CMPE 282 Cloud Services
Docker Lab

Instructor: Kong Li

Content

- Docker Requirements
- Linux VM
- VM Creation
- Docker Installation
- Docker Start/Stop
- Docker tutorial
- Useful Docker commands
- HW

CMPE vCenter Server Lab Rules

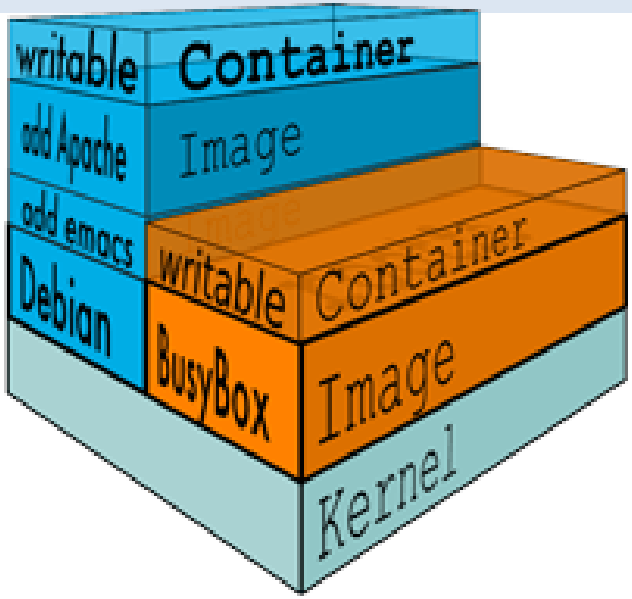
- You have permission only on
  - Resource Pool: **CMPE282 SEC1**
  - VM folder: **CMPE LABS/CMPE282 SEC1/workspace**
  - Datastore: **Classroom**

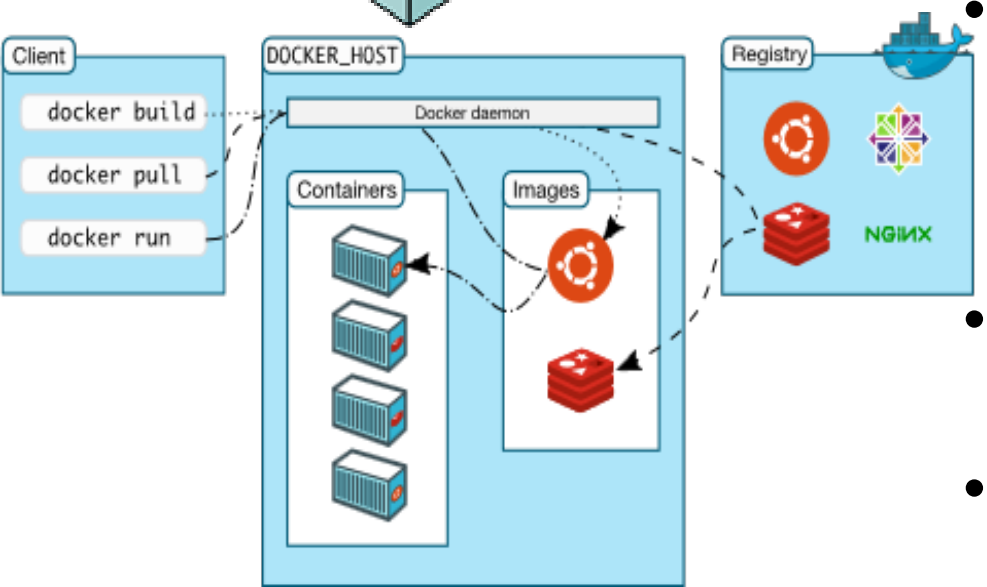
Your own area: a folder **YourName-L3SID** under workspace

- Naming convention for any newly-created VM/template/vApp
  - Naming convention: **<YourName>-<os><version>-<L3SID>** , e.g., john-ub1404-123
  - If necessary (avoid more naming collision), append **-1**, **-2**, etc at the end
- VM creation rules:
  - You can create VM only based on template, ISO file, or OVF/OVA **provided by the instructor**
    - Any created VM must connect to a specified network **without** internet access, unless allowed by instructor
  - You are **not** allowed to create VMs based on **your** uploaded ISO file or OVF/OVA
- Connection to vCenter Server and VM console
  - Web client: supported browser + several plug-ins (client support + remote console)
  - VM console (Web client or VI client): require SJSU VPN
  - ssh
- It is a shared environment
  - Be responsible - never disrupt other users
  - Clean up (**power off** or **delete** VM) as soon as you finish – penalty if you fail to do so
  - **Any malicious action will face discipline**

Login acct: vsphere.local\cmpe282_sec1_student

Docker





- Docker Engine
  - Based on Linux container LXC
  - Images (left)
    - Union File system: Multiple layers of FS + writable layer
  - Share/deploy faster, scale easily
  - Docker Hub, Docker Registry (left)
- Built-in orchestration – Swarm
  - cluster, scheduling, placement
  - Integrated or standalone: since v1.12
- Docker Machine: auto provision hosts across multi-platforms
- Docker Compose: define multi-container apps

Docker Requirement

- Docker engine >= 1.12
- Editions: Enterprise, Community
- Platforms: 64-bit OS
  - Linux (desktop recommended): kernel version >= 3.10
    - Ubuntu: 14.04 or higher, http://www.ubuntu.com/download/desktop
    - CentOS: 7 or higher, https://www.centos.org/
    - Fedora: 24 or higher, https://getfedora.org/
    - VMware Photon OS: https://vmware.github.io/photon/
  - Mac OS X: 10.11 or higher, on 2010 or newer Mac, MMU virtualization
  - Windows: 10 pro, Hyper-V
  - AWS: EC2, auto scaling, ELB, Availability Zones, VPC
  - Microsoft Azure, Google Cloud, etc

Docker Engine Installation

- https://docs.docker.com/engine/installation/
- Ubuntu (docs.docker.com/engine/installation/linux/docker-ce/ubuntu/#install-docker-ce)
  - Using Repository

    $ sudo apt-get update

    $ sudo apt-get install apt-transport-https ca-certificates curl software-properties-common

    $ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

    $ sudo add-apt-repository \

    "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

    $ sudo apt-get update

    $ sudo apt-get install docker-ce=17.03.2~ce-0~ubuntu-xenial

  - Manually
    - Download the .deb package file from https://download.docker.com/linux/ubuntu/dists/

    $ sudo dpkg -i /path/to/package.deb

  - If run docker via non-root user, “sudo usermod -aG docker <login>”, re-login
- Quick test: $ docker run hello-world

If curl not installed
$ sudo apt-get update
$ sudo apt-get install curl

Docker Start/Stop

- System V
  - Start docker daemon: sudo service docker start
  - Stop docker daemon: sudo service docker stop
  - Status docker daemon: sudo service docker status
- SystemD
  - Start docker daemon: sudo systemctl start docker
  - Boot start docker: sudo systemctl enable docker
  - Docker daemon status: sudo systemctl status docker
- Enable root login via SSH (not recommended; only if you need to)
  - Disabled in CMPE vCenter VM templates
  - Modify /etc/ssh/sshd_config
    - change from “PermitRootLogin no” to “PermitRootLogin yes”
  - Restart sshd daemon
    - Service restart sshd, OR systemctl restart sshd

Hello World
docs.docker.com/get-started/

- Make sure docker engine is up

  $ service docker start OR systemctl start docker

- hello-world test

  $ docker run hello-world

- Ubuntu echo Hello world

  $ docker run ubuntu:16.04 /bin/echo 'Hello world'

  Hello world

  image — cmd to exec

- Interactive container

  $ docker run **-t -i** ubuntu:16.04 /bin/bash

  root@af8bae53bdd3:/#

  root@af8bae53bdd3:/# pwd

  /

  root@af8bae53bdd3:/# ls

  bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr

  root@af8bae53bdd3:/# exit

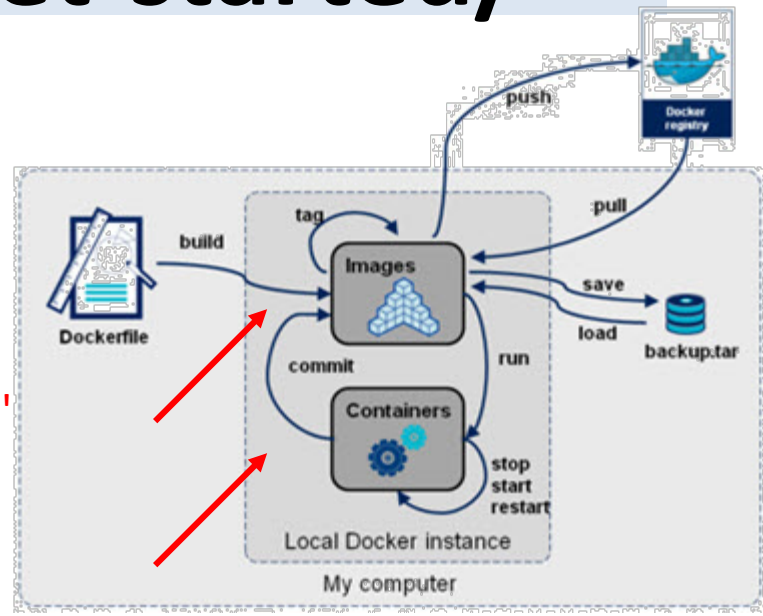


Image: layers of FS
Container: **UnionFS** - layers merged
- Image: read-only
- writable layer: read/write (copy-on-write)

Hello World (cont'd)

- A daemonized Hello world

  $ docker run **-d** ubuntu:16.04 /bin/sh -c "while true; do echo hello world; sleep 10; done"

  1e5535038e285177d5214659a068137486f96ee5c2e85a4ac52dc83f2ebe4147

  $ docker ps

  CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

  1e5535038e28 ubuntu:14.04 /bin/sh -c 'while tr 2 minutes ago Up 1 minute insane_babbage

  $ docker **exec** -t -i insane_babbage bash

  root@af8bae53bdd3:/# exit

  $ docker **logs** insane_babbage

  hello world

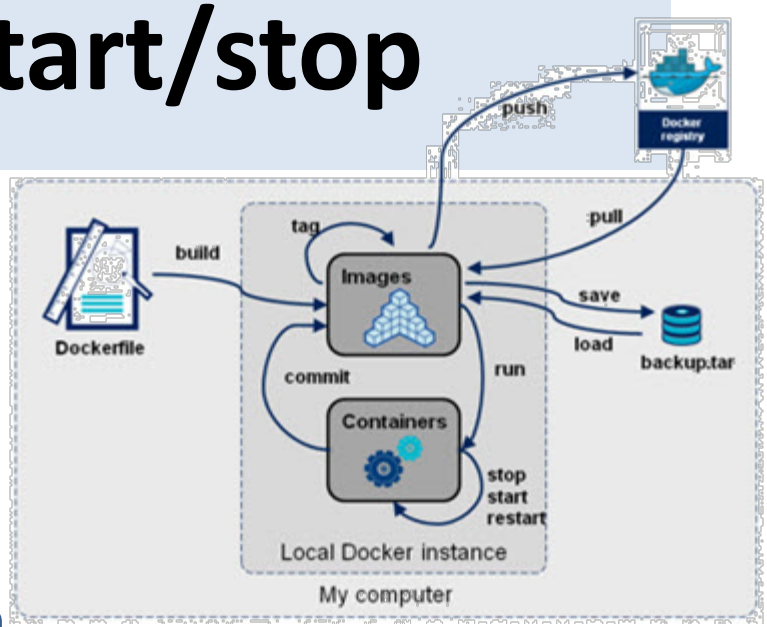
  …

  $ docker **stop** insane_babbage

  insane_babbage

  $ docker ps

Container: run/start/stop



- docker --help
- docker version
- docker ps
- docker logs CONTAINER
- docker stop CONTAINER
- docker attach --help
- Web app container

Detach: ctrl-p ctrl-q

$ docker run -d **-P** training/webapp python app.py

Expose all ports; auto mapping

$ docker ps **-l**

Last docker container

CONTAINER ID IMAGE COMMAND CREATED STATUS

PORTS NAMES

bc533791f3f5 training/webapp:latest python app.py 5 seconds ago Up 2 seconds

0.0.0.0:49155->5000/tcp nostalgic_morse

Container:5000 auto mapped to localhost:49155

$ docker ps **-a**

$ docker run -d -p **80:5000** training/webapp python app.py

Container:5000 mapped to localhost:80

$ docker logs -f nostalgic_morse

Container: run/start/stop (cont'd)

- Processes within container

  $ docker **top** nostalgic_morse
- Inspect container

  $ docker **inspect** nostalgic_morse
- Find IP of container

  $ docker inspect **-f** '{{ .NetworkSettings.IPAddress }}' nostalgic_morse

  172.17.0.5
- Stop, start, and remove container (run == create + start)

  $ docker stop nostalgic_morse

  nostalgic_morse

  $ docker ps -l

  $ docker start nostalgic_morse

  nostalgic_morse

  $ docker **rm** nostalgic_morse

  Error: Impossible to remove a running container, please stop it first or use -f
  2014/05/24 08:12:56 Error: failed to remove one or more containers

Container: Images

- List current images currently available on host

  $ docker images

- Pull images (from Docker Hub) to host

  $ docker **pull** centos

- Search images on Docker Hub (or visit https://hub.docker.com/)

  $ docker **search** centos

- See docs.docker.com/get-started/part2
- Build images from Dockerfile

  $ **docker build -t friendlyhello .**

  $ docker run **-p** 4000:80 friendlyhello

  $ docker run **-d** -p 4000:80 friendlyhello

  - Visit URL (or curl) http://localhost:4000

- Build images from a container and then commit to a new image

  $ docker commit -m "build manually" 0b2616b0e5a8 friendlyhello2

Image: tree-like layers, each layer built on top of another
Each instruction in Dockerfile becomes history (layer, FS diff) of the image

```
# Use Python runtime as a parent image
FROM python:2.7-slim
# Set the working directory to /app
WORKDIR /app
# Copy the current dir into container:/data
ADD . /app
# Install needed packages in requirements.txt
RUN pip install -r requirements.txt
# map port 80 to 80
EXPOSE 80
# Define environment variable
ENV NAME World
# Run app.py when the container launches
CMD ["python", "app.py"]
```

Container: Images (cont'd)

- Tag an image, login, and then push image to Hub

  username/repository:tag

  $ docker **tag** friendlyhello cmpe/get-started:part1

  $ docker **login** --username=cmpe --email=youremail@company.com

  $ docker **push** cmpe/get-started:part1

- Remove images from host

  $ docker **rmi** cmpe/get-started:part1

- **Image** save/load

  FS only; no runtime state

  - A Docker *image* can be *saved* to a tarball and *loaded* back
    - preserve history of image (all layers. tags, metadata, etc.); Size larger
    - Can roll back to a layer

- **Container** export/import

  No change; read-only

  - A Docker boots (instantiates) an image and adds additional layer(s) on top of it
  - A Docker *container* can be *exported* to a tarball and *imported* back
    - Flattened, no history (no chain of layers); Size smaller
    - Cannot roll back to a layer

- Container migration from host 1 to host 2?

Docker: Services, Stack, App, Compose, Machine, Swarm

- service == a component (of a distributed app) running in a container

  $ docker service --help

- stack == a group of related services that share dependencies, and can be orchestrated and scaled together

  $ docker stack --help

- app == a single stack, or multiple stacks
- docker-compose: define multi-container apps
  - Install: docs.docker.com/compose/install/
- docker-machine: auto provision hosts
  - Install: docs.docker.com/machine/install-machine/
- Swarm: a Dockerized cluster of multiple hosts
- Canvas: Files/src/docker-friendlyhello

Docker: Compose

- docs.docker.com/get-started/part3
- docker-compose.yml

$ docker swarm init

Current host: swarm manager

$ docker stack deploy -c docker-compose-1.yml getstartedlab

$ docker stack ls

$ docker stack services getstartedlab

$ docker stack ps getstartedlab

- Scale: change replicas, in-place update

$ docker stack deploy -c docker-compose-1.yml getstartedlab

$ docker stack ps getstartedlab

$ docker stack rm getstartedlab

- Leave swarm: $ docker swarm leave --force

instances

Load balanced overlay network

```
version: "3"
services:
  web:
    # replace username/repo:tag
    image: friendlyhello
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "80:80"
    networks:
      - webnet
networks:
  webnet:
```

Docker: Swarm

- docs.docker.com/get-started/part4
- Swarm: a Dockerized cluster of multiple hosts (nodes)
  - manual setup or use docker-machine
  - One Swarm manager: $ docker swarm init
    - Exec cmds on swarm manager directly, or indirectly via docker-machine
  - Multiple workers: $ docker swarm join --token <token> <ip>:<port>
- On swarm manager:

Each node has different hostname

$ docker node ls (or $ docker-machine ls)

$ docker stack deploy -c docker-compose-1.yml getstartedlab

$ docker stack ps getstartedlab

Image available on all nodes, if not pulled from Hub

$ docker service ls

$ docker network ls

- http visit any host in swarm – it will be load-balanced across containers across all hosts in the swarm

Docker: Stack

- docs.docker.com/get-started/part5
  - Visualizer

$ docker stack rm getstartedlab

$ docker stack deploy -c docker-compose-2.yml getstartedlab

$ docker stack ps getstartedlab

  - http visit swarm master port 8080
  - add redis to .yml (right)

$ docker stack rm getstartedlab

$ docker stack deploy -c docker-compose-3.yml getstartedlab

$ docker stack ps getstartedlab

  - http visit swarm master port 8080
  - http visit any node port 80 in cluster

- docs.docker.com/get-started/part6

host:./data
container: /data

```
version: "3"
services:
  web:
    # replace username/repo:tag
    image: friendlyhello
    deploy:
      replicas: 5
      restart_policy:
        condition: on-failure
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
    ports:
      - "80:80"
    networks:
      - webnet
  redis:
    image: redis
    ports:
      - "6379:6379"
    volumes:
      - ./data:/data
    deploy:
      placement:
        constraints: [node.role == manager]
    networks:
      - webnet
networks:
  webnet:
```

Docker: Network

- Docker has two types of networks
  - Bridged: limited to single host
  - Overlay: multi-hosts (i.e., for Swarm mode)
- Docker auto creates three networks: bridge (docker0), host, none
  - The default bridge network connects to a net IF called docker0 (172.17.x.y/16)

    $ docker network ls

    $ ip addr (or ifconfig -a)
- The default bridge network (connected to docker0)
  - Member containers can talk with each other, and host, by using IP
    - If specifying "--link", container name can be used (DNS implication)
  - Containers are created in the default bridge network, unless "--net" specified
- Legacy: link container web to container db (DNS implication)
  - Format: --link ContainerID:DnsAlias

$ docker run -d **--name db** training/postgres

$ docker run -d -P **--name web --link=db:db** training/webapp python app.py

$ docker network inspect bridge

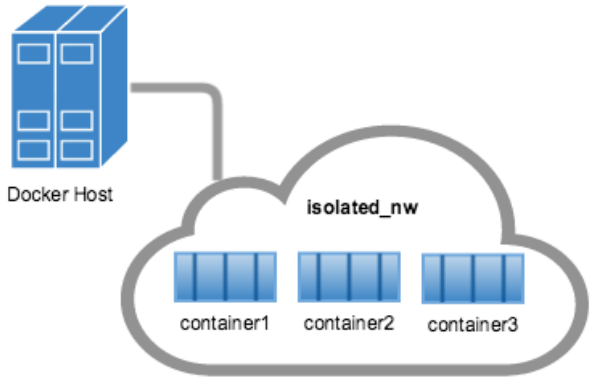
Docker: Bridged Network

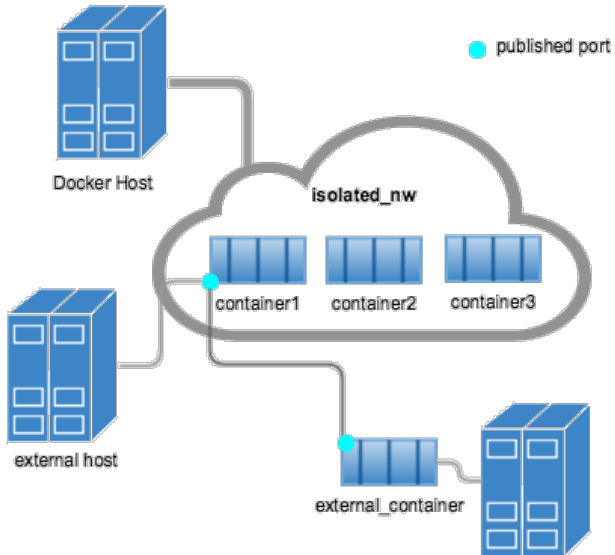
- Create a user-defined network isolated_nw with bridge driver (also new IF)

  $ docker network create -d bridge isolated_nw

  $ docker network ls
- Run a container and connect to isolated_nw

  $ docker run -d **--name db --net=isolated_nw** training/postgres
- Containers in isolated_nw can talk with each other using IP (or name)
- Containers can only talk within networks but not across networks
- A container attached to two networks can talk with member containers in either network

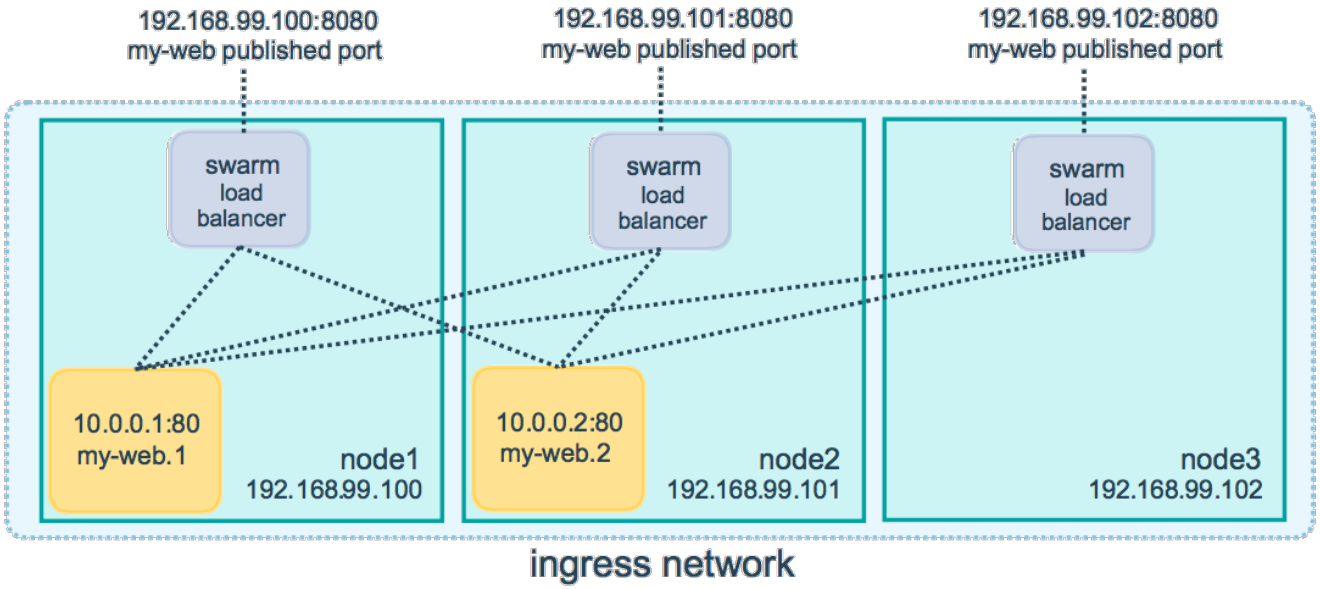




Docker: Overlay Network

- Overlay
  - swarm mode routing mesh: enables each node in the swarm to accept connections on published ports for any service running in the swarm, even if there's no task running on the node
  - Docker Swarm does this automatically for you

$ docker service create --name my-web --publish 8080:80 --replicas 2 nginx



Useful Docker Commands

- docker version
- docker info
- docker ps
- docker ps -a
- docker start containerid
- docker stop containerid
- docker rm containerid
- docker run -d --name=… --net=…
- docker run -it --name=… --net=…
- docker cp …
- docker exec …
- docker attach …
- docker logs containerid
- docker top containerid
- docker exec -it containerid proc
- docker images
- docker rmi image
- docker build …
- docker stack ls
- docker service ls
- docker network ls
- docker inspect containerid

HW

- See Canvas

References

- http://blogs.vmware.com/workstation/2015/04/installing-project-photon-vmware-workstation.html
- https://docs.docker.com/userguide/dockerizing/
- https://docs.docker.com/linux/
- Upgrade Photon 1.0TP2 with Docker 1.9.1: http://www.virtuallyghetto.com/
- https://docs.docker.com/engine/userguide/networkingcontainers/
- http://www.virten.net/2015/04/basic-commands-for-vmware-photon-and-docker/
- http://images.techhive.com/assets/media-resource/16373/ifw_dd_docker.pdf