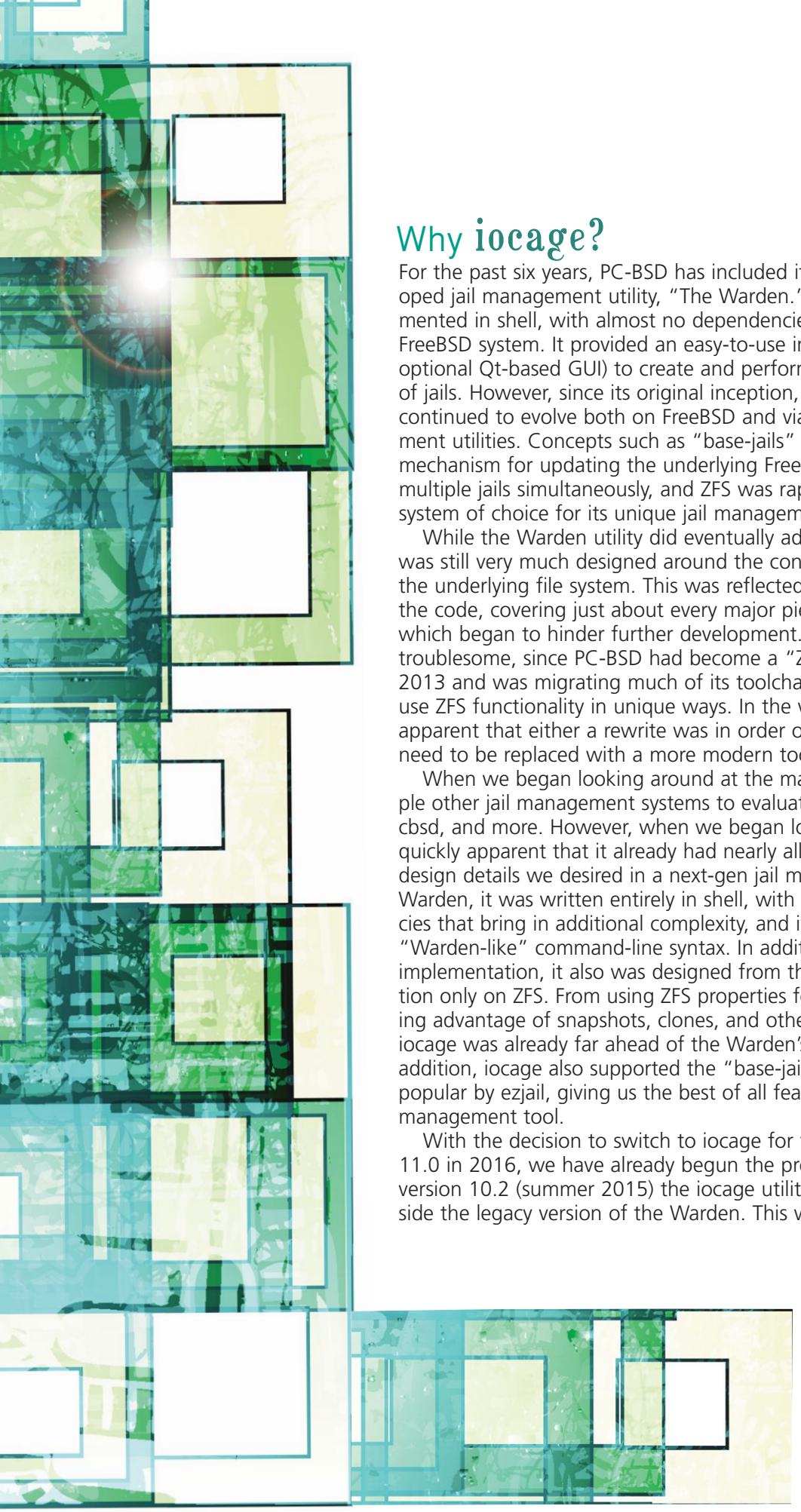
Migrating

Jail Management from Warden to iocage

With the upcoming release of PC-BSD/TrueOS 11.0 in 2016, we have begun the process of migrating to some new tools and utilities that come bundled with the system. Some of the major ones include switching back to FreeBSD's boot-loader (away from GRUB), moving to the Lumina Desktop as our primary environment, and converting from the Warden jail management utility over to iocage. In this article we will be taking a more-in-depth look at iocage, how it compares to Warden, and the reasons for the move.

By Kris Moore & Brandon Schneider



Why iocage?

For the past six years, PC-BSD has included its own in-house-developed jail management utility, "The Warden." This utility was implemented in shell, with almost no dependencies apart from the base FreeBSD system. It provided an easy-to-use interface (with an optional Qt-based GUI) to create and perform basic management of jails. However, since its original inception, jail management has continued to evolve both on FreeBSD and via other jail management utilities. Concepts such as "base-jails" became popular as a mechanism for updating the underlying FreeBSD base-system on multiple jails simultaneously, and ZFS was rapidly becoming the file system of choice for its unique jail management possibilities.

While the Warden utility did eventually add ZFS functionality, it was still very much designed around the concept of using UFS as the underlying file system. This was reflected internally in much of the code, covering just about every major piece of functionality, which began to hinder further development. This was particularly troublesome, since PC-BSD had become a "ZFS-only" system in 2013 and was migrating much of its toolchain to understand and use ZFS functionality in unique ways. In the winter of 2014, it was apparent that either a rewrite was in order or that Warden would need to be replaced with a more modern tool going forward.

When we began looking around at the market, there were multiple other jail management systems to evaluate, from ezjail, qjail, cbsd, and more. However, when we began looking at iocage, it was quickly apparent that it already had nearly all the features and design details we desired in a next-gen jail manager. First, like Warden, it was written entirely in shell, with no external dependencies that bring in additional complexity, and it used a very similar "Warden-like" command-line syntax. In addition to its native shell implementation, it also was designed from the ground up to function only on ZFS. From using ZFS properties for jail settings, to taking advantage of snapshots, clones, and other native ZFS features, iocage was already far ahead of the Warden's ZFS functionality. In addition, iocage also supported the "base-jail" concepts, made popular by ezjail, giving us the best of all features in a single jail management tool.

With the decision to switch to iocage for the upcoming PC-BSD 11.0 in 2016, we have already begun the process for end users. In version 10.2 (summer 2015) the iocage utility was included alongside the legacy version of the Warden. This was done to give users

and developers time to play with the new utility for a period before the conversion takes place. A migration utility will become available this fall, which allows moving Warden existing jails to iocage automatically, and will then be included in 11.0-RELEASE.

Since iocage is a command-line tool, we are also currently in development of a new GUI that will replace the original Warden Qt UI. The new GUI in development is web-based as a part of the AppCafe project, with the goal of being useful both in a desktop such as PC-BSD and a server-based headless system such as FreeNAS or TrueOS. This new AppCafe interface is under heavy development at the moment, with a planned release for FreeNAS 10 and PC-BSD 11.0 in 2016. Aside from its support for system package management (via pkgng), the new AppCafe will also be using iocage exclusively to deal with jails in a few unique ways.

In a more traditional jail management role, the AppCafe web-interface will act as a front-end to iocage directly, giving easy control for creating, removing, and performing basic management of jails. Since Brandon Schneider (iocage co-developer, alongside the original author, Peter Toth) joined us at iXsystems in 2015, some new features have been added to iocage that make creating and distributing jails much easier than before. The new mechanism for jail distribution uses the VCS tool "git" to do the initial checkout of a prebuilt jail environment ready for execution. This allows content creators to easily create jails using pkgng or other methods, then commit and push their changes to a public git server, such as GitHub. From there clients can run a single iocage command to fetch this jail repo to their local box and begin execution. This new model will become the basis of the AppCafe "App Cages" feature, which will let us bundle applications such as Plex Media Server and others in a ready-to-run fashion. Additionally, it provides unique ways to verify updates and view diffs/logs of changes in a readily understood fashion using git. These features are already in a working state and included in the PC-BSD 11.0-CURRENT branch releases, giving developers and users an early preview of what is coming for the 11.0-RELEASE.

iocage Primer/ Inside iocage Internals

A quick primer on how iocage functions. Firstly, we use ZFS properties to configure everything we do. We have no configuration file, the only exception being enabling the service in rc.conf. Most of iocage's nomenclature is kept consistent with ZFS's. We try to stay the same when the function is something both ZFS and iocage support. For the rest of the commands, we go with what we think is most self-explanatory. Our jail naming uses a randomly generated UUID, so we can avoid any naming conflicts.

Let's get started with the tool! If you have multiple zpools, then iocage will pick the first one it finds. So we normally do an 'iocage activate POOL' to begin with. We use what are called "bases." These bases are the RELEASEs you have fetched with iocage and form the basis of our basejails. In this example, we will use 10.2-RELEASE. To fetch it we issue an 'iocage fetch release=10.2-RELEASE" and let iocage do its job. When it's finished, we have a new base ready to be used. While the terminology for fetching and creating differ for specifying which version of FreeBSD you want to use, it's because once a RELEASE is fetched, the meaning has changed for iocage.

So we have picked a pool to use for iocage, and fetched a RELEASE. All that's left is to create a jail. Like ZFS, we allow the user to specify properties they would like to set during creation. For this example, we will be using a static IP and assigning the jail a name, which we call a "tag." Here we go! Type 'iocage create tag="example" ip4_addr="DEFAULTl192.168.1.100/24" base="10.2-RELEASE"'. That will create a jail named "example," and give it the IPv4 address 192.168.1.100. The "DEFAULT" is a special keyword that tells iocage to figure out the default interface to use. We specify the base in this instance, but iocage will default to the version your host machine is running. This jail will use what we call "shared networking mode." The other type you can use is VIMAGE, which is a virtual networking stack that is quite versatile. IPv6 is also supported for both modes.

Since our jail is now created, let's start it. 'iocage start example' and the jail will come right up. We can verify it's running with 'iocage list' or 'iocage get state example'. Which looks like this:

```
~% iocage list
JID  UUID                                   BOOT  STATE  TAG      TYPE
1    89b2f41a-76b2-11e5-8df9-d05099728dbf  off   up     example  basejail
```

The jail is now started—time to add a pkg! Simply running 'iocage console example' will log us right into the jail and allow us to start interacting with it as if it was a physical machine. In this instance, that means installing a pkg. So we can do 'pkg install tmux' and, shortly after, we have tmux installed in a jail. It's really that easy. iocage allows for a lot of advanced-usage scenarios and all other sorts of things. For that, I encourage you to read our manpage and visit our documentation: https://iocage.readthedocs.org/en/latest/index.html.

That covers the primer for using iocage.

As a tool, we aim to be very user friendly, even if you have never used jails before. Having prior jail knowledge is certainly a plus. Any question you may have can be answered on our Google Group: https://groups.google.com/forum/#!forum/iocage.

Deep Dive

Now it's time to do a bit of a deep dive into the latest work on iocage, which is a total rewrite of our basejails. We call them "basejails," because they use a common shared base that is faster and easier for a user to update. This also has the benefit of substantial space savings. Our basejails have become our default jail type now in the latest development version.

Our basejail structure is pretty straightforward. In iocage we have the jail that lives in '/iocage/jail/UUID/root'. UUID will be replaced with whatever UUID was generated during your jail creation. Mine for this example is "89b2f41a-76b2-11e5-8df9-d05099728dbf," as every one is unique. Since these are read-only mounts, the user cannot manipulate that data. We use the excellent unionfs file system to mount them as an overlay, allowing the user to add files, change files that existed, and even remove files, all without touching the read-only layer—unionfs is pretty magical. We do this by having our read-write mounts located in the directory called "_". This is the list of directories used with the basejails and where they get mounted:

Once the jail has been started, those directories are mounted and using it is identical for the user.

One thing I have mentioned is tags, and I haven't elaborated on what they mean to you, the user. Tags allow you to name a jail so that you're able to interact with it using iocage and do not have to remember the UUID. But if you prefer to use the UUIDs, you are welcome to do so! This means you can do every operation by simply using a tag. Tags are very convenient and I would suggest using them. If a tag is not supplied during creation, the tag will be the date the jail was created. This also applies to you if you're a user who would like to copy some files into a jail before it's started, or remove some once it is stopped. For our example I can change my directory to '/iocage/tags/example/_' instead of '/iocage/jails/ 89b2f41a-76b2-11e5-8df9-d05099728dbf/_'. It saves a lot of typing!

How We Use ZFS

One of the last things we will touch on with iocage is how we actually use ZFS. Every one of our jails has its own ZFS dataset. This allows you to take the jail, move it to a different system, and iocage will know just what to do when you use it. So when you set a property for the jail with 'iocage set prop=value', we are actually changing the ZFS properties on the jail's dataset. That means all the settings for each jail follow it around wherever you go, even with a brand new install of iocage that has another machine's disks that were previously used with iocage on that machine. This is really handy when you're moving zpools from host to host and you don't want to reconfigure your setup.

We also allow snapshotting of a jail. You can supply a snapshot name when you call 'iocage snapshot'. So this means you can do 'iocage snapshot -r example@before' which will recursively take a snapshot of the jail. This allows you to roll the jail back to whatever point in time you made that snapshot. This has the benefit of getting your jail configured when you don't want to

```
_/etc       ➔ /iocage/jails/89b2f41a-76b2-11e5-8df9-d05099728dbf/root/etc
_/root      ➔ /iocage/jails/89b2f41a-76b2-11e5-8df9-d05099728dbf/root/root
_/usr/home  ➔ /iocage/jails/89b2f41a-76b2-11e5-8df9-d05099728dbf/root/usr/home
_/usr/local ➔ /iocage/jails/89b2f41a-76b2-11e5-8df9-d05099728dbf/root/usr/local
_/usr/ports ➔ /iocage/jails/89b2f41a-76b2-11e5-8df9-d05099728dbf/root/usr/ports
_/var       ➔ /iocage/jails/89b2f41a-76b2-11e5-8df9-d05099728dbf/root/var
```

lose whatever state it was in before. You can tinker with peace of mind. You can also mount these snapshots and explore them, which can be invaluable.

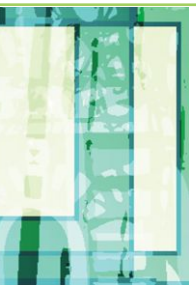
iocage tries to leverage everything it can with ZFS as it is a very powerful file system. This means you can even clone your jails or make templates out of them. All of this only takes up the differential space, which means you can freely experiment without having to worry. Templates are a feature that let you configure a jail just how you'd like it and make many more jails off of that template. This is going to be easier soon, as we will introduce batch jail creation to the tool. This will allow you to have a consistent naming scheme and customized jail base, and be able to have them numbered.

Hopefully this has given you a good idea of what iocage can offer and makes you want to learn more. As iocage is very flexible, there are simply too many use-cases to cover in this article. We love hearing the cool ways the tool is used, so feel free to show us your setup. Version 2.0 brings a lot of new, exciting things to iocage. Just some of the goodies coming soon are:

- Plugins
- A brand new basejail implementation
- Rewrites of many things such as snapshot, import/export, cloning, templates
- Ability to easily mount ports, src, and linprocfs
- Batch jail creation
- and a whole lot more!

There are a lot of other fun things iocage can do, and we invite you to try them yourself!

https://github.com/iocage/iocage



KRIS MOORE is the founder and lead developer of the PC-BSD project. He is also the co-host of the popular BSD Now (bsdnow.tv) video podcast. When not at home programming, he travels around the world giving talks and tutorials on various BSD-related topics at Linux and BSD conferences alike. He currently lives in Tennessee (USA) with his wife and five children and enjoys video gaming in his (very limited) spare time.

BRANDON SCHNEIDER is one of the two developers for the iocage project. He currently lives in Minnesota (USA). During his free time, he enjoys playing and talking about gaming and anything related to technology. He can be reached on Twitter @bschneider0922.