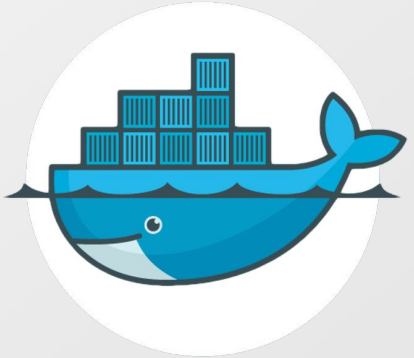

MICHIGAN IT
SYMPOSIUM
2016
UNIVERSITY OF
MICHIGAN

2016 IT Symposium: An Introduction to Docker



Bob Killen: Advanced Research Computing
Chris Kretler: ITS - Teaching and Learning

Please Do!

1. Download or git clone: https://github.com/chriskretler/nginx-docker-demo
2. If using the zip download, unpack in a clean local directory
3. docker pull nginx

Who are we?

- Chris Kretler
- ckretler@umich.edu
- github.com/chriskretler
- With U since 2007
- Helping development groups on campus get to know and use Docker
- Currently leading ITS effort to deploy docker apps on OpenShift

- Bob Killen
- rkillen@umich.edu
- github.com/mrbobbytables
- @mrbobbytables
- University employee for 14 years
- Over 3 years of experience with Docker
- Deployed Docker in a production setting
- Experience with all major container orchestration platforms (Cattle, Kubernetes, Mesos and Swarm)

UNIVERSITY OF MICHIGAN

Overview

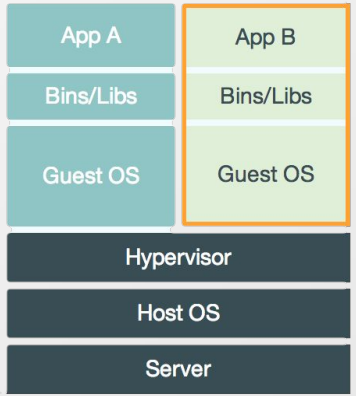
- **Docker Overview**
- Benefits of Using Docker
- Brief Tutorial
- More Features

What is Docker?

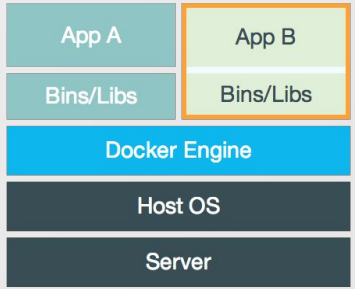
- An Open Source platform for developing, distributing, and running applications.
- Built around the concept of a ‘container’.
  - Lightweight OS level virtual machine.
  - Bundles an application, its dependencies, and other run-time requirements into an immutable, redistributable image.
- Powered by Linux Container Subsystem
  - Provides isolation of resources - CPU, memory, and network via kernel namespaces and cgroups.
- Available on multiple platforms and architectures
  - Linux (x86_64 and ARM)
  - Windows 10, server 2016
  - OSX 10.10+

What Docker is not

- A VM.
  - Docker is not a hypervisor, and a container is not a full virtual machine*
- Persistent.
  - A container's default behaviour is to be ephemeral.
- Secure by default.
  - A container is not a security panacea. It does have the capability of being more secure, but effort is required.



Virtual Machine



Container

* Containers are run within a VM on non Linux Platforms

Docker Core Components

Engine	Image	Registry
		
• The Docker Daemon running on a host. • Manages building, storing, and running images on a specific host.	• The item that is executed by the engine. • Images consists of a manifest and collection of read-only layers generated by a Dockerfile.	• The service that acts as an image repository. • Example: Docker hub

Docker Workflow

Build	Ship	Run
• Build, develop and join the components of your application together into an image or set of images.	• Push your image to a registry making it widely available.	• Deploy, manage and use the same containers your application was developed on in a production environment.

Build - Images

- Docker images are built from a ‘Dockerfile’.
- Dockerfiles contains a list of commands and actions that are used by the Docker Engine to assemble an image. Examples include ‘FROM,’ ‘RUN’, ‘COPY’, ‘EXPOSE’ and ‘CMD’.

helloworld (demo1)

```
FROM alpine
RUN echo "hello world" > example
CMD ["cat", "example"]
```

Dockerfile Command Reference: https://docs.docker.com/engine/reference/builder/

Build - Images

helloworld (demo1)

```
FROM alpine
RUN echo "hello world" > example
CMD ["cat", "example"]
```

- Use Alpine as the base image.
- Run the command writing ‘hello world’ to ‘example’
- Instruct the container to execute the ‘cat example’ command when run.

```
muninn:demo1 bob$ docker build -t helloworld .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM alpine
 ---> baa5d63471ea
Step 2 : RUN echo "hello world" > example
 ---> Running in 931cf474a585
 ---> 3f2c16891e5b
Removing intermediate container 931cf474a585
Step 3 : CMD cat example
 ---> Running in b92112c29e03
 ---> 70289614bb95
Removing intermediate container b92112c29e03
Successfully built 70289614bb95
```

```
muninn:demo1 bob$ docker run helloworld
hello world
```

Build - Images - Layers

- An image is composed of layers.
- Each layer has an associated UUID and checksum.
- If multiple images do the same thing within their Dockerfiles, only a single layer for that step will be created.

helloworld (demo1)

```
FROM alpine
RUN echo "hello world" > example
CMD ["cat", "example"]
```

layers (demo2)

```
FROM alpine
RUN echo "hello world" > example
CMD [ "echo", "I'm in a container~!"]
```

Build - Images - Layers

```
muninn:demo1 bob$ docker history helloworld
IMAGE               CREATED             CREATED BY                                      SIZE
70289614bb95        About a minute ago  /bin/sh -c #(nop)  CMD ["cat" "example"]        0 B
3f2c16891e5b        About a minute ago  /bin/sh -c echo "hello world" > example          12 B
baa5d63471ea        3 weeks ago         /bin/sh -c #(nop) ADD file:7afbc23fda8b0b3872   4.803 MB
```

baa5d3471ea

3f2c16891e5b

Base layers are the same.

```
muninn:demo2 bob$ docker history layers
IMAGE               CREATED             CREATED BY                                      SIZE
d3cb4c2894ff        4 seconds ago       /bin/sh -c #(nop)  CMD ["echo" "I'm in a cont   0 B
3f2c16891e5b        34 minutes ago      /bin/sh -c echo "hello world" > example          12 B
baa5d63471ea        3 weeks ago         /bin/sh -c #(nop) ADD file:7afbc23fda8b0b3872   4.803 MB
```

Build - Images - FROM

- Images inherit the parent layers via the FROM directive.
- Encourages the workflow pattern of chaining images together.
- Docker Hub and many other build systems can trigger automatic rebuild of a container upon update of the parent.

from (demo3)

```
FROM layers
CMD [ "cat", "example"]
```

```
muninn:demo3 bob$ docker run from
hello world
```

```
IMAGE          CREATED          CREATED BY                                      SIZE
121b43b4f1e9   46 minutes ago   /bin/sh -c #(nop)  CMD ["cat" "example"]        0 B
d3cb4c2894ff   19 hours ago     /bin/sh -c #(nop)  CMD ["echo" "I'm in a cont   0 B
3f2c16891e5b   19 hours ago     /bin/sh -c echo "hello world" > example         12 B
baa5d63471ea   3 weeks ago      /bin/sh -c #(nop) ADD file:7afbc23fda8b0b3872   4.803 MB
```

Ship - Push

- A built image is tagged with a namespace and tag.
- A Namespace is usually a user or organization name combined with the name of the container itself.
- A tag is a little piece of metadata about the image, usually related to the version of the application in the image or the version of the image itself.
- Hub Name Format: <namespace>:<tag> - **mrbobbytables/layers:1**
- Private Registry Name Format: <private registry url>/<namespace>:<tag>
    - **registry.example.com/mrbobbytables/layers:1**
- Only pushes layers that are not stored within the registry already.

```
muninn:demo2-layers bob$ docker tag layers mrbobbytables/layers:1
muninn:demo2-layers bob$ docker push mrbobbytables/layers:1
The push refers to a repository [docker.io/mrbobbytables/layers]
af7da5be3005: Layer already exists
011b303988d2: Layer already exists
1: digest: sha256:d8a2fefd97c0c2b86d2f2e6457ed5fb6c6d64c5017056e6145c47a7263b7a3a3 size: 735
muninn:demo2-layers bob$ docker pull mrbobbytables/layers:1
1: Pulling from mrbobbytables/layers

Digest: sha256:d8a2fefd97c0c2b86d2f2e6457ed5fb6c6d64c5017056e6145c47a7263b7a3a3
Status: Image is up to date for mrbobbytables/layers:1
```

Ship - Pull

- Pulling an image only pulls layers not currently downloaded.

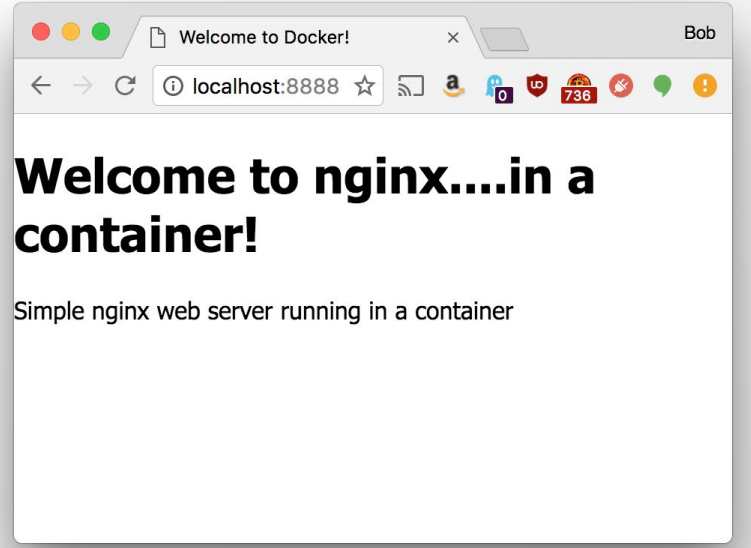
```
muninn2:demo2-layers bob$ docker pull mrbobbytables/layers:1
1: Pulling from mrbobbytables/layers

Digest: sha256:d8a2fefd97c0c2b86d2f2e6457ed5fb6c6d64c5017056e6145c47a7263b7a3a3
Status: Downloaded newer image for mrbobbytables/layers:1
```

Run

- Upon run, containers are assigned a unique id and randomly generated name (unless supplied)
- Resources must be explicitly mapped or exposed (ports, volumes etc.)
- These resources can be shared with other containers, or exposed on the host itself.
- It’s best to think of these exposed resources as a service, or the consumable portion of the container.

Run

```
muninn2:demo4-run bob$ docker run -d -v $(pwd):/usr/share/nginx/html -p 8888:80 nginx:stable-alpine
320a6676b00c995e3e1a47c2085219244e08a496bff0272a976ea53282e37415
```



- **-d**
  - tells docker to ‘daemonize’ the instance.
- **-v $(pwd):/usr/share/nginx/html**
  - mounts the current working directory to the /usr/share/nginx/html directory within the container.
- **-p 8888:80**
  - instructs the engine to map the host port 8888 to the container port 80.
- **nginx:stable-alpine**
  - The name of the image to run

Run

- Docker Run Reference:
  https://docs.docker.com/engine/reference/run/

Overview

- Docker Overview
- **Benefits of Using Docker**
- Brief Tutorial
- More Features

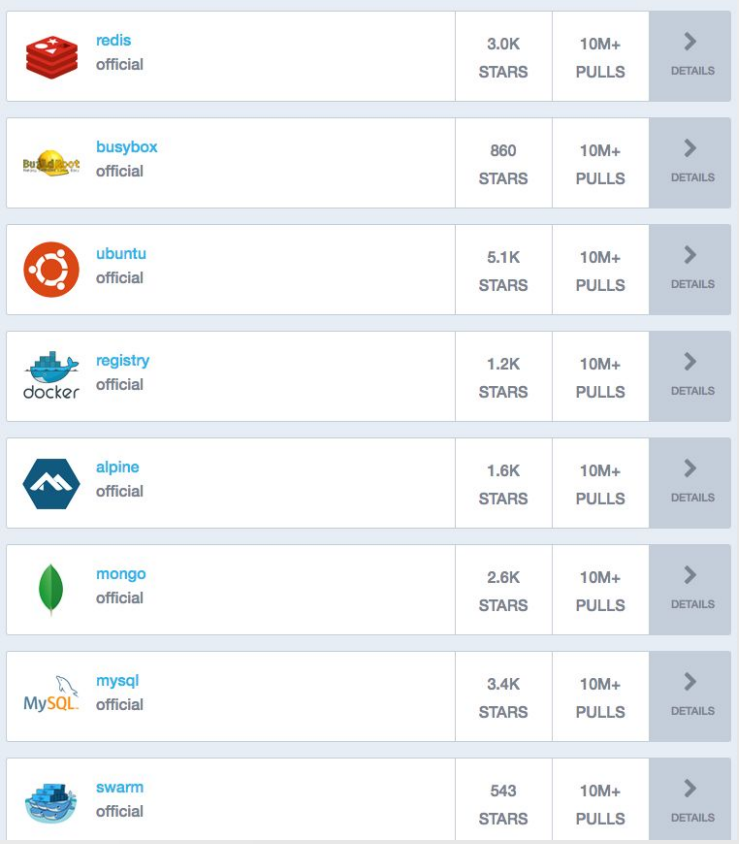
Benefits: Development

- Application environmental dependencies are bundled & specified:
  - No conflicts with existing applications in multi-tenanted situation.
  - What you build on your laptop will run the same way in any environment.
  - CI: Promotion with confidence.


```
FROM ubuntu:16.04
MAINTAINER Teaching and Learning Devs <its-tl-dev@umich.edu>

# execute this separate to allow caching
RUN apt-get update

# install dependencies and link node to nodejs, as ubuntu doesn't do this by default.
RUN apt-get install -y nodejs curl libmysqlclient-dev python-dev gunicorn python-pip
django-filter python-ldap npm git libssl-dev libffi-dev xmlsec1 && \
    ln -s /usr/bin/nodejs /usr/bin/node

# install dependent python packages that aren't available as pre-built libraries
RUN pip install --upgrade pip && \
    pip install Django==1.9 cryptography==1.5.1 mysql-python djangorestframework
django-crispy-forms whitenoise requests djangosaml2 coverage

# install node packages that aren't available as pre-built libraries
RUN npm install -g bower grunt grunt-cli

# expose port, do this before source code install to minimize layering
EXPOSE 8000

# create place for app to run from
WORKDIR /app/
COPY . /app/

RUN echo '{ "allow_root": true }' > /root/.bowerrc

# Install Ocellus
RUN cd mbofui && \
    bower install && \
    npm install && \
    grunt docker

# copy settings file and launch django
CMD python manage.py migrate; gunicorn --workers=4 --bind=0.0.0.0:8000
hacks_mbof.wsgi:application
```

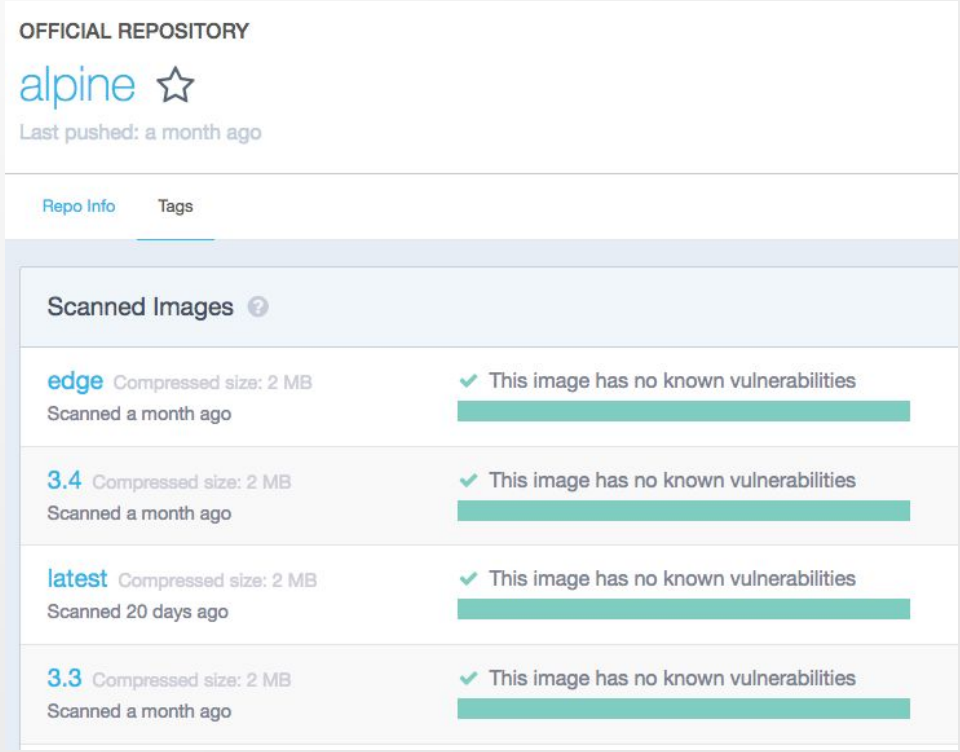
Benefits: Development

- Image registries (Docker Hub) facilitates experimentation with different platforms
  - Officially maintained environments for Python, Ruby, NodeJS, Java etc.
  - Components like redis, mongo, elasticsearch, postgress
  - latest / specific tags



Benefits: Operations

- Robust production ecosystem:
  - Zero-downtime deployments
  - Automatic scaling
  - Lower resource usage
  - Application separation
- Operations can maintain base images with organizational requirements bundled in private registries.



Overview

- Docker Overview
- Benefits of Using Docker
- **Brief Tutorial**
- More Features

Demo

- You are going to have a working web application at the end of this demo!
- Download repo: https://github.com/chriskretler/nginx-docker-demo
- Application overview
- Docker build based on downloaded files
- Docker run & again
- Modify text blob index.htm
- Docker build to create image
  - Talk about caching and layers
- Docker run
  - See your updates

Overview

- Docker Overview
- Benefits of Using Docker
- Brief Tutorial
- **More Features**

Docker-Compose

- Compose is a lightweight tool built to define and run multi-container applications.
- It uses a simple, but feature complete yaml DSL to describe container infrastructure in a '*docker-compose.yml*' file.
- It manages container infrastructure by breaking it down to 3 major components:
  - Services
  - Networks
  - Volumes

docker-compose.yml

compose-wordpress/docker-compose.yml

```yaml
version: '2'
services:
  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    networks:
      - backend
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_PASSWORD: wordpress
  db:
    image: mysql:5.7
    networks:
      - backend
    volumes:
      - db_data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: wordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
  lb:
    image: dockercloud/haproxy
    depends_on:
      - wordpress
    ports:
      - 80:80
    networks:
      - backend
      - public
    environment:
      - ADDITIONAL_SERVICES=composewordpress:wordpress
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
```

```yaml
networks:
  public:
    driver: bridge
  backend:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 192.168.223.0/24
          gateway: 192.168.223.1
volumes:
    db_data:
      driver: local
```

Compose-Networks

- Networks directive define consumable networks for the services.
- There are a variety of drivers, but the most common include:
  - Bridge (default)
  - Host
  - Overlay (multi-host capable)
- Containers can be attached to more than one network.
  - A '*frontend*' or '*public*' network will usually map a load balancer to one or more web servers.
  - A backend network will map the web servers to the database or application servers.

```
networks:
  public:
    driver: bridge
  backend:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 192.168.223.0/24
          gateway: 192.168.223.1
```

Compose-Volumes

```
volumes:
    db_data:
      driver: local
```

- The volumes directive contains named volumes only and does **NOT** include standard bind mounted volumes.
- Volumes created this way do not map to a user-definable directory on the host unless using a 3rd party driver (‘*local-persist*’).
  - https://github.com/CWSpear/local-persist
- Intended purpose is to consume storage in a cluster deployment.
- For cluster deployments, evaluate EMC’s rexray driver:
  - https://github.com/codedellemc/rexray

Compose-Services

- A service represents a container and defines the ways it interacts with external systems as well as other containers.
- Example: '*depends_on*' - configures startup order. The '*wordpress*' container will not attempt to start until the '*db*' container is up and running.
- Services can be attached to explicitly defined networks.
- Volumes in the service context can reference both bind mounts and named volumes.

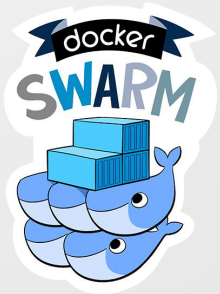
```
version: '2'
services:
  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    networks:
      - backend
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_PASSWORD: wordpress
  db:
    image: mysql:5.7
    networks:
      - backend
    volumes:
      - db_data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: wordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
  lb:
    image: dockercloud/haproxy
    depends_on:
      - wordpress
    ports:
      - 80:80
    networks:
      - backend
      - public
    environment:
      - ADDITIONAL_SERVICES=composewordpress:wordpress
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
```

Orchestration

kubernetes

MESOS

RANCHER



Orchestration

Kubernetes	Mesos	Rancher	Swarm
• Developed by Google • Large Community • Built for Web Scale • Very flexible • Complex • High learning curve	• Apache Project. • Designed to be the '*kernel*' of your datacenter • Best for Data Science applications • Complex • High Learning Curve	• Easy to use • Can manage other orchestration systems • Large catalog of services already built • Large system overhead • Not as fault tolerant as other systems	• Easiest to deploy and use • Extensive security built into engine • Best when bringing a compose based stack • Not as flexible as other options

Where to go from here

- Demo Files:
  - https://github.com/arc-ts/umitsm16-docker
- Docker Labs (Tutorials):
  - https://github.com/docker/labs
- Docker Run Reference:
  - https://docs.docker.com/engine/reference/run/
- Dockerfile Command Reference:
  - https://docs.docker.com/engine/reference/builder/
- Dockerfile Best Practices:
  - https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/
- Dockerfile Linter:
  - http://hadolint.lukasmartinelli.ch/

Where to go from here

Additional Questions

rkillen@umich.edu

ckretler@umich.edu

ARC-TS hosting Docker / Containerization workshops in the future.

http://arc-ts.umich.edu