FreeBSD containers in production

(a NSFW guide)

Yan Ka, Chiu

Who am I

- FreeBSD user for 6-8 years
- B.A. in Mathematics
- Functional programming with Scala / Haskell / Erlang
- System programming with C / C++ / Rust

What do I do

- Live streaming + e-commerce company
  - Backend
  - DevOps
  - Basically everything except frontend
- FreeBSD on AWS

Container(?)

- "Containers are lightweight packages of your application code together with dependencies such as specific versions of programming language runtimes and libraries required to run your software services." -- Google

Container

- Environmental context of processes and services
  - file system (files and packages)
  - network (ip address and routes)
  - device nodes
  - sysv ipc
  - privileges
  - ...

Why container

- Generalized Continuous integration and continuous delivery
- Observability (trace by jid)
- Scalability (up and down)
- Privilege management

Container on FreeBSD

Contexts

- Root filesystem
- devfs ruleset
- Jail parameters
  - Sysv message queues
  - Sysv shared memory
  - sysctl

Container on FreeBSD

Base tools / utilities

- Jail (3)
- Jail (8)
  - utility to manage jail(s)
  - using jail(3)
  - (fake) life cycle management

Existing utilities

- Bastille
- locage
- Pot
- runj

Bastille

- Pros
  - ZFS is optional
  - Bastillefile (composable template/receipt)
  - Great Maintainability
- Cons
  - Jails are state preserving
  - "Dead" jail awareness is lacking

locage

- Pros
  - Plugins (Declarative Template)
  - Can manage stopped jail
- Cons
  - Require ZFS
  - Jails are state preserving

Pot

- Pros:
  - Great integration with Consul and Nomad (Orchestration)
  - Basically what you would want.
- Cons:
  - Require ZFS

So what do we actually use?

Why do we build our own tool

- ZFS on Root is not available on AWS
- Want declarative image* definition / manifest
- Want image registry
- Bonus: Jail over NFS

Container Image

Requirement

- Distributable
- Should not be considered trusted
- Self-Documented, e.g. usage and parameters
- Privileges should not be grant automatically

Container Image

Format

- JSON manifest
- Filesystems layers store as OCI compatible archive
- Privileges and requirements must explicitly documented
  - devfs rules requirements
  - ports it provides
  - sysv ipc
  - etc...

Container Image

Privilege / security model

- Host policy
  - Container can be spawned from an image without manual intervention if and only if the all requested privileges permit by host
  - Manual override is possible

Container Image
devfs ruleset handling

- Host defined open rules and close rules
- Host defined upper limit of usable rules
- jail_devfs_rules := host_open_rules || img_requested_rules || host_close_rules
- Housekeeping daemon keep track of the mappings between rules and ruleset id

Container Image

Creating image

- Only happen on ZFS enabled nodes
- Abuse ZFS
  - ZFS clone staged file system of the parent image
  - Apply changes, usually via a heavily modified Bastillefile
  - Create file system layer from `ZFS diff` output

Container root over NFS

- Remotely mount container's root via NFS
- union mount /var
- run it like a normal jail