
  Versions: false
  Enabled: true
Kernel:
  CommandLine: quiet loglevel=0
```

Create a ZFSBootMenu image:

```
generate-zbm
```

Configure EFI boot entries

```
mount -t efivarfs efivarfs /sys/firmware/efi/efivars
```

Direct

```
apt install efibootmgr
```

```
efibootmgr -c -d "$BOOT_DISK" -p "$BOOT_PART" \
  -L "ZFSBootMenu (Backup)" \
  -l \\EFI\\ZBM\\VMLINUZ-BACKUP.EFI

efibootmgr -c -d "$BOOT_DISK" -p "$BOOT_PART" \
  -L "ZFSBootMenu" \
  -l \\EFI\\ZBM\\VMLINUZ.EFI
```

rEFInd

```
apt install refind
```

```
refind-install
rm /boot/refind_linux.conf

cat << EOF > /boot/efi/EFI/zbm/refind_linux.conf
"Boot default"  "quiet loglevel=0 zbm.skip"
"Boot to menu"  "quiet loglevel=0 zbm.show"
EOF
```

See also:

Some systems can have issues with EFI boot entries. If you reboot and do not see the above entries in your EFI selection screen (usually accessible through an F key during POST), you might need to use a well-known EFI file name. See *Portable EFI* for help with this. Your existing ESP can be used, in place of an external USB drive.

Refer to *zbm-kcl.8* and *zfsbootmenu.7* for details on configuring the boot-time behavior of ZFSBootMenu.

52.1.9 Prepare for first boot

Exit the chroot, unmount everything

```
exit
```

```
umount -n -R /mnt
```

Export the zpool and reboot

```
zpool export zroot
reboot
```

CHAPTER

FIFTYTHREE

FEDORA

53.1 Workstation 37 UEFI

Contents

This guide can be used to install Fedora onto a single disk with or without ZFS encryption.

It assumes the following:

- Your system uses UEFI to boot
- Your system is x86_64
- You're mildly comfortable with ZFS, EFI and discovering system facts on your own (`lsblk`, `dmesg`, `gdisk`, ...)

Download Fedora Workstation Live , write it to a USB drive and boot your system in EFI mode.

Confirm EFI support:

```
# dmesg | grep -i efivars
[    0.301784] Registered efivars operations
```

53.1.1 Configure Live Environment

Switch to a root account

```
sudo -i
```

Source `/etc/os-release`

The file */etc/os-release* defines variables that describe the running distribution. In particular, the *$ID* variable defined within can be used as a short name for the filesystem that will hold this installation.

```
source /etc/os-release
export ID="$ID"
```

Install updated ZFS packages

```
rpm -e --nodeps zfs-fuse
dnf install -y https://zfsonlinux.org/fedora/zfs-release-2-2$(rpm --eval "%{dist}").
↪noarch.rpm
dnf install -y https://dl.fedoraproject.org/pub/fedora/linux/releases/${VERSION_ID}/
↪Everything/x86_64/os/Packages/k/kernel-devel-$(uname -r).rpm
dnf install -y zfs
modprobe zfs
```

Generate `/etc/hostid`

```
zgenhostid -f 0x00bab10c
```

53.1.2 Define disk variables

For convenience and to reduce the likelihood of errors, set environment variables that refer to the devices that will be configured during the setup.

For many users, it is most convenient to place boot files (*i.e.*, ZFSBootMenu and any loader responsible for launching it) on the the same disk that will hold the ZFS pool. However, some users may wish to dedicate an entire disk to the ZFS pool or create a multi-disk pool. A USB flash drive provides a convenient location for the boot partition. Fortunately, this alternative configuration is easily realized by simply defining a few environment variables differently.

Verify your target disk devices with `lsblk`. `/dev/sda` and `/dev/sdb` used below are examples.

First, define variables that refer to the disk and partition number that will hold **boot files**:

Single Disk

```
export BOOT_DISK="/dev/sda"
export BOOT_PART="1"
export BOOT_DEVICE="${BOOT_DISK}${BOOT_PART}"
```

Separate Boot Device

```
export BOOT_DISK="/dev/sdb"
export BOOT_PART="1"
export BOOT_DEVICE="${BOOT_DISK}${BOOT_PART}"
```

Next, define variables that refer to the disk and partition number that will hold the **ZFS pool**:

Single Disk

```
export POOL_DISK="/dev/sda"
export POOL_PART="2"
export POOL_DEVICE="${POOL_DISK}${POOL_PART}"
```

Separate Boot Device

```
export POOL_DISK="/dev/sda"
export POOL_PART="1"
export POOL_DEVICE="${POOL_DISK}${POOL_PART}"
```

53.1.3 Disk preparation

Wipe partitions

```
wipefs -a "$POOL_DISK"
wipefs -a "$BOOT_DISK"

sgdisk --zap-all "$POOL_DISK"
sgdisk --zap-all "$BOOT_DISK"
```

Create EFI boot partition

```
sgdisk -n "${BOOT_PART}:1m:+512m" -t "${BOOT_PART}:ef00" "$BOOT_DISK"
```

Create zpool partition

```
sgdisk -n "${POOL_PART}:0:-10m" -t "${POOL_PART}:bf00" "$POOL_DISK"
```

53.1.4 ZFS pool creation

Create the zpool

Unencrypted

```
zpool create -f -o ashift=12 \
 -O compression=lz4 \
 -O acltype=posixacl \
 -O xattr=sa \
 -O relatime=on \
 -o autotrim=on \
 -m none zroot "$POOL_DEVICE"
```

Encrypted

```
echo 'SomeKeyphrase' > /etc/zfs/zroot.key
chmod 000 /etc/zfs/zroot.key

zpool create -f -o ashift=12 \
 -O compression=lz4 \
 -O acltype=posixacl \
 -O xattr=sa \
 -O relatime=on \
 -O encryption=aes-256-gcm \
 -O keylocation=file:///etc/zfs/zroot.key \
 -O keyformat=passphrase \
 -o autotrim=on \
 -m none zroot "$POOL_DEVICE"
```

Note: It's out of the scope of this guide to cover all of the pool creation options used - feel free to tailor them to suit your system. However, the following options need to be addressed:

- `encryption=aes-256-gcm` - You can adjust the algorithm as you see fit, but this will likely be the most performant on modern x86_64 hardware.
- `keylocation=file:///etc/zfs/zroot.key` - This sets our pool encryption passphrase to the file `/etc/zfs/zroot.key`, which we created in a previous step. This file will live inside your initramfs stored *on* the ZFS boot environment.
- `keyformat=passphrase` - By setting the format to `passphrase`, we can now force a prompt for this in `zfsbootmenu`. It's critical that your passphrase be something you can type on your keyboard, since you will need to type it in to unlock the pool on boot.

Enable zpool.cache

To more quickly discover and import pools on boot, we need to set a pool cachefile:

```
zpool set cachefile=/etc/zfs/zpool.cache zroot
```

Create initial file systems

```
zfs create -o mountpoint=none zroot/ROOT
zfs create -o mountpoint=/ -o canmount=noauto zroot/ROOT/${ID}
zfs create -o mountpoint=/home zroot/home

zpool set bootfs=zroot/ROOT/${ID} zroot
```

Note: It is important to set the property `canmount=noauto` on any file systems with `mountpoint=/` (that is, on any additional boot environments you create). Without this property, the OS will attempt to automount all ZFS file systems and fail when multiple file systems attempt to mount at `/`; this will prevent your system from booting. Automatic mounting of `/` is not required because the root file system is explicitly mounted in the boot process.

Also note that, unlike many ZFS properties, `canmount` is not inheritable. Therefore, setting `canmount=noauto` on `zroot/ROOT` is not sufficient, as any subsequent boot environments you create will default to `canmount=on`. It is necessary to explicitly set the `canmount=noauto` on every boot environment you create.

Export, then re-import with a temporary mountpoint of `/mnt`

Unencrypted

```
zpool export zroot
zpool import -N -R /mnt zroot
zfs mount zroot/ROOT/${ID}
zfs mount zroot/home
```

Encrypted

```
zpool export zroot
zpool import -N -R /mnt zroot
zfs load-key -L prompt zroot
```

```
zfs mount zroot/ROOT/${ID}
zfs mount zroot/home
```

Verify that everything is mounted correctly

```
# mount | grep mnt
zroot/ROOT/fedora on /mnt type zfs (rw,relatime,xattr,posixacl)
zroot/home on /mnt/home type zfs (rw,relatime,xattr,posixacl)
```

Update device symlinks

```
udevadm trigger
```

53.1.5 Install Fedora

```
mkdir /run/install
mount /dev/mapper/live-base /run/install

rsync -pogAXtlHrDx \
 --stats \
 --exclude=/boot/efi/* \
 --exclude=/etc/machine-id \
 --info=progress2 \
 /run/install/ /mnt
```

Copy files into the new install

Unencrypted

```
mv /mnt/etc/resolv.conf /mnt/etc/resolv.conf.orig
cp -L /etc/resolv.conf /mnt/etc
cp /etc/hostid /mnt/etc
mkdir -p /mnt/etc/zfs
cp /etc/zfs/zpool.cache /mnt/etc/zfs
```

Encrypted

```
mv /mnt/etc/resolv.conf /mnt/etc/resolv.conf.orig
cp /etc/hostid /mnt/etc
cp -L /etc/resolv.conf /mnt/etc
mkdir -p /mnt/etc/zfs
cp /etc/zfs/zpool.cache /mnt/etc/zfs
cp /etc/zfs/zroot.key /mnt/etc/zfs
```

Chroot into the new OS

```
mount -t proc proc /mnt/proc
mount -t sysfs sys /mnt/sys
mount -B /dev /mnt/dev
mount -t devpts pts /mnt/dev/pts
chroot /mnt /bin/bash
```

53.1.6 ZFS Configuration

Configure Dracut to load ZFS support

Unencrypted

```
cat << EOF > /etc/dracut.conf.d/zol.conf
nofsck="yes"
add_dracutmodules+=" zfs "
omit_dracutmodules+=" btrfs "
EOF
```

Encrypted

```
cat << EOF > /etc/dracut.conf.d/zol.conf
nofsck="yes"
add_dracutmodules+=" zfs "
omit_dracutmodules+=" btrfs "
install_items+=" /etc/zfs/zroot.key "
EOF
```

Install required packages

```
source /etc/os-release

rpm -e --nodeps zfs-fuse

dnf install -y https://dl.fedoraproject.org/pub/fedora/linux/releases/${VERSION_ID}/
↪Everything/x86_64/os/Packages/k/kernel-devel-$(uname -r).rpm

dnf --releasever=${VERSION_ID} install -y \
  https://zfsonlinux.org/fedora/zfs-release-2-2$(rpm --eval "%{dist}").noarch.rpm

dnf install -y zfs zfs-dracut
```

53.1.7 Install and configure ZFSBootMenu

Set ZFSBootMenu properties on datasets

Unencrypted

Assign command-line arguments to be used when booting the final kernel. Because ZFS properties are inherited, assign the common properties to the ROOT dataset so all children will inherit common arguments by default.

```
zfs set org.zfsbootmenu:commandline="quiet loglevel=4 rhgb" zroot/ROOT
```

Encrypted

Assign command-line arguments to be used when booting the final kernel. Because ZFS properties are inherited, assign the common properties to the ROOT dataset so all children will inherit common arguments by default.

```
zfs set org.zfsbootmenu:commandline="quiet loglevel=4 rhgb" zroot/ROOT
```

Setup key caching in ZFSBootMenu.

```
zfs set org.zfsbootmenu:keysource="zroot/ROOT/${ID}" zroot
```

Create a vfat filesystem

```
mkfs.vfat -F32 "$BOOT_DEVICE"
```

Create an fstab entry and mount

```
cat << EOF >> /etc/fstab
$( blkid | grep "$BOOT_DEVICE" | cut -d ' ' -f 2 ) /boot/efi vfat defaults 0 0
EOF

mkdir -p /boot/efi
mount /boot/efi
```

Install ZFSBootMenu

Prebuilt

Fetch a prebuilt ZFSBootMenu EFI executable, saving it to the EFI system partition:

```
mkdir -p /boot/efi/EFI/ZBM
curl -o /boot/efi/EFI/ZBM/VMLINUZ.EFI -L https://get.zfsbootmenu.org/efi
cp /boot/efi/EFI/ZBM/VMLINUZ.EFI /boot/efi/EFI/ZBM/VMLINUZ-BACKUP.EFI
```

Source

Install all packages required to build a ZFSBootMenu image on Fedora:

```
dnf install -y \
  perl-YAML-PP \
  perl-Sort-Versions \
  perl-boolean \
  git \
  fzf \
  mbuffer \
  kexec-tools
```

Install all packages required to build a ZFSBootMenu image on Fedora:

```
dnf install -y \
  perl-YAML-PP \
  perl-Sort-Versions \
  perl-boolean \
  git \
  fzf \
  mbuffer \
  kexec-tools
```

Download the latest ZFSBootMenu source and install on the host:

```
mkdir -p /usr/local/src/zfsbootmenu
cd /usr/local/src/zfsbootmenu
curl -L https://get.zfsbootmenu.org/source | tar -zxv --strip-components=1 -f -
make core dracut
```

Download the latest ZFSBootMenu source and install on the host:

```
mkdir -p /usr/local/src/zfsbootmenu
cd /usr/local/src/zfsbootmenu
curl -L https://get.zfsbootmenu.org/source | tar -zxv --strip-components=1 -f -
make core dracut
```

Configure *generate-zbm(5)* by ensuring that the following keys appear in `/etc/zfsbootmenu/config.yaml`:

```
Global:
  ManageImages: true
  BootMountPoint: /boot/efi
Components:
   Enabled: false
EFI:
  ImageDir: /boot/efi/EFI/zbm
  Versions: false
  Enabled: true
Kernel:
  CommandLine: quiet loglevel=0
```

Configure *generate-zbm(5)* by ensuring that the following keys appear in `/etc/zfsbootmenu/config.yaml`:

```
Global:
  ManageImages: true
  BootMountPoint: /boot/efi
Components:
```

(continues on next page)

(continued from previous page)

```
   Enabled: false
EFI:
  ImageDir: /boot/efi/EFI/zbm
  Versions: false
  Enabled: true
Kernel:
  CommandLine: quiet loglevel=0
```

Create a ZFSBootMenu image:

```
generate-zbm
```

Configure EFI boot entries

```
mount -t efivarfs efivarfs /sys/firmware/efi/efivars
```

Direct

```
efibootmgr -c -d "$BOOT_DISK" -p "$BOOT_PART" \
  -L "ZFSBootMenu (Backup)" \
  -l \\EFI\\ZBM\\VMLINUZ-BACKUP.EFI

efibootmgr -c -d "$BOOT_DISK" -p "$BOOT_PART" \
  -L "ZFSBootMenu" \
  -l \\EFI\\ZBM\\VMLINUZ.EFI
```

rEFInd

```
dnf install -y refind
```

```
refind-install
rm /boot/refind_linux.conf

cat << EOF > /boot/efi/EFI/zbm/refind_linux.conf
"Boot default"  "quiet loglevel=0 zbm.skip"
"Boot to menu"  "quiet loglevel=0 zbm.show"
EOF
```

See also:

Some systems can have issues with EFI boot entries. If you reboot and do not see the above entries in your EFI selection screen (usually accessible through an F key during POST), you might need to use a well-known EFI file name. See *Portable EFI* for help with this. Your existing ESP can be used, in place of an external USB drive.

Refer to *zbm-kcl.8* and *zfsbootmenu.7* for details on configuring the boot-time behavior of ZFSBootMenu.

Reset resolv.conf

```
mv /etc/resolv.conf.orig /etc/resolv.conf
```

53.1.8 Prepare for first boot

Exit the chroot, unmount everything

```
exit
```

```
umount -n -R /mnt
```

Export the zpool and reboot

```
zpool export zroot
reboot
```

CHAPTER

FIFTYFOUR

OPENSUSE

54.1 Leap 15.4 UEFI

Contents

54.1.1 Preparation

This guide can be used to install openSUSE Leap onto a single disk with or without ZFS encryption.

It assumes the following:

- Your system uses UEFI to boot
- Your system is x86_64
- You're mildly comfortable with ZFS, EFI and discovering system facts on your own (`lsblk`, `dmesg`, `gdisk`, ...)

Download openSUSE Leap 15.4 , write it to a USB drive and boot your system in EFI mode.

Confirm EFI support:

```
# dmesg | grep -i efivars
[    0.301784] Registered efivars operations
```

54.1.2 Configure Live Environment

Disable automounting

```
gsettings set org.gnome.desktop.media-handling automount false
```

Switch to a root account

```
sudo -i
```

Source `/etc/os-release`

The file */etc/os-release* defines variables that describe the running distribution. In particular, the *$ID* variable defined within can be used as a short name for the filesystem that will hold this installation.

```
source /etc/os-release
export ID="$ID"
```

Enable filesystems repository

```
zypper -n addrepo https://download.opensuse.org/repositories/filesystems/$(lsb_release -
↪rs)/filesystems.repo
zypper refresh
```

Install updated ZFS packages

```
zypper -n install zfs zfs-kmp-default
modprobe zfs
```

Generate `/etc/hostid`

```
zgenhostid -f 0x00bab10c
```

54.1.3 Define disk variables

For convenience and to reduce the likelihood of errors, set environment variables that refer to the devices that will be configured during the setup.

For many users, it is most convenient to place boot files (*i.e.*, ZFSBootMenu and any loader responsible for launching it) on the the same disk that will hold the ZFS pool. However, some users may wish to dedicate an entire disk to the ZFS pool or create a multi-disk pool. A USB flash drive provides a convenient location for the boot partition. Fortunately, this alternative configuration is easily realized by simply defining a few environment variables differently.

Verify your target disk devices with `lsblk`. `/dev/sda` and `/dev/sdb` used below are examples.

First, define variables that refer to the disk and partition number that will hold **boot files**:

Single Disk

```
export BOOT_DISK="/dev/sda"
export BOOT_PART="1"
export BOOT_DEVICE="${BOOT_DISK}${BOOT_PART}"
```

Separate Boot Device

```
export BOOT_DISK="/dev/sdb"
export BOOT_PART="1"
export BOOT_DEVICE="${BOOT_DISK}${BOOT_PART}"
```

Next, define variables that refer to the disk and partition number that will hold the **ZFS pool**:

Single Disk

```
export POOL_DISK="/dev/sda"
export POOL_PART="2"
export POOL_DEVICE="${POOL_DISK}${POOL_PART}"
```

Separate Boot Device

```
export POOL_DISK="/dev/sda"
export POOL_PART="1"
export POOL_DEVICE="${POOL_DISK}${POOL_PART}"
```

54.1.4 Disk preparation

Wipe partitions

```
wipefs -a "$POOL_DISK"
wipefs -a "$BOOT_DISK"

sgdisk --zap-all "$POOL_DISK"
sgdisk --zap-all "$BOOT_DISK"
```

Create EFI boot partition

```
sgdisk -n "${BOOT_PART}:1m:+512m" -t "${BOOT_PART}:ef00" "$BOOT_DISK"
```

Create zpool partition

```
sgdisk -n "${POOL_PART}:0:-10m" -t "${POOL_PART}:bf00" "$POOL_DISK"
```

54.1.5 ZFS pool creation

Create the zpool

Unencrypted

```
zpool create -f -o ashift=12 \
 -O compression=lz4 \
 -O acltype=posixacl \
 -O xattr=sa \
 -O relatime=on \
 -o autotrim=on \
 -m none zroot "$POOL_DEVICE"
```

Encrypted

```
echo 'SomeKeyphrase' > /etc/zfs/zroot.key
chmod 000 /etc/zfs/zroot.key

zpool create -f -o ashift=12 \
 -O compression=lz4 \
 -O acltype=posixacl \
 -O xattr=sa \
 -O relatime=on \
 -O encryption=aes-256-gcm \
 -O keylocation=file:///etc/zfs/zroot.key \
 -O keyformat=passphrase \
 -o autotrim=on \
 -m none zroot "$POOL_DEVICE"
```

Note: It's out of the scope of this guide to cover all of the pool creation options used - feel free to tailor them to suit your system. However, the following options need to be addressed:

- `encryption=aes-256-gcm` - You can adjust the algorithm as you see fit, but this will likely be the most performant on modern x86_64 hardware.
- `keylocation=file:///etc/zfs/zroot.key` - This sets our pool encryption passphrase to the file `/etc/zfs/zroot.key`, which we created in a previous step. This file will live inside your initramfs stored *on* the ZFS boot environment.
- `keyformat=passphrase` - By setting the format to `passphrase`, we can now force a prompt for this in `zfsbootmenu`. It's critical that your passphrase be something you can type on your keyboard, since you will need to type it in to unlock the pool on boot.

Enable zpool.cache

To more quickly discover and import pools on boot, we need to set a pool cachefile:

```
zpool set cachefile=/etc/zfs/zpool.cache zroot
```

Create initial file systems

```
zfs create -o mountpoint=none zroot/ROOT
zfs create -o mountpoint=/ -o canmount=noauto zroot/ROOT/${ID}
zfs create -o mountpoint=/home zroot/home

zpool set bootfs=zroot/ROOT/${ID} zroot
```

Note: It is important to set the property `canmount=noauto` on any file systems with `mountpoint=/` (that is, on any additional boot environments you create). Without this property, the OS will attempt to automount all ZFS file systems and fail when multiple file systems attempt to mount at `/`; this will prevent your system from booting. Automatic mounting of `/` is not required because the root file system is explicitly mounted in the boot process.

Also note that, unlike many ZFS properties, `canmount` is not inheritable. Therefore, setting `canmount=noauto` on `zroot/ROOT` is not sufficient, as any subsequent boot environments you create will default to `canmount=on`. It is necessary to explicitly set the `canmount=noauto` on every boot environment you create.

Export, then re-import with a temporary mountpoint of `/mnt`

Unencrypted

```
zpool export zroot
zpool import -N -R /mnt zroot
zfs mount zroot/ROOT/${ID}
zfs mount zroot/home
```

Encrypted

```
zpool export zroot
zpool import -N -R /mnt zroot
zfs load-key -L prompt zroot
```

```
zfs mount zroot/ROOT/${ID}
zfs mount zroot/home
```

Verify that everything is mounted correctly

```
# mount | grep mnt
zroot/ROOT/leap on /mnt type zfs (rw,relatime,xattr,posixacl)
zroot/home on /mnt/home type zfs (rw,relatime,xattr,posixacl)
```

Update device symlinks

```
udevadm trigger
```

54.1.6 Install Leap

Enable software repositories

```
zypper --root /mnt ar https://download.opensuse.org/distribution/leap/$(lsb_release -rs)/
↪repo/non-oss non-oss
zypper --root /mnt ar https://download.opensuse.org/distribution/leap/$(lsb_release -rs)/
↪repo/oss oss
zypper --root /mnt ar https://download.opensuse.org/update/leap/$(lsb_release -rs)/oss␣
↪update-oss
zypper --root /mnt ar https://download.opensuse.org/update/leap/$(lsb_release -rs)/non-
↪oss update-nonoss
zypper --root /mnt ar https://download.opensuse.org/repositories/filesystems/$(lsb_
↪release -rs)/filesystems.repo
zypper --root /mnt refresh
```

Note: Enter **a** to always trust the key.

Add base packages

```
zypper --root /mnt install -t pattern enhanced_base
```

Add package management

```
zypper --root /mnt install zypper yast2
```

Copy files into the new install

Unencrypted

```
rm /mnt/etc/resolv.conf
cp -L /etc/resolv.conf /mnt/etc
cp /etc/hostid /mnt/etc
mkdir -p /mnt/etc/zfs
cp /etc/zfs/zpool.cache /mnt/etc/zfs
```

Encrypted

```
rm /mnt/etc/resolv.conf
cp /etc/hostid /mnt/etc
cp -L /etc/resolv.conf /mnt/etc
mkdir -p /mnt/etc/zfs
cp /etc/zfs/zpool.cache /mnt/etc/zfs
cp /etc/zfs/zroot.key /mnt/etc/zfs
```

Chroot into the new OS

```
mount -t proc proc /mnt/proc
mount -t sysfs sys /mnt/sys
mount -B /dev /mnt/dev
mount -t devpts pts /mnt/dev/pts
chroot /mnt /bin/bash
```

Basic system configuration

```
update-ca-certificates
echo 'LANG=en_US.UTF-8' > /etc/locale.conf
echo 'YOURHOSTNAME' > /etc/hostname
echo -e '127.0.1.1\tYOURHOSTNAME' >> /etc/hosts
passwd
```

54.1.7 ZFS Configuration

Configure Dracut to load ZFS support

Unencrypted

```
cat << EOF > /etc/dracut.conf.d/zol.conf
nofsck="yes"
add_dracutmodules+=" zfs "
omit_dracutmodules+=" btrfs "
EOF
```

Encrypted

```
cat << EOF > /etc/dracut.conf.d/zol.conf
nofsck="yes"
add_dracutmodules+=" zfs "
omit_dracutmodules+=" btrfs "
install_items+=" /etc/zfs/zroot.key "
EOF
```

Install kernel packages

```
zypper -n install kernel-default kernel-firmware
```

Install ZFS

```
zypper -n install zfs zfs-kmp-default
```

Build Kernel Modules

```
dracut --regenerate-all --force
```

54.1.8 Install and configure ZFSBootMenu

Set ZFSBootMenu properties on datasets

Unencrypted

Assign command-line arguments to be used when booting the final kernel. Because ZFS properties are inherited, assign the common properties to the ROOT dataset so all children will inherit common arguments by default.

```
zfs set org.zfsbootmenu:commandline="quiet loglevel=4 rhgb" zroot/ROOT
```

Encrypted

Assign command-line arguments to be used when booting the final kernel. Because ZFS properties are inherited, assign the common properties to the ROOT dataset so all children will inherit common arguments by default.

```
zfs set org.zfsbootmenu:commandline="quiet loglevel=4 rhgb" zroot/ROOT
```

Setup key caching in ZFSBootMenu.

```
zfs set org.zfsbootmenu:keysource="zroot/ROOT/${ID}" zroot
```

Create a vfat filesystem

```
mkfs.vfat -F32 "$BOOT_DEVICE"
```

Create an fstab entry and mount

```
cat << EOF >> /etc/fstab
$( blkid | grep "$BOOT_DEVICE" | cut -d ' ' -f 2 ) /boot/efi vfat defaults 0 0
EOF

mkdir -p /boot/efi
mount /boot/efi
```

Install ZFSBootMenu

Prebuilt

Fetch a prebuilt ZFSBootMenu EFI executable, saving it to the EFI system partition:

```
mkdir -p /boot/efi/EFI/ZBM
curl -o /boot/efi/EFI/ZBM/VMLINUZ.EFI -L https://get.zfsbootmenu.org/efi
cp /boot/efi/EFI/ZBM/VMLINUZ.EFI /boot/efi/EFI/ZBM/VMLINUZ-BACKUP.EFI
```

Source

Install all packages required to build a ZFSBootMenu image on Fedora:

```
zypper -n install -y \
  perl-YAML-PP \
  perl-Sort-Versions \
  perl-boolean \
  git \
  fzf \
  mbuffer \
  kexec-tools
```

Install all packages required to build a ZFSBootMenu image on Fedora:

```
zypper -n install -y \
  perl-YAML-PP \
  perl-Sort-Versions \
  perl-boolean \
  git \
  fzf \
  mbuffer \
  kexec-tools
```

Download the latest ZFSBootMenu source and install on the host:

```
mkdir -p /usr/local/src/zfsbootmenu
cd /usr/local/src/zfsbootmenu
curl -L https://get.zfsbootmenu.org/source | tar -zxv --strip-components=1 -f -
make core dracut
```

Download the latest ZFSBootMenu source and install on the host:

```
mkdir -p /usr/local/src/zfsbootmenu
cd /usr/local/src/zfsbootmenu
curl -L https://get.zfsbootmenu.org/source | tar -zxv --strip-components=1 -f -
make core dracut
```

Configure *generate-zbm(5)* by ensuring that the following keys appear in `/etc/zfsbootmenu/config.yaml`:

```
Global:
  ManageImages: true
  BootMountPoint: /boot/efi
Components:
   Enabled: false
```

(continues on next page)

(continued from previous page)

```
EFI:
  ImageDir: /boot/efi/EFI/zbm
  Versions: false
  Enabled: true
Kernel:
  CommandLine: quiet loglevel=0
```

Configure *generate-zbm(5)* by ensuring that the following keys appear in `/etc/zfsbootmenu/config.yaml`:

```
Global:
  ManageImages: true
  BootMountPoint: /boot/efi
Components:
   Enabled: false
EFI:
  ImageDir: /boot/efi/EFI/zbm
  Versions: false
  Enabled: true
Kernel:
  CommandLine: quiet loglevel=0
```

Create a ZFSBootMenu image:

```
generate-zbm
```

Configure EFI boot entries

```
mount -t efivarfs efivarfs /sys/firmware/efi/efivars
```

```
zypper -n install efibootmgr
```

Direct

```
efibootmgr -c -d "$BOOT_DISK" -p "$BOOT_PART" \
  -L "ZFSBootMenu (Backup)" \
  -l \\EFI\\ZBM\\VMLINUZ-BACKUP.EFI

efibootmgr -c -d "$BOOT_DISK" -p "$BOOT_PART" \
  -L "ZFSBootMenu" \
  -l \\EFI\\ZBM\\VMLINUZ.EFI
```

rEFInd

```
zypper -n install refind
```

```
refind-install
rm /boot/refind_linux.conf

cat << EOF > /boot/efi/EFI/zbm/refind_linux.conf
"Boot default"  "quiet loglevel=0 zbm.skip"
```

(continues on next page)

(continued from previous page)

```
"Boot to menu"  "quiet loglevel=0 zbm.show"
EOF
```

See also:

Some systems can have issues with EFI boot entries. If you reboot and do not see the above entries in your EFI selection screen (usually accessible through an F key during POST), you might need to use a well-known EFI file name. See *Portable EFI* for help with this. Your existing ESP can be used, in place of an external USB drive.

Refer to *zbm-kcl.8* and *zfsbootmenu.7* for details on configuring the boot-time behavior of ZFSBootMenu.

54.1.9 Prepare for first boot

Exit the chroot, unmount everything

```
exit
```

```
umount -n -R /mnt
```

Export the zpool and reboot

```
zpool export zroot
reboot
```

CHAPTER

FIFTYFIVE

THIRD-PARTY RESOURCES

The ZFSBootMenu Wiki hosts several user-contributed guides.

The following links are not hosted by ZFSBootMenu, but may provide useful information.

55.1 Void Linux

- Void Linux ZFSBootMenu install scripts

55.2 Ubuntu

- Ubuntu server ZFSBootMenu install script
- Ubuntu ZFSBootMenu install script - desktop and server

55.3 Arch Linux

- Arch Wiki: Using ZFSBootMenu for UEFI
- Arch User Repository: ZFSBootMenu
- Arch User Repository: ZFSBootMenu EFI binaries
- Arch Linux ZFSBootMenu install scripts

55.4 Fedora

- Install and boot Fedora with ZFSBootMenu

CHAPTER

FIFTYSIX

MAIN SCREEN

56.1 Keyboard Shortcuts

[ENTER] **boot**

Boot the selected boot environment, with the listed kernel and kernel command line visible at the top of the screen.

[MOD+K] **kernels**

Access a list of kernels available in the boot environment.

[MOD+S] **snapshots**

Access a list of snapshots of the selected boot environment. New boot environments can be created here.

[MOD+D] **set bootfs**

Set the selected boot environment as the default for the pool.

The operation will fail gracefully if the pool can not be set *read/write*.

[MOD+E] **edit kcl**

Temporarily edit the kernel command line that will be used to boot the chosen kernel in the selected boot environment. This change does not persist across reboots.

[MOD+T] **revert kcl**

Revert the temporary kernel command line set via *[MOD+E]*.

[MOD+P] **pool status**

View the health and status of each imported pool.

[MOD+R] **recovery shell**

Execute a Bash shell with minimal tooling, enabling system maintenance.

[MOD+J] **jump into chroot**

Enter a chroot of the selected boot environment. The boot environment is mounted *read/write* if the zpool is imported *read/write*.

[MOD+W] **import read/write**

If possible, the pool behind the selected boot environment is exported and then re-imported in *read/write* mode.

This is not possible if any of the following conditions are met:

- The version of ZFS in ZFSBootMenu has detected unsupported pool features, due to an upgraded pool.
- The system has an active **resume**, indicating that the pool is currently in use.

Upon successful re-import in *read/write* mode, each of the boot environments on this pool will be highlighted in *red* at the top of the screen.

[MOD+O] **sort order**

Cycle the sorting key through the following list:

- **name** Use the filesystem or snapshot name
- **creation** Use the filesystem or snapshot creation time
- **used** Use the filesystem or snapshot size

The default sort key is *name*.

[MOD+L] **view logs**

View logs, as indicated by *[!]*. The indicator will be yellow for warning conditions and red for errors.

CHAPTER

FIFTYSEVEN

SNAPSHOT MANAGEMENT

57.1 Keyboard Shortcuts

[ENTER] **duplicate**

Creation method: *zfs send | zfs recv*

This creates a boot environment that does not depend on any other snapshots, allowing it to be destroyed at will. The new boot environment will immediately consume space on the pool equal to the *REFER* value of the snapshot.

A duplicated boot environment is commonly used if you need a new boot environment without any associated snapshots.

The operation will fail gracefully if the pool can not be set *read/write*.

If *mbuffer* is available, it is used to provide feedback.

[MOD+X] **clone and promote**

Creation method: *zfs clone , zfs promote*

This creates a boot environment that is not dependent on the origin snapshot, allowing you to destroy the file system that the clone was created from. A cloned and promoted boot environment is commonly used when you've done an upgrade but want to preserve historical snapshots.

The operation will fail gracefully if the pool can not be set *read/write*.

[MOD+C] **clone**

Creation method: *zfs clone*

This creates a boot environment from a snapshot with out modifying snapshot inheritence. A cloned boot environment is commonly used if you need to boot a previous system state for a short time and then discard the environment.

The operation will fail gracefully if the pool can not be set *read/write*.

[MOD+N] **snapshot creation**

This creates a new snapshot of the currently selected boot environment. A new snapshot is useful if you need to repair a boot environment from a chroot, to allow for easy roll-back of the changes.

The operation will fail gracefully if the pool can not be set *read/write*.

[MOD+D] **diff**

Compare the differences between snapshots and filesystems. A single snapshot can be selected and a diff will be generated between that and the current state of the filesystem. Two snapshots can be selected (and deselected) with the tab key and a diff will be generated between them.

The operation will fail gracefully if the pool can not be set *read/write*.

[MOD+J] **jump into chroot**

Enter a chroot of the selected boot environment snapshot. The snapshot is always mounted read-only.

[MOD+O] **sort order**

Cycle the sorting key through the following list:

- **name** Use the filesystem or snapshot name
- **creation** Use the filesystem or snapshot creation time
- **used** Use the filesystem or snapshot size

The default sort key is *name*.

[MOD+L] **view logs**

View logs, as indicated by *[!]*. The indicator will be yellow for warning conditions and red for errors.

[MOD+R] **roll back snapshot**

Roll back a boot environment to the selected snapshot. This is a destructive operation that will not proceed without affirmative confirmation.

CHAPTER

FIFTYEIGHT

DIFF VIEWER

58.1 Keyboard Shortcuts

Column 1 descriptions

- The path has been removed

+ The path has been created

M The path has been modified

R The path has been renamed

Column 2 descriptions

B Block device

C Character device

/ Directory

> Door

| Named pipe

@ Symbolic link

P Event port

= Socket

F Regular file

CHAPTER

FIFTYNINE

KERNEL MANAGEMENT

59.1 Keyboard Shortcuts

[ENTER] **boot**

Immediately boot the chosen kernel in the selected boot environment, with the kernel command line shown at the top of the screen.

[MOD+D] **set default**

Set the selected kernel as the default for the boot environment.

The ZFS property *org.zfsbootmenu:kernel* is used to store the default kernel for the boot environment.

The operation will fail gracefully if the pool can not be set *read/write*.

[MOD+U] **unset default**

Inherit the ZFS property *org.zfsbootmenu:kernel* from a parent if present, otherwise unset the property.

The operation will fail gracefully if the pool can not be set *read/write*.

[MOD+L] **view logs**

View logs, as indicated by *[!]*. The indicator will be yellow for warning conditions and red for errors.

CHAPTER

SIXTY

ZPOOL HEALTH

60.1 Keyboard Shortcuts

[MOD+R] **rewind checkpoint**

If a pool checkpoint is available, the selected pool is exported and then imported with the *--rewind-to-checkpoint* flag set.

The operation will fail gracefully if the pool can not be set *read/write*.

[MOD+L] **view logs**

View logs, as indicated by *[!]*. The indicator will be yellow for warning conditions and red for errors.

CHAPTER

SIXTYONE

RECOVERY SHELL

61.1 Common Commands

zfsbootmenu | zbm

Launch the interactive boot environment menu.

zfs-chroot *dataset*

Enter a chroot of the specified boot environment. The boot environment is mounted *read/write* if the zpool is imported *read/write*.

zkexec *dataset kernel initramfs*

Directly *kexec* a kernel and initramfs from a boot environment, allowing any kernel and initramfs to be loaded into memory and immediately booted.

zreport

List ZFS module, pool and dataset details for bug reports.

zbmcmdline

Show the aggregated commandline from */etc/cmdline*, */etc/cmdline.d/* and */proc/cmdline*.

set_rw_pool *pool*

Export, then re-import the pool *read/write*.

set_ro_pool *pool*

Export, then re-import the pool *read-only*.

mount_zfs *dataset*

Mount the filesystem at a unique location and print the mount point.

mount_efivarfs *mode*

Mount or remount *efivarfs* as read-write or read-only.

help

View the online help system.

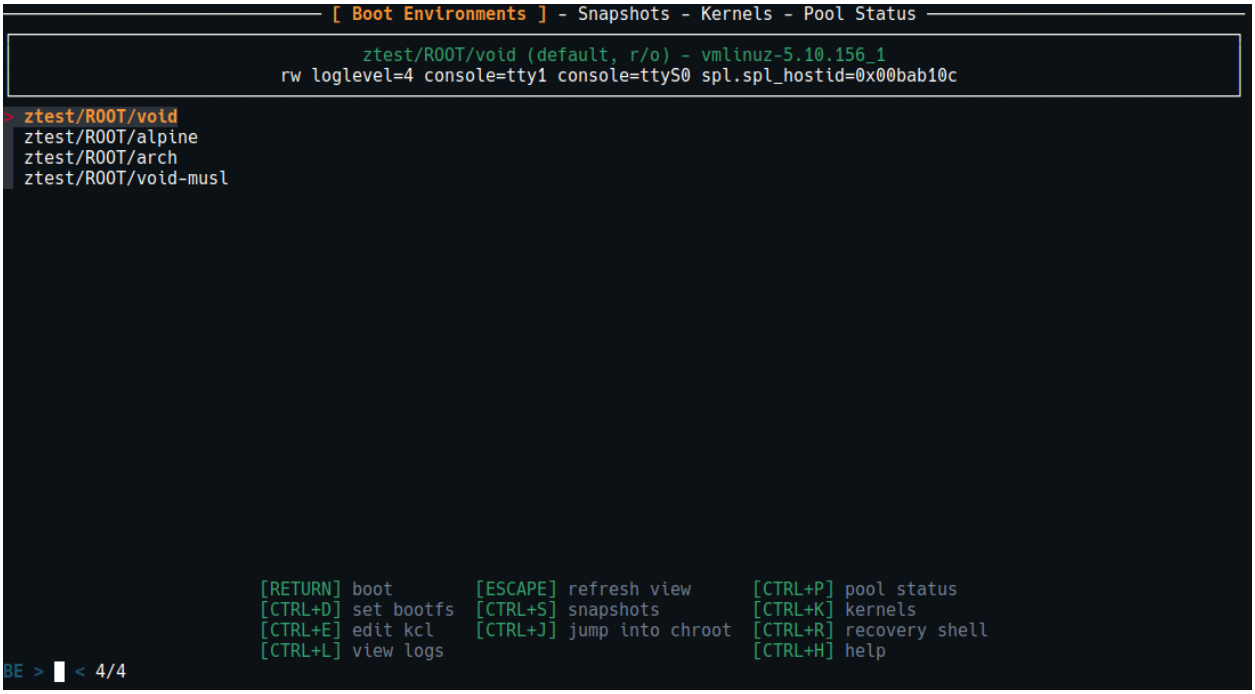
logs

View warning/error/debug logs.

ZFSBootMenu is a bootloader that provides a powerful and flexible discovery, manipulation and booting of Linux on ZFS. Originally inspired by the FreeBSD bootloader, ZFSBootMenu leverages the features of modern OpenZFS to allow users to choose among multiple "boot environments" (which may represent different versions of a Linux

distribution, earlier snapshots of a common root, or entirely different distributions), manipulate snapshots in a pre-boot environment and, for the adventurous user, even bootstrap a system installation via `zfs recv`.

In essence, ZFSBootMenu is a small, self-contained Linux system that knows how to find other Linux kernels and initramfs images within ZFS filesystems. When a suitable kernel and initramfs are identified (either through an automatic process or direct user selection), ZFSBootMenu launches that kernel using the `kexec` command.



CHAPTER

SIXTYTWO

OVERVIEW

In broad strokes, ZFSBootMenu works as follows:

- Via direct EFI booting, an EFI boot manager like `rEFInd`, a BIOS bootloader like `syslinux`, or some other means, boot a ZFSBootMenu image (as either a self-contained UEFI application or a dedicated Linux kernel and initramfs).
- Find all healthy ZFS pools and import them (or, at the user's option, find and import only a specific pool).
- If appropriate, select a preferred boot environment:
  - If the ZFSBootMenu command line specifies a pool preference, and that pool has been imported, prefer the filesystem indicated by its `bootfs` property (if defined).
  - If the ZFSBootMenu command line specifies no pool preference or the preferred pool is not found, prefer the filesystem indicated by the `bootfs` property (if defined) on the first-found pool.
  - If a suitable `bootfs` has been identified, start an interruptable countdown (by default, 10 seconds) to automatically boot that environment.
  - If no `bootfs` value can be identified or the automatic countdown was interrupted, search all imported pools for filesystems that set `mountpoint=/` and contain Linux kernels and initramfs images in their `/boot` subdirectories. Present a list of matching environments for user selection via `fzf`.
- Mount the filesystem representing the selected boot environment and find either the highest versioned kernel or a specifically selected kernel version in its `/boot` directory.
- Using `kexec`, load the selected kernel and its initramfs image into memory, setting the kernel command line with the contents of the `org.zfsbootmenu:commandline` property for that filesystem.
- Unmount all ZFS filesystems.
- Boot the final kernel and initramfs.

At this point, the system will be booting into your usual OS-managed kernel and initramfs, along with any arguments needed to correctly boot your system.

Whenever ZFSBootMenu encounters natively encrypted ZFS filesystems that it intends to scan for boot environments, it will prompt the user to enter a passphrase as necessary.

62.1 Distribution Agnostic

ZFSBootMenu is capable of booting just about any Linux distribution. Major distributions that are known to boot without requiring any special configuration include:

- Void
- Arch
- Alpine
- Gentoo
- Fedora
- Debian and its descendants (Ubuntu, Linux Mint, Devuan, etc.)

Red Hat and its descendants (RHEL, CentOS, etc.) are expected to work as well but have never been tested.

ZFSBootMenu provides several configuration options that can be used to fine-tune the boot process for nonstandard configurations.

62.2 Easily Deployed and Extensively Configurable

Each release includes pre-generated boot images, based on Void Linux, that should work for the majority of users. Images are distributed for `x86_64` platforms both as monolithic UEFI applications as well a separate kernel an initramfs image that are suitable for use on both UEFI and legacy BIOS systems. Users of other platforms or that require custom configurations can build local images, running the ZFSBootMenu image generator either in a host installation or in the controlled environment of an OCI (Docker) container.

Modern UEFI platforms provide a wide range of *options for launching ZFSBootmenu*. For legacy BIOS systems, `syslinux` is a convenient choice. A *syslinux guide for Void Linux* describes the `syslinux` installation and configuration process in the context of a broader Void Linux installation.

62.2.1 Local Installation

The ZFSBootMenu repository includes a Makefile with targets to install the *generate-zbm* builder, all necessary components, manual pages and some convenient helpers. A local ZFSBootMenu installation requires some additional software that may be available as packages in your distribution or may need to be manually installed. The following components are required or recommended for inclusion in the bootloader image:

- fzf
- kexec-tools
- mbuffer (recommended, but not required)

In addition, `generate-zbm` requires a few Perl modules:

- perl Sort::Versions
- perl YAML::PP

- perl boolean

If you will create unified EFI executables (which bundles the kernel, initramfs and command line), you will also need a an EFI stub loader, which is typically included with systemd-boot or gummiboot.

Most or all of these software components may be available as packages in your distribution.

Locally created ZFSBootMenu images use your your regular system kernel, ZFS drivers and user-space utilities. The ZFSBootMenu image is constructed using standard Linux initramfs generators. ZFSBootMenu is known to work and is explicitly supported with:

- dracut
- mkinitcpio

Building a custom image is known to work in the following configurations:

- With `mkinitcpio` or `dracut` on Void (the `zfsbootmenu` package will make sure all prerequisites are available)
- With `mkinitcpio` or `dracut` on Arch
- With `dracut` on Debian or Ubuntu (installed as `dracut-core` to avoid replacing the system `initramfs-tools` setup)

Configuration of the ZFSBootMenu build process is accomplished via a *YAML configuration file* for `generate-zbm`.

62.2.2 Building in a Container

The official ZFSBootMenu release images are built in a standard Void Linux OCI container that provides a predictable environment that is known to be supported with ZFSBootMenu. The container entrypoint provides full access to all of the configurability of ZFSBootMenu, and a helper script simplifies the process or running the container and managing the images that it produces. The *ZFSBootMenu container guide* provides a detailed description of the containerized build process as well as a straightforward example of local image management using the helper script.

62.3 ZFS Boot Environments

The concept of a "boot environment" is very loosely defined in ZFSBootMenu. Fundamentally, ZFSBootMenu treats any filesystem that appears to be an operating system root and contains an identifiable Linux kernel and initramfs as a boot environment. A *primer* provides more details about the identification process.

62.3.1 Command-Line Arguments

When booting a particular enviornment, ZFSBootMenu reads the `org.zfsbootmenu:commandline` *property* for that filesystem to discover kernel command-line arguments that should be passed to the kernel it will boot.

Note: Do not set a `root=` option (or any similar indicator of the root filesystem) in this property; ZFSBootMenu will add an appropriate `root=` argument when it boots the environment and will actively suppress any conflicting option.

Because ZFS properties are inherited by default, it is possible to set the `org.zfsbootmenu:commandline` property on a common parent to apply the same KCL arguments to multiple environments. Setting the property locally on individual boot environments will override the common defaults.

As a special accommodation, the substitution keyword `%{parent}` in the KCL property will be recursively expanded to whatever the value of `org.zfsbootmenu:commandline` would be on the parent dataset. This allows, for example, mixing options common to multiple environments with those specific to each:

```
zfs set org.zfsbootmenu:commandline=""zfs.zfs_arc_max=8589934592"" zroot/ROOT
zfs set org.zfsbootmenu:commandline="%{parent} loglevel=4" zroot/ROOT/void.2019.11.01
zfs set org.zfsbootmenu:commandline="loglevel=7 %{parent}" zroot/ROOT/void.2019.10.04
```

will cause ZFSBootMenu to interpret the KCL for `zroot/ROOT/void.2019.11.01` as:

```
zfs.zfs_arc_max=8589934592 loglevel=4
```

while the KCL for `zroot/ROOT/void.2019.10.04` would be:

```
loglevel=7 zfs.zfs_arc_max=8589934592
```

To simplify the manipulation of command-line parameters for boot environments, the *zbm-kcl* helper facilitates live review and edits.

62.4 Run-time Configuration of ZFSBootMenu

ZFSBootMenu may be configured via a combination of *command-line parameters* and *ZFS properties* that are described in detail in the *zfsbootmenu(7)* manual page. For users of pre-built UEFI executables, the *zbm-kcl* helper script provides a convenient way to modify the embedded ZFSBootMenu command line without requiring the creation of a custom image.

62.5 Signature Verification and Prebuilt EFI Executables

All release assets, including EFI executables and kernel/initramfs pairs, are signed with signify, which provides a simple method for verifying that the contents of the file are as this project intended. Once you've installed `signify` (that's left as an exercise, although Void Linux provides the `signify` package for this purpose), just download the desired assets from the ZFSBootMenu release page, download the file `sha256.sig` alongside it, and run:

```
signify -C -x sha256.sig
```

You will need the public key used to sign ZFSBootMenu executables. The key is available at releng/keys/zfsbootmenu.pub. Install this file as `/etc/signify/zfsbootmenu.pub` if you like; this key can be used for all subsequent verifications. Otherwise, look at the `-p` command-line option for `signify` to provide a path to the key.

The signature file `sha256.sig` also includes a signature for the source tarball corresponding to the release. If this file is not present alongside the EFI bundle and the signature file, `signify` will complain about its signature. This error message is OK to ignore; alternatively, tell `signify` to verify only the EFI bundle, or download the source tarball alongside the other files.

The signify key `zfsbootmenu.pub` may itself be verified; alongside the public key is the GPG signature releng/keys/zfsbootmenu.pub.gpg, produced with a personal key from @ahesford, one of the members of the ZFSBootMenu project. This personal key is also available on public key servers. To verify the `signify` key, download the key `zfsbootmenu.pub` and its signature file `zfsbootmenu.pub.gpg`, then run:

```
gpg --recv-key 0x312485BE75E3D7AC
gpg --verify zfsbootmenu.pub.gpg
```

Note: On some distributions, `gpg` may instead by `gpg2`.