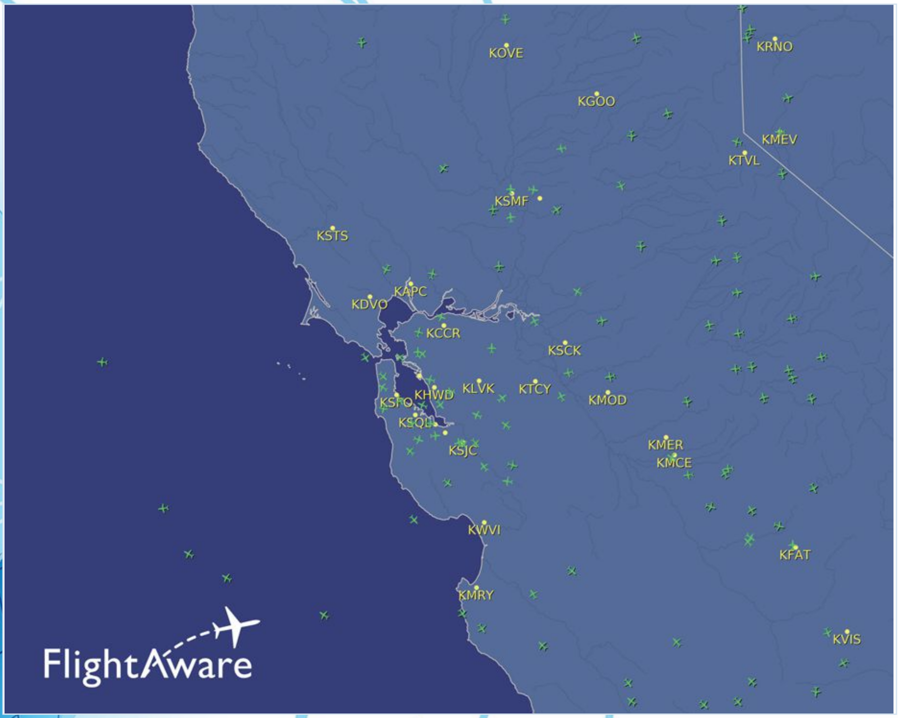
SEE TEXT ONLY

By Sean Kelly

FlightAware and *FreeBSD*

It *is* the greatest, mostly.



If you've not heard of us, FlightAware is the world's largest flight-tracking data company, providing over 10,000 aircraft operators and service companies with global flight-tracking solutions. We provide free flight tracking to the general public on our website and via our mobile apps, but we also have other commercial products and services used by aircraft operators around the world. We accomplish this by aggregating data from our own ground stations, government data sources covering over 55 countries, and other satellite-based providers. We then take all this data, interpret it, and turn it into more holistic views of what is going on. As you might expect, this requires a fair amount of processing and storage.

At FlightAware, our business to date has been built on top of FreeBSD, and we are very strong proponents of it. We have four employees who are inactive members of the FreeBSD Project, including our CEO and CTO. Up until very recently, FreeBSD was used almost entirely to power our site and services with only a handful of exceptions. This, however, is starting to change and I'd like to explain why. But first, let's cover our history with FreeBSD and how we've been using it.

Our CEO used to run ftp2.freebsd.org and he contributed to the ports tree. He has been using FreeBSD since the 2.0-ALPHA days and continues using it both professionally and personally to this day. Our CTO goes further back, to the beginnings of 386BSD, where he maintained patch kits and later contributed loopback detection to the T1 driver. Going back even further, one of our developers did a lot of hacking on 2BSD and 4.1BSD and later helped with the *FreeBSD Handbook*. Finally, I started using FreeBSD at the release of 4.0 and have contributed some patches as well as the initial implementation of the software watchdog subsystem.

Along with FreeBSD, we use the Tcl programming language extensively. This includes our website, which is written in Tcl and served by Apache's mod_rivet extension. Several ports of our Tcl packages exist in the ports tree, including speedtables, casstcl, yajl-tcl, tcllauncher, tclreadline, and tclbsd. These ports are maintained by tcltk@ and Pietro Cerruti. While Pietro is not a FlightAware employee, he generously updates ports to coincide with our software releases.

Some of the reasons that we do use FreeBSD include the stability and robustness of the network stack, the cohesive community, the holistic view of the system from kernel to userland, and ZFS. All these things combine to create an optimum and trustworthy open-source environment for our servers and services.

How We Use It

Like many others, we run web servers, Varnish servers, mail servers, and DNS servers on FreeBSD. While not entirely unusual, there are several other things we do on the FreeBSD platform that may interest others a bit more. These include a custom FreeBSD installer, high transaction RDBMS, a few different in-memory databases using shared memory, and, of course, all of these utilize the fantastic ZFS filesystem.

Installation: A Server Is Born

While bsdinstall is a great improvement on sysinstall, we still needed something that provided much more automation. Almost all of our servers start with the same base install, so it was unnecessary for us to repeatedly punch the same information into bsdinstall. Instead, we developed a set of scripts that incorporate bsdinstall and bsdconfig. These scripts run from a mfsBSD PXE environment and do the following:

- Destroy any existing hardware RAID configuration and partitions
- Wipe any existing ZFS labels on disks
- Create a new zpool on all internal disks
- Create all the appropriate ZFS filesystems to support boot environments
- Pull down the FreeBSD installer files and untar them
- Create a user
- Configure all network parameters

Once that is complete, the machine reboots and we're prompted with a menu where we can select the components we'd like installed. Each component is managed by a shell function inside of a shell script, making it easy to manage and modify. When we need a new component, we just create a new function.

All this evolved over time. Ideally, we'd like to be able to have an installer that will do a PXE boot, find a configuration file matching a MAC address, and do a full headless installation from that configuration file. It would be even better if this was built into FreeBSD so that FreeBSD had an easy zero-touch deployment system that was standardized and each shop didn't have to roll their own.

Traditional Databases: Make It Fast

While it is less novel than our custom installer, another use for FreeBSD at FlightAware is to power our PostgreSQL database cluster. We chose FreeBSD for this especially due to the data integrity features provided by ZFS. Another nice benefit is the LZ4 compression which yields a compression ratio of around 2.15x with a negligible performance impact. The cluster itself is a standard PostgreSQL 9.4 setup using the native streaming replication support. We use pgpool to load balance read-only transactions while all write transactions go to the master. This allows us to spread queries out over a larger number of servers, increasing our transaction processing potential due to most transactions being read-only.

In October 2015, we transitioned our PostgreSQL servers away from 24 hard disks in ZFS mirror pairs to use four NVMe SSDs in mirrored pairs. This was a huge performance gain once we got past issues with TRIM that eventually resulted in us disabling it. The NVMe subsystem timed out upon zpool create since ZFS was trying to TRIM the entire pool and NVMe gave up waiting on the SSD to do it. We also had issues during normal operation when TRIM was executed on Samsung SSDs that caused several outages. When PostgreSQL would clean up pg_xlog files, it would essentially briefly stall I/O due to TRIM. We've since switched to Intel SSDs which don't seem to exhibit the problem, but we left TRIM off anyway.

To back up all of this, we developed a custom toolset in Tcl that manages ZFS snapshots. The tool takes a snapshot on a database server, sends it to another host where it is staged, and from there the snapshot is dispersed to various sites. These same tools also manage the retention policy for the snapshots, keeping a certain amount of hourly, daily, and weekly snapshots on hand.

In-Memory Databases: Making Fast Faster

Along with our PostgreSQL databases, we also have internal software called Birdseye. Birdseye is an in-memory database written in Tcl and C, and based on our open-source speedtables project. Birdseye uses shared memory to store approximately 24 hours of positions for all aircraft that we know about. Multiple Birdseye processes on a single machine share the same shared memory segments so that more clients can be serviced concurrently. Clients, such as the website, can then connect to these Birdseye servers and make all sorts of queries to very quickly discern what a plane is doing, has done, or even what is happening within an arbitrary geographical box.

Along with Birdseye, we have another in-memory database of sorts called Superbird. Superbird, too, is based on our speedtables project and is written in Tcl with some generated C. Superbird will periodically read in any changes that have been made to PostgreSQL tables and keep them in an in-memory speedtables database. This is leveraged by the website so that it is possible to make faster queries of busy tables by accessing a local in-memory cached copy rather than needing to access an off-host PostgreSQL server. In other words, Superbird is a database caching layer utilizing update polling.

As you can see, shared memory is an important technology for us. We use it quite extensively with our internal tools as well as with third party projects like PostgreSQL. We've found that the overall performance of SHM has gotten better in the last several years and hope that it continues to do so. We noticed significant improvements with FreeBSD 9 and a bit again with FreeBSD 10.

Pain Points: Not Everything Is Perfect

Now that we've covered how FreeBSD works for us, we'd be remiss to not discuss our pain points. FlightAware is rapidly growing, as is our need to be able to deploy new systems and services at an ever-increasing rate. Further, data growth and increasing ingestion rates are causing us to reevaluate how we store, process, and analyze our datasets. This is where our choice of FreeBSD becomes harder to continue and justify.

Installation: An Arm of Servers

One amazing thing about FreeBSD is that it can be installed on a system and then mostly forgotten. You obviously want to install updates and so forth, but the system is generally rock solid and requires very little ongoing maintenance and care. As a result, we tend to treat our FreeBSD systems as unique rather than as a fleet of entirely dispensable resources. We'll add and remove software, but rarely do we find the need to do a clean install. The problem with this is that the approach doesn't scale well. As a result, we're starting to treat operating system installs as a commodity.

As I discussed earlier, we've built our own FreeBSD installer that mostly works for us to address this change. It isn't great and we'd like to add more features, such as the ability for it to TFTP a configuration file for entirely headless installs. Maybe there would be a base configuration file and then there could be a MAC address-based or SMBIOS system serial number-based override file. But instead of us implementing this for our own tools, it'd be fantastic if this was just part of the FreeBSD installer.

The FreeBSD installer should support entirely headless installations with documented scripting functionality. This is something that the Linux world conquered long ago with the Red Hat kickstart system or the Debian preconfiguration file. The installer could be PXE booted and then use DHCP options to direct it to where it can download configuration data. This would make it trivial to deploy or redeploy hundreds of servers at once.

Containers and Isolation: When Jail Isn't Enough

Now that I've shared my vision for how the FreeBSD installation process could be improved, let me explain what it is we want to do with fleets of heedlessly installed servers.

We are rapidly moving away from deploying services on a standard system installation. Like a lot of cloud scale companies, we are adopting a containerization approach that will allow us to easily deploy many services on a single piece of hardware without the software and library dependencies that this traditionally introduces. We'd like to have every base installation be identical and serve as nothing but a platform for hosting application containers. This simplifies installation and management.

As one potential path toward this, we did some evaluation of jails and found them mostly serviceable. However, we opted not to move in that direction due to several issues we faced. One glaring issue is the lack of per-jail shared memory. FreeBSD jails' shared memory is not isolated from each other or from the base system. The lack of separation means services like PostgreSQL and Birdseye can't be securely deployed in adjacent jails without special care to avoid conflict or exploitation. For example, running PostgreSQL in adjacent jails is risky due to the pgsql user having the same UID and thus each jail having access to the other jail's IPC resources.

Despite all of this, a lot of progress seems to have been made by adding RCTL to GENERIC, but we'd also like to see more focus in this area. VIMAGE should be made stable and added, as well. Further, we'd like to see more namespacing for things like shared memory implemented to provide true isolation. This is one area where Linux is ahead and has resulted in technologies like Docker and Kubernetes thriving.

The actual container mechanism is important, but how it is managed is also a big deal. As it is, FreeBSD has many ways to manage jails that are in various states of support and development. We've got ezjail, iocage, CBSD, jail(8)'s /etc/jail.conf, and many other homegrown tools and derivatives. While choice is almost always a good thing, this diverse ecosystem also makes it harder for automation and orchestration tools to target support for jails. It would be fantastic if there was a way as part of base to bring all of the diverse features under one roof—one set of APIs and commands to rule them all.

64-bit Java: Twice as Good as Before

As I already mentioned, our datasets are growing as is the amount of data we ingest. This has caused us to turn to newer technologies like Cassandra, Kafka, and Spark to store, move, and make sense of it all. All these have one thing in common: they run on the Java Runtime Environment.

There are not a lot of options for Java on FreeBSD. You can build and use the OpenJDK open-source implementation of Java which can be compiled into a native 64-bit implementation. You can also use the 32-bit Linux JRE from Oracle through the FreeBSD Linux emulation. However, only until very recently have 64-bit Linux system calls been supported and we've not successfully gotten the 64-bit Oracle JRE to run on FreeBSD. Finally, The FreeBSD Foundation used to offer a FreeBSD native binary for the Oracle JRE, but that is no longer available.

So you're probably wondering what our problem is and why we don't just use OpenJDK. The answer is that we've encountered issues with it that were hard to pin down. With Kafka, we saw periodic message corruption using OpenJDK that we didn't see with the Oracle JRE. Not to mention that the Oracle JRE is the authoritative implementation of Java which matters in a production environment. After that, we chose to use the Oracle JRE at least until we can circle back and retest and help fix OpenJDK.

We want 64-bit and we've chosen Oracle's JRE as the path forward. That means that we had to turn to Linux in order to run our Kafka, Cassandra, and Spark environments. As we continue to build out these services, this will result in us having more and more Linux.

System Tuning: My Server Isn't That Wimpy

Another area that could use more attention is out-of-the-box tuning for a FreeBSD install. While many defaults seem to be tuned for the absolute least common denominator of support, it'd be great to implement some way to have profiles that can be selected during install. Let me explain with concrete examples.

Right now, we compile our Tcl interpreter with an additional CFLAGS value of -DFD_SETSIZE=4096 to override the default of 1024 from <sys/select.h>. We know that select() isn't ideal and we've put out a bounty for Tcl kqueue support, but at the same time 1024 seems really small for modern day network applications. It'd be great to see this increased so that FreeBSD systems could be ready to handle all of the concurrent connections that it can otherwise effortlessly support without further tweaking by the user.

Default log rotation sizes are also something that needs some attention. According to the /etc/newsyslog.conf on a fresh FreeBSD install, most log files should be rotated after they grow to be 100KB in size. I don't know about you, but my /var/log/messages and my /var/log/auth.log don't take very long to reach that. In an era where you can get 10TB hard disks and 1TB SSDs, this seems like an obvious candidate for a profile system with options for embedded, server, desktop, etc., so that the target rotation size is more reasonable. Even 100KB seems small for a SD card-based system.

While I'm complaining about logging, this seems like the opportune time to suggest some features. Can we get an include directive for /etc/syslog.conf? newsyslog.conf has one, so it seems reasonable that syslog should as well. Some improvements to the ! program specifier in syslog.conf would be nice too. As it is, in order to log all messages from postgres to /var/log/postgres.log, I have to do something like:

```
# Capture PostgreSQL logs
!postgres
*.* /var/log/postgres.log

# Capture all other logs
!-postgres
*.* /var/log/all.log
```

It would be nice if there was a way to have a program specifier that matched anything except other defined program specifiers. Then you can have a catch-all section without having to list the exclusions out specifically. This will also work around another issue I encountered: the maximum line length of syslog.conf is too short for listing these out.

Finally, another area that could be improved is overall network stack tuning. If you search Google, you can find lots of different pages that tell you sysctls and tunables that you should be setting to maximize your network stack and 10GigE NIC throughput. It would be great if there was a profile selection during install that could tune things to be great for Varnish, nginx, Apache, and so forth, as well as GigE, 10GigE, or 40GigE networking. As it is, it is sort of a dark art to get it set up and fully understand it.

Debugging: Because Things Break

What would you do if the zpool command dumped core? For me, I'd fire up the debugger and hope to see where it crashed by loading up the core file. Unfortunately, the base system lacks debugging symbols when you install a release version of FreeBSD. Why is this? Are they too big, just like those 100KB log files? It would be nice if base binaries weren't stripped, or at least an optional system component included symbol files for all of the base binaries and libraries.

Similarly, the ports tree defaults to building software without symbols. Most ports have a DEBUG option you can enable that will cause the software to be built with symbols but generally without any compiler optimizations enabled. This isn't ideal for production. I'd like to see a universal option to build a port with optimization but also with debugging symbols, ideally by default. I understand that the debugging will be a bit harder with something built using -O or -O2, but it is sure better than nothing.

Finally, everything should have DTRACE by default. DTRACE is an amazing tool that FlightAware has used on more than one occasion to understand a performance problem or system failure. Both PostgreSQL and Tcl conveniently have extensive DTRACE support. However, ports does not enable this by default. In our opinion, every port should default to building with DTRACE. It isn't needed on a day-to-day basis but, like debugging symbols, it is invaluable when a problem eventually does appear.

Wrapping Up

I hope I've been able to make it clear that FlightAware really does prefer and advocate for FreeBSD. That being said, there is always room for improvement and I set out to explain here what could be improved in FreeBSD to make it work better for us and presumably others. It is time to start looking at ways to make it easier to increase the number of installs in the wild and make it easier to treat FreeBSD as a commodity platform. ●

SEAN KELLY has been a member of the FlightAware team since 2012. He serves as the Director of IT Operations. In this role, Sean designs and oversees the growth and maintenance of both the worldwide infrastructure that powers FlightAware as well as all of the IT systems powering FlightAware's two corporate offices. Sean is also a former committer of the FreeBSD Project and has been using FreeBSD since at least the 4.0 days.