What's In This Section

- Check versions of our docker cli and engine
- Create a Nginx (web server) container
- Learn common container management commands
- Learn Docker networking basics
- Requirements: Have latest Docker installed from last Section

What We Covered

- command: `docker version`
  - verified cli can talk to engine
- command: `docker info`
  - most config values of engine
- docker command line structure
  - old (still works): `docker <command> (options)`
  - new: `docker <command> <sub-command> (options)`

Container VS Virtual Machine

- Containers aren't Mini-VM's
- They are just processes
- Limited to what resources they can access (file paths, network devices, running processes)
- Exit when process stops
- docker run --name mongo -d mongo
- docker top mongo
- ps aux
- docker stop mongo
- ps aux

This Lecture

- image vs. container
- run/stop/remove containers
- check container logs and processes

Image vs. Container

- An Image is the application we want to run
- A Container is an instance of that image running as a process
- You can have many containers running off the same image
- In this lecture our image will be the Nginx web server
- Docker's default image "registry" is called Docker Hub (hub.docker.com)

docker container run --publish 80:80 nginx

1. Downloaded image 'nginx' from Docker Hub
2. Started a new container from that image
3. Opened port 80 on the host IP
4. Routes that traffic to the container IP, port 80

What happens in 'docker container run'

1. Looks for that image locally in image cache, doesn't find anything
2. Then looks in remote image repository (defaults to Docker Hub)
3. Downloads the latest version (nginx:latest by default)
4. Creates new container based on that image and prepares to start
5. Gives it a virtual IP on a private network inside docker engine
6. Opens up port 80 on host and forwards to port 80 in container
7. Starts container by using the CMD in the image Dockerfile

Example Of Changing The Defaults

change version of image

```
docker container run --publish 8080:80 --name webhost -d nginx:1.11 nginx
-T
```

change host listening port

change CMD run on start

Assignment: Manage Multiple Containers

- docs.docker.com and `--help` are your friend
- Run a `nginx`, a `mysql`, and a `httpd` (apache) server
- Run all of them `--detach` (or `-d`), name them with `--name`
- nginx should listen on `80:80`, httpd on `8080:80`, mysql on `3306:3306`
- When running `mysql`, use the `--env` option (or `-e`) to pass in `MYSQL_RANDOM_ROOT_PASSWORD=yes`
- Use `docker container logs` on mysql to find the random password it created on startup
- Clean it all up with `docker container stop` and `docker container rm` (both can accept multiple names or ID's)
- Use `docker container ls` to ensure everything is correct before and after cleanup

What's Going On In Containers

- `docker container top` - process list in one container
- `docker container inspect` - details of one container config
- `docker container stats` - performance stats for all containers

Getting a Shell Inside Containers

- `docker container run -it` - start new container interactively
- `docker container exec -it` - run additional command in existing container
- Different Linux distros in containers

Docker Networks: Concepts

- Review of `docker container run -p`
- For local dev/testing, networks usually "just work"
- Quick port check with `docker container port <container>`
- Learn concepts of Docker Networking
- Understand how network packets move around Docker

Docker Networks Defaults

- Each container connected to a private virtual network "bridge"
- Each virtual network routes through NAT firewall on host IP
- All containers on a virtual network can talk to each other without -p
- Best practice is to create a new virtual network for each app:
  - network "my_web_app" for mysql and php/apache containers
  - network "my_api" for mongo and nodejs containers

Docker Networks Cont.

- "Batteries Included, But Removable"
  - Defaults work well in many cases, but easy to swap out parts to customize it
- Make new virtual networks
- Attach containers to more then one virtual network (or none)
- Skip virtual networks and use host IP (--net=host)
- Use different Docker network drivers to gain new abilities
- and much more...

Docker Networks: CLI Management

- Show networks `docker network ls`
- Inspect a network `docker network inspect`
- Create a network `docker network create --driver`
- Attach a network to container `docker network connect`
- Detach a network from container `docker network disconnect`

Docker Networks: Default Security

- Create your apps so frontend/backend sit on same Docker network
- Their inter-communication never leaves host
- All externally exposed ports closed by default
- You must manually expose via `-p`, which is better default security!
- This gets even better later with Swarm and Overlay networks

Docker Networks: DNS

- Understand how DNS is the key to easy inter-container comms
- See how it works by default with custom networks
- Learn how to use `--link` to enable DNS on default bridge network

Docker Networks: DNS

- Containers shouldn't rely on IP's for inter-communication
- DNS for friendly names is built-in if you use custom networks
- You're using custom networks right?
- This gets way easier with Docker Compose in future Section

Assignment Requirements: CLI App Testing

- Know how to use -it to get shell in container
- Understand basics of what a Linux distribution is like Ubuntu and CentOS
- Know how to run a container 😬

Assignment: CLI App Testing

- Use different Linux distro containers to check `curl` cli tool version
- Use two different terminal windows to start bash in both `centos:7` and `ubuntu:14.04`, using `-it`
- Learn the `docker container run –rm` option so you can save cleanup
- Ensure `curl` is installed and on latest version for that distro
  - ubuntu: `apt-get update && apt-get install curl`
  - centos: `yum update curl`
- Check `curl --version`

Assignment Requirements: DNS RR Test

- Know how to use -it to get shell in container
- Understand basics of what a Linux distribution is like Ubuntu and CentOS
- Know how to run a container 😬
- Understand basics of DNS records

Assignment: DNS Round Robin Test

- Ever since Docker Engine 1.11, we can have multiple containers on a created network respond to the same DNS address
- Create a new virtual network (default bridge driver)
- Create two containers from `elasticsearch:2` image
- Research and use `—network-alias search` when creating them to give them an additional DNS name to respond to
- Run `alpine nslookup search` with `--net` to see the two containers list for the same DNS name
- Run `centos curl -s search:9200` with `--net` multiple times until you see both "name" fields show