Survey of Container Build Tools

Michael Ducy
Community & Evangelism - Sysdig

Agenda

- How containers should be used
- Problems with Dockerfile paradigm
- Specific tools
- Summary

About me

Spent the last 4.5 years at Chef

Cloud, Automation, Performance & Capacity, Ops

Ask me about goats (goatcan.do)

Triton, Maroon, Buckeye

Review of “What’s a Container”

Namespaces

Control what a process can see.

- PID
- Mount
- Network
- UTS
- IPC
- User
- Cgroup

Cgroups

Control what a process can use.

- Memory
- CPU
- Blkio
- Cpuacct
- Cpuset
- Devices
- Net_prio
- Freezer

Thanks to **Jess Frazelle (@jessfraz)** for this slide **https://goo.gl/7fVKFa**

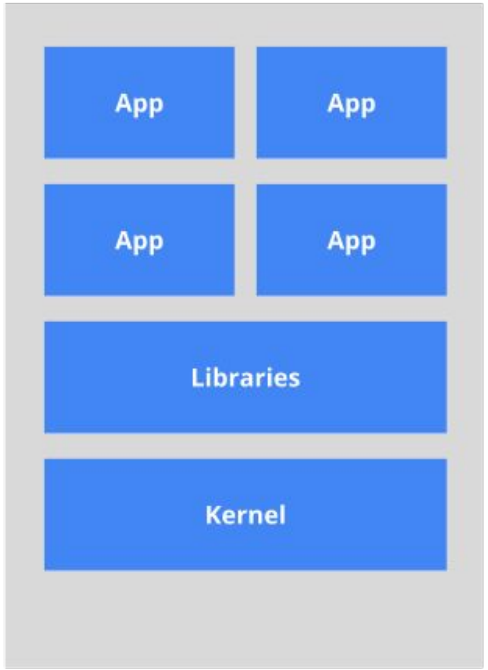
Containers vs. Zones vs. Jails vs. VMs

Containers	**Zones**	**Jails**	**VMs**
Cgroups	First class	First class	First class
Namespaces	concept	concept	concept
LSMs			

Read more about this here: https://blog.jessfraz.com/post/containers-zones-jails-vms/

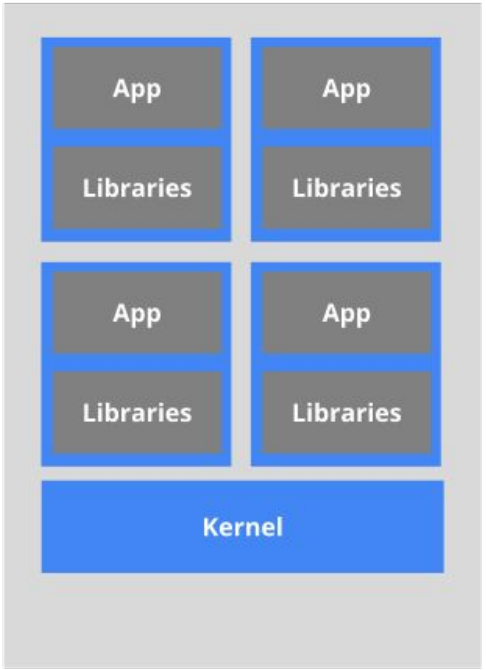
What's a Container

The old way: Applications on host



Heavyweight, non-portable
Relies on OS package manager

The new way: Deploy containers



Small and fast, portable
Uses OS-level virtualization

Problem with the Dockerfile paradigm

Builds aren’t deterministic or reproducable.

Programmability through Bash

Easy to turn a container image into a “VM”

Lack of visibility into what’s really in the final image

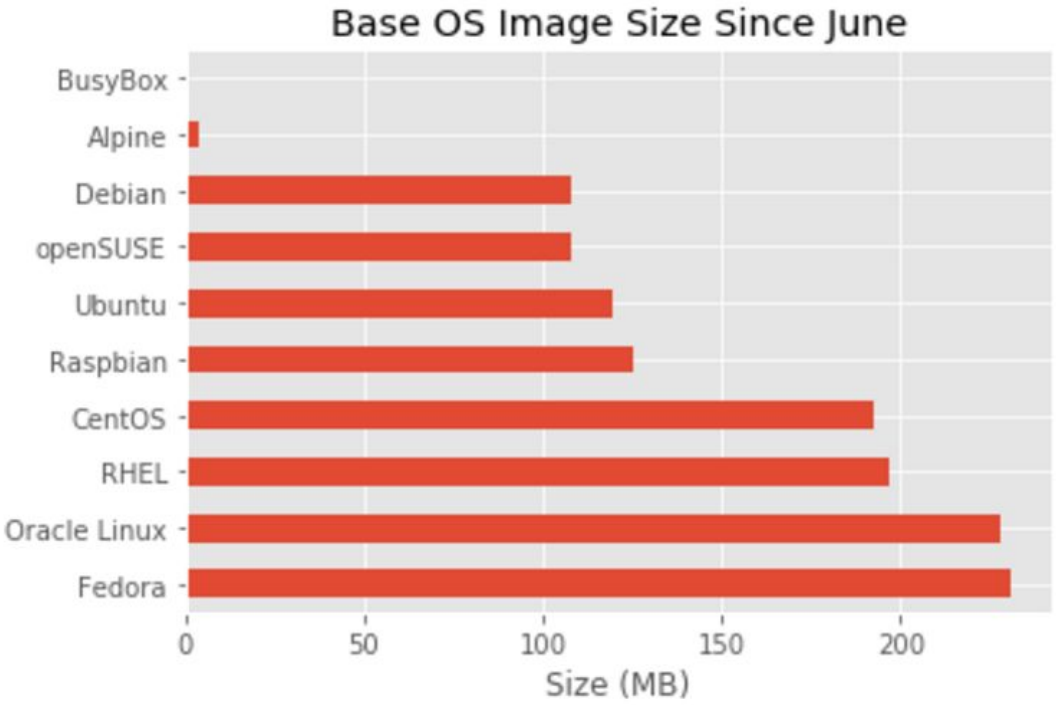
Bottom up approach vs application down

Real World Numbers Show the Problem

7 to 1 container to host ratio (DataDog survey, April 2017)

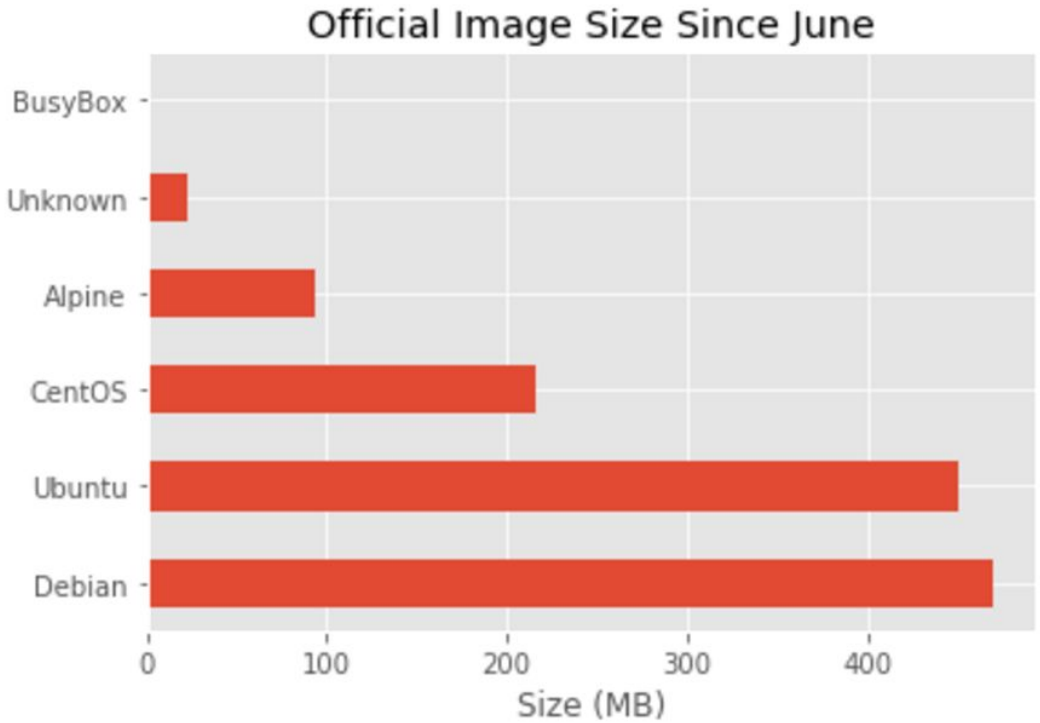
10 to 1 container to host ratio (Sysdig survey April 2017)

Base OS Image Size



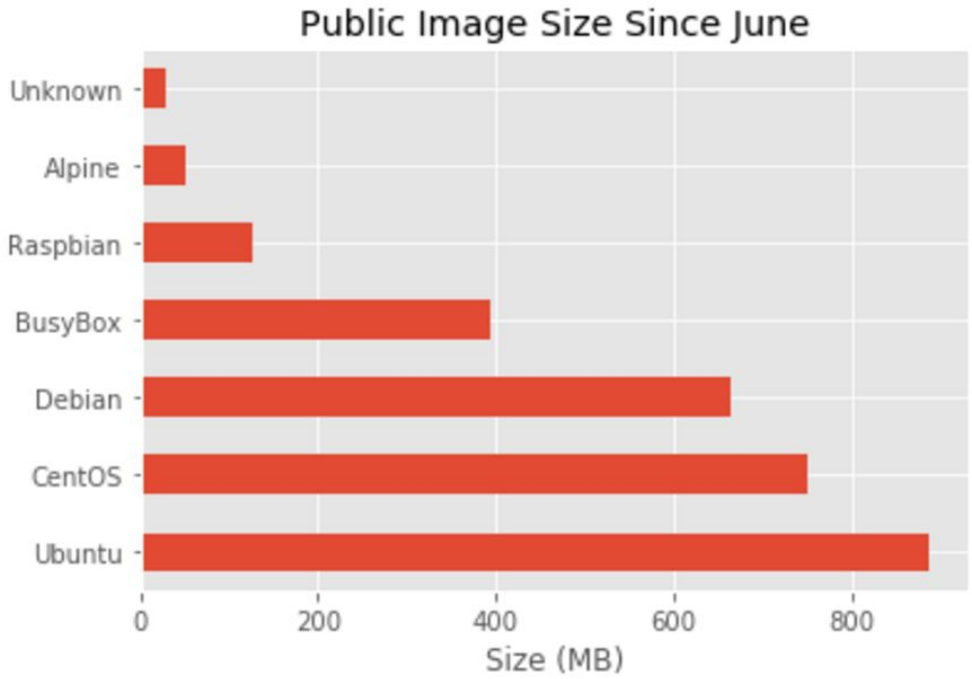
https://anchore.com/blog/breakdown-of-operating-systems-of-dockerhub/

Official Image Size



https://anchore.com/blog/breakdown-of-operating-systems-of-dockerhub/

Public Image Size



Sysdig

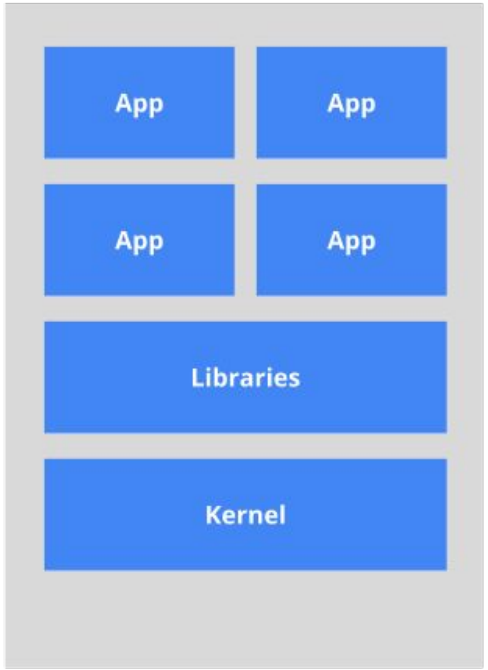
https://anchore.com/blog/breakdown-of-operating-systems-of-dockerhub/

@mfdii

What can we infer?

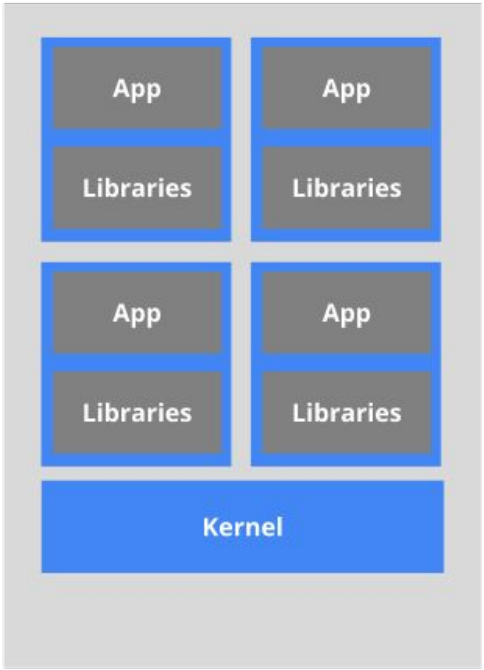
What can we infer?

The old way: Applications on host



Heavyweight, non-portable
Relies on OS package manager

The new way: Deploy containers



Small and fast, portable
Uses OS-level virtualization

What can we infer?

Nobody knows how to package their application in a container.

Lots of choices

Traditional Build Tools

- buildah
- nixos-container
- ansible-container
- Smith

- Distroless

Source-to-Container Tools

- Buildkit
- Source-to-Image (s2i)
- Habitat

buildah

buildah

buildah

Project from Project Atomic - https://github.com/projectatomic/buildah

OCI and Docker Image Formats

Doesn’t require a container runtime to build containers.

Allows for some _interesting_ uses which are 0_0 LOLWUT

Not deterministic.

nixos-container

Builds containers based on NixOS

Imperative and Declarative approaches

Containers can be auto-rebuilt when host OS updates

Ability to get very granular with software installed in a container due to Nix's packaging approach

nixos-container

Warning: Currently, NixOS containers are not perfectly isolated from the host system. This means that a user with root access to the container can do things that affect the host. So you should not give container root access to untrusted users.

nixos-container

Warning: Currently, NixOS conta ystem. This means that a user with root access to the cont u should not give container root access to untrusted users.

ansible-container

Declarative approach (mostly)

Allows you to build multiple containers from one container.yml (like docker-compose)

Allows you to take advantage of Ansible expertise/playbooks you already have

https://github.com/ansible/ansible-container

Smith

Smith

Focuses on building “microcontainers”

Principles of microcontainers

1. A microcontainer only contains the process to be run and its direct dependencies.
2. The microcontainer has files with no user ownership or special permissions beyond the executable bit.
3. The root filesystem of the container should be able to run read-only. All writes from the container should be into a directory called `/write`. Any unique config that an individual container instance will need should be placed into a directory called `/read`. Ephemeral files such as pid files can be written to `/run`.

Smith

Declarative (mostly)

Builds containers that are **significantly** smaller (httpd built with Smith is 3% of Dockerhub image)

Can use yum packages or Docker base images for source of binaries

https://hackernoon.com/how-to-build-a-tiny-httpd-container-ae622c37db39

What you want in a container

Distroless

Declarative builds leveraging Google's Bazel

'"Distroless" images contain only your application and its runtime dependencies. They do not contain package managers, shells any other programs you would expect to find in a standard Linux distribution.'

Provides stripped down base images

Support for language runtimes: Java, Python, Go, C, Node, dotnet

Source-to-Image

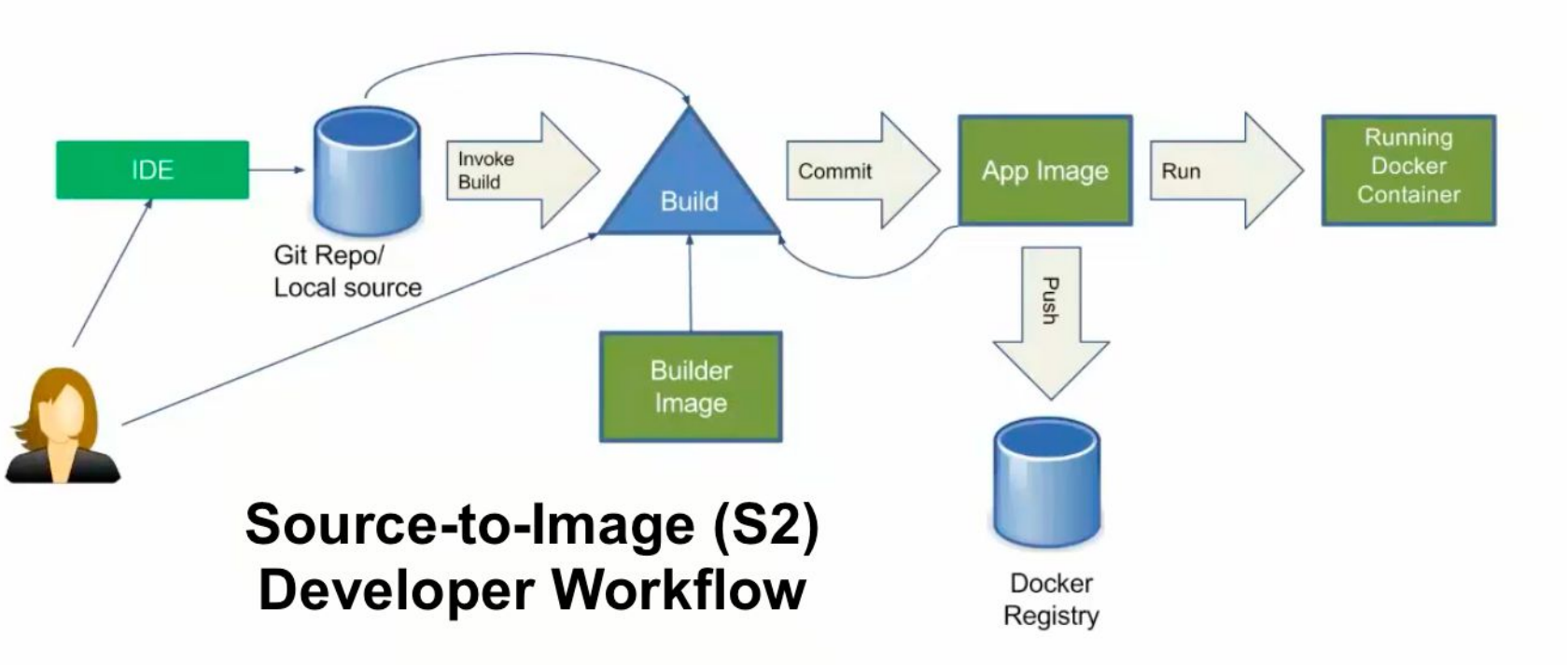
“Source-to-Image (S2I) is a toolkit and workflow for building reproducible Docker images from source code.”

One image can be used for builds, one can be used to run the artifact.

Allows build environments to be tightly versioned and controlled for reproducibility

Allows for control/security of the build environment through the use of build containers

Source-to-Image



Buildkit

“BuildKit is a toolkit for converting source code to build artifacts in an efficient, expressive and repeatable manner.”

Frontend’s specify a definition for how your software is built.

Buildkit takes this human readable definition (frontend) and transforms it into low level build (LLB) definition.

Exporters allow build artifacts to be exported in a variety of formats not just container formats.

Focuses on “How can we create generic primitives for a build system?”

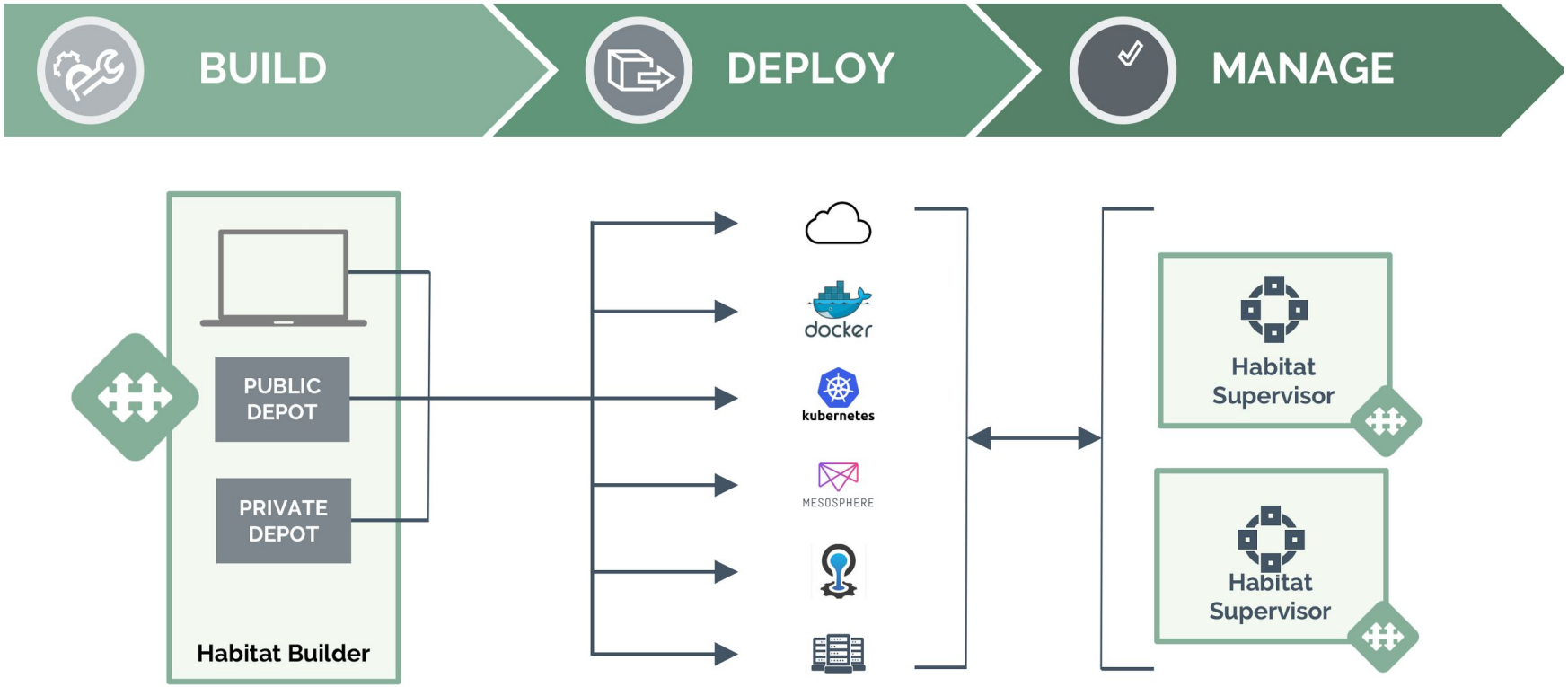
https://blog.mobyproject.org/introducing-buildkit-17e056cc5317

Buildkit


Frontends
LLB
Exporters
Dockerfile
...
concurrent execution
instruction caching
automatic storage management
cache import/export
tbd: remote workers
image
local files
oci image
oci bundle
vm
registry
...

Habitat

Habitat


BUILD
DEPLOY
MANAGE
PUBLIC DEPOT
PRIVATE DEPOT
Habitat Builder
docker
kubernetes
MESOSPHERE
Habitat Supervisor
Habitat Supervisor

Habitat

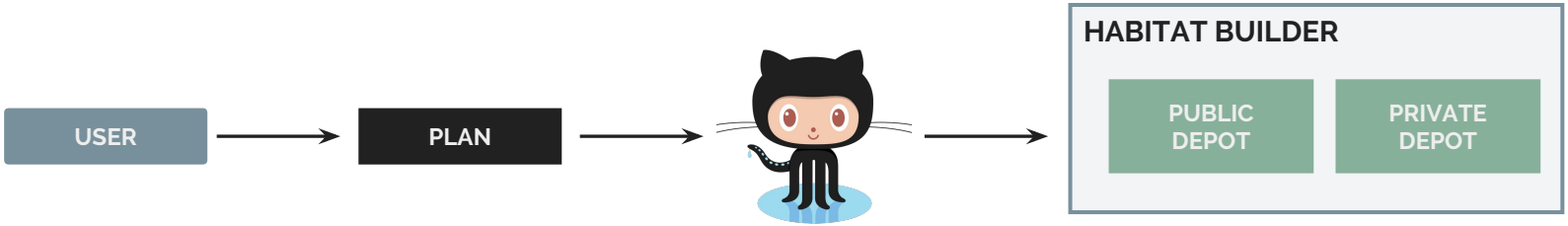
- Consistent process for packaging all apps across all architectures.
- Scaffolding for key languages: Node.js, Ruby, Go, Java.

- Export to variety of different formats:
  - Docker
  - ACI
  - CloudFoundry
  - Kubernetes
  - Mesos
  - tar

- Built in supervisor for Service Discovery, Templated Configs, Clustering Topologies, Health Check APIs, and more

Habitat Build Service

Habitat provides a cloud native build and packaging system



Developer adds a `plan.sh` to **define the build phases** of their software.

Scaffolding for common languages can be used provides sane defaults.

Developer **explicitly declares dependencies**, **required backing services**, and what **services/ports are exposed**.

Runtime lifecycle hooks can be defined for the application (`start, reconfigure`, etc).

Templated configuration files can be included.

Habitat Builder watches a GitHub repository for changes, when code is merged a build is performed based on the repo's `plan.sh`.

Build artifact includes **runtime lifecycle hooks, configuration,** and **application binaries** or **code**.

Build artifact **metadata** includes **dependent services, exposed services/ports,** and **required dependencies**.

Artifacts are stored in a **Habitat Builder depot**.

Lots of choices

- buildah
  - Still coupling yourself to the operating system paradigms
- nixos-container
  - Niche OS, with it's own insecure container model. Have to eat NixOS
- ansible-container
  - Great if you've committed to Ansible.
  - Ansible Service Bus for stateful services
- Smith
  - Microcontainers are what you want (++)
  - Janky to pull apart container images and still leverages OS package managers (to an extent)
- Distroless
  - Removes the OS (++), very language specific
  - Bazel is not the most approachable tool and real world examples are minimal

Lots of choices

- Buildkit
  - Very interesting approach to solve the problem of building software in general.
  - Versioned build environments possible
  - Still too early, examples sparse, frontends for languages non-existent
- s2i
  - Versioned, secured build environments
  - Library of build/run images but built on OS paradigm
- Habitat
  - Easily describes software builds in BASH
  - Export formats for multiple platforms (++)
  - Does the right thing to determine what a build artifact needs to run
  - Have to eat the Supervisor which doesn't fit with Kubernetes paradigms
  - Software libraries provided OOTB are not well maintained

Summary

Container Build Tools still have a long way to go.

Each tool has ++ and --

Some tools sacrifice “best practice” for approachability

Some tools make things overly complex

What do we need?

- Buildpack type model for Source Code building
- Declarative container build manifest generated from the build
- Exporter to create container image of choice with only the app and deps

Thank you

Slides:

https://www.slideshare.net/MichaelDucy/survey-of-container-build-tools

ALSO Sysdig is hiring

https://sysdig.com/jobs/