What is Docker Swarm?

- Docker Swarm is a technique in which we join multiple Docker Engines running on different hosts and use them together as a cluster

Docker Swarm Components:

Node: A node is an individual Docker Engine participating in the swarm

Swarm: Is a Mode which consists of multiple Docker hosts which run in a cluster

Service: A service is the definition of the tasks to execute on the swarm nodes.

- ✓**Which Image to use**
- ✓**How many Containers to run**
- ✓**Which commands to execute inside containers.**
- ✓**Which Port, Volumes, Network etc to use**

Task: A task carries a Docker container and the commands to run inside the container.

- ✓Once a task is assigned to a node, it cannot move to another node.
- ✓It can only run on the assigned node or fail.

- **Manager nodes:** Is the machine we communicate with
- Assigns to other nodes
- Manager nodes handle cluster management tasks: maintaining cluster state, scheduling services, and serving swarm mode HTTP API endpoints.
- The managers maintain a consistent state of the swarm and services running on it,

Worker nodes are those where actual containers run

Creating a Docker Swarm:

- The Docker engine runs with swarm mode disabled by default.
- To run Docker in swarm mode, you can either create a new swarm or have the node join an existing swarm.

❑ **Creating a Docker Swarm**

```
$ docker swarm init or docker swarm init --advertise-add <IP>
```

❑ **Adding a node worker to the Swarm Cluster**

```
$ docker swarm join --token <TOKEN> <MANAGER-IP>
```

❑ **Viewing the current Swarm nodes**

```
$ docker node ls
```

❑ **Creating a Service**

```
$ docker service create --name <NAME>  --replicas <#> -p <HP:CP> IMAGE
```

```
$ docker service create -p 80:80 --name web nginx
$ docker service create --replicas 3 -p 80:80 --name web nginx
```

```
$ docker service ls
```

```
$ docker service ps <ServiceName>
```

❑ **Scaling up and Scaling down**

```
$ docker service scale web=3
```

❑ **Details about a service**

```
$ docker service inspect –pretty <servicename>
```

❑ **Deleting a Service**

```
$ docker service rm redis
```

STATUS	NODE PARTICIPATION
BLANK	Worker Node
LEADER	Manager Node (Primary)
REACHABLE	Candidate to become Leader node (Promote)
UNAVAILABLE	Manager node that cannot communicate with other nodes
DRIAN	Do not assign/create Containers

Node Status identifies the node participation in the swarm management.

❑ Managing Nodes

```
$ docker node inspect --pretty <NODE>
```

```
$ docker node promote
```

```
$ docker node demote
```

```
$ docker node update --availability drain <NODE>
```

```
$ docker node update --availability active <NODE>
```

Rolling Updates

- When updating a service, you can define how many containers should be updated at a time and what should happen if the new containers start failing.
- Roll back to the previous version of a service

```
$ docker service update --image <IMAGE> <Service>
```

```
$ docker service update --replicas=5 <Service>
```

```
$ docker service update --rollback <Service>
```

```
$ docker service update --update-failure-action=rollback <Service>
```

Replicated and Global Mode

Replicated:

- A **replicated service** specifies a number of identical tasks you want to run
- Swarm runs services in Replicated mode by default

Global:

- A **global service** runs one replica per node on all the nodes in the swarm, with no pre-specified number of tasks/nodes.
- Every new node added will get the replica created and viceversa

Example : When we have to run some daemon process on each node like **logstash**, **Datadog**, **Prometheus**, **ceph** etc

```
$ docker service create -p 80:80 --name web --mode global nginx
```

Deploy the stack to the swarm

- A stack is a group of interrelated services that share dependencies, and can be orchestrated and scaled together.
- A single stack is capable of defining and coordinating the functionality of an entire application
- Use docker-compose.yml of version '3.0' to define the stack & swarm to run on multiple machines

```
$ docker stack deploy -c docker-compose.yml <STACKNAME>
```

```
$ docker stack ls
```

```
$ docker stack ps <STACKNAME>
```

```
$ docker stack services <STACKNAME>
```

```
$ docker stack rm <STACKNAME>
```

```
---
version: "3.4"
services:
  demoweb:
    image: nginx
    ports:
      - 80:80
    volumes:
      - /tmp:/usr/share/nginx/html
    deploy:
      mode: replicated
      replicas: 2
```

Container Health Check?

- Your container is running. But is it healthy?
- Add a health check to the Dockerfile

```
HEALTHCHECK [OPTIONS] CMD command
Ex: HEALTHCHECK CMD curl --fail http://localhost:80/ || exit 1
```

```
$ docker build -t myapache .
$ docker run --rm --name myapachetest -p 80:80 myapache
```

- See the health status

```
$ docker inspect --format='{{json .State.Health}}' <ContainerID>
```

```
$ docker ps
```

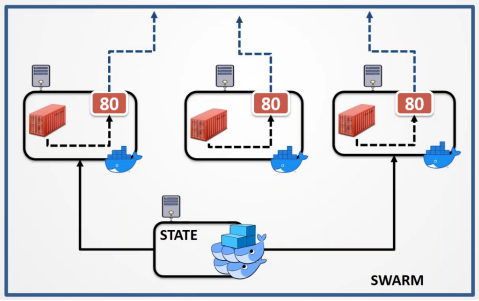
- Configure the health check using a compose file

```
---
version: "3.4"
services:
  demoweb:
    image: nginx
    ports:
      - 80:80
    volumes:
      - /tmp:/usr/share/nginx/html
    deploy:
      mode: replicated
      replicas: 2
    healthcheck:
      test: ls
      interval: 10s
      timeout: 10s
      retries: 3
      start_period: 10s
```

```
$ docker stack deploy -c docker-compose.yml web
```

Service Discovery and Load balancing:

- The swarm manager uses **ingress load balancing** to expose the services you want to make available externally to the swarm
- The ingress controller will do **round robin** by default



Docker config

- When you grant a newly-created or running service access to a config, the config is mounted as a file in the container.
- The location of the mount point within the container defaults to **/<config-name>** in Linux containers. In Windows containers, configs are all mounted into **C:\ProgramData\Docker\configs**
- We can explicitly define the location using the **target** option.

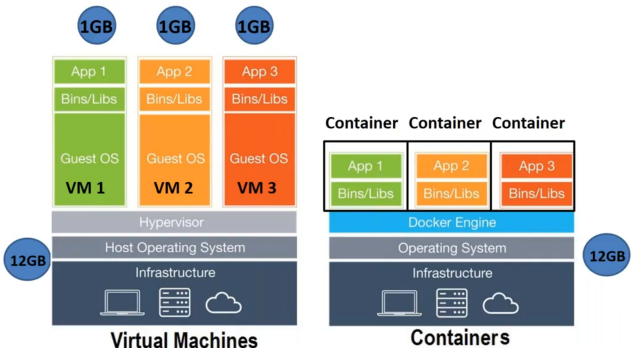
```
$ docker config create <configname> <file>
```

```
$ docker service create --name testservice \
--config src=testconfig,target=/var/www/colors.html \
-p 80:80 httpd
```

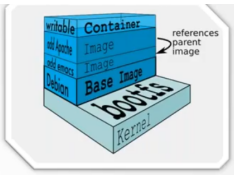
```
---
version: "3.4"
services:
  demoweb:
    image: nginx
    ports:
      - 80:80
    volumes:
      - /tmp:/usr/share/nginx/html
    deploy:
      mode: replicated
      replicas: 2
    configs:
    - source: sample.conf
      target: /tmp/sample.conf
      mode: 0755
configs:
  sample.conf:
    external: true
```

```
$ docker config create sample.conf sample.conf
$ docker stack deploy -c docker-compose.yml web
```

Virtualization vs Containers



Docker images

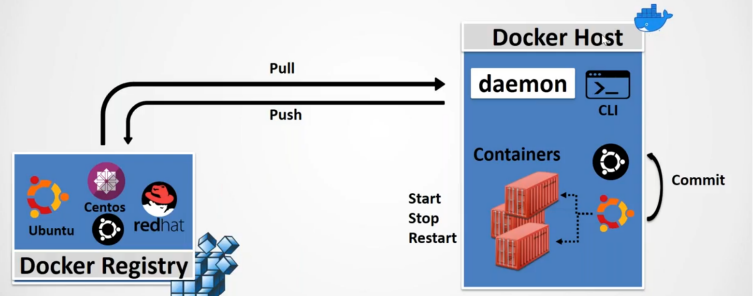


- Docker image is made up of file systems layered over each other
- A Docker base image is nothing but an OS user space minus the kernel
- Base is a boot filesystem, bootfs uses a *Union File System*
- Root filesystem stays in read-only mode
- Basically a tar file
- When a container is launched from an image, Docker mounts a read-write filesystem on top of any layers below

Docker Registry

- Docker Registry is an **Centralized** storage location for Docker Images
- Docker Registry helps in Image management, Image Scanning & Image Signing workflow
- DockerHub is the default online Public registry
- However we can have our Private registry using DockerHub, AWS ECR, Azure Container Registry, Google Container Registry, Jfrog Container Registry

Docker System




```
$ apt update
$ apt install -y docker.io
$ docker info

$ docker run --name <Cname> -it|-d <Image> <FirstCmd>
 - download the Image from the Docker Registry (Dockerhub)
 - create a new container
 - unique ID assigned to the container
 - Container will be started
 - Attach the container to the terminal interactively
 - First command/script/application will be executed

$ docker ps
$ docker ps -a
$ docker start <CID|CName>
$ docker attach <CID|CName>
$ docker stop <CID|CName>
$ docker rm <CID|CName>
$ docker logs -f <CID|CName>
$ docker exec -it <CID|CName> <cmd>

$ docker run -it centos /bin/bash
$ docker run --name test00 -d centos /bin/sh -c "while  true; do echo hello Adam; sleep 5; done"
$ docker exec -it test00 /bin/bash

$ docker run --name <Cname> -it|-d -p <hp>:<cp> -v <hd>:<cd> <Image> <FirstCmd>

$ docker run -it centos /bin/bash
$ docker run --name test00 -d centos /bin/sh -c "while  true; do echo hello Adam;
$ docker exec -it test00 /bin/bash

$ docker run --name mynginx -d -p 80:80 nginx
$ docker run --name myjenkins -d -p 8080:8080 jenkins

$ docker run --name test01 -v /tmp/HOST:/tmp/CONTAINER -it centos /bin/bash
```

Building Images Interactively

- Create a new container and make some changes
  ```
  $ docker run --name adam_the_container -i -t ubuntu
    % apt-get install vim
    % exit
  ```
- Inspect the changes
  ```
  $ docker diff adam_the_container
  ```
- Commit & run your image
  ```
  $ docker commit adam_the_container
  $ docker commit adam_the_container myfirstImage
  $ docker run -it <newImageId>
  ```
- Tagging images
  ```
  $ docker tag <newImageId> myfirstImage
  ```

- Stopping a container
  ```
  $ docker stop adam_new
  ```
- List containers
  ```
  $ docker ps
  ```
  -a = lists all containers
  -q = shows only container ID
- Deleting a container
  ```
  $ docker rm adam_new
  ```
- Inspecting the container's processes
  ```
  $ docker top adam_new
  ```

```
$ docker inspect <containerid>
$ docker inspect --format '{{ .NetworkSettings.IPAddress }}' <containerid>
$ docker inspect --format='{{ .State.Running }}' <containerid>
```

- Container naming
  ```
  $ docker run –name adam_the_container  -it ubuntu //bin/bash
  ```
- Starting a stopped container
  ```
  $ docker start adam_the_container
  ```
- Attaching a container
  ```
  $ docker attach adam_the_container
  ```
- Creating daemonized containers
  ```
  $ docker run --name adam_new -d ubuntu //bin/sh -c "while
  true; do echo hello Adam; sleep 5; done“
  ```
- Fetch logs of a container
  ```
  $ docker logs –f adam_new
  ```

```
Dockerfile
==========
INSTRUCTIONS OS_CMD/OPTIONS
------------ --------------
FROM         <BaseImageName>
RUN          <OS-CMD/SCRIPT>
CMD          <OS-CMD/SCRIPT> # This will get executed in the begining during runtime
             ["executable","arg1","arg2"]
ENTRYPOINT   ["executable","arg1","arg2"]
ENV          <EnvVariable> <Value>
USER         <userId>
WORKDIR      <dir>
COPY         <Host-Src> <Image-Dest>
ADD          <Host-Src> <Image-Dest>
EXPOSE       <Port>

==========
FROM ubuntu:12.04
MAINTAINER ADAM
RUN apt-get update && apt-get install -y apache2 && apt-get clean && rm -rf /var/lib/apt/lists/*
ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2
EXPOSE 80
CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]
======

<Registry>/<repository>/<Image>:<Tag>

$ docker login <Registry>
$ docker tag <localimage>:<tag> <Registry>/<repository>/<Image>:<Tag>
  docker tag myapache:build102 adamtravis/myapache:build102
$ docker push <Registry>/<repository>/<Image>:<Tag>
  docker push adamtravis/myapache:build102
```