

Stop using portmaster, portupgrade and ports on your servers and switch to packages.

# Poudriere

BY BRYAN DREWERY

Setting up your own package builds with Poudriere takes only a few minutes and will save you a lot of time in the future.

Since November 2013, FreeBSD has provided official packages for Pkg, formerly known as pkgng. The 10 release also brought the first signed packages. The project uses Poudriere for package building.

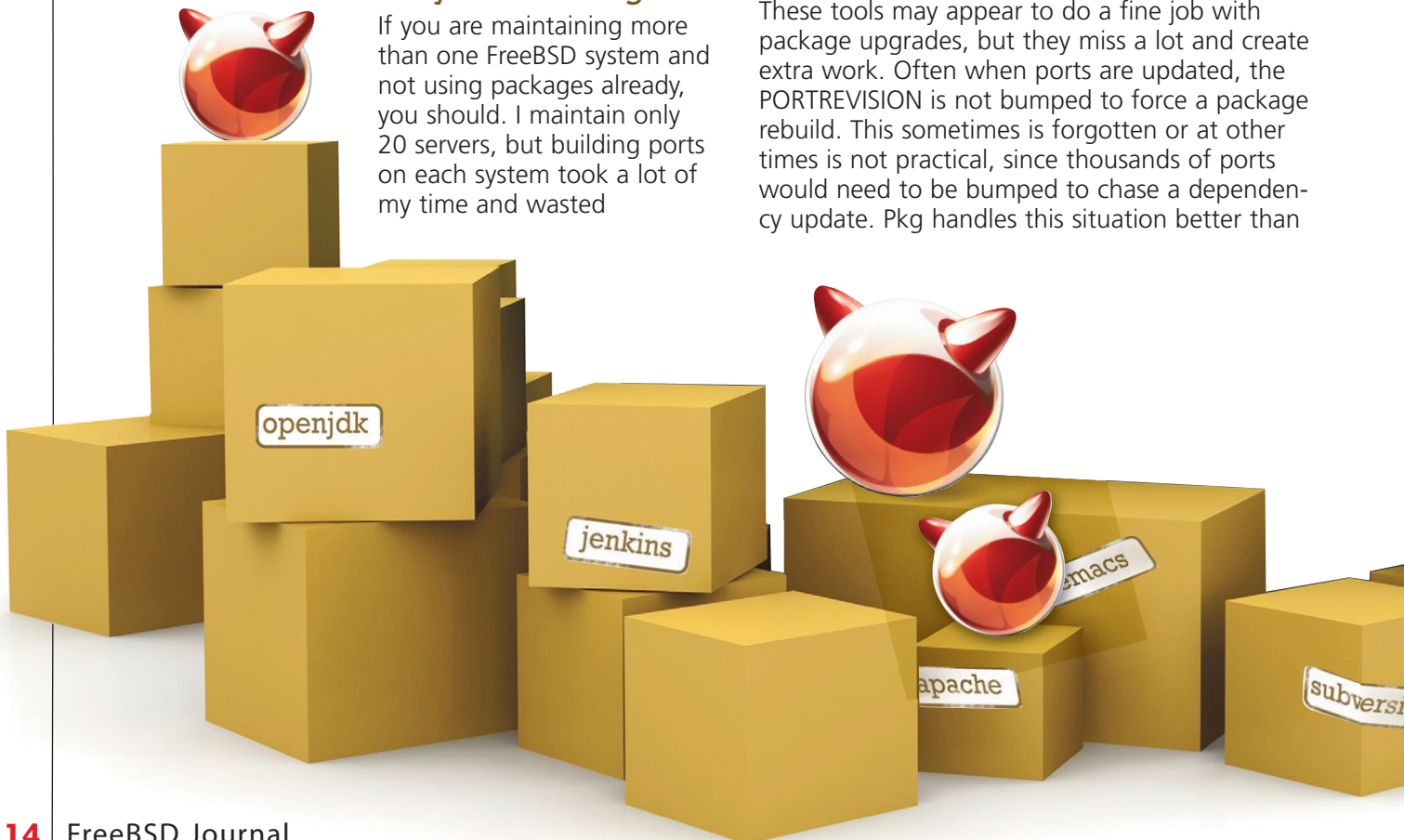
## Why Use Packages

If you are maintaining more than one FreeBSD system and not using packages already, you should. I maintain only 20 servers, but building ports on each system took a lot of my time and wasted

resources on production machines.

When building ports on multiple servers, it is very easy to get their options or versions out of sync. By building packages once on one system, I lessened the load on my systems, lessened the amount of work I had to do and made all my systems consistent. Instead of dealing with the same failure on each system, I only need to handle it on the build system.

Until Pkg was available, I never really considered using packages. The old style pkg\_install packages were fine for initial system installation, but there was no built-in way to upgrade, except to remove all and install a new set. You had to use a tool such as portmaster or portupgrade and have an INDEX or a ports tree checked out. These tools may appear to do a fine job with package upgrades, but they miss a lot and create extra work. Often when ports are updated, the PORTREVISION is not bumped to force a package rebuild. This sometimes is forgotten or at other times is not practical, since thousands of ports would need to be bumped to chase a dependency update. Pkg handles this situation better than



the old system. Pkg can also detect when the selected options for installed packages have changed from the available remote packages and will reinstall them automatically. The old tools would require recursively reinstalling packages sometimes and not others. There was too much manual work involved with the old package system. The goal of Pkg is to have a built-in upgrade process that removes manual intervention. There is still some work to do on removing some of the manual intervention, but it is already far better than the old system.

## Custom Package Options

Why would you need to deviate from the official packages? The ports framework provides options support for ports to change build-time configuration. Not all applications support run-time configuration. Some applications must be compiled differently depending on which features are enabled. Others have options simply to lessen the amount of features and dependencies in the default port. For server administrators, this can quickly lead to finding that some of the default packages do not meet their requirements. One common example is that PHP comes in CGI mode by default without any support for `apache+mod_php` or the more flexible PHP-FPM. Another common issue with the default packages is that they come with X11 support which may be undesirable on non-desktop environments. Perhaps you have custom ports or custom patches for some ports. By building your own packages you regain control over which options packages are built with and how often updates are available.

Some other reasons to build your own packages are when you are dealing with restrictive

licenses for which the FreeBSD project is unable to ship packages or if your system is highly customized and not ABI compatible with FreeBSD.

There are a few ways to get custom packages. Pkg supports using multiple repositories. It can be set up to use the official FreeBSD repository as a primary and a custom one as a secondary. Pkg is not limited by the number of repositories it can track and they can be reordered for priority. The problem with multiple repositories is that it can currently be difficult to maintain. When Pkg detects that an installed package has different options or dependencies from a repository it is tracking, the package will be reinstalled from potentially any remote version. You can either lock the package during upgrades with `pkg lock PKGNAME` and `pkg unlock PKGNAME` or bind it to a specific repository with `pkg annotate -A PKGNAME repository REPOSITORYNAME`. There is also the subtle problem of keeping the ports tree for your custom repository in sync with the FreeBSD packages. Since packages are built from a ports tree snapshot taken once a week, if your custom repository does not match it may lead to conflicts. It is much simpler to just build an entire package set of just what you need with the options that you want. On your systems you would only track your one repository and not include the FreeBSD one. This also has the benefit of using your own infrastructure for distributing packages which can speed up upgrades substantially.

## Building Packages

For the longest time Tinderbox was the popular go-to tool for building packages. Other people would just install all ports on one system and then create packages from that system and copy them to other systems. This method is not recommended because the packages are created in an unclean environment that is constantly growing larger and more polluted. Even using portmaster with ports today and creating Pkg packages from those for distribution is not recommended for the same reasons. It is better to use a system designed for creating package sets.

Poudriere (roughly pronounced poo-dree-year, French for “powder keg”) was written as a faster and simpler replacement for Tinderbox. It was written by the Pkg author Baptiste Daroussin and is now mostly maintained by me along with Baptiste and some other contributors. It has quickly become the de-facto FreeBSD port testing and package building tool. It is the



# Poudriere

official build cluster tool and is also used by the FreeBSD Ports project for testing sweeping patches in what are called “exp-runs”. It is written in POSIX shell and is slowly being moved to C components. Unlike Tinderbox, it has no dependencies and does not require a database. It has been greatly optimized to be highly parallel in all operations. It uses jails to build ports in sandboxed environments in very strict conditions. Jail creation is done once with a simple command. During builds, the jail is cloned automatically for each CPU being used to give ports a clean place to build. Builds can occur on UFS, ZFS or the TMPFS file systems. UFS is high I/O, low RAM, slow build and TMPFS is low I/O, high RAM and very fast build. It is also configurable such that only some parts of the build use TMPFS while others use UFS/ZFS to allow some compromise on lower memory machines. An amd64 host can also build i386 packages with no extra effort. Packages can be built for the current host version or older. For example, if the host machine is 9.2, it can build 9.2, 9.1 and 8.3 package sets.

Poudriere does incremental builds by default to only rebuild what is needed. The incremental build checks for changed options, missing dependencies, changed dependencies, new versions, and changed pkgnames. If any of those changed it will rebuild that port. This also causes anything depending on that port to be rebuilt. This is sometimes overkill, but ensures that no port change is missed in package creation. There is also built-in ccache support which can help port rebuilding time when dependencies change. Build times of package sets vary, but on a system with multiple CPU and enough RAM, a few hundred ports can typically build in an hour or two.

Poudriere has a read-only, real-time web

interface that allows monitoring the status of builds. This interface does not require any server-side CGI or scripting support, as Poudriere just writes out a status file in JSON and then web interface uses it. It is not as nice as the Tinderbox interface, but there are plans to improve it more in the future. The 3.1 version has been incrementally improved to be more responsive and allow searching and sorting each sub-list of packages. See screen shot.

Poudriere also has a feature called a “set”. This allows having multiple saved options, `make.conf` files and resulting package sets for each named “set”. This removes the need to have multiple jails for the same target version/architecture. For example, this can be used to create a PHP 5.3 package set named “php53” and a PHP 5.5 package set named “php55” using one jail on the build system. When building the set would be specified with “-z setname”, i.e. “`bulk -a php53 -j 91amd64`” would produce packages in `/usr/local/poudriere/data/packages/91amd64-default-php53`. The “default” refers to the ports tree, which can also be changed with the “-p” option.

The upcoming Poudriere 3.1 release also brings some interesting new features. One of the major ones is named `ATOMIC_PACKAGE_REPOSITORY`. It prevents the repository from being modified until a build is completed. Currently in 3.0 the repository has packages deleted at startup and packages being modified during build, thus disallowing serving it directly over http. This is enabled by default. It works by hard-link copying the package directory into a `.building` directory during startup, then when the build completes the `.building` directory is renamed to a `.real_TIMESTAMP` directory and the top-level `.latest` symlink in the repository is updated to point to the new build. There are still potential problems with changing the repository during a pkg upgrade job, but the window for problems is far smaller than without this (Box 1).

This feature also allows doing dry-runs with `bulk` to see what would be done by using `bulk -n`.

Atomic package repository also allows keeping old package sets. This is not enabled by default but can be enabled by setting `KEEP_OLD_PACKAGES` and `KEEP_OLD_PACKAGES_COUNT`. By default, 5 sets are kept. With this you could rollback a system by changing the `.latest` symlink to an old set and then running `pkg upgrade -f` on a server to force it to


The screenshot shows the Poudriere 3.1 Web Interface. At the top, there's a header with 'Jail: 91amd64-default', 'Build: 2013-12-22\_10621621s', and 'SVN: https://svn.us-east.freebsd.org/ports/head/337197'. Below this is a progress bar showing 'Total: 24536', 'Built: 100%', 'Failed: 0%', 'Skipped: 0%', 'Ignored: 0%', and 'To build: 10751'. The main content is divided into two sections: 'Builders' and 'Built ports'. The 'Builders' section has a table with columns 'Id', 'Origin', and 'Status'. The 'Built ports' section has a table with columns 'Package', 'Origin', and 'Status'. The 'Failed ports' section has a table with columns 'Package', 'Origin', 'Phase', 'Skipped', and 'L'. The 'Built ports' table shows various packages like 'pkg-1.2.4\_1', 'devtool-1.2.4\_1', 'chroot-0.13\_1', etc. The 'Failed ports' table shows 're-0.4.6'.

Poudriere 3.1 Web Interface Preview

```

/usr/packages/exp-91amd64-commit-test # ls -al
total 13
drwxr-xr-x  7 root wheel 12 Mar  2 01:59 ./
drwxr-xr-x 26 root wheel 32 Mar  2 01:13 ../
lrwxr-xr-x  1 root wheel 16 Mar  2 01:59 .latest@ -> .real_1393747164
drwxr-xr-x  4 root wheel  7 Mar  1 16:59 .real_1393714735/
drwxr-xr-x  4 root wheel  7 Mar  2 00:40 .real_1393742366/
drwxr-xr-x  4 root wheel  7 Mar  2 00:58 .real_1393743542/
drwxr-xr-x  4 root wheel  7 Mar  2 01:05 .real_1393743901/
drwxr-xr-x  4 root wheel  7 Mar  2 02:00 .real_1393747164/
lrwxr-xr-x  1 root wheel 11 Nov 19 17:20 All@ -> .latest/All
lrwxr-xr-x  1 root wheel 14 Nov 19 17:20 Latest@ -> .latest/Latest
lrwxr-xr-x  1 root wheel 19 Nov 19 17:20 digests.txz@ -> .latest/digests.txz
lrwxr-xr-x  1 root wheel 23 Nov 19 17:20 packagesite.txz@ -> .latest/packagesite.txz

```



**Box 1.**  
**Atomic package  
repository layout**

reinstall all packages from the remote repository. This would downgrade all to the old set.

Another upcoming feature for 3.1 is named *poudriered*. It will allow non-root usage of *poudriere* through a socket to a root daemon. This will allow queueing jobs as well as queueing a job for all jails. It is configurable in a similar way as *sudo* to be able to restrict subcommands

and even arguments to specific users and groups. More improvements, such as daemon privilege separation, are planned for 3.2/4.0.

Setup and usage of *poudriere* is simple and fast. Install *poudriere*, create a jail, checkout a ports tree, create a file with a list of ports, optionally create a private/public keypair for a

```

$ pkg install ports-mgmt/poudriere
$ cp /usr/local/etc/poudriere.conf.sample /usr/local/etc/poudriere.conf
# Modify configuration.
$ vim /usr/local/etc/poudriere.conf
# Create a ports tree in /usr/local/poudriere/ports/default
$ poudriere ports -c -m svn+https

# Create a jail from a snapshot
$ poudriere -j 10amd64 -v 10.0-RELEASE -a amd64
# Create a head jail from src
$ poudriere -j head-amd64 -v head -a amd64 -m svn+https

# Create a list of port origins (cat/port), 1 per line.
$ vim /usr/local/etc/poudriere.d/ports.list


# Create a public/private keypair for the repository
$ cd /etc/ssl
$ openssl genrsa -out repo.key 2048
$ chmod 0400 repo.key
$ openssl rsa -in repo.key -out repo.pub -pubout
# Configure poudriere to use your public key
$ echo "PKG_REPO_SIGNING_KEY=/etc/ssl/repo.key" >> /usr/local/etc/poudriere.conf

# Create a make.conf
$ echo "WITH_PKGNG=yes" >> /usr/local/etc/poudriere.d/make.conf

# Configure options for the build
$ poudriere options -f /usr/local/etc/poudriere.d/ports.list

# Build packages
$ poudriere bulk -j 91amd64 -f /usr/local/etc/poudriere.d/ports.list

```



**Box 2.**  
**Typical  
Poudriere setup**



# Poudriere

signed repository and then build! (Box 2).

If you need to build multiple sets, then you should use the “-z” flag when using “options”, and “bulk” commands, and also setup a `SET-make.conf` in `/usr/local/etc/poudriere.d` with any set-specific configuration.

The official Poudriere site has a guide for creating and maintaining repositories. The manual page is also online here.

There is a guide for using Jenkins to do scheduled builds of packages which is documented well in a 3 part series here.

## How FreeBSD Builds Packages

The FreeBSD project used to build packages only for releases and occasionally for the STABLE branches. The old package builders used a distributed system named Portbuild. It would use a large cluster of smaller 2GB-4GB machines to build packages. This was error-prone and slow, mostly due to the older machines. A full build could still take a week. Today packages are built using single large machines using Poudriere. The FreeBSD Foundation was nice enough to purchase several 24-32 CPU 96GB machines to replace the old cluster. Using the new systems with Poudriere, the entire ports tree can be built from scratch in about 16 hours on one machine.

Packages are built for the oldest release of each branch. These packages are supposed to be ABI/KBI compatible with all future releases on those branches as well as the STABLE branch for that release. This means that packages built for 8.3 will work on 8.4 but are not guaranteed to work on 9.x. For official FreeBSD package builds, every Tuesday night a snapshot of the ports tree is made and packages begin building. Currently packages are built from 8.3, 9.1, 10.0, and head for i386 and amd64. The quarterly ports branch is also built for 10.0 on both i386 and amd64. This adds up to 10 separate package sets that must be built each week. We split these into two separate servers, one for i386 and the other for amd64. Not all 24,000 ports are built every week for every set, since Poudriere is smart enough to only build what needs to be rebuilt.

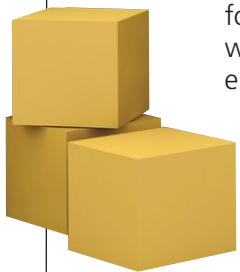
It takes just a few days for all sets to be built. After each individual package set is built, its repository is generated and signed by our signing server. An example of how we sign packages is in the `pkg-repo.8`

manual page. Then the packages are uploaded, using `rsync`, to our content delivery network for public consumption.

## Meta Packages

For managing my servers I use meta-packages. These are packages that only depend on other packages and do not install any files themselves. They can be created from a port that does no actual building. By installing meta-packages onto your servers, you guarantee that each server will have the same packages installed as long as they each install only the few meta-packages you create. Using meta-packages also makes removing unneeded packages much simpler. `Pkg` has a feature that tracks which packages have specifically requested to be installed and which were pulled in automatically as dependencies. If you were to type `pkg install` for all packages on a system, then `pkg autoremove` would never remove anything. If instead you were to only install the one meta-package and it pulled in 100 dependencies, then in the future updated the meta-package to no longer require some other package, `pkg autoremove` would properly detect and remove that package.

I take this a step further and have role-based meta-packages. For example, I have a base meta-package that contains all packages that all my servers need. This would be things such as the configuration management tool. I then have meta-packages for each type of server that depend on only what it needs and the base meta-package. For example, my DNS server uses the `dns-server` meta-package that depends on `bind` and some other DNS tools and the base meta-package. My web application jails all have a `web-server` meta-package that depends on `PHP`, `nginx` and the base meta-package. This simplifies management on the servers, as only a handful of packages need to be explicitly installed and monitored. My method is to use a ports tree overlay to create my meta-packages, but you could also just create the packages directly by using the `pkg` manifest format. For the ports tree overlay, I just `rsync` my git-tracked tree over the top of an SVN checkout of ports. For Poudriere only the meta-packages need to be specified to build and it will then build all needed dependencies as well. I have a guide on my blog for managing role based servers with meta-packages.



## Deployment

The official FreeBSD package URL uses an SRV DNS record to advertise which mirrors are available, but otherwise is just plain HTTP. If your network is completely internal there is not much to do for deployment. All that is needed is an internal FTP or HTTP server to serve up the package repository. The servers I host are spread out all over the world and are not in one network. At first I tried using just one HTTP server for serving the packages, but quickly found that updating all of them at once would severely slow everything down. If you have a large pipe this may still be an option. I ended up using Amazon S3 and have had a much better experience. I wrote a bulk hook for my builds using s3sync to upload the packages. I host 5GB of package sets on there and update servers weekly. This comes out to just a few dollars or less a month for updating my 20 servers.

To configure a server to use your repository, create a `/usr/local/etc/pkg/repos/MYREPO.conf` file, and also place your repository public key in `/usr/local/etc/pkg/repos/MYREPO.pem` (Box 3).

This configuration requires setup of ABI symlinks in the repository. This is a one-time operation. It allows you to use the same repository configuration on all servers without changing which release and arch it uses. Pkg will change the value of ABI when it fetches packages. It should look something like this (Box 4). As for keeping servers up-to-date, since I am a Ports committer and also a Pkg developer, I like to observe all upgrades that I can to find any issues. I'm also just paranoid and like to make sure upgrades go smoothly, so I manually run `pkg upgrade` on my servers and don't use an automated crontab for it. My systems have a lot of applications built outside of ports, so I must save shared libraries until those applications can be rebuilt. This is similar to what `portupgrade` and `portmaster` with the `-p` flag do. An example of this can be found on my github. Manually running my upgrade script is not a problem for me since I only maintain a handful of servers. However, Pkg is supported by puppet, salt and ansible. Do be warned though that some manual intervention is still required with package upgrades occa-

```
MYREPO: {
  url: "http://url.to.your.repository/${ABI}",
  enabled: true,
  signature_type: "pubkey",
  pubkey: "/usr/local/etc/pkg/repos/MYREPO.pem"
}

# Optionally disable the FreeBSD repo.
FreeBSD: {
  enabled: false
}
```

### Box 3. Typical pkg repository configuration

sionally. This usually only occurs anymore when the origin of a package changes and requires running `pkg set -o old/origin:new/origin`. The most common case is when something like Perl is updated. You can detect this case and other cases of conflicting packages in a script by running `pkg upgrade -Fy` which will list all conflicting packages in the upgrade. The ports framework and Pkg currently do not have a means to handle replaced packages automatically, but eventually will. These cases are still currently documented in the `/usr/ports/UPDATING` file. You can keep an eye on this file in ports svnweb.

Pkg discussion takes place on the `freebsd-pkg` mailing list and on IRC in `#pkgng` on Freenode. Poudriere discussion takes place on IRC in `#poudriere` on Freenode. Feel free to stop by with any questions or ideas you have. •

---

**Bryan Drewery has managed shared hosting services with FreeBSD since 2004. He joined the project in 2012 as a Ports committer, is a member of Portmgr and has recently become a Src committer. He is the current upstream maintainer of Portupgrade, Portmaster, and a developer on Pkg and Poudriere. In Portmgr, he helps with the Ports framework, managing the package build systems, package building and testing ports patches on them.**

```
# ls -al /usr/local/poudriere/data/packages
total 19
drwxr-xr-x 7 root wheel 15 Jul 11 2013 ./
drwxr-xr-x 26 root wheel 32 Mar 2 01:13 ../
drwxr-xr-x 2 root wheel 2 Jul 8 2013 10amd64/
drwxr-xr-x 7 root wheel 39 Mar 2 10:05 83amd64/
drwxr-xr-x 7 root wheel 39 Mar 2 10:30 83i386/
lrwxr-xr-x 1 root wheel 7 Jul 8 2013 freebsd:10:x86:64@ -> 10amd64
lrwxr-xr-x 1 root wheel 6 Jan 26 2013 freebsd:8:x86:32@ -> 83i386
lrwxr-xr-x 1 root wheel 7 Jan 26 2013 freebsd:8:x86:64@ -> 83amd64
```

### Box 4. ABI symlink setup