The FreeBSD Appliance

Leveraging FreeBSD and Strategic Scripting to Deliver Storage and Virtualization Services

Michael Dexter

AsiaBSDCon 2023, Tokyo, Japan

Abstract

The FreeBSD Operating System has traditionally been viewed as a complete server and desktop solution or a collection of core components for commercial appliance development. It has benefited from decades of academic, volunteer, and vendor contribution of core components including its TCP/IP stack, multiple packet filters, Jail containers, the CAM/CTL storage infrastructure, VNET and Netgraph virtual network stacks, the OpenZFS file system and volume manager, and the bhyve hypervisor, all with a unified source tree and build environment. Many of these components have enabled high-profile storage and networking product ecosystems but less-obvious developments are regularly occurring: FreeBSD is experiencing extensive refinement in addition to major feature development, making for an unprecedented "out of the box" user experience. FreeBSD 14.0-RELEASE will include subtle but powerful features including:

- Extensive compilation options
- Reproducible Builds
- `makefs(8) -t zfs`
- Jailed `nfsd(8)` and bhyve
- Emerging Packaged Base
- `nullfs(5)-f` file mounts
- `CTL(4)`/`virtio_scsi(4)` support
- Expanded `libxo(3)` and `nvlist(9)` support

This paper describes the new abilities enabled by these small and seemingly-unrelated features and their ability to reduce the need for highly-customized FreeBSD appliance distributions. It will also describe strategies for following the "CURRENT" development branch of FreeBSD without becoming a full-time release engineer. Finally, it will outline how contemporary FreeBSD provides a meaningful storage and virtualization platform with minimal supplementary utilities.

Introduction

There are many ways to view an open source operating system: As a “product” that happens to be “free as in freedom”, as a platform on which other projects and products are built, or as a collection of production-ready implementations of open standards including RFCs. The FreeBSD operating system and its ancestral Berkeley Software Distribution appease each of these perspectives resulting in equally visible and invisible use of them. Common to every use of a flexible operating system like FreeBSD is the building, or compilation of the operating system, and the Release Engineering or delivery of the installable results. FreeBSD has an established and documented build and release process but has steadily incorporated small but significant refinements that together allow for unprecedented flexibility for alternative building and releasing strategies using in-base facilities. Alternative or custom releases of FreeBSD can include minimal and single-purpose systems, computer science education-oriented systems, and appliances delivering network services.

The FreeBSD Release Process

Like most “BSDs”, FreeBSD is a self-hosted operating system with a unified source tree that is officially augmented with ported third party software or “ports” which are pre-compiled into “packages”. The FreeBSD source tree is generally compiled as a whole using the procedures described in `build(7)`. The build supports cross-compilation for different instruction set architectures and the results are archived as “distribution sets”. These sets include `base.txz` for the base operating system, `src.txz` for the corresponding sources, and supplemental archives such as debugging information and snapshots of the ported software and documentation source trees. The distribution sets are in turn installed by the installer `bsdinstall(8)` which assists with system configuration via an `ncurses(3X)`-based user interface. Alternatively, the operating system can be installed directly from its build objects using a sequence of `make(1)` commands as described in `build(7)`. Transparent to the user are hundreds of build options and kernel configuration options that determine what components are included, excluded, or modified.

FreeBSD Compilation Options

The FreeBSD compilation options take the form of:

- `make(1)` options for traditional `make(1)` parameters that control the build
- `src-env.conf(5)` `environ(7)` options such as `WITH_META_MODE`
- Kernel `MODULES_OVERRIDE` described in `make.conf(5)`
- Kernel `options` such as `TMPFS`/`tmpfs(5)`
- Kernel `device`(s) such as `em`/`em(4)`
- Build options described in `src.conf(5)` such as `WITH_LLDB` and `WITHOUT_LLDB`

`make(1)` options follow the conventions established in the 1970's. `src-env.conf(5)` "source environments" options control the `make(1)` environment with options such as `WITH_META_MODE` that reuses cached build artifacts to only recompile changed components, making for significantly faster builds. `MODULES_OVERRIDE` describes what select modules should only be compiled. Kernel `options` determine what features are compiled into the kernel such as the `tmpfs(5)` memory-backed file system. Kernel `device`(s) determine what hardware or virtual device drivers should be compiled, such as the `em(4)` hardware Ethernet driver. Finally, the build options described in `src.conf(5)` determine what userland components should or should not be compiled.

Official FreeBSD Releases represent a curated collection of the above compilation options and a user can recreate a FreeBSD `release(7)` including installation media with the following commands:

```
cd /usr/src
make buildworld buildkernel
cd /usr/src/release
make release
```

These steps produce binary objects in `/usr/obj` and support concurrent compilation jobs with the `-j` flag to `make(1)` which should not exceed the available number of CPU threads. The authenticity of the resulting release is further guaranteed with the `WITH_REPRODUCIBLE_BUILD` build option, which follows the guidelines of the Reproducible Builds[1] project. The net result is a build strategy that is highly consistent but the inherent flexibility of which is largely unexplored.

`makefs(8) -t zfs`

While the FreeBSD `build(7)` relies primarily on on `make(1)` and `cc(1)` and can be built on other operating systems[2], the `release(7)` process relies on additional utilities such as `makefs(8)` to perform additional steps, namely producing bootable installation media and "virtual" machine images in a variety of formats including CD-ROM, "memstick", .qcow2, RAW, VHD, and VMDK. Of the "RAW" virtual machine image exhibits a number of unique qualities:

- It is built from the `/usr/obj` compiled object directory rather than extracted distribution sets
- It can be imaged to a physical machine boot device such as a solid state drive
- It performs a `growfs(8)` operation on boot, filling the remaining space of the boot device
- It supports BIOS and UEFI booting
- It is configured for DHCP networking
- It supports root on ZFS using '`makefs -t zfs`' with the `VMFS=ZFS make(1)` option as of FreeBSD 14 and later

These qualities result in meaningful system boot images that build efficiently and work on a variety of hardware and virtual machines.

OccamBSD

The OccamBSD[3] project employs the various FreeBSD compilation options to build minimal and purpose-built RAW, ISO, and "memstick" boot images and provides the basis for the FreeBSD Appliance strategies described in this paper. OccamBSD uses build profiles to perform `build(7)` and `release(7)` operations such as these to generate a minimum system that be booted on the `bhyve(8)`/`vmm(4)` hypervisor:

```
target="amd64"
target_arch="amd64"
cpu="HAMMER"
makeoptions=""

build_options="WITHOUT_AUTO_OBJ
WITHOUT_UNIFIED_OBJDIR
WITHOUT_INSTALLLIB WITHOUT_BOOT
WITHOUT_LOADER_LUA WITHOUT_LOCALES
WITHOUT_ZONEINFO WITHOUT_DYNAMICROOT
WITHOUT_FP_LIBC WITHOUT_VI"

kernel_modules="virtio"

kernel_options="SCHED_ULE
GEOM_PART_GPT FFS GEOM_LABEL CD9660
MSDOSFS TSLOG"

kernel_devices="pci loop ether acpi
uart ahci scbus cd pass virtio
virtio_pci virtio_blk vtnet
virtio_scsi virtio_balloon"
```

The resulting operating system is under 150MB in size and boots in seconds. Adding features such as OpenZFS and networking requires only selecting the appropriate combination of compilation options. A firm coupling of userland components to their dependent kernel resources does not exist and this association must be performed manually. In and of itself, OccamBSD provides various insights into the true "base" components of FreeBSD:

- Revealing `build(7)`, `src.conf(5)`, and `release(7)` bugs
- Revealing abandoned userland components such as `mlxcontrol(8)` that can lack built corresponding kernel `device`(s)
- Revealing components that lack build options such as `nfsd(8)`
- Revealing under-documented subsystems such as `boot(8)` whose documented `FILES` may be out of sync with the operating system as a result of vigorous development
- Revealing opportunities for the addition or removal of components to or from the base operating system

The OccamBSD strategy also provides a narrowed context for operating systems education, auditing, documentation, and quality assurance considering that it isolates the lowest common denominator of FreeBSD used by virtually all users in all environments.

FreeBSD Configuration

The lightly-configured bootable RAW images produced by the FreeBSD `release(7)` process require further configuration for all but the simplest deployments. "Modern" system and "cloud" deployment employs extensive automation in which systems are considered commodity "livestock" rather than unique "pets". This philosophy is realized through countless available configuration tools and ecosystems that broadly demonstrate a common quality: idempotence. Idempotence[5] is the "property of certain operations in mathematics and computer science whereby they can be applied multiple times without changing the result beyond the initial application". In practice this dictates that a system should have a predetermined desired state and its configuration tools should work to achieve and maintain that state. At an extreme, a purpose-built, idempotent system should be incapable of performing anything but its desired functionality.

FreeBSD has long focused on an operator experience that is consistent and intuitive, embracing the "Principle Of Least Astonishment" or POLA[4]. In service of this inherent consistency, FreeBSD provides the `sysrc(8)` utility for making `rc.conf(5)` and `loader.conf(5)` configuration file changes in a safe manner. While `sysrc(5)` is not yet idempotent, idempotence can be achieved with judicious use of the `test(1)` assumption, string, and return value testing, and '`sysrc -c`' "check":

```
hostname="occambsd"
if [ "$( sysrc -c hostname=$hostname )" ] ; then
        echo "Hostname $hostname is correct"
        logger "Hostname $hostname is correct"
else
        echo ; echo "Setting hostname $hostname"
        logger "Setting hostname $hostname"
        sysrc hostname="$hostname"
        service hostname restart
fi
```

Should `sysrc(5)` prove inadequate for a specific configuration setting, traditional utilities with pattern matching such as `sed(1)` can be used with far less orientation than any given general-purpose configuration management system. Such knowledge should also prove more enduring than knowledge of proprietary vendor-maintained syntax.

In-Base Services and Facilities

FreeBSD includes many facilities and services including in-kernel NFS, iSCSI, and Fibre Channel servers, a mail server, FTP and TFTP servers, and an SSH server. Each of these have been proven with production workloads and are included in many FreeBSD-based products and services. While the configuration complexity of these services can range from trivial to complex, an idempotent approach can be taken and employ facilities such as the `rc.local(8)` script that is executed at boot time.

Third Party Services and Facilities

For services and facilities not provided by the FreeBSD base operating system, the `ports(7)` collection of ported third party software and `pkg(8)` binary package manager provide over 55,000 additional components and options. In service of FreeBSD appliances, `pkg(8)` supports the `-r` root directory flag that can specify a unique root directory for package installation. `pkg(8)` will determine the version and ABI of a given root directory and install the correct packages when cross-building and releasing a release. '`pkg -r`' root directories can include the FreeBSD release media:

```
cd /usr/obj/usr/src/amd64.amd64
cd release
pkg -r disc1 install -y tmux
chroot disc1 /etc/rc.d/ldconfig start
make cdrom
```

If employed with a hardware-imaged root-on-ZFS RAW "virtual" machine image that can be configured in advance or idempotently on boot, this strategy can deliver commodity "livestock" systems that can be easily re-imaged or updated via boot environments. While the configured system can include packages, it need not include the `pkg(8)` facility to manage them, reducing the attack surface of the system in addition to the reductions provided by custom compilation. Should the system require customized packages, the `poudriere(8)` bulk package builder provides the infrastructure to facilitate and automate this task. Finally, the FreeBSD "Packaged Base" initiative PkgBase[6] aims to fully package the base operating system, allowing for system updates with `pkg(8)` and presumably selectively add and remove base components without recompilation. Together these upstream tools reduce the burden on downstream release engineers tasked with tracking the FreeBSD CURRENT development branch.

Service Containment

The FreeBSD `jail(8)` facility has provided a container infrastructure since the 4.0-RELEASE of the FreeBSD and has steadily introduced features such as hierarchical/nested Jail support, VNET and Netgraph virtual network stacks, resource controls, NFSd and bhyve hypervisor containment, and `nullfs(5) -f` file mount support available in FreeBSD 14. With the introduction of '`nullfs -f`' file mounts, a Jail's root file system can be constructed significantly of read-only directories and files mounted from outside the Jail. Combined with the OpenZFS[7] file system and volume manager, write support can be dynamically enabled or disabled, and directories/data sets can be dynamically mounted or unmounted. In further service of reduced attack surfaces, Jail userlands can be populated with selective compilation as outlined above, or select population using techniques such as '`ldd -f '%p\n' `which bhyve``' to identify the dependencies of a given binary. Essential base support files such as `/libexec/ld-elf.so.1` are easily identified with limited experimentation. Combined, all of these facilities can provide a highly-flexible, security-conscious "cradle to grave" appliance environment for the delivery of in-base services including `nfsd(8)`, `sshd(8)`, and the `bhyve(8)` hypervisor using entirely in-base tools and select idempotent scripting. Any such appliance can be augmented with packaged third party software or external appliances packed as virtual machine images.

`CTL(4)`/`virtio_scsi(4)` Storage

The FreeBSD appliance strategies outlined in this paper make extensive use of in-base resources but special attention must be given to the potential of OpenZFS, the CAM Target Layer `CTL(4)`, and `virtio_scsi(4)` in combination. In addition to the dynamic read-only and mounting/unmounting abilities of OpenZFS, the file system and volume manager can also provide ZVOL block devices. ZVOL synthetic block devices and RAW disk images can be provided as hot-pluggable SCSI devices via `CTL(4)` to a virtual machine's `virtio_scsi(4)` virtual SCSI bus. The result is highly-flexible storage infrastructure for `bhyve(8)` virtual machines.

`libxo(3)` and `nvlist(9)`

Finally, the steady incorporation of the `libxo(3)` library for producing XML, JSON, and HTML output from in-base utilities, and the `nvlist(9)` library for reading configuration data name/value pairs with utilities such as `bhyve(8)` promises to provide a highly-structured and API-friendly administrative environment for operators. The subtle but powerful innovations made in the FreeBSD operating system outlined in this paper are significantly reducing dependence on external configuration and management facilities. This trend empowers system operators by allowing them to focus on learning the often-timeless syntax and conventions of this highly-capable unified computing environment.

Acknowledgements

The author would like to thank the many FreeBSD developers who have resolved countless issues revealed by this research over the last five years. The FreeBSD Release Engineering team also deserves appreciation for maintaining the infrastructure that enables this research. This work would obviously also not be possible without the individuals who have contributed the powerful new features outlined in this paper.

References

1. reproducible-builds.org
2. wiki.freebsd.org/BuildingOnNonFreeBSD
3. github.com/michaeldexter/occambsd
4. en.wikipedia.org/wiki/Principle_of_least_astonishment
5. en.wikipedia.org/wiki/Idempotence
6. wiki.freebsd.org/PkgBase
7. openzfs.org