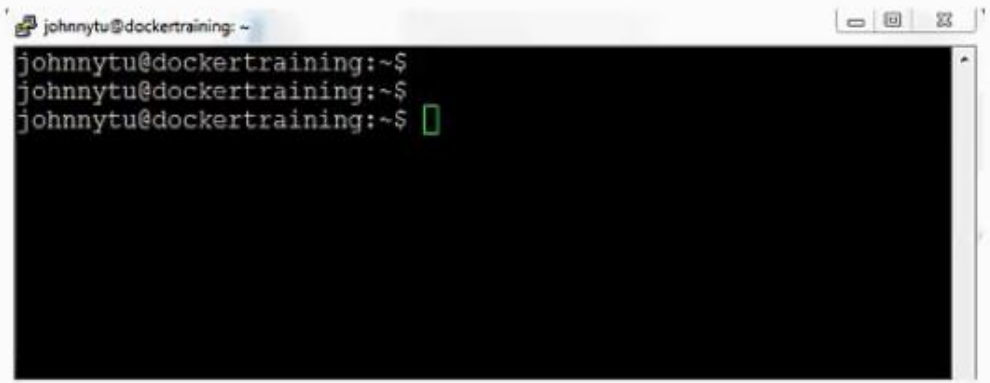
Docker Operations

Raghu
raghuopsdev@gmail.com

Session Logistics

- 1 hour including exercise time
- You will need
  - Linux machine (preferably Ubuntu)
  - Use your own or use a cloud provider
    - Amazon EC2
    - Digital Ocean
- Be familiar with Linux command line
- Have taken the Introduction to Docker and Docker Fundamentals course



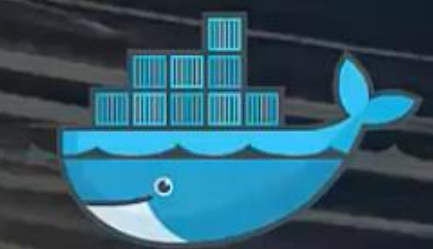
What we have Learned

- Intro to Docker
  - Benefits of Container based virtualization
  - Docker concepts and terms
  - Simple container examples
- Docker Fundamentals
  - Building images
  - Managing containers
  - Push images to Docker Hub
  - Basic container networking
  - Continuous integration with Docker

Agenda

- Troubleshooting Containers
- Overview of Security Practices
- Private Registry
- Intro to Docker Machine
- Intro to Docker Swarm
- Intro to Docker Compose
- Building micro service applications with Docker

Container Troubleshooting



docker

Container logging

- Container PID 1 process output can be viewed with **`docker logs`** command
- Will show whatever PID 1 writes to stdout

View the output of the containers PID 1 process

```
docker logs <container name>
```

View and follow the output

```
docker logs -f <container name>
```

```
johnnytu@docker-demo:~$ docker run -d tomcat
3e9e3fbc8e667371145cc9c44ac7f877d989ba782d72054889edf45ae6d3db78
johnnytu@docker-demo:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                CREATED             STATUS              PORTS               NAMES
3e9e3fbc8e66        tomcat:latest       "catalina.sh run"      4 seconds ago       Up 3 seconds        8080/tcp            compassionate_bell
a7700f3e7091        postgres:latest     "/docker-entrypoint.   3 days ago          Up 3 days           5432/tcp            dbms
johnnytu@docker-demo:~$ docker logs compassionate_bell
21-Apr-2015 07:40:23.873 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server version:        Apache Tomcat/8.0.21
21-Apr-2015 07:40:23.877 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server built:          Mar 23 2015 14:11:21 UTC
21-Apr-2015 07:40:23.878 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server number:        8.0.21.0
21-Apr-2015 07:40:23.879 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Name:              Linux
21-Apr-2015 07:40:23.890 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Version:           3.13.0-43-generic
```

```
johnnytu@docker-demo:~$ docker logs -f --tail 1 compassionate_bell
21-Apr-2015 07:41:40.303 INFO [main] org.apache.catalina.startup.Catalina.start Server startup in 75859 ms
```

Container application logs

- Typically, apps have a well defined log location
- Map a host folder to the application's application log folder in the container
- In this way, you can view the log generated in the container from your host folder

Run a container using nginx image and mount a volume to map the /nginxlogs folder in the host to the /var/log/nginx folder in the container

```
docker run -d -P -v /nginxlogs:/var/log/nginx nginx
```

```
johnnytu@docker-demo:~$ docker run -d -P -v /container/logs/nginx:/var/log/nginx nginx
6bb35370d691d954504f8052a372aa8991e43a24b04f047edd135c04a04795bd
johnnytu@docker-demo:~$
```

```
johnnytu@docker-demo:~$ docker run -d -P -v /container/logs/nginx:/var/log/nginx nginx
6bb35370d691d954504f8052a372aa8991e43a24b04f047edd135c04a04795bd
johnnytu@docker-demo:~$ cd /container/logs/nginx/
johnnytu@docker-demo:/container/logs/nginx$ ls
access.log  error.log
johnnytu@docker-demo:/container/logs/nginx$
johnnytu@docker-demo:/container/logs/nginx$ tail -f access.log
^C
johnnytu@docker-demo:/container/logs/nginx$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
     NAMES
6bb35370d691        nginx:1.7           "nginx -g 'daemon of   24 seconds ago      Up 24 seconds       0.0.0.0:49161->443/tcp, 0.0.0.0:49162->80/t
cp   drunk_wozniak
a7700f3e7091        postgres:latest     "/docker-entrypoint.   3 days ago          Up 3 days           5432/tcp
     dbms
johnnytu@docker-demo:/container/logs/nginx$ tail -f access.log
101.176.203.123 - - [21/Apr/2015:11:38:41 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like
 Gecko) Chrome/41.0.2272.118 Safari/537.36" "-"
101.176.203.123 - - [21/Apr/2015:11:38:41 +0000] "GET /favicon.ico HTTP/1.1" 404 571 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (
KHTML, like Gecko) Chrome/41.0.2272.118 Safari/537.36" "-"
101.176.203.123 - - [21/Apr/2015:11:38:48 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like G
ecko) Chrome/41.0.2272.118 Safari/537.36" "-"
```

Check container logs

EXERCISE

1. Run a new container using the tomcat image
   `docker run -d -P tomcat`
2. View the container log
   `docker logs <container id>`
3. On your host machine, create a folder `/container/logs/nginx`
4. Run a new container using the NGINX image and mount the `/container/logs/nginx` folder into `/var/log/nginx`
   ```
   docker run -d -P -v
   /container/logs/nginx:/var/log/nginx nginx
   ```
5. Look inside your `/container/logs/nginx` folder and notice the new log files from the container

Inspecting a container

- **`docker inspect`** command displays all the details about a container
- Outputs details in JSON array
- Use grep to find a specific property

Display all details of the specified container

```
docker inspect <container name>
```

Display the IP address of the specified container

```
docker inspect <container name> | grep IPAddress
```

```
johnnytu@docker-demo:~$
johnnytu@docker-demo:~$ docker inspect --format {{.NetworkSettings.IPAddress}} drunk_wozniak
172.17.0.81
```

Starting and stopping Docker daemon

- If you started Docker as a service, use `service` command to stop, start and restart the Docker daemon
  - `sudo service docker stop`
  - `sudo service docker start`
  - `sudo service docker restart`
- If not running as a service, run Docker executable in daemon mode to start the daemon
  `sudo docker -d &`
- If not running as a service, send a SIGTERM to the Docker process to stop it
  - Run `pidof docker` to find the Docker process PID
  - `sudo kill $(pidof docker)`

Docker daemon upstart configuration file

- Located in `/etc/default/docker`
- Use `DOCKER_OPTS` to control the startup options for the daemon when running as a service
- Restart the service for changes to take effect

```
sudo service docker restart
```

Start daemon with log level of debug and allow connections to an insecure registry at the domain of myserver.org

```
DOCKER_OPTS="--log-level debug --insecure-registry
myserver.org:5000"
```

Docker daemon logging

- Start the docker daemon with `--log-level` parameter and specify the logging level
- Levels are (in order from most verbose to least):
  - Debug
  - Info
  - Warn
  - Error
  - Fatal

Run docker daemon with debug log level (log written on terminal)

```
sudo docker -d --log-level=debug
```

Configuring in DOCKER_OPS (log output will be written to /var/log/upstart/docker.log)

```
DOCKER_OPTS="--log-level debug"
```

Setup daemon logging

EXERCISE

1. Stop the Docker service or process that is currently running
   ```
   sudo service docker stop
   ```
   OR
   ```
   sudo kill $(pidof docker)
   ```
2. Start Docker in daemon mode and specify the debug logging level
   ```
   sudo docker -d --log-level=debug
   ```
3. Run a few Docker commands and observe the log output

Security

docker

Linux containers and security

- Docker helps make applications safer as it provides a reduced set of default privileges and capabilities
- Namespaces provide an isolated view of the system. Each container has its own
  - IPC, network stack, root file system etc…
- Processes running in one container cannot see and effect processes in another container
- Control groups (Cgroups) isolate resource usage per container
  - Ensures that a compromised container won't bring down the entire host by exhausting resources

Quick security considerations

- Docker daemon needs to run as root
- Only ensure that trusted users can control the Docker daemon
  - Watch who you add to docker group
- If binding the daemon to a TCP socket, secure it with TLS
- Use Linux hardening solution
  - Apparmor
  - SELinux
  - GRSEC

Further reading

- http://docs.docker.com/articles/security/
- http://www.slideshare.net/jpetazzo/docker-linux-containers-lxc-and-security

- Docker security is additive and simple and should be used alongside best practices for Securing Linux distributions

Private Registry

docker

Private Registry

- Allows you to run your own registry instead of using Docker Hub
- Multiple options
  - Run registry server using container
  - Docker Hub Enterprise
- **Two versions**
  - Registry v1.0 for Docker 1.5 and below
  - Registry v2.0 for Docker 1.6

Setting up a private registry

- Run the registry server inside a container
- Use the **registry** image at https://registry.hub.docker.com/u/library/registry/
- Image contains a pre-configured version of registry v2.0

Run a new container using the registry image

```
docker run -d -p 5000:5000 registry:2.0
```

```
johnnytu@docker-demo2:~$ docker run -d -p 5000:5000 registry:2.0
Unable to find image 'registry:2.0' locally
2.0: Pulling from registry
```

```
johnnytu@docker-demo2:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
a883b977dc88        registry:2.0        "registry cmd/regist     5 seconds ago       Up 5 seconds        0.0.0.0:5000->5000/tcp   modest_wright
```

Push and pull from private registry

- First tag the image with host IP or domain of the registry server, then run `docker push`

Tag image and specify the registry host

```
docker tag <image id>  myserver.net:5000/my-app:1.0
```

Push image to registry

```
docker push myserver.net:5000/my-app:1.0
```

Pull image from registry

```
docker pull myserver.net:5000/my-app:1.0
```

```
johnnytu@docker-demo2:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
a883b977dc88        registry:2.0        "registry cmd/regist   About an hour ago   Up About an hour    0.0.0.0:5000->5000/tcp   modest_wright

johnnytu@docker-demo2:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
hello-world         latest              91c95931e552        4 days ago          910 B
registry            2.0                 ebbd55f108b8        5 days ago          545.7 MB
johnnytu@docker-demo2:~$
johnnytu@docker-demo2:~$ docker tag 91c95931e552 localhost:5000/myhello-world:1.0
johnnytu@docker-demo2:~$ docker images
REPOSITORY                      TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
hello-world                     latest              91c95931e552        4 days ago          910 B
localhost:5000/myhello-world    1.0                 91c95931e552        4 days ago          910 B
registry                        2.0                 ebbd55f108b8        5 days ago          545.7 MB
johnnytu@docker-demo2:~$
johnnytu@docker-demo2:~$ docker push localhost:5000/myhello-world:1.0
The push refers to a repository [localhost:5000/myhello-world] (len: 1)
91c95931e552: Image already exists
a8219747be10: Image successfully pushed
Digest: sha256:e2c17c8d4c888c3dbe59c608f4c3339bcb81b2ea802403906c3120355d10a17e
johnnytu@docker-demo2:~$
```

```
johnnytu@docker-demo3:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
johnnytu@docker-demo3:~$
johnnytu@docker-demo3:~$ docker pull 104.131.142.17:5000/myhello-world:1.0
FATA[0000] Error response from daemon: v1 ping attempt failed with error: Get https://104.131.142.17:5000/v1/_ping: tls: oversized reco
rd received with length 20527. If this private registry supports only HTTP or HTTPS with an unknown CA certificate, please add `--insec
ure-registry 104.131.142.17:5000` to the daemon's arguments. In the case of HTTPS, if you have access to the registry's CA certificate,
 no need for the flag; simply place the CA certificate at /etc/docker/certs.d/104.131.142.17:5000/ca.crt
johnnytu@docker-demo3:~$
johnnytu@docker-demo3:~$
```

```
johnnytu@docker-demo3:~$ sudo service docker stop
[sudo] password for johnnytu:
docker stop/waiting
johnnytu@docker-demo3:~$
johnnytu@docker-demo3:~$ sudo vim /etc/default/docker
```

```
# Docker Upstart and SysVinit configuration file

# Customize location of Docker binary (especially for development testing).
#DOCKER="/usr/local/bin/docker"

# Use DOCKER_OPTS to modify the daemon startup options.
DOCKER_OPTS="--insecure-registry 104.131.142.17:5000"

# If you need Docker to use an HTTP proxy, it can also be specified here.
#export http_proxy="http://127.0.0.1:3128/"

# This is also a handy place to tweak where Docker's temporary files go.
#export TMPDIR="/mnt/bigdrive/docker-tmp"
~
```

Restart the Docker Service

```
johnnytu@docker-demo3:~$ docker pull 104.131.142.17:5000/myhello-world:1.0
1.0: Pulling from 104.131.142.17:5000/myhello-world

a8219747be10: Pull complete
91c95931e552: Already exists
Digest: sha256:e2c17c8d4c888c3dbe59c608f4c3339bcb81b2ea802403906c3120355d10a17e
Status: Downloaded newer image for 104.131.142.17:5000/myhello-world:1.0
johnnytu@docker-demo3:~$
```

```
johnnytu@docker-demo3:~$ docker tag nginx 104.131.142.17:5000/mynginx:1.0
johnnytu@docker-demo3:~$ docker push 104.131.142.17:5000/mynginx:1.0
The push refers to a repository [104.131.142.17:5000/mynginx] (len: 1)
637d3b2f5fb5: Image already exists
073bafc48d63: Image successfully pushed
99d6f607f867: Image successfully pushed
```

```
johnnytu@docker-demo2:~$ docker pull localhost:5000/mynginx:1.0
1.0: Pulling from localhost:5000/mynginx
```

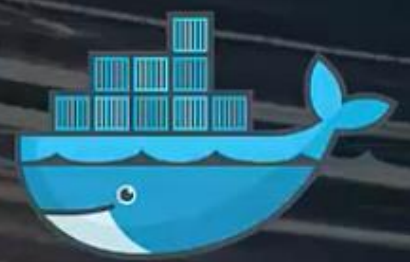
To Check the image in the registry

```
johnnytu@docker-demo2:~$ curl -v -X GET http://localhost:5000/v2/mynginx/tags/list
* Hostname was NOT found in DNS cache
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 5000 (#0)
> GET /v2/mynginx/tags/list HTTP/1.1
> User-Agent: curl/7.35.0
> Host: localhost:5000
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: application/json; charset=utf-8
< Docker-Distribution-Api-Version: registry/2.0
< Date: Wed, 22 Apr 2015 06:16:47 GMT
< Content-Length: 34
<
{"name":"mynginx","tags":["1.0"]}
```

Further resources

- Registry v1.0
  - https://github.com/docker/docker-registry
- Registry v2.0
  - https://github.com/docker/distribution
  - https://docs.docker.com/registry/overview/

Intro to Docker Machine



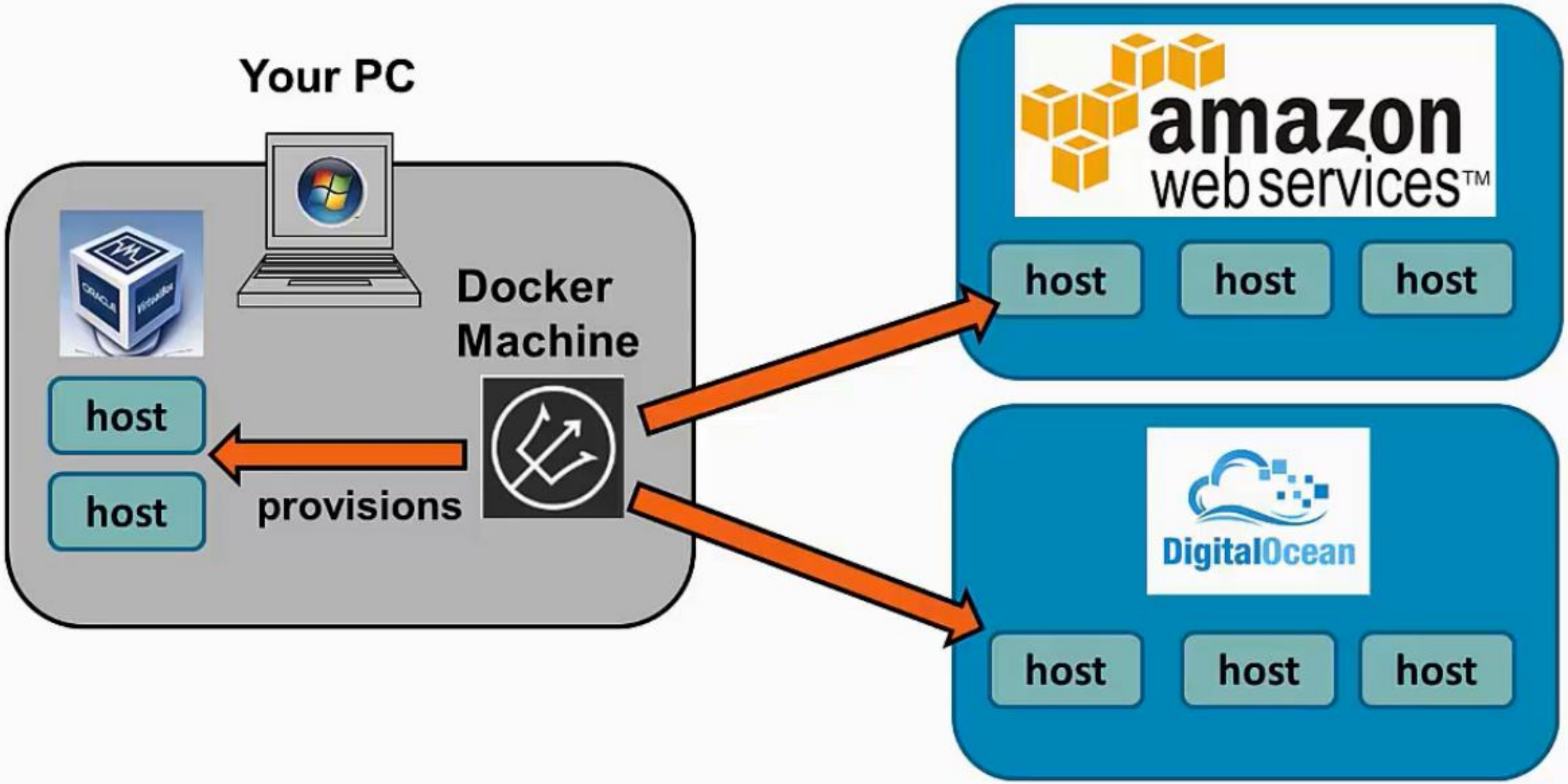
docker

Docker Machine overview

Docker Machine *is a tool that automatically provisions Docker hosts and installs the Docker Engine on them*

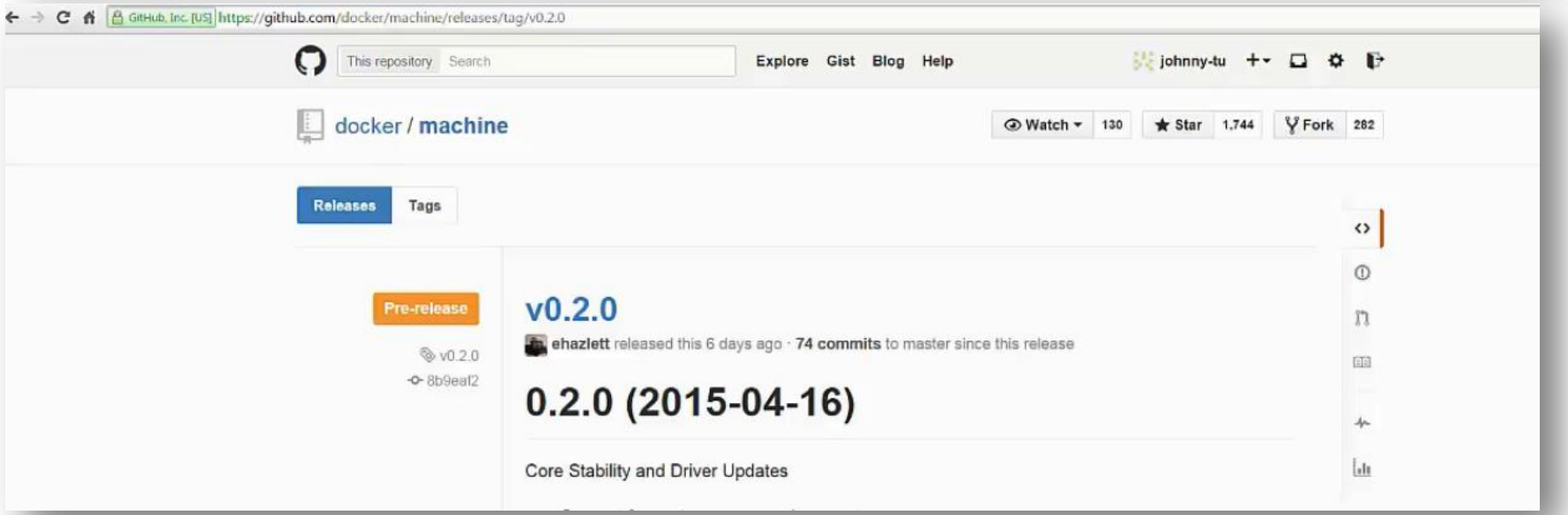
- Create additional hosts on your own computer
- Create hosts on cloud providers (e.g. Amazon AWS, DigitalOcean etc…)
- Machine creates the server, installs Docker and configures the Docker client

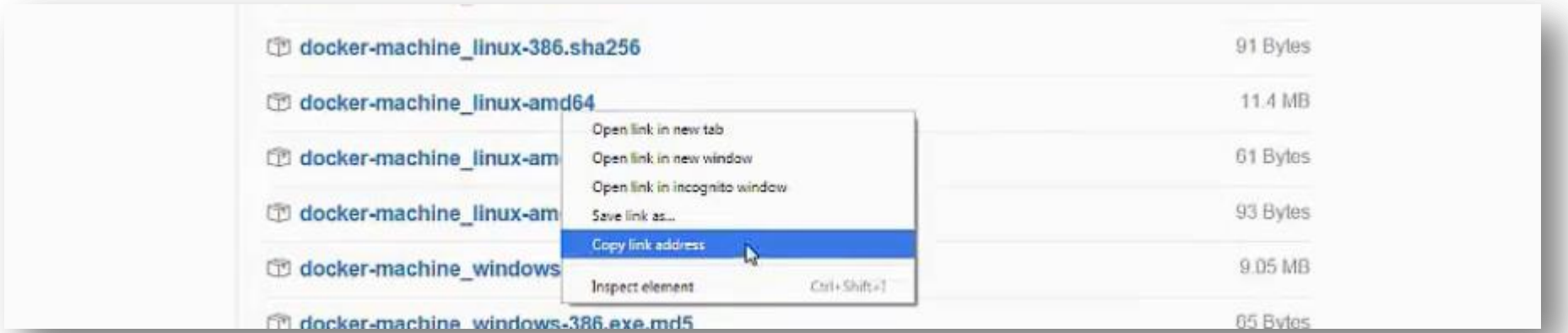
Docker Machine overview



Installing Machine

- Download the binary for the operating system at https://github.com/docker/machine/releases
- Place the binary into a folder of your choice
- Add the folder to your system environment PATH






```
johnnytu@docker-demo2:~$ wget https://github.com/docker/machine/releases/download/v0.2.0/docker-machine_linux-amd64
--2015-04-22 06:40:04--  https://github.com/docker/machine/releases/download/v0.2.0/docker-machine_linux-amd64
Resolving github.com (github.com)... 192.30.252.131
Connecting to github.com (github.com)|192.30.252.131|:443... connected.
HTTP request sent, awaiting response... 302 Found
```

```
johnnytu@docker-demo2:~$ mv docker-machine_linux-amd64 docker-machine
johnnytu@docker-demo2:~$ ls
docker-machine
johnnytu@docker-demo2:~$ mv docker-machine /usr/local/bin/
mv: cannot move 'docker-machine' to '/usr/local/bin/docker-machine': Permission denied
johnnytu@docker-demo2:~$ sudo !!
sudo mv docker-machine /usr/local/bin/
[sudo] password for johnnytu:
johnnytu@docker-demo2:~$ cd /usr/local/bin
johnnytu@docker-demo2:/usr/local/bin$ ls
docker-machine
johnnytu@docker-demo2:/usr/local/bin$
```

```
johnnytu@docker-demo2:/usr/local/bin$ sudo chmod 775 docker-machine
johnnytu@docker-demo2:/usr/local/bin$ ls -l
total 11684
-rwxrwxr-x 1 johnnytu johnnytu 11961936 Apr 16 00:30 docker-machine
```

$ docker-machine –v
<It displays the Version>

Creating a host

- Use **`docker-machine create`** command and specify the driver to use
- Use virtualbox driver if creating hosts on a Windows or Mac
  - Need to have Virtual Box installed (https://www.virtualbox.org/)

Create a host named "testhost" on the current machine, using Virtual Box.

```
docker-machine create --driver virtualbox testhost
```

Provisioning hosts in the cloud

- Each cloud provider has different options on the **docker-machine create** command
- See https://docs.docker.com/machine/#drivers as reference

Example with DigitalOcean

```
docker-machine create
      --driver digitalocean \
      --digitalocean-access-token <your access token> \
      --digitalocean-size 2gb \
      testhost
```

Amazon Web Services (AWS) EC2

https://docs.docker.com/machine/examples/aws/#step-1-sign-up-for-aws-and-configure-credentials

List your machines

- **`docker-machine ls`** command displays all the host machines that have been provisioned
- Can easily see hosts across different cloud providers

```
johnnytu@dockerexamples:~$ docker-machine ls
NAME        ACTIVE   DRIVER          STATE     URL                         SWARM
host3       *        digitalocean    Running   tcp://45.55.183.123:2376
testhost2            digitalocean    Running   tcp://45.55.146.79:2376
johnnytu@dockerexamples:~$
```

Docker machine SSH

- Allows us to connect to a provisioned host using SSH
- Logs in using the SSH key that is created when creating the machine

Connect to host3 using SSH

```
docker-machine ssh host3
```

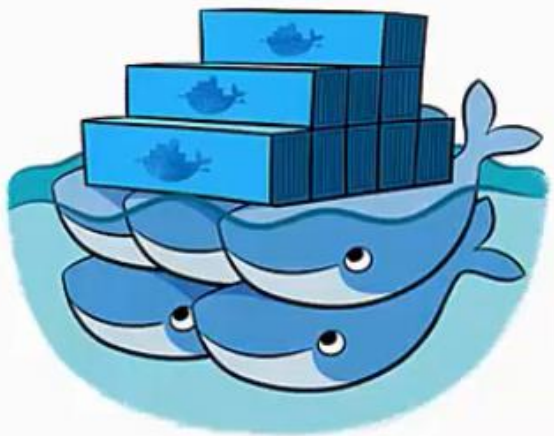
Intro to Docker Swarm

docker

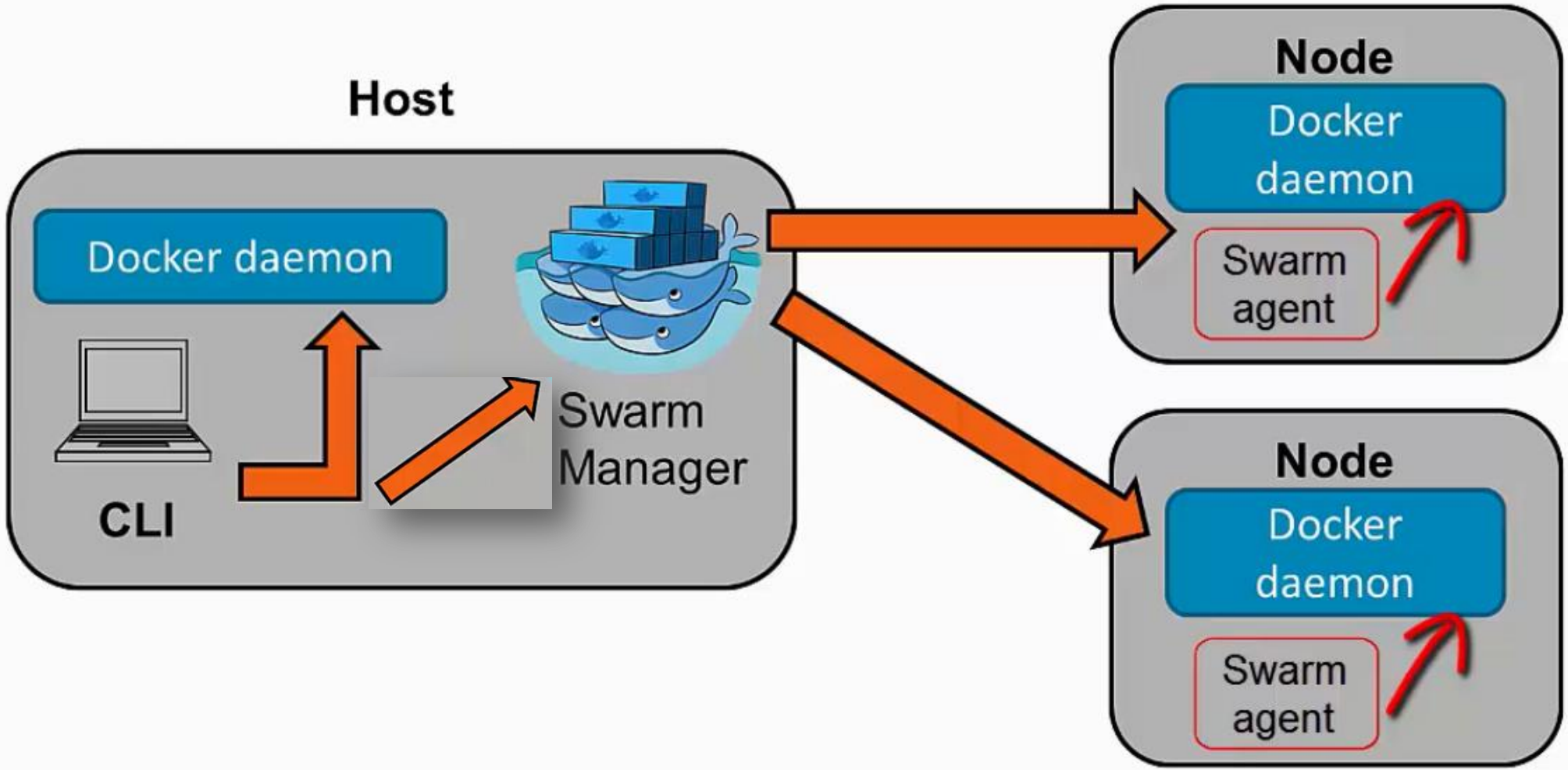
What is Docker Swarm?

Docker Swarm *is a tool that clusters Docker hosts and schedules containers*

- Turns a pool of host machines into a single virtual host
- Ships with simple scheduling backend
- Supports many discovery backends
  - Hosted discovery
  - etcd
  - Consul
  - ZooKeeper
  - Static files



How Swarm works



Setup process (using hosted discovery)

- On the machine that you will use as the Swarm master, run a command to create the cluster
- Start Swarm master
- For each node with Docker installed, run a command to start the Swarm agent
- **Note:** Agents can be started before or after the master

Installing and running Swarm

- Most convenient option is to use the Swarm image on Docker Hub https://registry.hub.docker.com/u/library/swarm/
- Swarm container is a convenient packaging mechanism for the Swarm binary
- Swarm containers can be run from the image to do the following
  - Create a cluster
  - Start the Swarm manager
  - Join nodes to the cluster
  - List nodes on a cluster

Create the Swarm cluster

- **`swarm create`** command will output the cluster token
- Token is an alphanumeric sequence of characters that identifies the cluster when using the hosted discovery protocol
- Copy this number somewhere

Run a container using the `swarm` image. We run the create command of the Swarm application inside and get the output on our terminal

`--rm` means to remove the container once it has finished running

```
docker run --rm swarm create
```

```
johnnytu@swarm-manager:~$ docker run --rm swarm create
Unable to find image 'swarm:latest' locally
latest: Pulling from swarm

511136ea3c5a: Pull complete
dc2cace3cb9c: Pull complete
132dac22a0c2: Pull complete
c578e06f7812: Pull complete
1dfbc304fc7f: Pull complete
9b5a856b703d: Pull complete
282cd8d4f06e: Pull complete
96b8c18d1208: Already exists
swarm:latest: The image you are pulling has been verified. Important: image verification is a tech preview feature and should not be relied on to
provide security.

Digest: sha256:fcb626ab012325ac29f90cafc31c4de8d3d9ef6ebd4b711af007059dd5f8cc61
Status: Downloaded newer image for swarm:latest
342f4713ffed258e5efdff4b044e8b2a
johnnytu@swarm-manager:~$ vim clustertoken
johnnytu@swarm-manager:~$
```

Start the Swarm manager

- Run a container that runs the `swarm manager`
- Make sure to map the `swarm` port in the container to a port on the host

```
docker run -d -P swarm manage token://<cluster token>
```

```
johnnytu@swarm-manager:~$ cat clustertoken
342f4713ffed258e5efdff4b044e8b2a

johnnytu@swarm-manager:~$ docker run -d -P swarm manage token://342f4713ffed258e5efdff4b044e8b2a
5bf560d5560f6f65e140cb67ff456576241c8b14485711117a9786ebb4f3d89a
johnnytu@swarm-manager:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                     NAMES
5bf560d5560f        swarm:latest        "/swarm manage token   4 seconds ago       Up 3 seconds        0.0.0.0:32768->2375/tcp   loving_lumiere

johnnytu@swarm-manager:~$
```

```
johnnytu@swarm-node1:~$
johnnytu@swarm-node1:~$
johnnytu@swarm-node1:~$
johnnytu@swarm-node1:~$
johnnytu@swarm-node1:~$ ps -ef |grep docker
root      2232     1  0 09:35 ?        00:00:00 /usr/bin/docker -d
johnnytu  2285  1357  0 09:36 pts/0    00:00:00 grep --color=auto docker
johnnytu@swarm-node1:~$
johnnytu@swarm-node1:~$ sudo service docker stop
docker stop/waiting
johnnytu@swarm-node1:~$ sudo vim /etc/default/docker
johnnytu@swarm-node1:~$
```

Node 1
198.199.95.245

```
DOCKER_OPTS=" H 0.0.0.0:2375"
```

Node 1
198.199.95.245

```
johnnytu@swarm-node1:~$ sudo service docker start
docker start/running, process 2354
johnnytu@swarm-node1:~$ docker ps
FATA[0000] Get http:///var/run/docker.sock/v1.18/containers/json: dial unix /var/run/docker.sock: no such file or directory. Are you trying to con
nect to a TLS-enabled daemon without TLS?
johnnytu@swarm-node1:~$
johnnytu@swarm-node1:~$ DOCKER_HOST=localhost:2375
johnnytu@swarm-node1:~$ export DOCKER_HOST
johnnytu@swarm-node1:~$ echo $DOCKER_HOST
localhost:2375
johnnytu@swarm-node1:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
johnnytu@swarm-node1:~$
johnnytu@swarm-node1:~$
```

```
johnnytu@swarm-node1:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
johnnytu@swarm-node1:~$
johnnytu@swarm-node1:~$ docker run -d swarm join --addr=198.199.95.245:2375 token://342f4713ffed258e5efdff4b044e8b2a
41d8f1fa04a172d76380227b22f8e8cc9430f8c30c7d71f394757efcec980c59
johnnytu@swarm-node1:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                CREATED             STATUS              PORTS               NAMES
41d8f1fa04a1        swarm:latest        "/swarm join --addr=   4 seconds ago       Up 3 seconds        2375/tcp            compassionate_mcclintoc
k
johnnytu@swarm-node1:~$
```

```
johnny@swarm-node2:~$ docker run -d swarm join --addr=162.243.134.165:2375 token://342f4713ffed258e5efdff4b044e8b2a
42d77992a8c7da7a6c8d42c9b7afbd113669e9917c3e8e5fc73f4b3280111d1b
johnny@swarm-node2:~$
```

Node 2
162.243.134.165

Connect the Docker client to Swarm

- Point your Docker client to the Swarm manager container
- Two methods:
  - Configuring the `DOCKER_HOST` variable with the Swarm IP and port
  - Run docker with `-H` and specify the Swarm IP and port
- Look at the container port mapping to find the Swarm port

Configure the DOCKER_HOST variable

```
export DOCKER_HOST=127.0.0.1:<swarm port>
```

Run docker client and specify the daemon to connect to

```
docker -H tcp://127.0.0.1:<swarm port>
```

```
johnnytu@swarm-manager:~$ cat clustertoken
342f4713ffed258e5efdff4b044e8b2a

johnnytu@swarm-manager:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                     NAMES
5bf560d5560f        swarm:latest        "/swarm manage token   2 hours ago         Up 2 hours          0.0.0.0:32768->2375/tcp   loving_lumiere

johnnytu@swarm-manager:~$ export DOCKER_HOST=127.0.0.1:32768
johnnytu@swarm-manager:~$ echo $DOCKER_HOST
127.0.0.1:32768
johnnytu@swarm-manager:~$
```

Verify the Docker client

- To ensure your Docker client is connected to Swarm, run **`docker version`**
- Server should say Swarm

```
johnnytu@dockerexamples:~$ docker version
Client version: 1.5.0
Client API version: 1.17
Go version (client): go1.4.1
Git commit (client): a8a31ef
OS/Arch (client): linux/amd64
Server version: swarm/0.1.0
Server API version: 1.16
Go version (server): go1.3.3
Git commit (server): 2acbea1
```

Checking your connected nodes

- Run **`docker info`**
- Since client is connected to Swarm, it will show the nodes

```
johnnytu@dockerexamples:~$ docker info
Containers: 5
Nodes: 2
 dockernode1: 192.241.228.93:2375
  └ Containers: 4
  └ Reserved CPUs: 0 / 1
  └ Reserved Memory: 0 B / 490 MiB
 dockernode2: 104.236.163.107:2375
  └ Containers: 1
  └ Reserved CPUs: 0 / 1
  └ Reserved Memory: 0 B / 490 MiB
```

```
johnnytu@swarm-manager:~$ docker info
Containers: 3
Strategy: spread
Filters: affinity, health, constraint, port, dependency
Nodes: 2
 swarm-node1: 198.199.95.245:2375
  └ Containers: 1
  └ Reserved CPUs: 0 / 1
  └ Reserved Memory: 0 B / 514.5 MiB
 swarm-node2: 162.243.134.165:2375
  └ Containers: 2
  └ Reserved CPUs: 0 / 1
  └ Reserved Memory: 0 B / 514.5 MiB
johnnytu@swarm-manager:~$
```

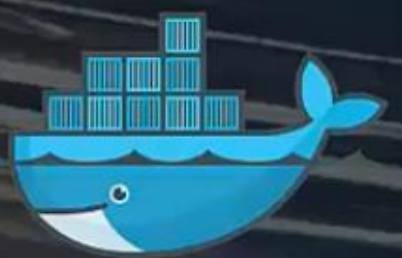
Run a container in the cluster

- Standard **`docker run command`**
- Swarm master decides which node to run the container on based on your scheduling strategy
- https://docs.docker.com/swarm/scheduler/strategy/
- Running **`docker ps`** will show which node a container is on

```
johnnytu@swarm-manager:~$ docker run -d -P nginx
861f952bf11b0bf61124f74de0d4dd8c5214c2df52b2c4d65e22c34843b1ecc6
johnnytu@swarm-manager:~$ docker run -d -P tomcat
209db9ce6a410e48d76b5d1e8b7157e5d35021149b595b312d5b3322f10546bd
johnnytu@swarm-manager:~$
```

```
johnnytu@swarm-manager:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
                    NAMES
209db9ce6a41        tomcat:latest       "catalina.sh run"        7 minutes ago       Up 7 minutes        198.199.95.245:32770->8080/tcp
                    swarm-node1/stupefied_colden
861f952bf11b        nginx:latest        "nginx -g 'daemon of   8 minutes ago       Up 8 minutes        198.199.95.245:32769->80/tcp, 198.199.95.24
5:32768->443/tcp    swarm-node1/compassionate_yalow
johnnytu@swarm-manager:~$
```

Intro to Compose



docker

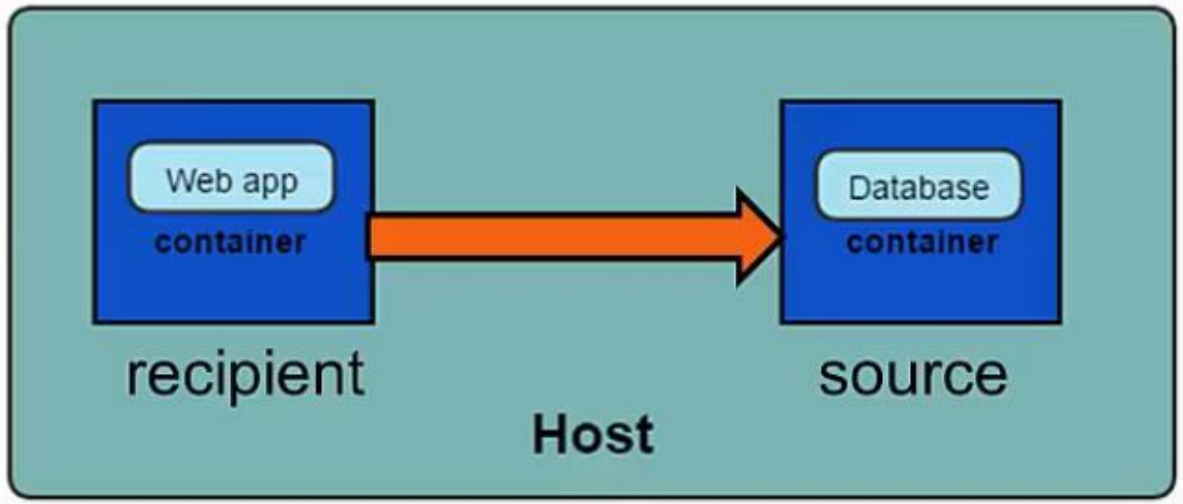
What is Compose?

*Docker **Compose** is a tool for creating and managing multi container applications*

- Containers are all defined in a single file called **docker-compose.yml**
- Each container runs a particular component / service of your application. For example:
  - Web front end
  - User authentication
  - Payments
  - Database
- Container links are defined
- Compose will spin up all your containers in a single command

Benefits of Compose

- **Quick recap on linking containers**
- Recipient container can access data on source container
- Starting up each container separately and linking them is not very practical



Configuring the Compose yml file

- Defines the services that make up your application
- Each service contains instructions for building and running a container

Build and image instruction

- **Build** defines the path to the Dockerfile that will be used to build the image
- Container will be run using the image built
- **Image** defines the image that will be used to run the container
- All services must have either a build or image instruction

Build image using Dockerfile in current directory

Use the latest redis Image from Docker Hub

```
javaclient:
  build: .
redis:
  image: redis
```

Links

- Same concept as container linking
- Specify `<service name>:<alias>`
- If no alias is specified, the service name will be used as the alias
- Creates an entry for the alias inside the container's /etc/hosts file

```
javaclient:
  build: .
  command: java HelloWorld
  links:
   - redis
redis:
  image: redis
```

Running your application

- Use **`docker-compose up`**
- Up command will
  - Build the image for each service
  - Create and start the containers

Installation


```
curl -L https://github.com/docker/compose/releases/download/1.18.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
```

```
chmod +x /usr/local/bin/docker-compose
```

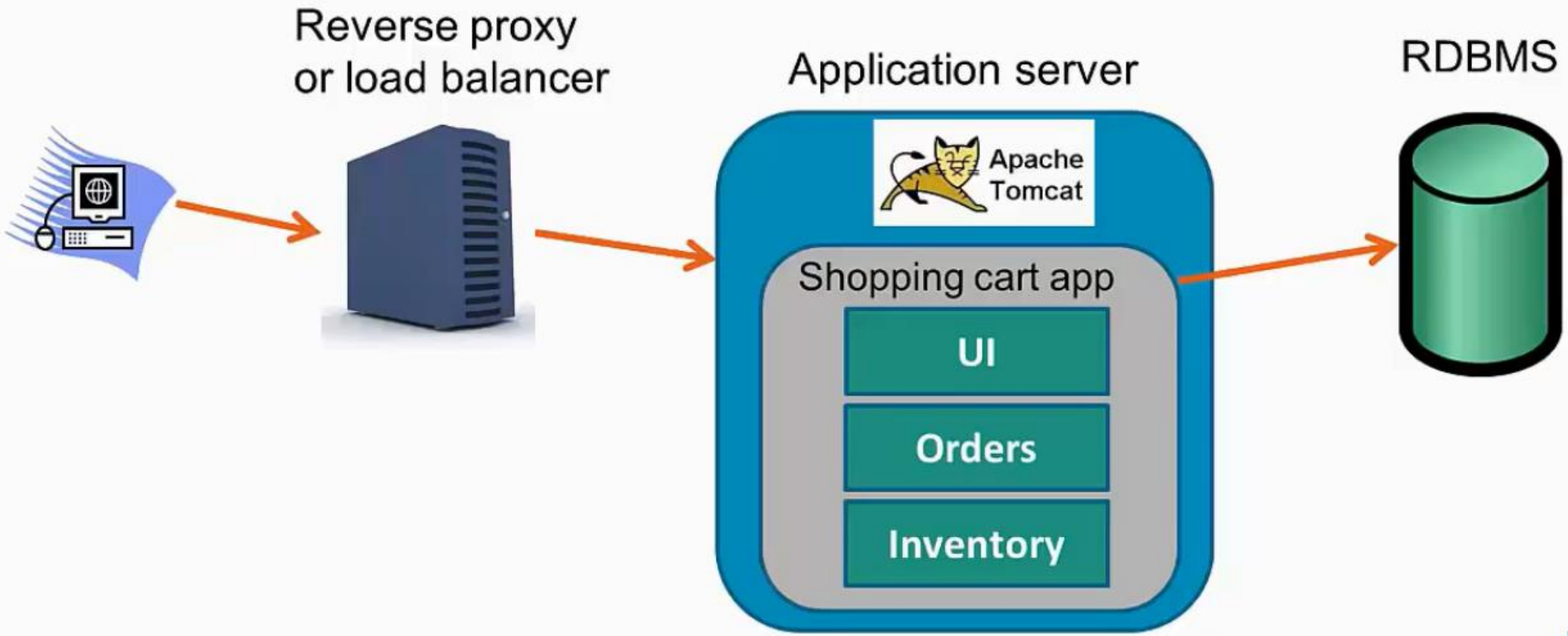
```
dovker-compose –version
cd HelloWorld/
Ls
cd /src
vi HelloWorld.java
cd ../HelloWorld
vi Dockerfile
//RUN Without Compose
docker build –t raghu/javahelloworld .
docker run –d –name redis redis
docker run –link redis:redis raghu/javahelloworld java HelloWorld
exit
vi docker-compose.yml
docker-compose up
docker ps
docker-compose up –d
docker ps
docker logs <container id>


docker logs helloworld_redis_1
```

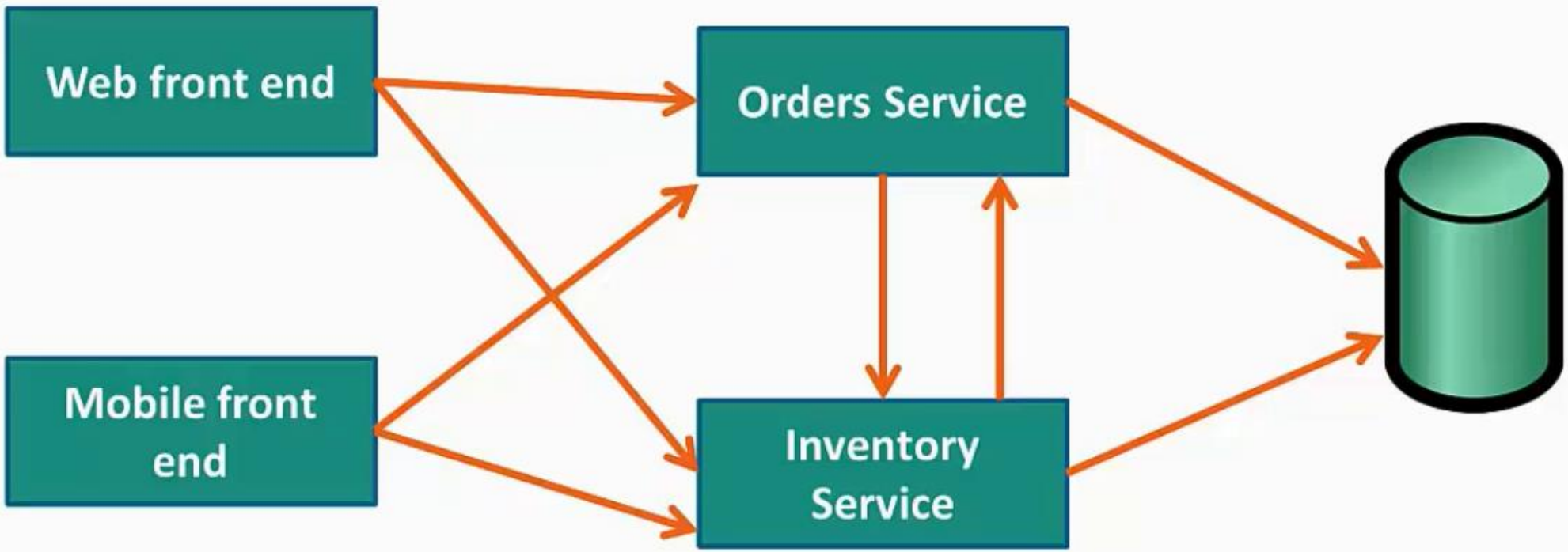
Building Micro Service Applications

docker

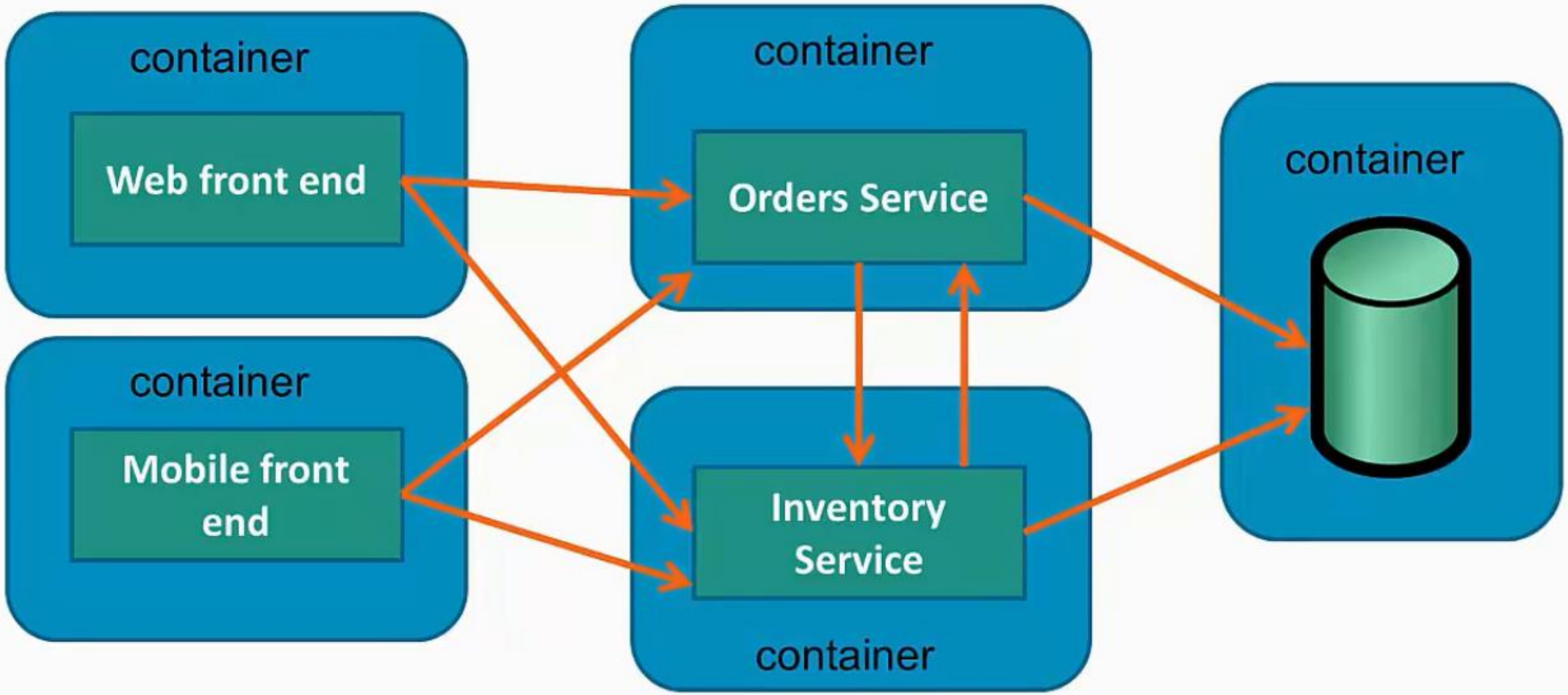
Traditional style monolithic architecture



Micro service architecture



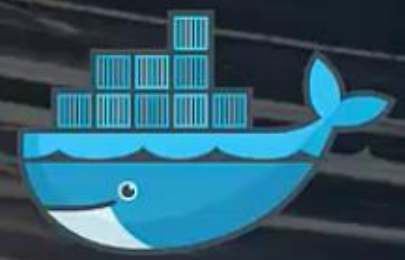
Micro service architecture



Advantages

- Each service can be developed and upgraded independently
- Easier for developers to understand
  - Only have to focus on their service
- If one service goes down, the application should still run, albeit with reduced functions
- Application is easier to troubleshoot
- The whole application does not have to be committed to one technology stack

Further Resources



docker

References

- Docker command line
  https://docs.docker.com/reference/commandline/cli/
- Dockerfile
  https://docs.docker.com/reference/builder/
- Docker Compose command line
  https://docs.docker.com/compose/cli/
- docker-compose.yml
  https://docs.docker.com/compose/yml/

Raghu
raghuopsdev@gmail.com