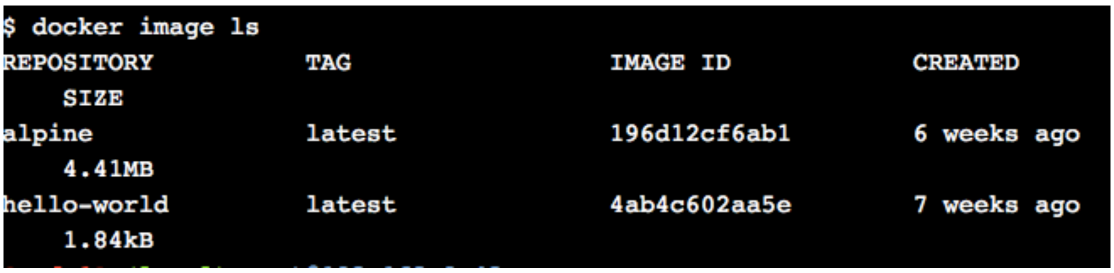
Docker is all about speed. Develop, build, test, deploy, update, recover faster. Maintenance and complexity drains budgets, so innovation suffers.

First Alpine Linux Containers let me learn the basic usage of container, docker images, docker container run and container isolation.



```
$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED
    SIZE
alpine              latest              196d12cf6ab1        6 weeks ago
    4.41MB
hello-world         latest              4ab4c602aa5e        7 weeks ago
    1.84kB
```

Fig 1. Docker Image

Video: What is a Container

Most containers are in Linux and people can run any number of processes inside the operating system. These processes share an address space and they share a process name space.

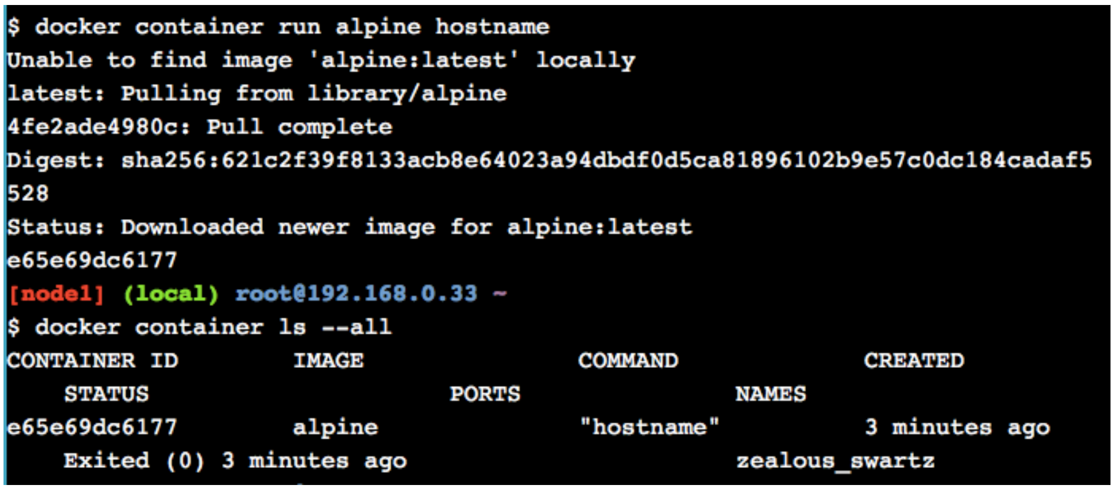
Container process and the container lifecycle are completely tightly coupled.

Container image is simply a binary representation. It is just a bunch of bits on a file system in the same way as a VMDK as a disk image and OVA as a image for a VM. It is basically an image that contains some binary state the interesting thing there about a container image at least in the way.

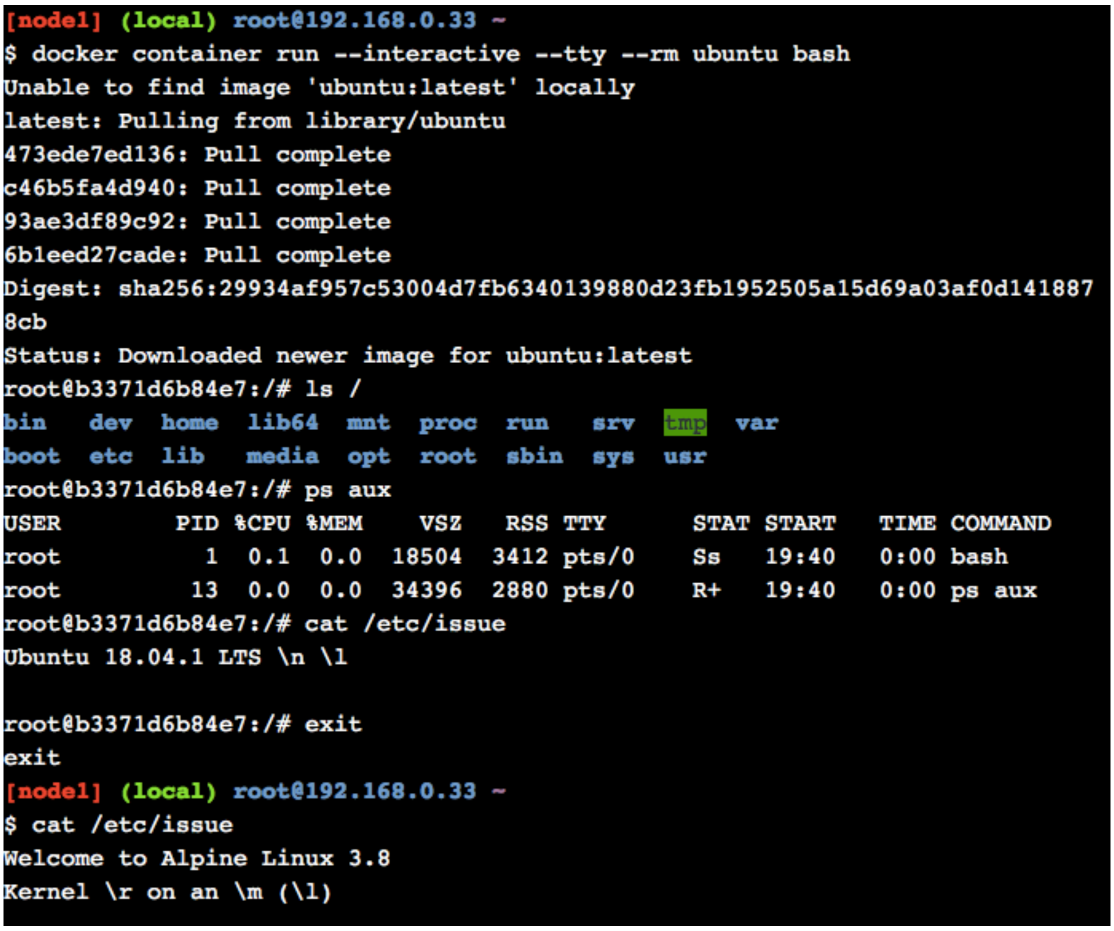
Docker file is basically an environment in a file in a text file.

A docker file ultimately ends up creating an image so we use docker files and dock a build to create this image tree of images that we can then use to instantiate containers. Docker file is typically a starting point for an image because it makes that really easy.

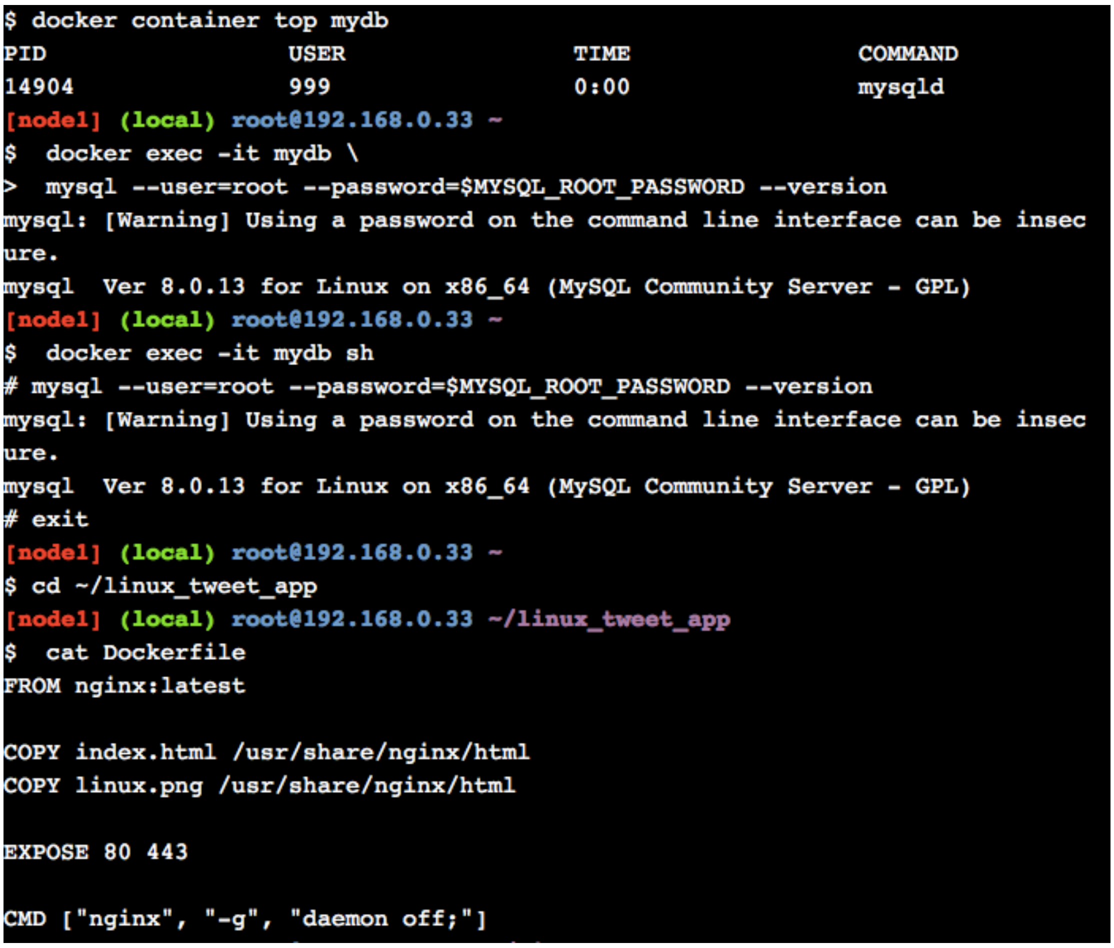
Lab: Docker Intro



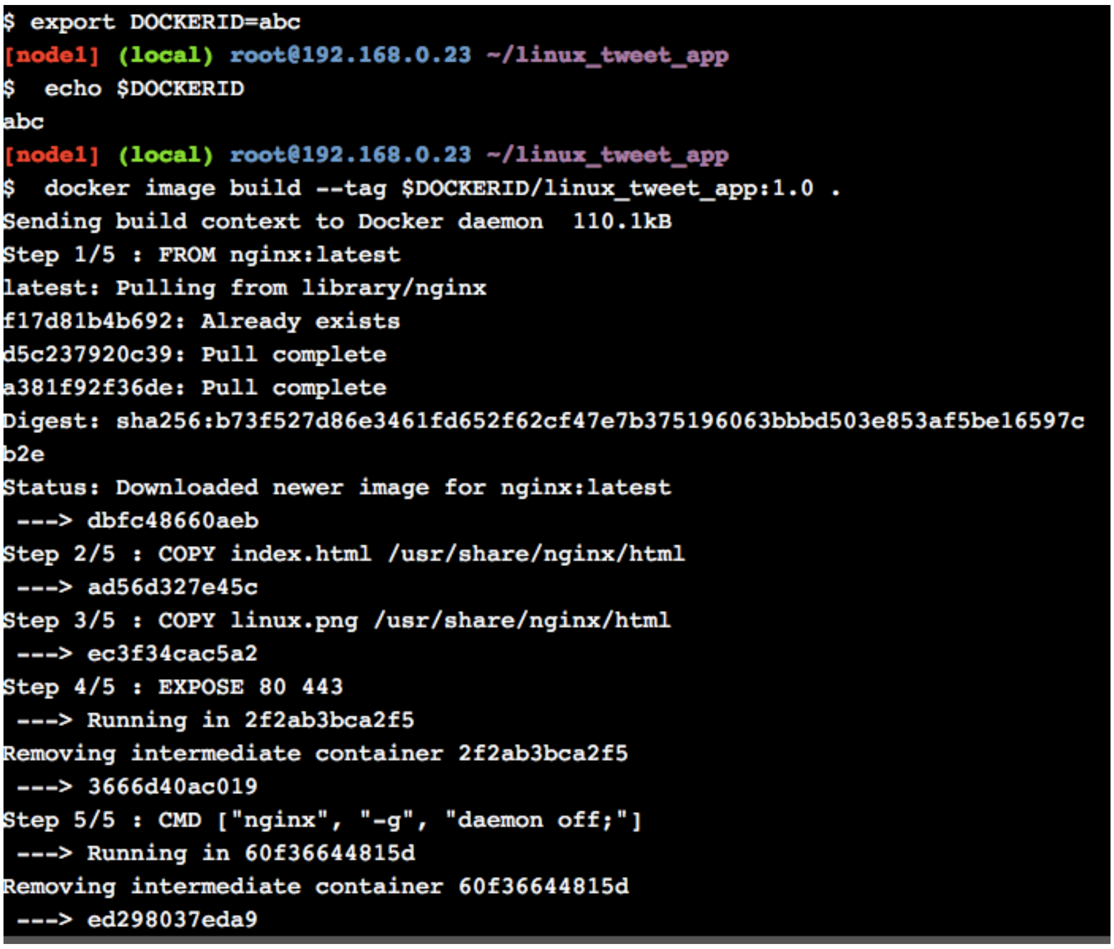
```
$ docker container run alpine hostname
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
4fe2ade4980c: Pull complete
Digest: sha256:621c2f39f8133acb8e64023a94dbdf0d5ca81896102b9e57c0dc184cadaf5
528
Status: Downloaded newer image for alpine:latest
e65e69dc6177
[node1] (local) root@192.168.0.33 ~
$ docker container ls --all
CONTAINER ID        IMAGE               COMMAND             CREATED
    STATUS                      PORTS               NAMES
e65e69dc6177        alpine              "hostname"          3 minutes ago
    Exited (0) 3 minutes ago                        zealous_swartz
```



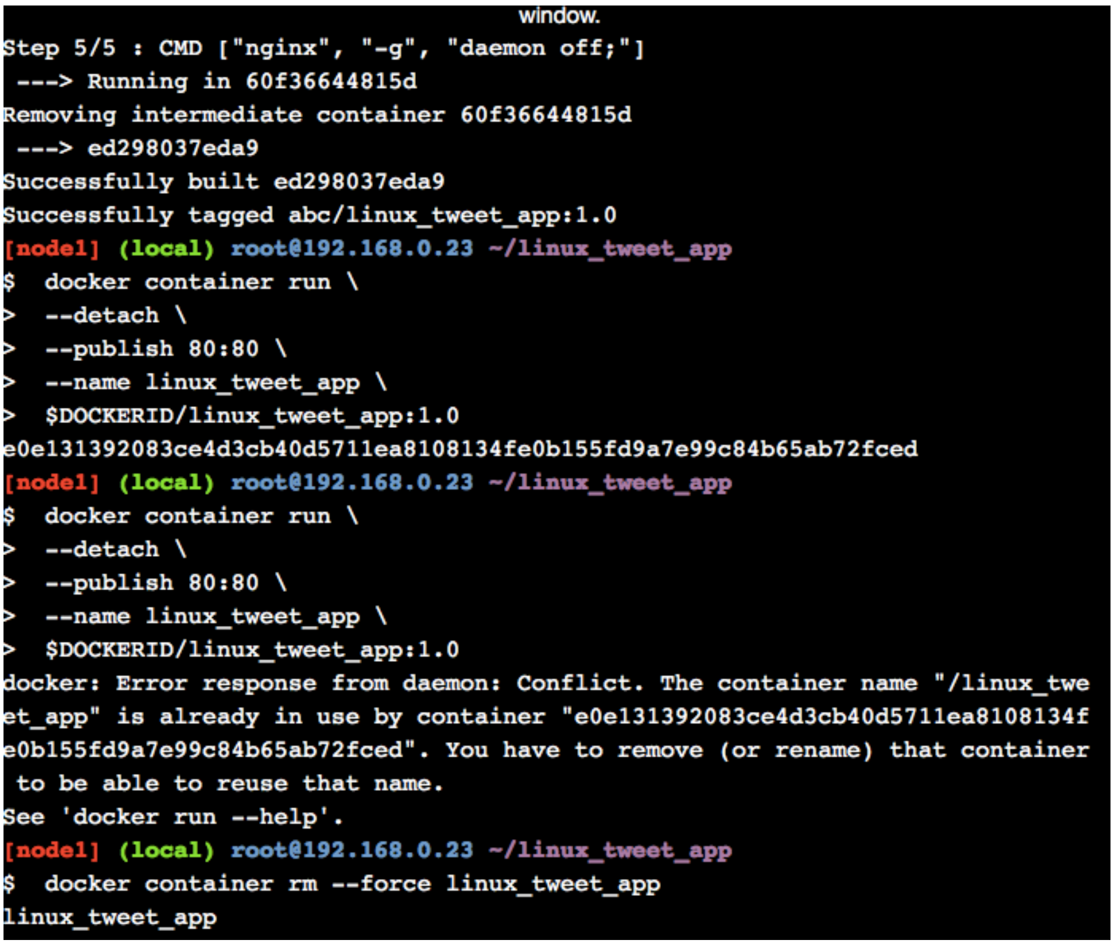
```
[node1] (local) root@192.168.0.33 ~
$ docker container run --interactive --tty --rm ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
473ede7ed136: Pull complete
c46b5fa4d940: Pull complete
93ae3df89c92: Pull complete
6b1eed27cade: Pull complete
Digest: sha256:29934af957c53004d7fb6340139880d23fb1952505a15d69a03af0d141887
8cb
Status: Downloaded newer image for ubuntu:latest
root@b3371d6b84e7:/# ls /
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
root@b3371d6b84e7:/# ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.1  0.0  18504  3412 pts/0    Ss   19:40   0:00 bash
root          13  0.0  0.0  34396  2880 pts/0    R+   19:40   0:00 ps aux
root@b3371d6b84e7:/# cat /etc/issue
Ubuntu 18.04.1 LTS \n \l

root@b3371d6b84e7:/# exit
exit
[node1] (local) root@192.168.0.33 ~
$ cat /etc/issue
Welcome to Alpine Linux 3.8
Kernel \r on an \m (\l)
```



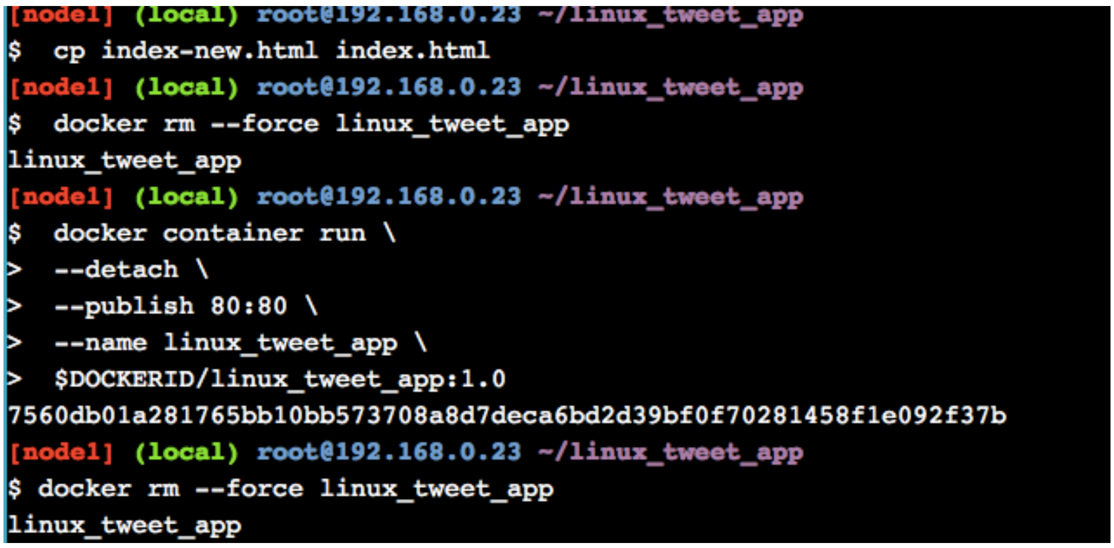
```
$ docker container top mydb
PID                 USER                TIME                COMMAND
14904               999                 0:00                mysqld
[node1] (local) root@192.168.0.33 ~
$  docker exec -it mydb \
>  mysql --user=root --password=$MYSQL_ROOT_PASSWORD --version
mysql: [Warning] Using a password on the command line interface can be insec
ure.
mysql  Ver 8.0.13 for Linux on x86_64 (MySQL Community Server - GPL)
[node1] (local) root@192.168.0.33 ~
$  docker exec -it mydb sh
# mysql --user=root --password=$MYSQL_ROOT_PASSWORD --version
mysql: [Warning] Using a password on the command line interface can be insec
ure.
mysql  Ver 8.0.13 for Linux on x86_64 (MySQL Community Server - GPL)
# exit
[node1] (local) root@192.168.0.33 ~
$ cd ~/linux_tweet_app
[node1] (local) root@192.168.0.33 ~/linux_tweet_app
$  cat Dockerfile
FROM nginx:latest

COPY index.html /usr/share/nginx/html
COPY linux.png /usr/share/nginx/html

EXPOSE 80 443

CMD ["nginx", "-g", "daemon off;"]
```



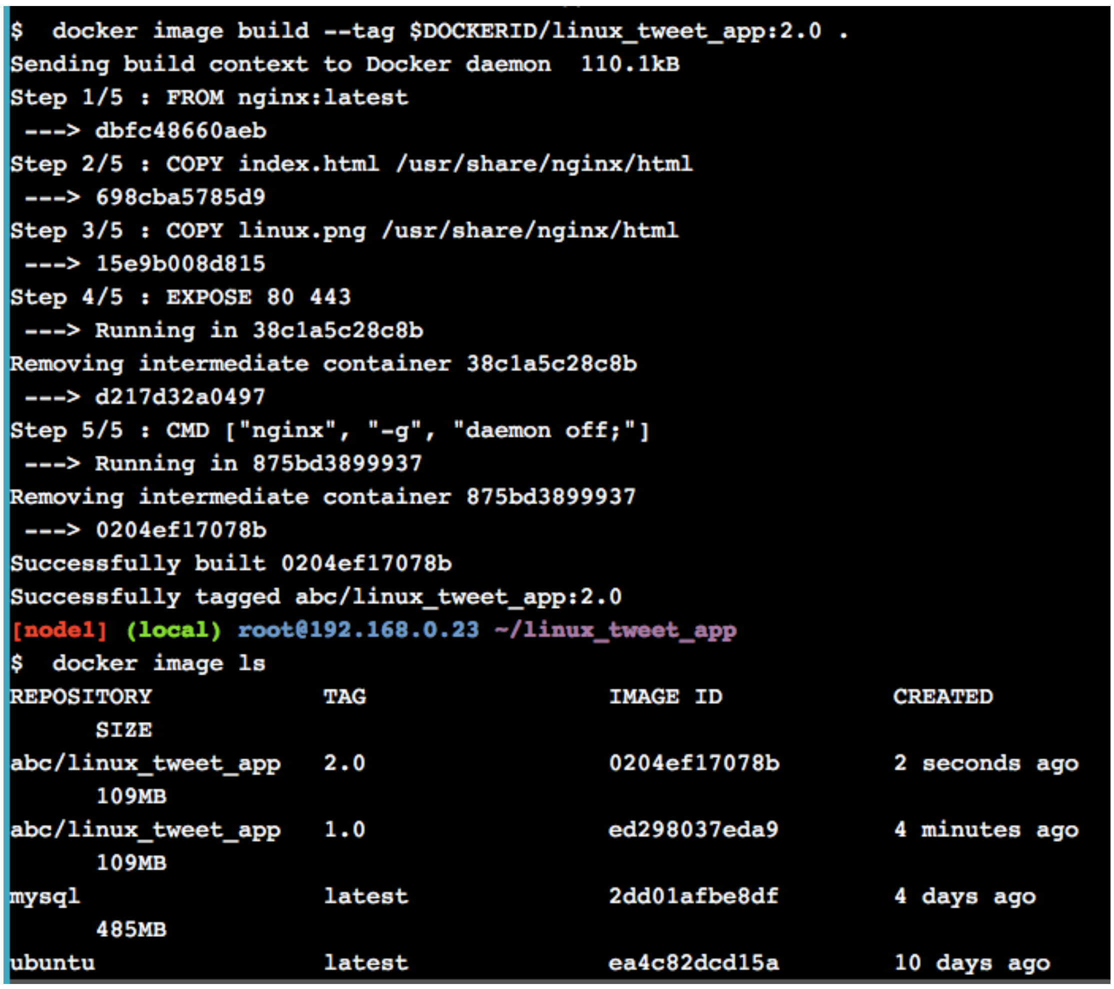
```
$ export DOCKERID=abc
[node1] (local) root@192.168.0.23 ~/linux_tweet_app
$  echo $DOCKERID
abc
[node1] (local) root@192.168.0.23 ~/linux_tweet_app
$  docker image build --tag $DOCKERID/linux_tweet_app:1.0 .
Sending build context to Docker daemon  110.1kB
Step 1/5 : FROM nginx:latest
latest: Pulling from library/nginx
f17d81b4b692: Already exists
d5c237920c39: Pull complete
a381f92f36de: Pull complete
Digest: sha256:b73f527d86e3461fd652f62cf47e7b375196063bbbd503e853af5be16597c
b2e
Status: Downloaded newer image for nginx:latest
 ---> dbfc48660aeb
Step 2/5 : COPY index.html /usr/share/nginx/html
 ---> ad56d327e45c
Step 3/5 : COPY linux.png /usr/share/nginx/html
 ---> ec3f34cac5a2
Step 4/5 : EXPOSE 80 443
 ---> Running in 2f2ab3bca2f5
Removing intermediate container 2f2ab3bca2f5
 ---> 3666d40ac019
Step 5/5 : CMD ["nginx", "-g", "daemon off;"]
 ---> Running in 60f36644815d
Removing intermediate container 60f36644815d
 ---> ed298037eda9
```



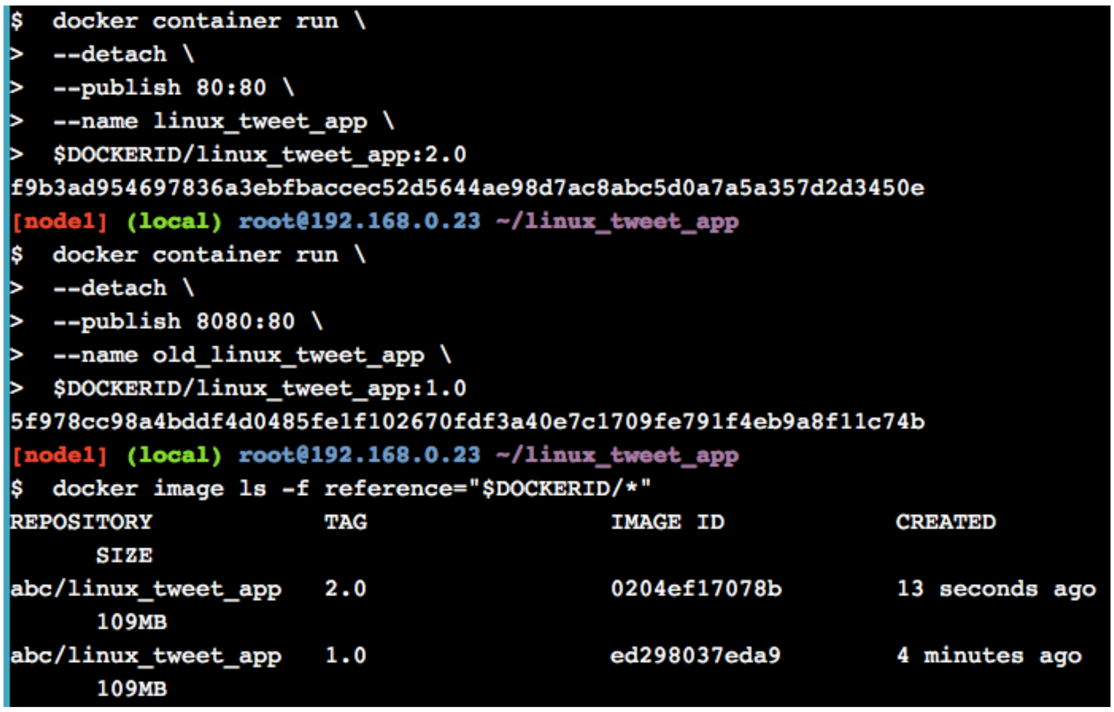
```
window.
Step 5/5 : CMD ["nginx", "-g", "daemon off;"]
 ---> Running in 60f36644815d
Removing intermediate container 60f36644815d
 ---> ed298037eda9
Successfully built ed298037eda9
Successfully tagged abc/linux_tweet_app:1.0
[node1] (local) root@192.168.0.23 ~/linux_tweet_app
$  docker container run \
>  --detach \
>  --publish 80:80 \
>  --name linux_tweet_app \
>  $DOCKERID/linux_tweet_app:1.0
e0e131392083ce4d3cb40d5711ea8108134fe0b155fd9a7e99c84b65ab72fced
[node1] (local) root@192.168.0.23 ~/linux_tweet_app
$  docker container run \
>  --detach \
>  --publish 80:80 \
>  --name linux_tweet_app \
>  $DOCKERID/linux_tweet_app:1.0
docker: Error response from daemon: Conflict. The container name "/linux_twe
et_app" is already in use by container "e0e131392083ce4d3cb40d5711ea8108134f
e0b155fd9a7e99c84b65ab72fced". You have to remove (or rename) that container
 to be able to reuse that name.
See 'docker run --help'.
[node1] (local) root@192.168.0.23 ~/linux_tweet_app
$  docker container rm --force linux_tweet_app
linux_tweet_app
```



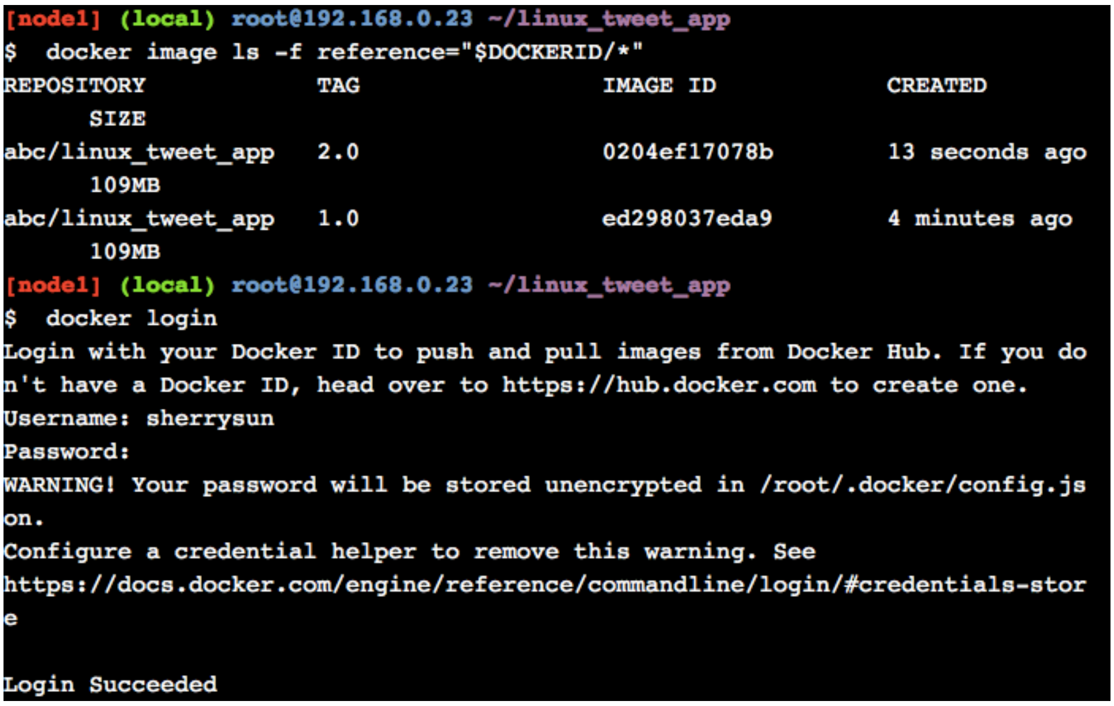
```
[node1] (local) root@192.168.0.23 ~/linux_tweet_app
$  cp index-new.html index.html
[node1] (local) root@192.168.0.23 ~/linux_tweet_app
$  docker rm --force linux_tweet_app
linux_tweet_app
[node1] (local) root@192.168.0.23 ~/linux_tweet_app
$  docker container run \
>  --detach \
>  --publish 80:80 \
>  --name linux_tweet_app \
>  $DOCKERID/linux_tweet_app:1.0
7560db01a281765bb10bb573708a8d7deca6bd2d39bf0f70281458f1e092f37b
[node1] (local) root@192.168.0.23 ~/linux_tweet_app
$ docker rm --force linux_tweet_app
linux_tweet_app
```



```
$  docker image build --tag $DOCKERID/linux_tweet_app:2.0 .
Sending build context to Docker daemon  110.1kB
Step 1/5 : FROM nginx:latest
 ---> dbfc48660aeb
Step 2/5 : COPY index.html /usr/share/nginx/html
 ---> 698cba5785d9
Step 3/5 : COPY linux.png /usr/share/nginx/html
 ---> 15e9b008d815
Step 4/5 : EXPOSE 80 443
 ---> Running in 38c1a5c28c8b
Removing intermediate container 38c1a5c28c8b
 ---> d217d32a0497
Step 5/5 : CMD ["nginx", "-g", "daemon off;"]
 ---> Running in 875bd3899937
Removing intermediate container 875bd3899937
 ---> 0204ef17078b
Successfully built 0204ef17078b
Successfully tagged abc/linux_tweet_app:2.0
[node1] (local) root@192.168.0.23 ~/linux_tweet_app
$  docker image ls
REPOSITORY            TAG                 IMAGE ID            CREATED
      SIZE
abc/linux_tweet_app   2.0                 0204ef17078b        2 seconds ago
      109MB
abc/linux_tweet_app   1.0                 ed298037eda9        4 minutes ago
      109MB
mysql                 latest              2dd01afbe8df        4 days ago
      485MB
ubuntu                latest              ea4c82dcd15a        10 days ago
```



```
$  docker container run \
>  --detach \
>  --publish 80:80 \
>  --name linux_tweet_app \
>  $DOCKERID/linux_tweet_app:2.0
f9b3ad954697836a3ebfbaccec52d5644ae98d7ac8abc5d0a7a5a357d2d3450e
[node1] (local) root@192.168.0.23 ~/linux_tweet_app
$  docker container run \
>  --detach \
>  --publish 8080:80 \
>  --name old_linux_tweet_app \
>  $DOCKERID/linux_tweet_app:1.0
5f978cc98a4bddf4d0485fe1f102670fdf3a40e7c1709fe791f4eb9a8f11c74b
[node1] (local) root@192.168.0.23 ~/linux_tweet_app
$  docker image ls -f reference="$DOCKERID/*"
REPOSITORY            TAG                 IMAGE ID            CREATED
      SIZE
abc/linux_tweet_app   2.0                 0204ef17078b        13 seconds ago
      109MB
abc/linux_tweet_app   1.0                 ed298037eda9        4 minutes ago
      109MB
```



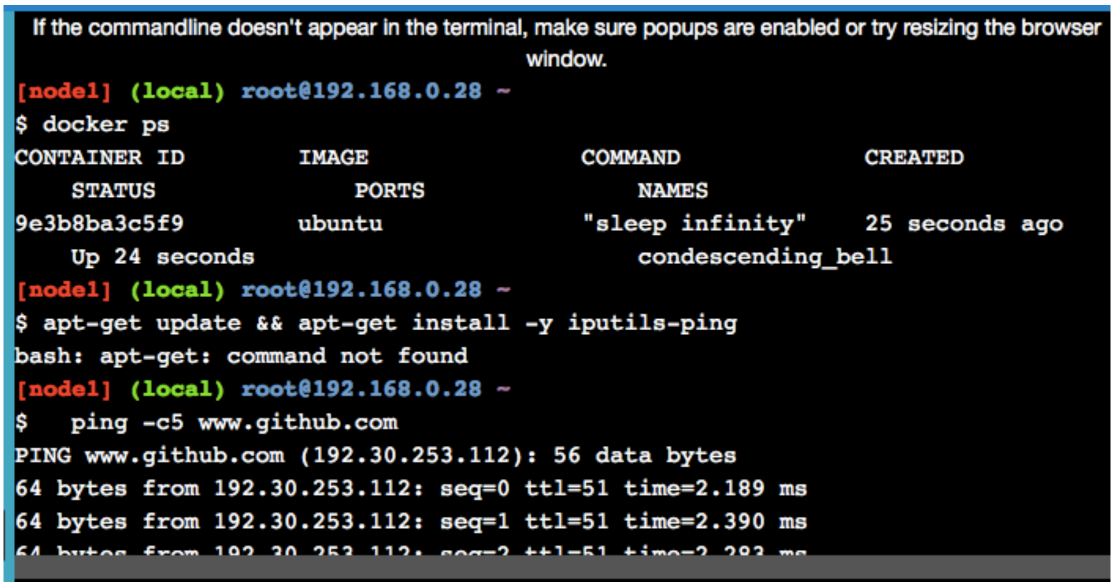
```
[node1] (local) root@192.168.0.23 ~/linux_tweet_app
$  docker image ls -f reference="$DOCKERID/*"
REPOSITORY           TAG                 IMAGE ID            CREATED
      SIZE
abc/linux_tweet_app   2.0                 0204ef17078b        13 seconds ago
      109MB
abc/linux_tweet_app   1.0                 ed298037eda9        4 minutes ago
      109MB
[node1] (local) root@192.168.0.23 ~/linux_tweet_app
$  docker login
Login with your Docker ID to push and pull images from Docker Hub. If you do
n't have a Docker ID, head over to https://hub.docker.com to create one.
Username: sherrysun
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.js
on.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-stor
e

Login Succeeded
```

Video: VMs Versus Containers Deep Dive

Containers in VM can be complimentary as containers can run inside virtual machines in a virtual machine. Everything but D hardware is comprised inside the disk image that makes up the virtual machine. Inside of this image we have the kernel the init system the user space programs and the application themselves. This depending on the size of the user space and the application can range from hundreds of megabytes to tens of gigabytes in a container depending on the type.

Boot time in a VM. We have several startup time of the application itself. The startup times can be divided in two sections. One is the system check section that includes the x86 post the EFI or boot check the kernel boot and the initial startup and then the process run initial part that we can call system check that includes the x86 post.

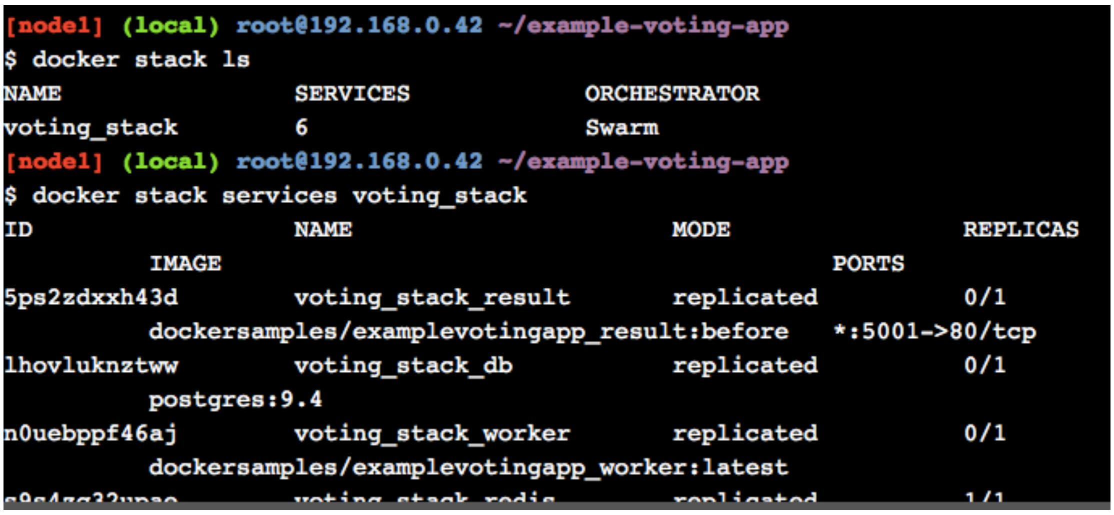
Lab: Docker Networking



```
If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser
window.
[node1] (local) root@192.168.0.28 ~
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
    STATUS              PORTS               NAMES
9e3b8ba3c5f9        ubuntu              "sleep infinity"    25 seconds ago
    Up 24 seconds                           condescending_bell
[node1] (local) root@192.168.0.28 ~
$ apt-get update && apt-get install -y iputils-ping
bash: apt-get: command not found
[node1] (local) root@192.168.0.28 ~
$   ping -c5 www.github.com
PING www.github.com (192.30.253.112): 56 data bytes
64 bytes from 192.30.253.112: seq=0 ttl=51 time=2.189 ms
64 bytes from 192.30.253.112: seq=1 ttl=51 time=2.390 ms
```

In this part, I learnt to list networks, inspect a network. The main part was to bridge networking and overlay networking.

Lab: Swarm Mode Introduction



```
[node1] (local) root@192.168.0.42 ~/example-voting-app
$ docker stack ls
NAME                SERVICES            ORCHESTRATOR
voting_stack        6                   Swarm
[node1] (local) root@192.168.0.42 ~/example-voting-app
$ docker stack services voting_stack
ID                  NAME                          MODE                REPLICAS
          IMAGE                                       PORTS
5ps2zdxxh43d        voting_stack_result           replicated          0/1
          dockersamples/examplevotingapp_result:before   *:5001->80/tcp
lhovluknztww        voting_stack_db               replicated          0/1
          postgres:9.4
n0uebppf46aj        voting_stack_worker           replicated          0/1
          dockersamples/examplevotingapp_worker:latest
```

In this part I learnt swarm mode has some different with compose. Swarm can tell Docker people want to run many Docker engines and want to coordinate operations across all of them. I learnt how to initialize swarm, show swarm members, clone the voting app, deploy the stack, and scaling an application.

Video: Kubernetes vs Swarm

Docker is a container platform with panoply of management options for any type of cloud set-up, providing fine control over applications independent of infrastructure. The tool we are interested in is Docker Swarm, a flexible container storage platform which some consider more straightforward to use than Kubernetes. Kubernetes is a tool allowing you to run multiple containers in parallel.

Video: Kubernetes in 5 Minutes

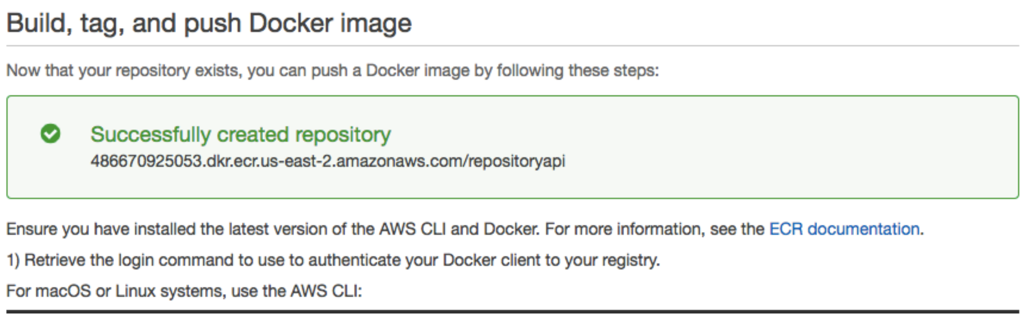
The fundamental premise behind kubernetes is that we can enforce what is called desired state management. A worker is a container host. One thing unique about a worker or the container host in a kubernetes environment is that it does have this cubelet process that runs which is response for communicating with the kubernets cluster services. So the cluster services the workers themselves that is what makes up the quote kubernetes cluster.

Video: Hadoop Intro

People can use more computers instead of one machine performing the job. People can use multiple machines and this is called distributed system.

The challenges of distributed system. Since multiple computers are used in the distributed system there are high chances of system failure. There is also a limit on the bandwidth programing. Complexity is also high because it is difficult to synchronize data and process. Hadoop is the solution to these challenges. Hadoop is a framework that allows for the distributed processing of large data sets across cluster of computers using simple programing models.

AWS Tutorial: Break a Monolith Application into Microservices



Build, tag, and push Docker image

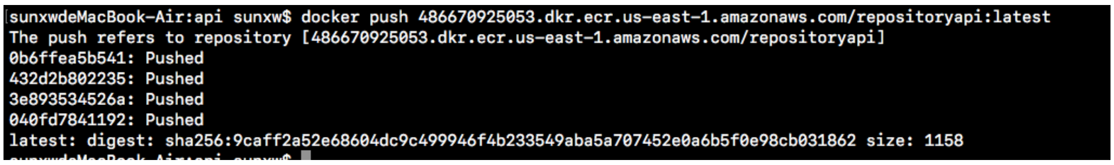
Now that your repository exists, you can push a Docker image by following these steps:

Successfully created repository
486670925053.dkr.ecr.us-east-2.amazonaws.com/repositoryapi

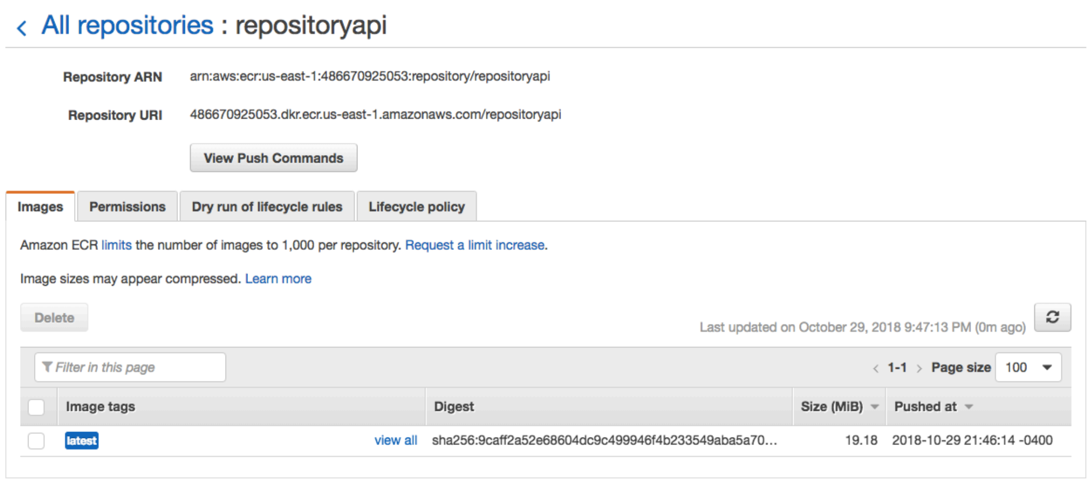
Ensure you have installed the latest version of the AWS CLI and Docker. For more information, see the ECR documentation.

1) Retrieve the login command to use to authenticate your Docker client to your registry.

For macOS or Linux systems, use the AWS CLI:



```
sunxwdeMacBook-Air:api sunxw$ docker push 486670925053.dkr.ecr.us-east-1.amazonaws.com/repositoryapi:latest
The push refers to repository [486670925053.dkr.ecr.us-east-1.amazonaws.com/repositoryapi]
0b6ffea5b541: Pushed
432d2b802235: Pushed
3e893534526a: Pushed
040fd7841192: Pushed
latest: digest: sha256:9caff2a52e68604dc9c499946f4b233549aba5a707452e0a6b5f0e98cb031862 size: 1158
```



< All repositories : repositoryapi

Repository ARN	arn:aws:ecr:us-east-1:486670925053:repository/repositoryapi
Repository URI	486670925053.dkr.ecr.us-east-1.amazonaws.com/repositoryapi

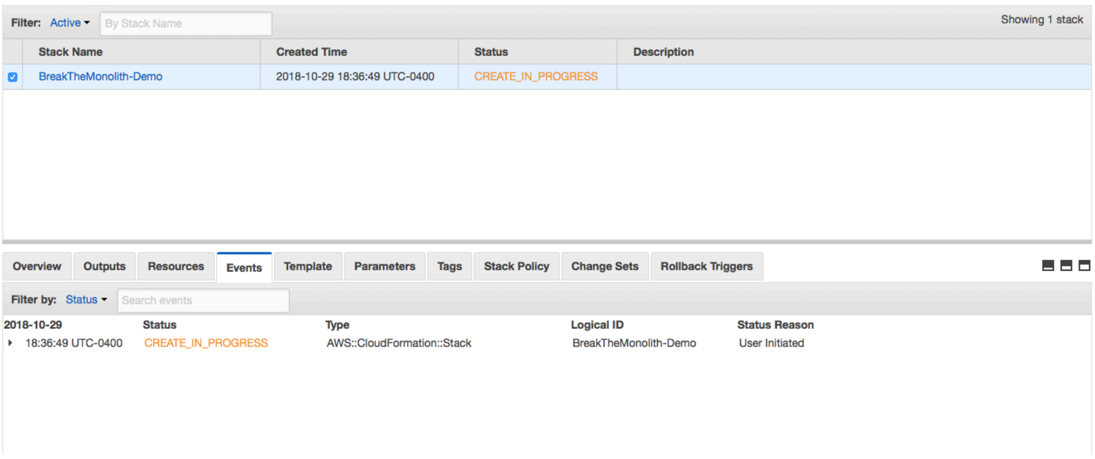
Amazon ECR limits the number of images to 1,000 per repository. Request a limit increase.

Image sizes may appear compressed. Learn more

Last updated on October 29, 2018 9:47:13 PM (0m ago)

Filter in this page — 1-1 — Page size 100

Image tags		Digest	Size (MiB)	Pushed at
latest	view all	sha256:9caff2a52e68604dc9c499946f4b233549aba5a70...	19.18	2018-10-29 21:46:14 -0400

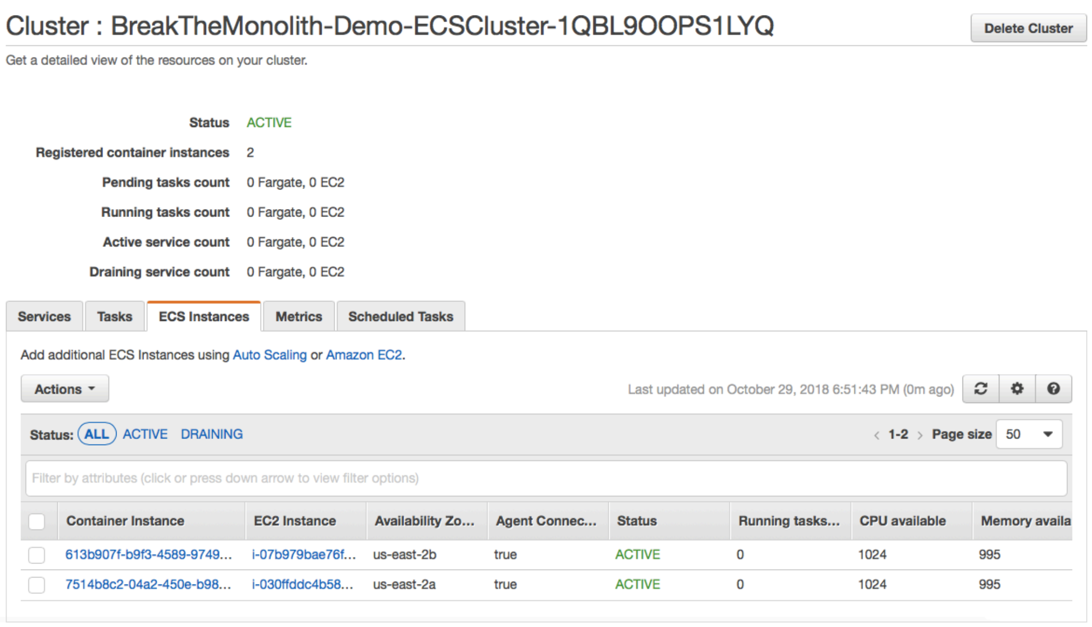


Filter: Active — By Stack Name — Showing 1 stack

Stack Name	Created Time	Status	Description
BreakTheMonolith-Demo	2018-10-29 18:36:49 UTC-0400	CREATE_IN_PROGRESS	

Filter by: Status — Search events

2018-10-29	Status	Type	Logical ID	Status Reason
18:36:49 UTC-0400	CREATE_IN_PROGRESS	AWS::CloudFormation::Stack	BreakTheMonolith-Demo	User Initiated



Cluster : BreakTheMonolith-Demo-ECSCluster-1QBL9OOPS1LYQ

Get a detailed view of the resources on your cluster.

Status	ACTIVE
Registered container instances	2
Pending tasks count	0 Fargate, 0 EC2
Running tasks count	0 Fargate, 0 EC2
Active service count	0 Fargate, 0 EC2
Draining service count	0 Fargate, 0 EC2

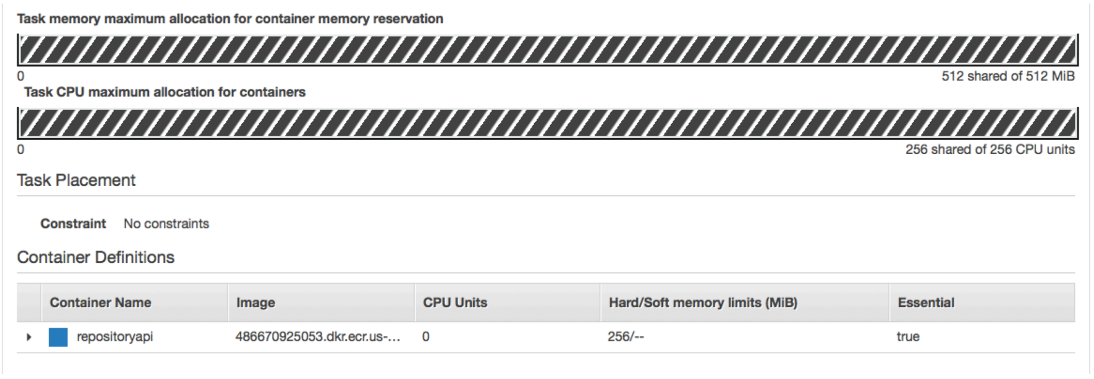
Add additional ECS Instances using Auto Scaling or Amazon EC2.

Last updated on October 29, 2018 6:51:43 PM (0m ago)

Status: ALL ACTIVE DRAINING — 1-2 — Page size 50

Filter by attributes (click or press down arrow to view filter options)

Container Instance	EC2 Instance	Availability Zo...	Agent Connec...	Status	Running tasks...	CPU available	Memory availa
613b907f-b9f3-4589-9749...	i-07b979bae76f...	us-east-2b	true	ACTIVE	0	1024	995
7514b8c2-04a2-450e-b98...	i-030ffddc4b58...	us-east-2a	true	ACTIVE	0	1024	995



Task memory maximum allocation for container memory reservation

0 — 512 shared of 512 MiB

Task CPU maximum allocation for containers

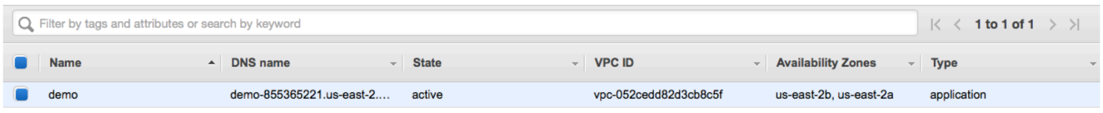
0 — 256 shared of 256 CPU units

Task Placement

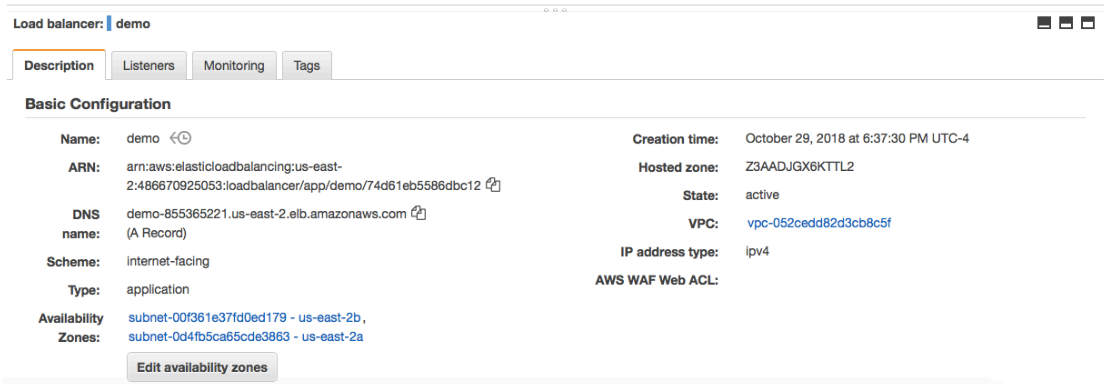
Constraint No constraints

Container Definitions

Container Name	Image	CPU Units	Hard/Soft memory limits (MiB)	Essential
repositoryapi	486670925053.dkr.ecr.us-...	0	256/--	true



Name	DNS name	State	VPC ID	Availability Zones	Type
demo	demo-855365221.us-east-2....	active	vpc-052cedd82d3cb8c5f	us-east-2b, us-east-2a	application



Load balancer: demo

Basic Configuration

Name:	demo
ARN:	arn:aws:elasticloadbalancing:us-east-2:486670925053:loadbalancer/app/demo/74d61eb5586dbc12
DNS name:	demo-855365221.us-east-2.elb.amazonaws.com (A Record)
Scheme:	internet-facing
Type:	application
Availability Zones:	subnet-00f361e37fd0ed179 - us-east-2b, subnet-0d4fb5ca65cde3863 - us-east-2a
Creation time:	October 29, 2018 at 6:37:30 PM UTC-4
Hosted zone:	Z3AADJGX6KTTL2
State:	active
VPC:	vpc-052cedd82d3cb8c5f
IP address type:	ipv4
AWS WAF Web ACL:	

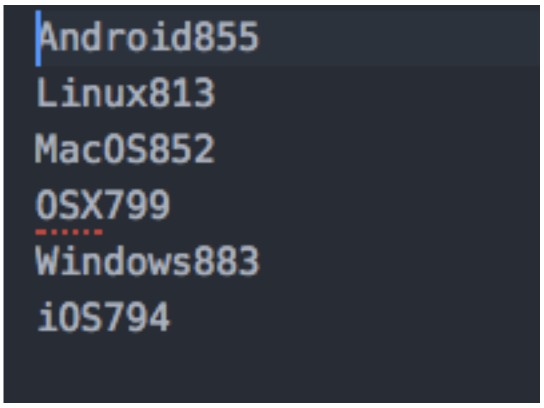
In this part, I mainly learnt how to containerize, deploy and break monolith. Plus, microservices is also an important part. Within a microservices architecture, each application component runs as its own service and communicates with other services via a well-defined API.

However, I also find some errors in the aws document provides by the link. For example, there is an error in the Containerize the Monolisth Step 4. The step of Build the Image and Push the image to ECR is not totally right. And the right step is provided in the link:

https://docs.aws.amazon.com/AmazonECR/latest/userguide/docker-push-ecr-image.html

Thus, this project provided me challenges to trouble shot, finding the way to specific problem I met when I did the project.

QwikLab: Analyze Big Data with Hadoop



```
Android855
Linux813
MacOS852
OSX799
Windows883
iOS794
```