What is Docker?

Presented by: Derek Nelson

Docker is a technology that uses virtualization to create and run *images*. Images can then be run across a wide variety of platfroms without any configuration.

It's a simple tech. Almost all use cases fit onto a single page.

Glossary

- **image** - An image is a Docker package which can be run or deployed to a registry.
- **Dockerfile** - A file used to define an image.
- **build** - The process of turning a Dockerfile into an image.
- **container** - A container is an instance of a docker image.
- **mount** - A directory bound from the host machine into the docker instance.
- **registry** - A place to store built docker images. Docker Hub is a popular registry.
- **compose** - A command line tool which is used to execute and store run configurations for one or many docker images.

Writing a Dockerfile

Example:

```
FROM ubuntu:19.10

RUN apt update && \
    apt install -y \
      libreoffice \
      libreoffice-java-common \
      openjdk-8-jre --no-install-recommends

RUN mkdir /root/input /root/output

CMD soffice --headless --convert-to pdf \
        /root/input/shane.txt --outdir /root/output
```

Writing a Dockerfile: Keywords

- **FROM** *{IMAGE_NAME}:{TAG}*
- **RUN** *{COMMAND}* - Runs a command as part of the build
- **CMD** *{COMMAND}* - Default command for image
- **ENV** *{VARIABLE_NAME}={VALUE}* - Setting environmental variables
- **USER** *{USERNAME}* - Change to user
- **WORKDIR** *{DIRECTORY}* - Change current working directory
- **COPY** *{HOST_DIRECTORY} {LOCAL_DIRECTORY}* - Copy host directory into image.

Docker CLI

Build

```
docker build -t {IMAGE_NAME}:{TAG} \
  {PATH_TO_DOCKERFILE | .}
```

Run

```
docker run --rm -it -v {HOST_PATH}:{DOCKER_PATH} \
        -p {HOST_PORT}:{DOCKER_PORT} {COMMAND}
```

- -it (interactive tty) : Allows you to interact using the CLI
- -v (mount volume) : Mounts a host path inside your Docker container
- -w (working directory) : Specifies the working directory
- -p (publish port) : Exposes a port
- --rm (remove) : Removes the container on exit

Docker CLI

Exec

(Like run but runs a specific command against a running container.)

Security

- Only use docker images from **known sources**
- Update packages

```
apt update && apt install -y
```

- Create an 'appuser' to run your program

```
useradd -d /app -s /sbin/nologo -u 1000 appuser
```

- Ensure ownership and rights of system directories

```
find $sysdirs -xdev -type d \
        -exec chown root:root {} \; \
        -exec chmod 0755 {} \;
```

Security (continued)

- Remove interactive shell for non-appuser users

```
sed -i -r '/^appuser:/! \
        s#^(.*):[^:]*$#\1:/sbin/nologin#' \
    /etc/passwd
```

Limiting attack *surface area*

Surface area refers to the number of vectors an image can be attacked from. This can be simplified to the number of things installed on your image.

- Use a multi-staged build to build your app and then put the finished build on a fresh docker image.
- Do not include dev dependencies.

Alpine Linux

Alpine linux is a very small security-oriented linux distro often paired with docker to create images with the smallest surface area.

Alpine docker images are also (*usually*) faster to download and startup than other distros. A base image is currently under 5MB.

Intro to Docker Compose

Docker Compose is a command line tool for specifying run configurations and running the Docker CLI.

Sample Docker Compose

```
version: '3'
services:
  web:
    build: .
    volumes:
      - ./host-dir:/app
      - ./another-host-dir:/working-directory
    ports:
      - "5000:5000"
  redis:
    image: "redis:alpine"
```

Glossary

- version: the version of the docker-compose spec the file is written using.
- services: contains all of the docker images contained in the run configuration.
- build: points to a Dockerfile or directory containing a Dockerfile.
- volumes: mounts host directories inside of a docker image.
- ports: exposes internal ports to the host machine.

Docker Compose CLI

Build

```
# Build everything
cd /path-to-docker-compose-yml;
docker-compose build;
```

Up

```
# Bring up everything
docker-compose up
```

Down

```
# Bring everything down
docker-compose down
```

Docker Compose CLI (continued)

Start

```
# Start a specific service
docker-compose start {SERVICE_NAME}
```

Stop

```
# Stop a specific service
docker-compose stop {SERVICE_NAME}
```

Tips

Docker **will** fill up your disk. Always use --rm with *docker run*.

If you do fill up your disk a quick way to clean up space is:

```
docker system prune -a
```