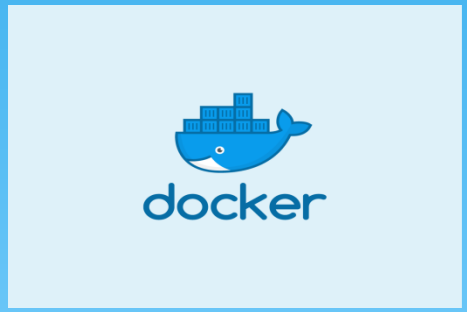
Docker Volumes, Compose & Swarm



Module Overview

* Manage Data in Docker (Volumes and Bind Mounts)
* Docker Compose: Defining & Running Multi Container Applications
* Docker Swarm: Managing a Cluster of Docker Engines

Docker Volumes

* Containers are usually immutable and ephemeral
* Docker Volumes :
  * Volumes have a life cycle that goes beyond the life cycle of containers
  * What happens tp sample.txt when container is deleted ?
    * *$ docker container run --name demo \ alpine /bin/sh -c 'echo "This is a test" > sample.txt'*
  * Makes special location outside of container UFS
  * Volumes connect with none, one or multiple containers at once
  * Not subject to commit, save or export commands
  * By default they have unique ID, but you can assign name

* //Persistent Data: Data Volumes
* *docker pull mysql*
* *docker image inspect mysql*
* *docker container run -d --name mysql2 -e MYSQL_ALLOW_EMPTY_PASSWORD=True mysql*
* *docker volume ls*
* *docker container stop mysql2*

//create a new container with -v option

```
docker container run ––detach ––name
mysql3 –e
MYSQL_ALLOW_EMPTY_PASSWORD=TRUE –v
mysql–db:/var/lib/mysql mysql
```

Docker Bind Mount

* A file or directory on the *host machine* is mounted into a container
* Two locations pointing to the same files
* Skips UFS and host files overwrite any in container
* Bind mount starts with a /

* docker container run -d --name nginx -p 80:80 -v $(pwd):/usr/share/nginx/html nginx

Docker Compose

* Compose is a tool for defining and running multi-container Docker applications
* Declarative way of creating multiple services
* Comprise of two separate but related things:
  * YAML file
  * CLI tool docker-compose

Installing docker-compose

- Run this command to download the current stable release of Docker Compose:

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

- Apply executable permissions to the binary:

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

- Test the installation.

```
$ docker-compose --version
```

Docker Compose Commands

```
# YAML Example:
version: '3'
services: # containers. same as docker run
  servicename:  # a friendly name. this is also the DNS name inside network
    image: # optional, if you use build
    command: # optional, replace the default CMD specified by image
    environment: # same as specified in docker run by -e
    volumes: # same as -v in docker run
  servicename2:
volumes: # optional, same as docker volume create
networks: # optional, same as docker network create
```

CLI Commands

```
docker-compose -up
docker-compose -down
docker-compose -help
```

Docker Container Challenges

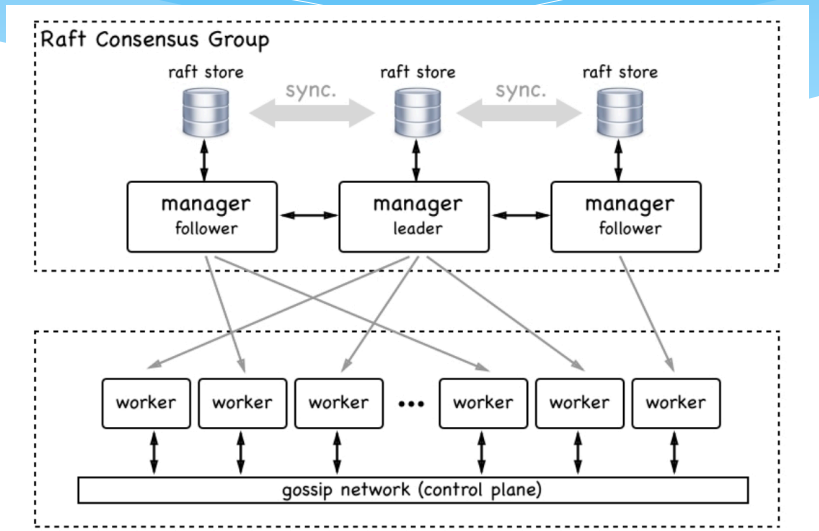
* How can we:
  * automate container lifecycle?
  * easily scale out/in/up/down?
  * ensure our containers are re-created if they fail?
  * replace containers without downtime (blue/green deploy)?
  * control/track where containers get started?
  * create cross-node virtual networks?
  * ensure only trusted servers run our containers?
  * tore secrets, keys, passwords and get them to the right container (and only that container)?

Docker Swarm

* Swarm mode: Built-in orchestration
* Cluster management integrated with Docker Engine
* Decentralized Design
* Declarative Service Model
* Scaling
* Desired state reconciliation
* Multi-host networking

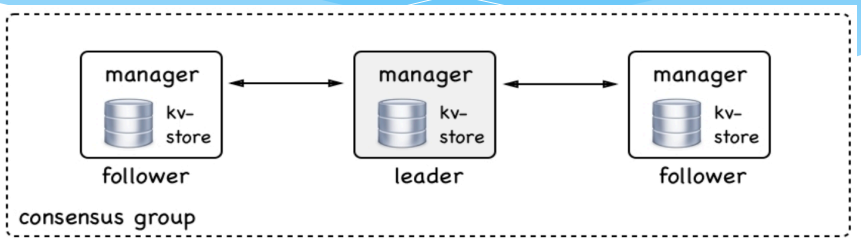
Swarm Architecture

* A raft consensus group of an odd number of manager nodes, and a group of worker nodes that communicate with each other over a gossip network, also called the control plane
* The manager nodes manage the swarm whilst the worker nodes execute the applications deployed into the swarm. Each manager has a complete copy of the full state of the swarm in its local raft store. Managers communicate with each other in a synchronous way and the raft stores are always in sync
* The workers, on the other hand, communicate with each other asynchronously for scalability reasons.



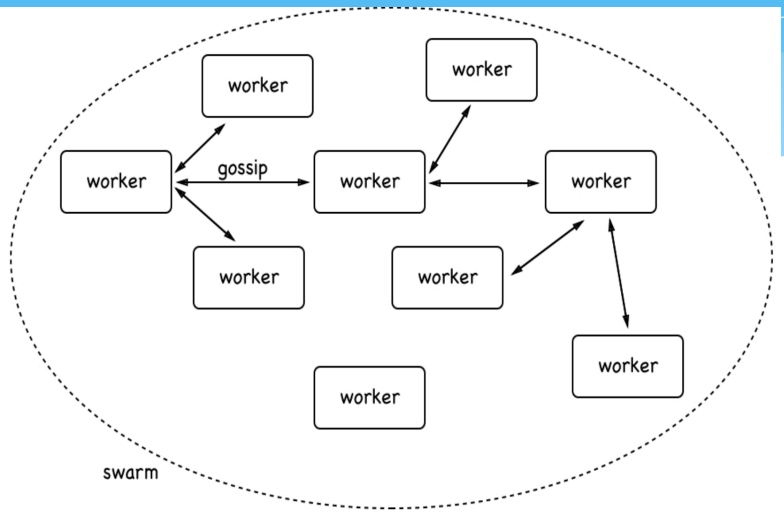
Swarm Managers

* Each Docker Swarm needs to have at least one manager node
* Manager nodes are not only responsible for managing the swarm but also for maintaining the state of the swarm
* All the swarm state is stored in a high performance key-value store (kv-store) on each manager node.



Swarm Workers

* A swarm worker node is meant to host and run containers that contain the actual application services. They are the workhorses of the swarm
* Worker nodes communicate with each other over the so-called control plane. They use the gossip protocol for their communication. This communication is asynchronous, which means that at any given time not all worker nodes must be in perfect sync
* Worker nodes are kind of passive. They never actively do something other than run the workloads that they get assigned by the manager nodes.



Swarm Services & Tasks

* Swarm service is the definition of the tasks to execute on the manager or worker nodes. It is the central structure of swarm system and the primary root of user interaction with the swarm.
* When you create a service, you specify which container image to use and which commands to execute inside running containers.
* In the replicated services model, the swarm manager distributes a specific number of replica tasks among the nodes based upon the scale set in the desired state
* A task carries a docker container and commands to run inside a docker container. It is the atomic scheduling unit of swarm.

Docker Swarm Commands

* Swarm status

docker info

* Swarm init

docker swarm init

* Swarm status

docker node ls

* Create new service

docker service create alpine ping 8.8.8.8

* List services

docker service ls

* Service Details

docker service ps <<$ServiceId>>

* Update Service to create 3 replicas

docker service update <<$ServiceId>> -- replicas 3

* Verify service status

docker service ls

* List Containers

docker container ps