Hands-On With Docker Containers

Peter Fales

Peter@Fales-Lorenz.net

Uniforum Chicago

March 22, 2018

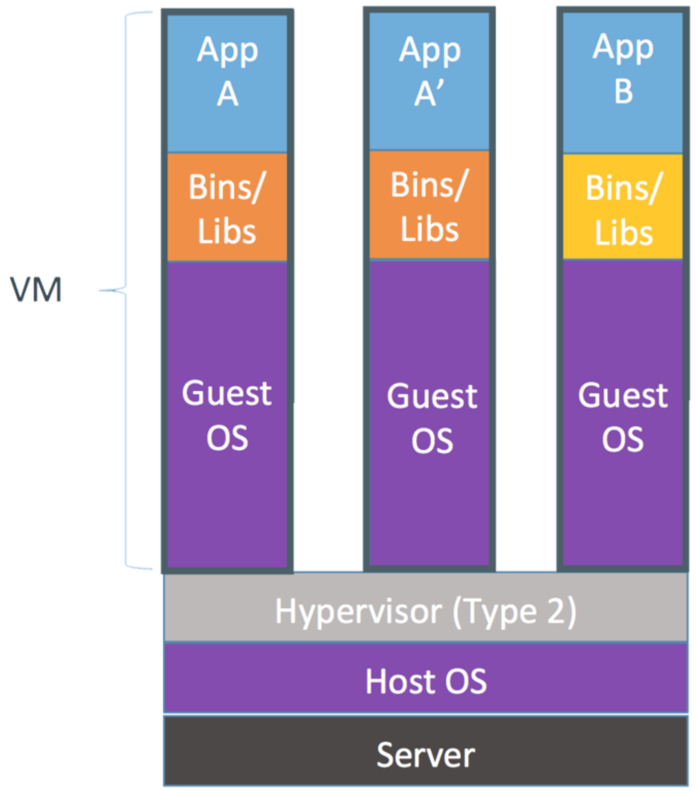
Topics

- Overview of Docker
  - Intro for new users – focus is on high level, introduce concepts, whet your appetite
- Motivation – why Docker
- Basic Concepts
- Demo of basic concepts and operations
  - Install Docker, start containers, create custom images, etc..

What is Docker

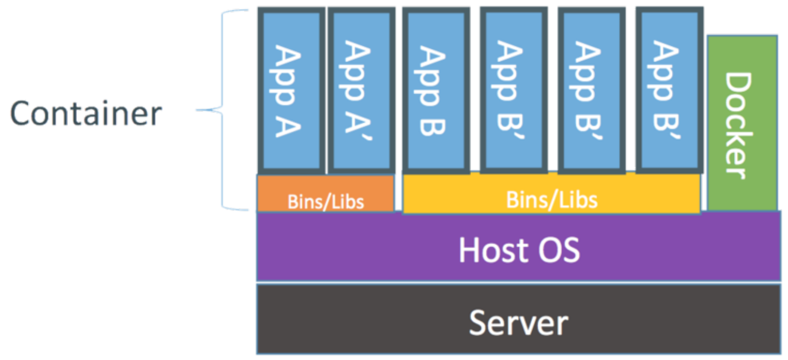
- Virtualized container execution environment
  - Builds on native linux facilities: cgroups, namespaces, networks, etc. for resource prioritization, limitation, and isolation
- Application deployment engine
  - Simplifies creation, distribution, execution of virtual images
  - Shipping container paradigm

Docker (Container) vs. VM



Containers are isolated, but share OS and, where appropriate, bins/libraries

...result is significantly faster deployment, much less overhead, easier migration, faster restart



Why Docker?

- Goals
  - Lightweight, fast, easy to use
  - Logical segregation, enhance consistency (reduce “it worked in dev”)
  - Reduce development cycle time
  - Encourages (but does not require) “microservices” architecture (modularity)
  - Layering allows for version control and rollback

Concepts

- Docker client and server (Docker Engine)
- Images
- Containers
- Registries/Repositories
- Dockerfile
- Volumes
- Networking

Docker Client and Server

- Heavy lifting done by server, **dockerd**, aka Docker Engine
  - Manages images and containers
- Controlled from:
  - Command line client, **docker**, either on local machine or over network
  - Restful API

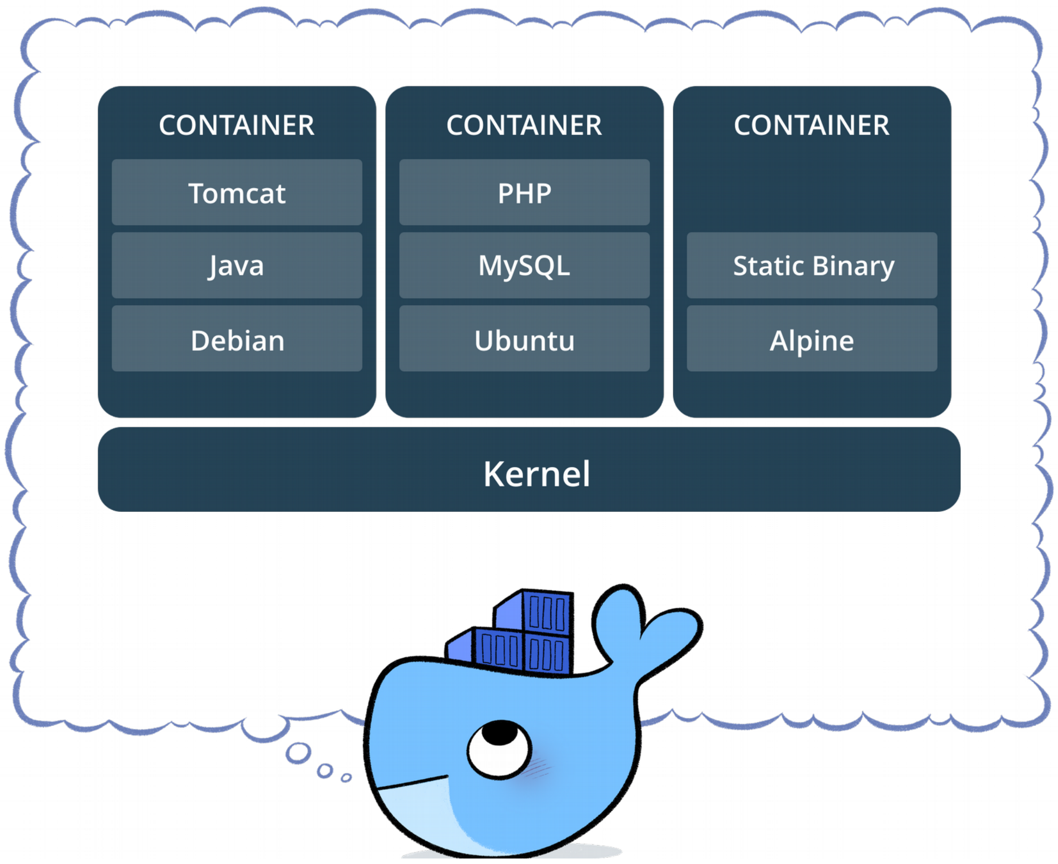
Docker Images

- Building blocks or “source code” for containers
- Layered using Union file systems
  - Add a file
  - Open a port
  - And lots more...
- Can be built interactively, but more typically
- Built from a “recipe” in a Dockerfile
  - Ensures that images are complete, consistent, and correct

Docker Containers

- Running instance of an image
- Multiple containers can run from one image
- Concept comes from standard shipping container:
  - Standard operations (create, start, stop)
  - Unique contents

Docker Containers



https://www.docker.com/sites/default/files/Package%20software.png

Registries/Repositories

- Centralized storage of images
- Facilitates sharing and distribution
- Can be
  - Public (e.g. docker.com/docker.io) – Default if not otherwise specified
  - Private – Privately hosted, or docker.com
- Named as host:port/namespace/image:

FULL: `localhost:5000/psfales/my_image`
OR WITH DEFAULTS: `ubuntu`

Dockerfile

- Step-by-step “recipe” for building a image
- Each step is a layer – caching can speed up rebuild process
- Numerous commands: ADD, RUN, EXPOSE, ENV, etc.

```
FROM  httpd
MAINTAINER Peter Fales
RUN apt-get update
ADD index.html /usr/local/apache2/htdocs
```

Volumes

- Makes a directory outside the container visible to the container
  - Permanent Data Store
  - Shared access from container and host
- Two types of Volumes
  - Native (host) file system
  - Volume Containers

Networking

- Very powerful & flexible – can be complex, but easy to get started
  - Expose ports to the outside (non-Docker) world
  - Link connectors via internal ports or named hosts
  - Overlay network extends over multiple hosts
  - Other mechanisms managed by Orchestration systems

Orchestration (examples)

- Docker Compose – create application stacks (web server, application server, database)
- Docker Swarm - create scalable clusters
- Docker-machine – alternative for managing machines and clusters, including cloud services
- Apache Mesos - “A distributed systems kernel” - API’s for resource management and scheduling across entire datacenter and cloud environments.
- Google Kubernetes - an open-source system for automating deployment, scaling, and management of containerized applications.

Why?

- Detailed recipes ensure consistency over time, or between development, test, and production (“Run anywhere”)
- Consistent environment – develop on desktop, deploy on cloud
- Lightweight – quick development cycle, not resource intensive

Why not?

- If you need kernel modifications
- If you need cross-platform (e.g. IIS, or other applications that run only on Windows)
- If you need rich user interfaces, such as X (but workarounds)
- May be ephemeral (workarounds)
- May increase security attack space
- Performance limitations of Union file system. (workarounds)

http://www.channelfutures.com/open-source/when-not-use-docker-understanding-limitations-containers

Docker Environments

- Linux host running Ubuntu, Debian, RHEL, CentOS, Scientific Linux, Fedora, or others
- OS X using “Docker for Mac” (virtual machine)
- Microsoft Windows using “Docker for Windows” (virtual machine)

Installation on Fedora

- Install Docker

```
# yum -y install docker
```

- Start Docker

```
# systemctl start docker
```

- Enable at boot time

```
# systemctl enable docker
```

- Test

```
# docker info
Containers: 0
Images: 0
...
```

Resources

- https://docs.docker.com/articles/dockerfile_best-practices/
- https://www.packtpub.com/networking-and-servers/learning-docker-second-edition
- https://www.dockerbook.com/
- https://github.com/wsargent/docker-cheat-sheet

Live Demo

Videos

- DOCKER (8 minutes)
  https://www.youtube.com/watch?v=pGYAg7TMmp0
- KUBERNETES (6.5 minutes)

  https://www.youtube.com/watch?v=R-3dfURb2hA

Q & A