Jetpack
Container runtime for FreeBSD

Maciej Pasternacki <maciej@3ofcoins.net>
Berlin DevOps 2015-01
1. OS-level virtualization across the ages

2. The container paradigm

3. Docker, Rocket, and breaking the monoculture

4. Jetpack: motivation, implementation, demo
OS-level virtualization

- Single host kernel, multiple isolated guest user spaces
- Not as universal as full virtualization
- Very small overhead
- Resources can be shared between instances
- Adjustable isolation level
NAME
   chroot – change root directory

LIBRARY
   Standard C Library (libc, -lc)

SYNOPSIS
   #include <unistd.h>

   int
   chroot(const char *dirname);

DESCRIPTION
   The dirname argument is the address of the pathname of a directory, terminated by an ASCII NUL. The chroot() system call causes dirname to become the root directory, that is, the starting point for path searches of pathnames beginning with ‘/’.

1982: Stone Age
1998–2012: Industrial Age

- **1998**: FreeBSD Jail
- **2001**: Linux-VServer, Virtuozzo
- **2005**: OpenVZ, Solaris Containers
- **2008**: Linux cgroups, LXC
1998-2012: Industrial Age

- Isolated filesystem, process tree, networking
- Restricted interaction between environments
- Restricted administrative system calls
- Resource usage limits
VM Mindset

Guest is a complete system:

- managed from the inside
- runs multiple services
- long-running and mutable
- opaque to host

Management overhead of a whole server
2013: Modern Age

- Jan 2013: Docker
- Dec 2014: CoreOS Rocket, App Container Specification
- Jan 2015: Jetpack
2013: Modern Age

- Inspired by PaaS, application-focused
- Guest managed from the outside
- Immutable, distributable images
- Fast copy-on-write provisioning
Container Mindset

- Layered storage
- Explicit interaction points
- Immutable images, volatile containers
- Service-oriented
Layered Storage
Interaction Points

- Command line arguments
- Environment variables
- Network ports
- Persistent volumes
- Stdin, stdout, stderr
- Exit status
Immutability

- Images, once built, are read-only
- Containers’ write layer is throwaway
- Volumes are persistent
Immutability

- **Images**, once built, can be repeatably reused
- **Containers** are exchangeable (upgrades!)
- **Volumes** declare user data
Service-oriented

- Well-defined images can be shared and reused across applications
- Containers can be meaningfully managed and monitored by host
- Management overhead of a service rather than whole machine
Docker

- First free container runtime
- Defined the paradigm
- Extremely fast and wide adoption
- Runs on Linux
- Implementation-driven
Docker

- Only free container runtime, until recently
- Prototyped the paradigm
- Extremely soon locked into early decisions
- Tied to Linux
- Implementation-defined

https://www.docker.com/
The management question, therefore, is not whether to build a pilot system and throw it away. You will do that. [...] Hence plan to throw one away; you will, anyhow.

(Fred Brooks, The Mythical Man-Month)
Rocket

- New container runtime by CoreOS
- Designed for “composability, security, and speed” (also simplicity & interoperability)
- Implementation follows (neutral) specification
- Breaks Docker monoculture
- Runs on Linux (uses systemd heavily)

https://coreos.com/blog/rocket/
App Container Specification

The "App Container" defines an image format, image discovery mechanism and execution environment that can exist in several independent implementations.

https://github.com/appc/spec
Jetpack

- (incomplete) App Container implementation for FreeBSD
- Written in Go
- Jails for process isolation & lockdown
- ZFS for layered storage
- Breaks Linux monoculture (hopefully)

https://github.com/3ofcoins/jetpack/
Jetpack: ZFS storage

- Snapshots/clones for layered storage
- Deduplication & compression conserves space
- Streaming allows easy distribution of complete set of images

https://github.com/3ofcoins/jetpack/
Jetpack: Building Images

jetpack image IMG build -dir=. COMMAND...

1. clone new container from IMG
2. copy build dir -dir to container
3. run COMMAND... inside container, in build dir
4. commit container’s rootfs without build dir as new image

https://github.com/3ofcoins/jetpack/
Jetpack: Building Images

jetpack image IMG build -dir=. COMMAND...

- COMMAND is toolchain-agnostic, it can be: ./setup.sh, make build, chef-solo...

- Doesn’t introduce a new file format
- A bsdmake include file is provided, but not required

https://github.com/3ofcoins/jetpack/
jetpack.image.mk

- Makefile include to simplify image building
- Prepares build dir on host
- `jetpack image ... build make build`

https://github.com/3ofcoins/jetpack/
jetpack.image.mk

.MAKEFLAGS: -I/usr/local/share/jetpack

PARENT_IMAGE = freebsd-base
PKG_INSTALL = nginx

build:
    # this runs after package is installed
    install -v -m 0640 -o root -g www \
        nginx.conf /usr/local/etc/nginx.conf

manifest.json:
    ./manifest.json.sh > @$

.include "jetpack.image.mk"

https://github.com/3ofcoins/jetpack/
PARENT_IMAGE = freebsd-base/release\$(RELEASE)
CLEAN_FILES  = entropy manifest.json
BUILD_VARS   = http_proxy

prepare:
  # this runs on host to prepare build dir
  dd if=/dev/random of=entropy bs=4096 count=1

build:
  sed -i '' 's|^Components.*|Components world/base|' \
     /etc/freebsd-update.conf
  install -v -m 0644 rc.conf /etc/rc.conf
  install -v -m 0600 entropy /entropy
  PAGER=cat freebsd-update -s update6.freebsd.org \ 
       fetch install
  rm -rf /var/db/freebsd-update/*

manifest.json:
  ./manifest.json.json.sh > $@
Jetpack: The Future

- Complete spec coverage
- Network stack separation (VIMAGE)
- Resource limiting (RCTL)
- Firewall/NAT rules management (pf)
- Image discovery & distribution
- A LOT MORE... it’s still a prototype!

https://github.com/3ofcoins/jetpack/
Demo Time!
https://github.com/3ofcoins/jetpack/

- https://github.com/appc/spec/
- https://coreos.com/blog/rocket/
- http://3ofcoins.net/2014/12/06/of-containers-dockers-rockets-and-daemons/
- https://www.docker.com/
- http://cryptome.org/cyberinsecurity.htm