1 Starting stopping and running container:

To start the docker daemon run:

```
$ systemctl start docker
```

To run container, container will first pull from docker and then run.

```
$ docker container run -it <container-to-download>
```

the following command prints processes and orders by resource consumption: Container namespace isolation prevents seeing other processes.

```
$ top
```

to get info on running containers:

```
$ docker container ls
```

to enter into existing container use:

```
$ docker container exec -it <sha-of-container> bash
```

types of isolation docker provides:

- PID is just one of the Linux namespaces that provides containers with isolation to system resources.
- MNT: Mount and unmount directories without affecting other namespaces.
- NET: Containers have their own network stack.
- IPC: Isolated interprocess communication mechanisms such as message queues.
- User: Isolated view of users on the system.
- UTC: Set hostname and domain name per container.

to run multiple containers: (note $-d or --detach$ runs container in background, $-p or --publish$ is to attach default port from container (determined from documentation) to your assigned port in your host, –name assigns port name so you can use name instead of the SHA)

```
$ docker container run --detach --publish <fpt>:<tpt> --name <name> <cont>
```

Destroy containers:

```
$ docker container stop <name or SHA>
```

and remove stopped containers

```
$ docker system prune
```

2 building and pushing docker image

example: add to DockerFile

```
FROM python:3.6.1-alpine # starting image to build your layers on top of
RUN pip install flask #-- commands required to set up image
CMD ["python","app.py"] #--executed command when starting the container
COPY app.py /app.py #--files copied in containers local directory
```

build docker image (-t is to name image)

```
$ docker image build -t python-hello-world .
```

to see log in application

```
$ docker container logs [container id]
```

push to registry:

```
$ docker login #enter username and password
$ docker tag python-hello-world [dockerhub username]/python-hello-world
#tag w/username
$ docker push jzaccone/python-hello-world #push
```

to deploy a change:

```
$ docker image build -t jzaccone/python-hello-world . #rebuild
$ docker push jzaccone/python-hello-world #repush
```

docker only rebuilds changes of changed layer and all layers above it. None below.

2.1 Understanding Layers

each line in the dockerfile represents a layer, the lower the line, the higher the layer layers are read only except toplayer which is copy-on-write which pulls up layers files when edits are made to lower files. Layers can be shared by containers can be seen from

```
$ docker image history python-hello-world
```

3 Swarms

in one instance of a terminal run:

```
$ docker swarm init --advertise-addr eth0 #run join swarm for child nodes
```

The command generates join token to ensure no malicious nodes join swarm

Now add a service: (–mount has hostname print out of node it is running on) –publish runs built in routing mesh

```
docker service create --detach=true --name nginx1 --publish 80:80
--mount source=/etc/hostname,target=/usr/share/nginx/html/index.html,
type=bind,ro nginx:1.12
```

Inspect services:

```
docker service ls #running containers
docker service ps nginx1 #running instances of service
curl localhost:80 #test the service, see which node is running
```

Scale up service

```
$ docker service update --replicas=5 --detach=true nginx1
nginx1 #replicas add that many instances
```

What Happens?

- The state of the service is updated to 5 replicas, which is stored in the swarm's internal storage.
- Docker Swarm recognizes that the number of replicas that is scheduled now does not match the declared state of 5.
- Docker Swarm schedules 4 more tasks (containers) in an attempt to meet the declared state for the service.

Sending a lot of requests will result in changing nodes in swarm which handle requests. see historical log from

```
$ docker service logs nginx1
```

apply updates:

```
$ docker service update --image nginx:1.13 --detach=true nginx1
```

docker Now updates. Recognizes desired and actual state are mismatched and forces correction in update.

If you leave swarm and watch with this command

```
watch -n 1 docker service ps nginx2
```

swarm automatically corrects itself

Node failures: in event of manager node failure, you want other manager nodes to pick up slack. three managers handles one, five handles two and seven -three