Running Postgres as a Unikernel

Every now and then we get the question from an interested party along the lines of "So when is this *not* a good fit?" after someone realizes that pretty much most any daemonized service running on a x86 linux instance in the cloud (eg: all modern web application software) is a great fit for unikernels.

There is, indeed, a certain class of software from the mid 90s and earlier that traditionally has not been a great fit and in the past I would not hesitate to immediately paint postgres as the de-facto classic SYS V poster-child with its gratuitous use of multiple processes, shared memory, IPC signalling and *ahem*, 1666 global variables. I'm not knocking postgres here. You have to keep in mind that postgres was birthed in 1996 and descended from Stonebraker (the OG database godfather), et al's Post-Ingres from 1986. Even in 1996 this was a time before we had "proper" threads, before we had commercialized virtualization in the form of vmware && the public cloud and before commodity SMP processors so you can't fault anyone for the design decisions made 27 years ago.

Anywho, to cut to the chase, you can now run postgres as a nanos unikernel.

This isn't the first time we took a look at running postgres as a unikernel. A few years ago one of our engineers took a look at how difficult it might be to port it. He made a lot of progress but after spending a handful of days getting rid of shared memory and converting the processes to threads, then running into signal issues decided to punt it down the road until someone really wanted it.

Fast forward to last weekend and there was a LWN post making the rounds on HN and lobste.rs about a conversation on the pgsql-hackers mailing list with the subject line of:

Re: Let's make PostgreSQL multi-threaded.

This immediately caught my eye and my reaction to the question being discussed was essentially "Does a bear shit in the woods?" In the world of software sometimes a catch-phrase will get repeated so often that it becomes its own echo chamber. I don't know if there are comp sci. departments out there pushing some of these ideas or whether it is a devrel trying to get edgy in their conference talk but one such idea is the "threads are bad" idea. This has done a lot of damage to the world of software. It comes from the fact that a lot of developers will immediately say "its hard to write multi-threaded code" ergo "threads are bad". This of course is a logically invalid argument but it is memes like this that explain why we see software like this not receive the benefits of newer architecture and newer hardware over a 27 year timeline. Keep in mind in 2023 there are vc-funded postgres companies popping up left and right. I spotted members from both Neon and EnterpriseDB on this thread.

end rant

Little did I know that Konstantin Knizhnik had **already** converted the codebase over to a threaded system. As soon as I saw that I decided it was time to take another look.

The way I go about seeing if something 'just works' or not is pretty straight-forward. I just try to run it. In some cases I've used the software before and in many I have not. Having used postgres in prod quite a lot over the years and having already looked at in the past with unikernel glasses on I kinda already knew the issues that needed to be dealt with.

We first start removing calls to geteuid and getuid. Then we get rid of some more permission checks. While Nanos has the concept of read-only files and exec protection and other permission related capabilities such as support for pledge and unveil it doesn't really have the notion of 'users' as there is only one program running and there is no interactive shell to spawn other programs. It is not a general purpose operating system and has no intention of being one. Removing calls like this is one way. Another method I'll use sometimes when I can't or won't modify the code is to just throw in an override library via LD_PRELOAD that looks something like this:

```
cat ~/zz/l/l.c
int geteuid() {
   return 22;
}

int getuid() {
   return 42;
}
```

Then in the package I can just set it via an env var:

```
"Env": {
    "LD_PRELOAD": "l/l"
  },
```

Again, these checks make a ton of sense and are definitely warranted in linux land but this is Nanos.

Next, since this was an older codebase I was working off of I had to change the name of a method that clashed with something that existed in my libc and deal with a compiler error, also from a newer version of gcc.

Earlier on we had created a patch to replace the shared memory call with a heap allocated area which removed methods such as IpcMemoryDetach and IpcMemoryDelete for a much simpler calloc. We don't really do shared memory as it is *usually* one of those things that implies multiple processes (but not always!).

That is until I saw that we could simply edit the pg_hba.conf file and set the dynamic_shared_memory_type to 'mmap' instead of 'posix'. Yes, yes Andy I see the dirty looks - let me continue my show and tell.

Finally we need to ensure we can access it - the package in question allows 10.0.2.2/32 by default which assumes you are using user-mode (default network in ops). For prod/other deploys you simply need to copy in the correct network which in this case is in /db/pg_hba.conf.

I also ended up ifdef'ng the HAVE_SYNC_FILE_RANGE block in src/backend/storage/file/fd.c in favor MS_ASYNC as we don't have SYNC_FILE_RANGE but can (and will probably) easily add later. You might be wondering how I discovered that outside the flood of "could not flush dirty data" error messages I saw. When diving into a codebase like this the '--trace' flag, which is our version of strace, is invaluable to understand what calls the program is actually issuing and in this case which ones might be missing or failing.

There are other sundry items of course that if you're curious you can just take a look at the package.manifest or the sysroot but after all of that - we're now running postgres on the Nanos unikernel:

```
➜  ~ ops pkg load eyberg/postgresql:11.3.0 -p 5432
running local instance
booting /Users/eyberg/.ops/images/postgres ...
2023-06-21 04:17:55.133 UTC [23437153024] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2023-06-21 04:17:55.136 UTC [23437153024] LOG:  listening on Unix socket "/tmp/.s.PGSQL.5432"
2023-06-21 04:17:55.173 UTC [23437153024] LOG:  could not resolve "localhost": Temporary failure 
2023-06-21 04:17:55.176 UTC [23437153024] LOG:  disabling statistics collector for lack of workin
2023-06-21 04:17:55.178 UTC [23437153024] HINT:  Enable the "track_counts" option.
2023-06-21 04:17:55.185 UTC [5052757568] LOG:  database system was shut down at 2023-06-20 19:04:
2023-06-21 04:17:55.197 UTC [23437153024] LOG:  database system is ready to accept connections
en1: assigned 192.168.68.117
en1: assigned FE80::50CC:A2FF:FEA8:EBB1
```

Now there is clearly a lot of extra tweaking/coding that could be done to make this a lot nicer but we'll leave that as an exercise for another day. I should state that postgres is a prime example of something that took a lot of patching to make work in Nanos whether it is Konstantin's port or the patches we had. The vast majority of software in the web application space doesn't take any patching - just proper configuration and normal packaging that'd you find for any platform. Til next time.

Deploy Your First Open Source Unikernel In Seconds

Get Started Now.

vms

COMPANY

Press

Investors

Careers

Affiliates

MARKETS

Energy	Life Sciences
Health	Retail
Finance	Gaming
Telecom	Education
Government	High Performance Computing

RESOURCES

DataSheets

WhitePapers

Videos

Blog

Customer Portal

LEARN

Docker vs Unikernels

CONTACT

(833) NANO-VMS

(833) 626-6867

Terms of Service Privacy Policy