SPICE WORLD 2019

Docker for Dummies

SPICEWORLD2019

Nick Leghorn

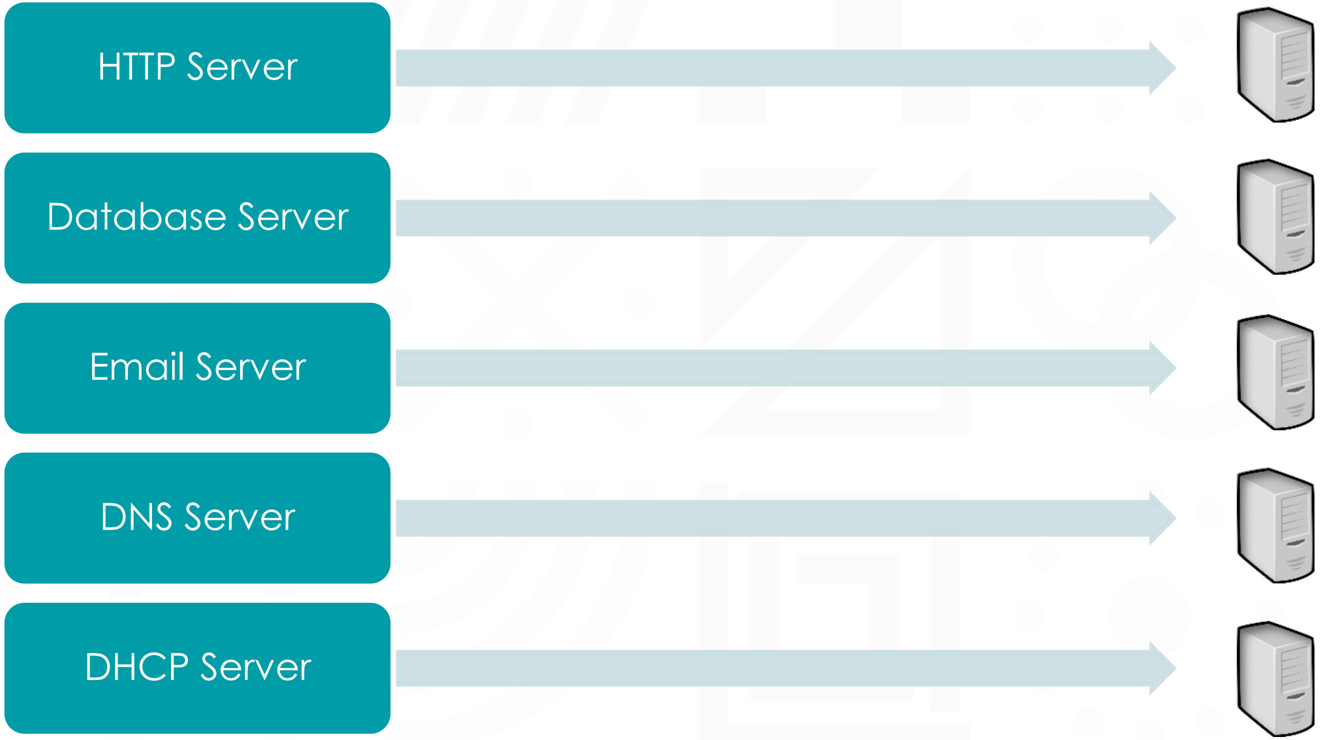
Manager of Security Engineering, Indeed.com

SPICEWORLD2019

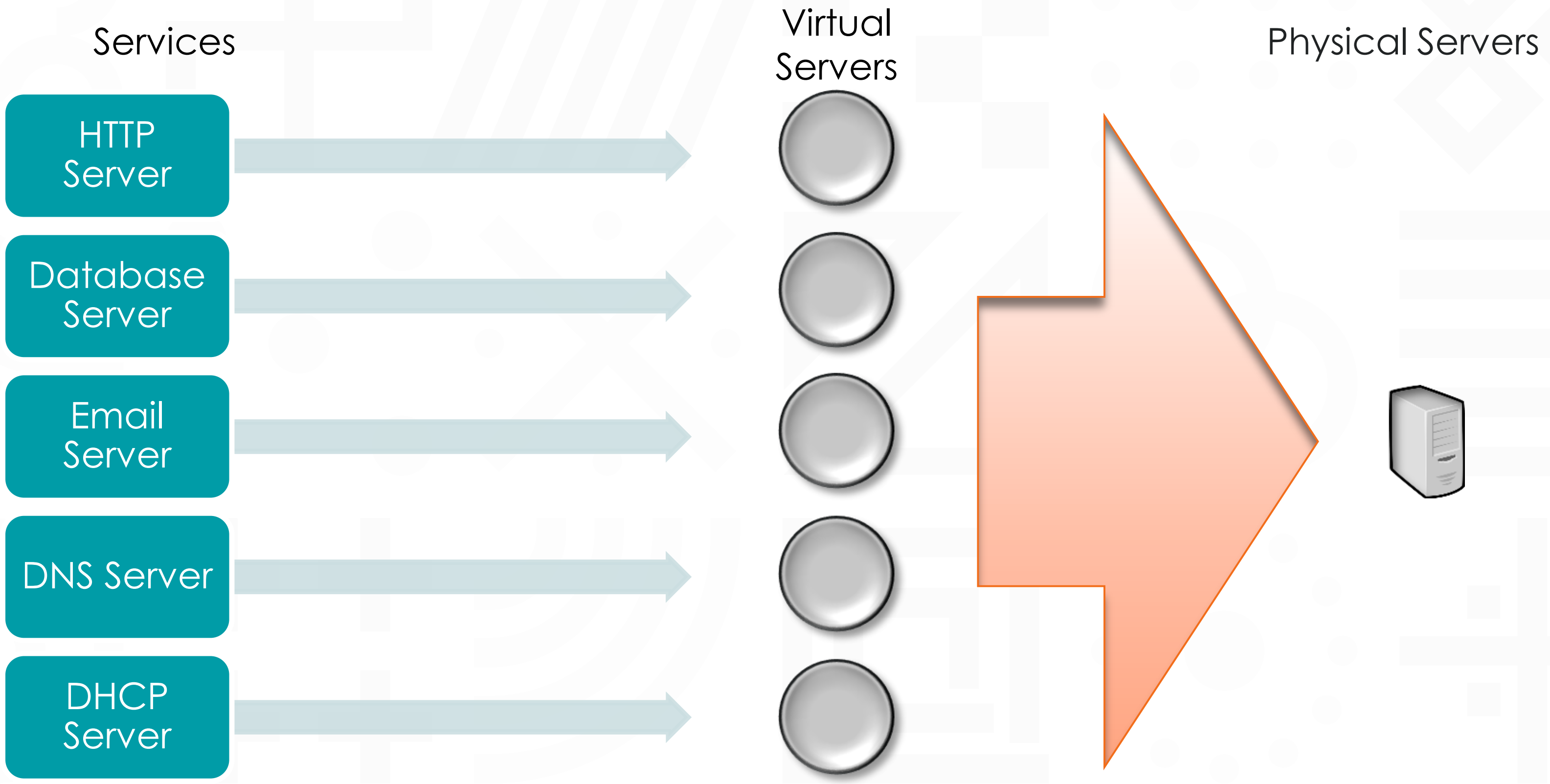
Agenda

1. What is Docker and Why Do I Care?
2. Setting Up A Standalone Docker System
3. Understanding The Bells and Whistles
4. Use Cases
    a. Standalone PiHole container
    b. HTTP Web server with mounted directory
    c. Deploy a LibreNMS Monitoring Suite on Linked Containers
5. Docker Swarm, Kubernetes, and Beyond

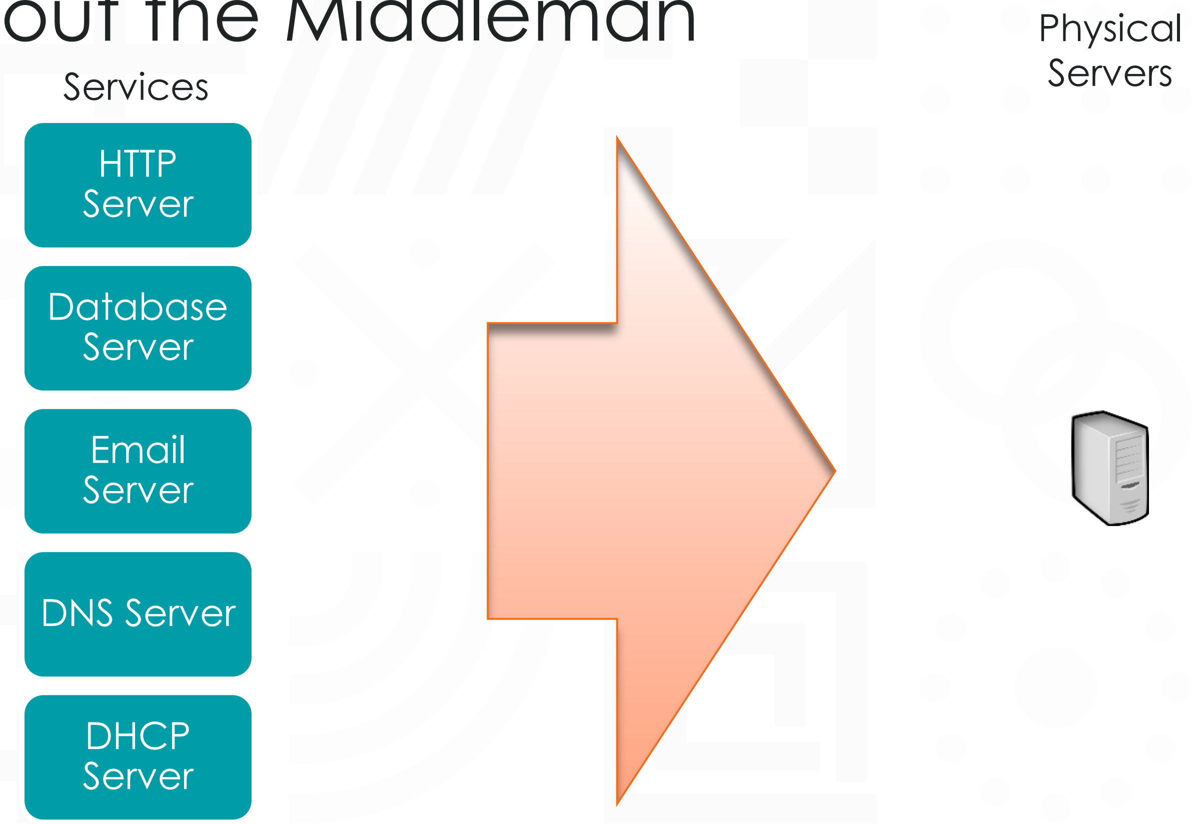
In The Beginning...



…And Then Came VMware

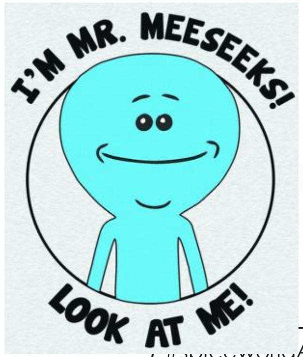


Docker: Like VMware, But Without the Middleman



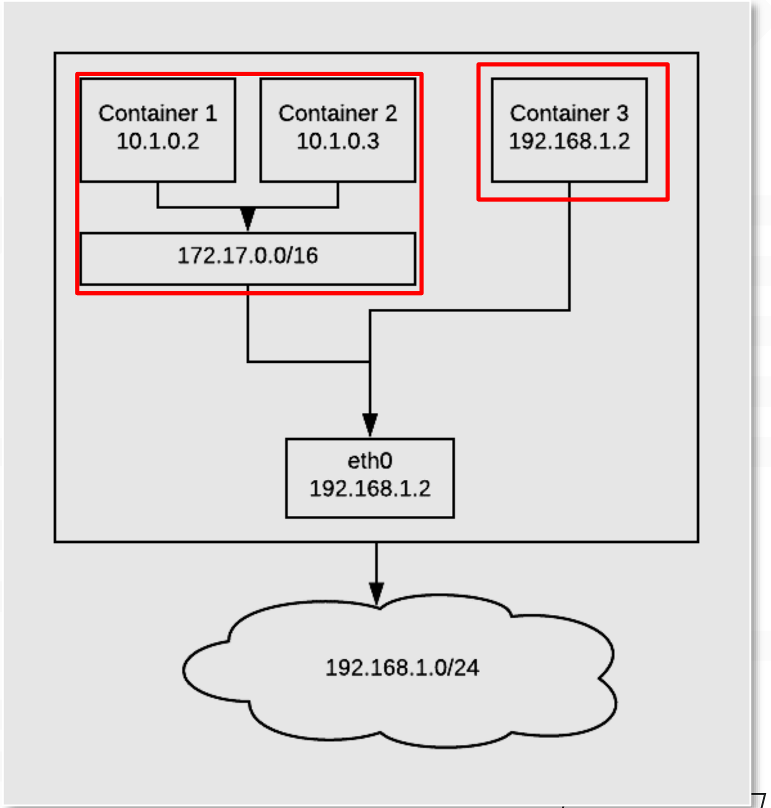
So, How Does It Work?

- Think of “containers” as Virtual Machines, each one is its own server
- Containers can be generated locally or pre-configured containers (images) can be downloaded from the “docker hub”
- Each container uses the kernel of the host system to operate but is isolated from everything else by default
- The container has a single purpose which it is assigned at startup called an “entrypoint”
- Once complete it shuts down automatically, saving state.



Networking for Docker Containers

- DEFAULT: Bridge (Docker creates a local VLAN for containers, acts as router to network)
    - Containers addressable by name
- Host (Container uses host network interface directly)
- Isolated Network (Like bridge, but isolated VLAN for specific containers)



Benefits of Docker

Better Resource Usage

- Only install the packages you need, not the bloat
- Manage resources for specific containers

Infrastructure as Code

- Deploy containers in seconds with scripted deployment
- “Treat containers like cattle, not cats”

Improved Security

- Each container has its own sandbox

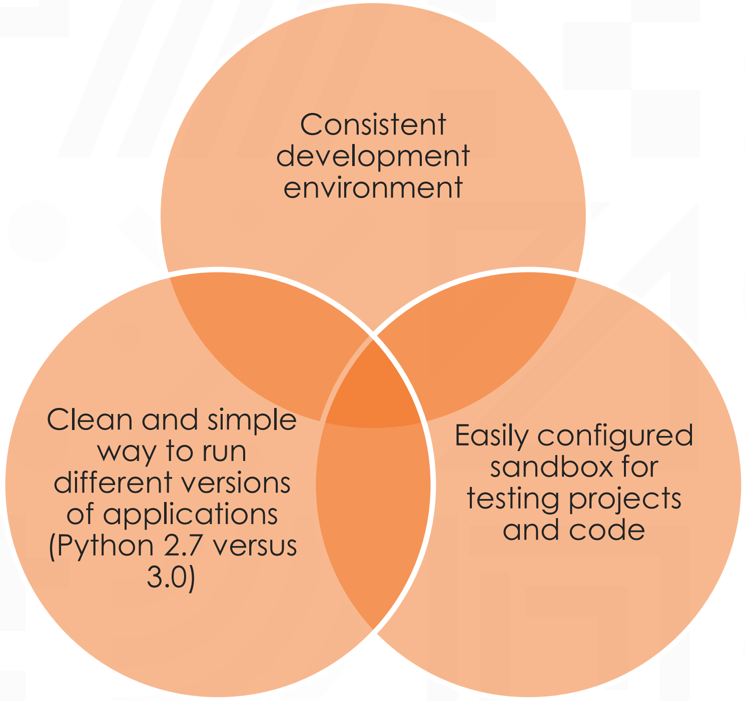
Popular Docker Images

- Database Servers
  - Oracle Database
  - Couchbase
  - Mongo
  - Mariadb
  - Redis
  - Postgres
- Database Helpers
  - Oracle instant client
- Base OS Images
  - Alpine
  - Ubuntu
  - Fedora
  - RHEL
- Web Servers
  - Nginx
  - Apache
- Docker Helpers
  - Traefik
- Development Environments
  - Java 8
  - Python
  - Busybox

Setting Up a Standalone Docker System

SPICEWORLD2019

Why?


Consistent development environment
Clean and simple way to run different versions of applications (Python 2.7 versus 3.0)
Easily configured sandbox for testing projects and code

Installing Docker

```
Sudo yum install docker [-y -q -e 0]
```

```
Sudo systemctl start docker
```

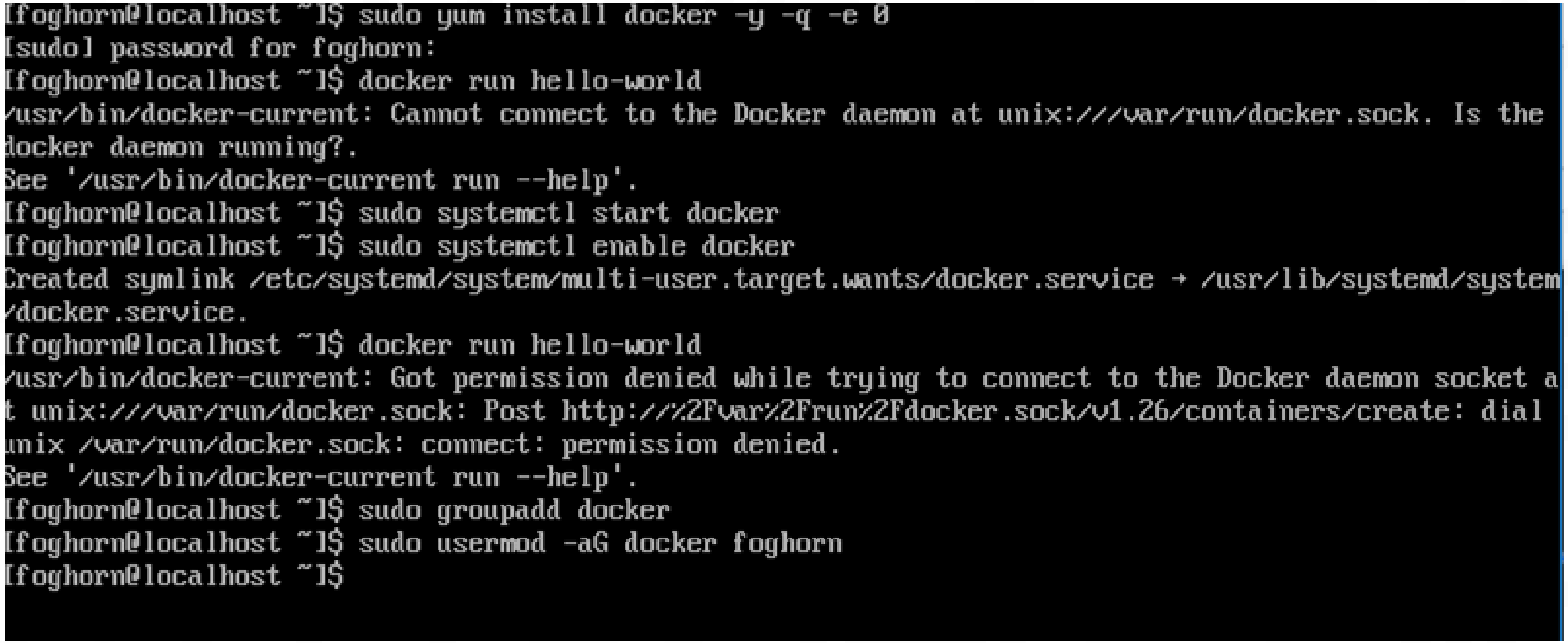
```
Sudo systemctl enable docker
```

Making Docker Accessible Without Sudo

```
Sudo groupadd docker

Sudo usermod -aG docker $user
```

[Log out and log back in]



```
[foghorn@localhost ~]$ sudo yum install docker -y -q -e 0
[sudo] password for foghorn:
[foghorn@localhost ~]$ docker run hello-world
/usr/bin/docker-current: Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the
docker daemon running?.
See '/usr/bin/docker-current run --help'.
[foghorn@localhost ~]$ sudo systemctl start docker
[foghorn@localhost ~]$ sudo systemctl enable docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system
/docker.service.
[foghorn@localhost ~]$ docker run hello-world
/usr/bin/docker-current: Got permission denied while trying to connect to the Docker daemon socket a
t unix:///var/run/docker.sock: Post http://%2Fvar%2Frun%2Fdocker.sock/v1.26/containers/create: dial
unix /var/run/docker.sock: connect: permission denied.
See '/usr/bin/docker-current run --help'.
[foghorn@localhost ~]$ sudo groupadd docker
[foghorn@localhost ~]$ sudo usermod -aG docker foghorn
[foghorn@localhost ~]$
```

Running Docker Containers

Docker Run versus Docker Compose

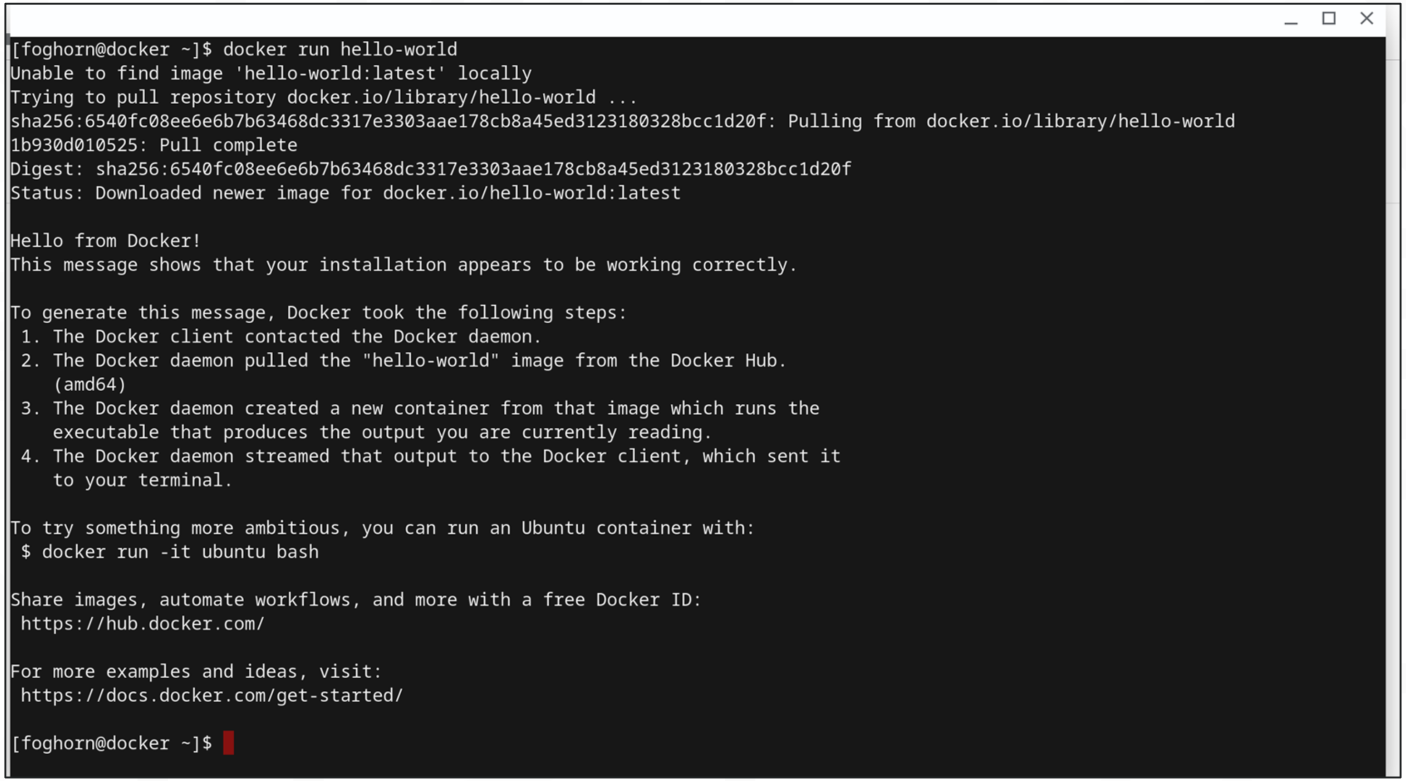
```
Docker run [image]
```

- Starts a single image / server
- Easy and good for standalone systems
- Start here!

```
Docker-compose up
```

- Orchestrate deploying multiple networks, containers and defining the links between them
- Requires a “dockerfile” called “docker-compose.yml” and installation of another tool
- We won’t be covering this in this “basic” overview - just know it exists!

Make Sure Docker is Properly Configured



```
[foghorn@docker ~]$ docker run hello-world
Unable to find image 'hello-world:latest' locally
Trying to pull repository docker.io/library/hello-world ...
sha256:6540fc08ee6e6b7b63468dc3317e3303aae178cb8a45ed3123180328bcc1d20f: Pulling from docker.io/library/hello-world
1b930d010525: Pull complete
Digest: sha256:6540fc08ee6e6b7b63468dc3317e3303aae178cb8a45ed3123180328bcc1d20f
Status: Downloaded newer image for docker.io/hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

[foghorn@docker ~]$
```

Understanding the Bells and Whistles

SPICEWORLD2019

Basic Container Management

See running containers

```
Docker ps
```

Start a stopped container

```
Docker start [container]
```

Stop a started container

```
Docker stop [container]
Docker kill [container]
```

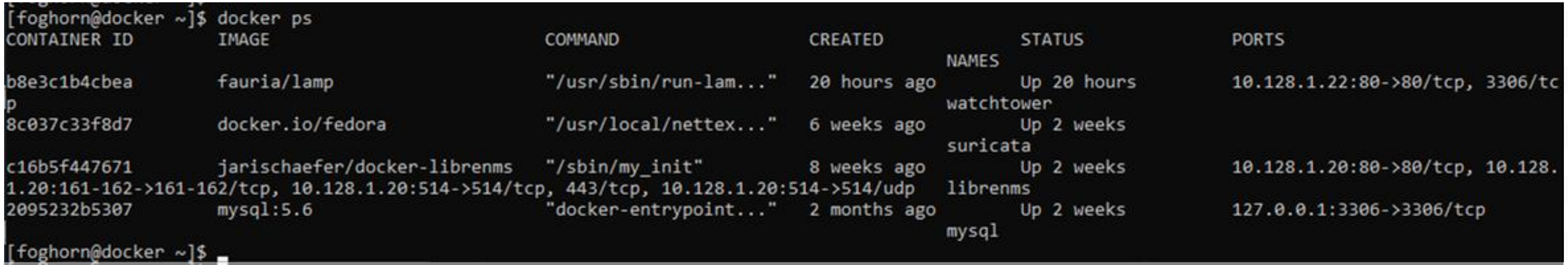
Delete a container

```
Docker rm [container]
```

Run a detached container

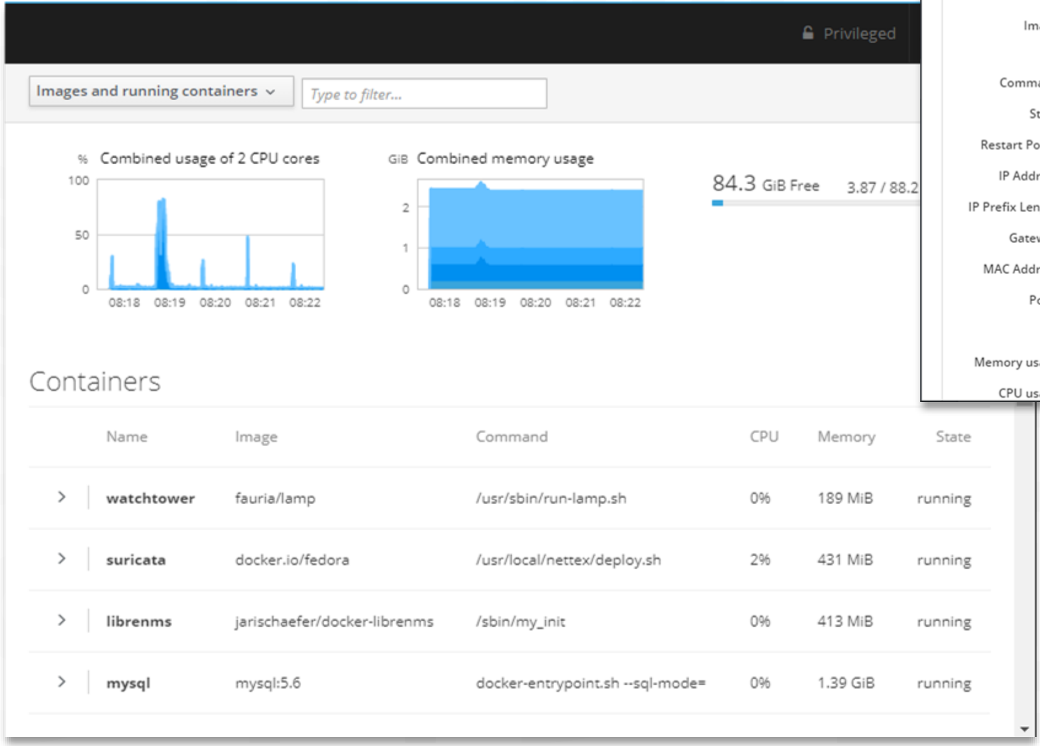
```
Docker run -d [container]
```

Docker PS



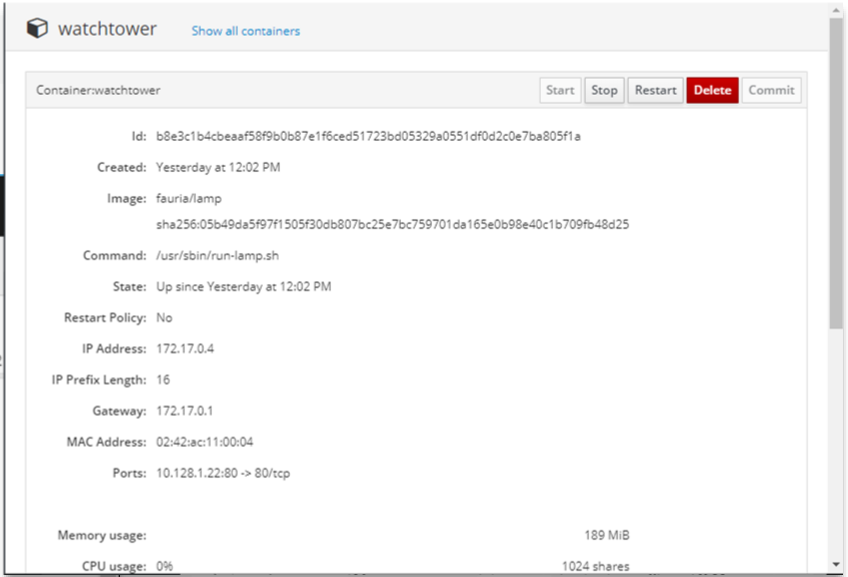
```
[foghorn@docker ~]$ docker ps
CONTAINER ID        IMAGE                         COMMAND                  CREATED             STATUS              PORTS
                                                                                    NAMES
b8e3c1b4cbea        fauria/lamp                   "/usr/sbin/run-lam..."   20 hours ago        Up 20 hours         10.128.1.22:80->80/tcp, 3306/tc
p                                                                                   watchtower
8c037c33f8d7        docker.io/fedora              "/usr/local/nettex..."   6 weeks ago         Up 2 weeks
                                                                                    suricata
c16b5f447671        jarischaefer/docker-librenms  "/sbin/my_init"          8 weeks ago         Up 2 weeks          10.128.1.20:80->80/tcp, 10.128.
1.20:161-162->161-162/tcp, 10.128.1.20:514->514/tcp, 443/tcp, 10.128.1.20:514->514/udp   librenms
2095232b5307        mysql:5.6                     "docker-entrypoint..."   2 months ago        Up 2 weeks          127.0.0.1:3306->3306/tcp
                                                                                    mysql
[foghorn@docker ~]$ ▪
```

Fedora Cockpit


Privileged
Images and running containers
Type to filter...
% Combined usage of 2 CPU cores
GiB Combined memory usage
84.3 GiB Free
3.87 / 88.2

Containers

	Name	Image	Command	CPU	Memory	State
›	**watchtower**	fauria/lamp	/usr/sbin/run-lamp.sh	0%	189 MiB	running
›	**suricata**	docker.io/fedora	/usr/local/nettex/deploy.sh	2%	431 MiB	running
›	**librenms**	jarischaefer/docker-librenms	/sbin/my_init	0%	413 MiB	running
›	**mysql**	mysql:5.6	docker-entrypoint.sh --sql-mode=	0%	1.39 GiB	running



Container:watchtower

Id:	b8e3c1b4cbeaaf58f9b0b87e1f6ced51723bd05329a0551df0d2c0e7ba805f1a
Created:	Yesterday at 12:02 PM
Image:	fauria/lamp
	sha256:05b49da5f97f1505f30db807bc25e7bc759701da165e0b98e40c1b709fb48d25
Command:	/usr/sbin/run-lamp.sh
State:	Up since Yesterday at 12:02 PM
Restart Policy:	No
IP Address:	172.17.0.4
IP Prefix Length:	16
Gateway:	172.17.0.1
MAC Address:	02:42:ac:11:00:04
Ports:	10.128.1.22:80 -> 80/tcp
Memory usage:	189 MiB
CPU usage:	0% 1024 shares

Docker Run [arguments] [image name]

Detached

By default, Docker will attach your terminal to the terminal running within the container you just started.

To enable a container to continue to run "in the background" you will need to "detach" the container from your terminal.

```
-d
```

Names

Naming docker containers make it easier to manage and address.

Docker will randomly assign a two word name to all unnamed containers.

```
--name testenv
```

Environment Variables

Most pre-configured docker containers will accept environment variables

Environment variables tell the container things like where to connect for database services, DNS names to use, or other configurable variables

```
-env dns=docker.nickleghorn.com
```

Publishing Ports

If you need your container to be available on the network as a service you can "publish" (think "map") a port from the docker network to the host network

REMEMBER: [host port]:[container port]

```
-p 8080:80
```

You can also specify an IP address to bind the port on

```
-p 10.128.1.224:8080:80
```

Default is to expose TCP. You can specify UDP as well.

```
-p 10.128.1.224:514:514/udp
```

Mounting Directories

Want to have a folder on your host OS available to a container?

REMEMBER: [host directory]:[container directory]

```
-v /home/foghorn/website:/var/www/html
```

Manual Container Operations

Copy a file FROM a container

```
Docker cp [container]:[/path/to/container/file] [/path/to/host/file]
```

Copy a file TO a container

```
Docker cp [/path/to/host/file] [container]:[/path/to/container/file]
```

Run a command in an existing container

```
Docker exec [container] [command]
```

Get an Interactive Shell in a Container

Is the container already running?

```
Docker exec -it [container] /bin/bash
```

Do you need to start the container?

```
Docker run -it [container] /bin/bash
```

Restart conditions

When should the container be restarted?

Always!

```
--restart=always
```

If the container fails, maximum twice

```
--restart=on-failure:2
```

Setting an Entrypoint

Most pre-built containers already have an entrypoint.

```
Docker run -d \
-v /test/:/usr/local/test/ \
--entrypoint /usr/local/test/start.sh \
fakecontainer
```

Use Case Examples

SPICEWORLD2019

Standalone PiHole Container / DNS Server

```
docker run -d \
    --name pihole \
    -p 53:53/tcp -p 53:53/udp \
    -p 67:67/udp \
    -p 80:80 \
    -p 443:443 \
    -e ServerIP="[INSERT IP HERE]" \
    -e WEBPASSWORD="[SET A PASSWORD]" \
    --restart=always \
    --cap-add=NET_ADMIN \
    --dns=127.0.0.1 --dns=1.1.1.1 \
    pihole/pihole:latest
```

HTTP Web Server with Mounted Directory

```
docker run -d \
  --name webserver \
  -p 127.0.0.1:8080:80\
  -v /home/foghorn/website/:/usr/local/apache2/htdocs/ \
  httpd:2.4

sudo chcon -Rt svirt_sandbox_file_t /home/foghorn/website
```

LibreNMS Deployment on Linked Containers

For extra homework and hands-on testing:

https://github.com/foghorn/librenmsdocker

Docker Swarm and Kubernetes

SPICEWORLD2019

#spiceworldATX

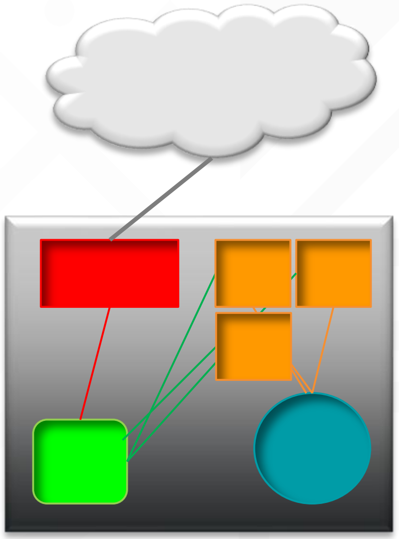
Docker is Just The Beginning

Docker just manages the containers on a single host

Docker swarm pools multiple servers to form shared resources and manages the running of containers within that “swarm”

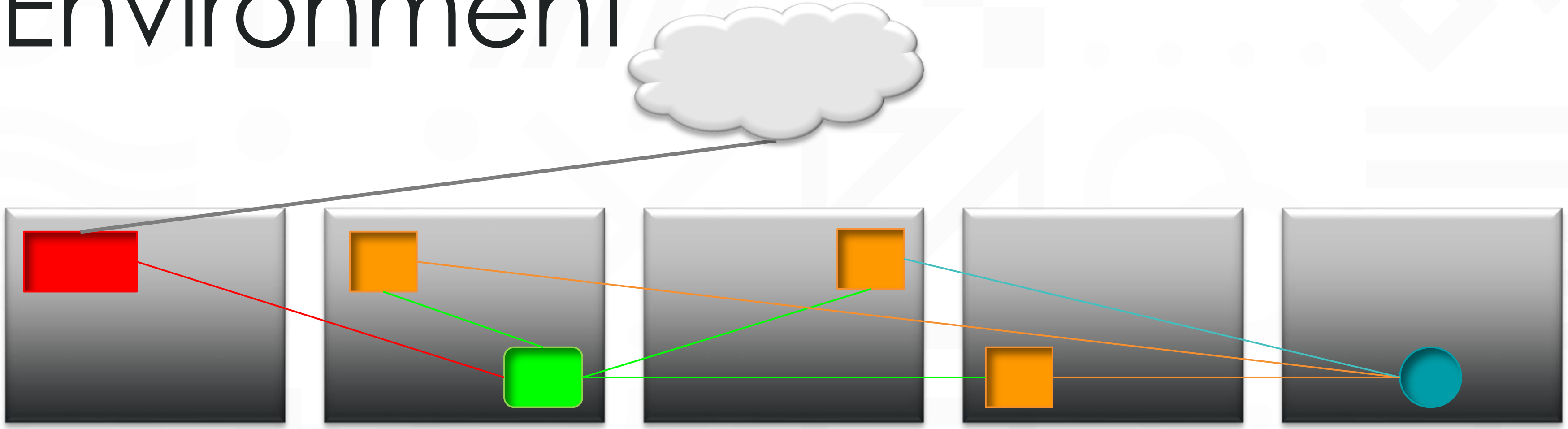
Kubernetes is like docker swarm but with more control over networking, load balancing, and other higher-level functions

Standalone Docker Environment



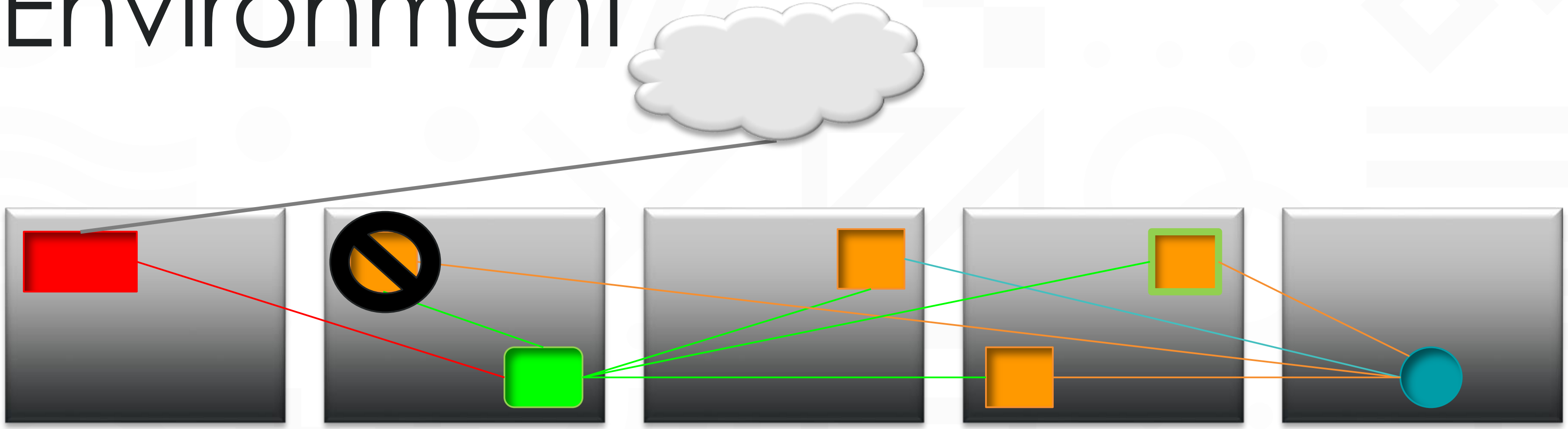
Firewall
Load Balancer
Web Hosts
Database

Kubernetes Environment



Firewall
Load Balancer
Web Hosts
Database

Kubernetes Environment



Firewall
Load Balancer
Web Hosts
Database

SPICEWORLD2019

#spiceworldATX

Review

Hopefully, you are leaving here today able to:

- Describe docker containers, their purpose, and how they operate
- Install docker on your local system
- Deploy and configure a docker container
- Understand the concept of Kubernetes and Docker Swarm

Instagram / Twitter

foghorn@NickLeghorn.com

GitHub

Website

Email

SPICEWORLD2019