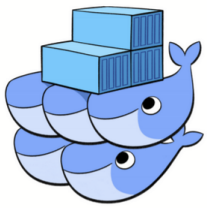
Docker Slides 1:

Why Docker Swarm?



- Existing knowledge
- Streamlined deployment process
- Scope and team size
- Alternatives: Kubernetes (heavier setup, more flexibility)

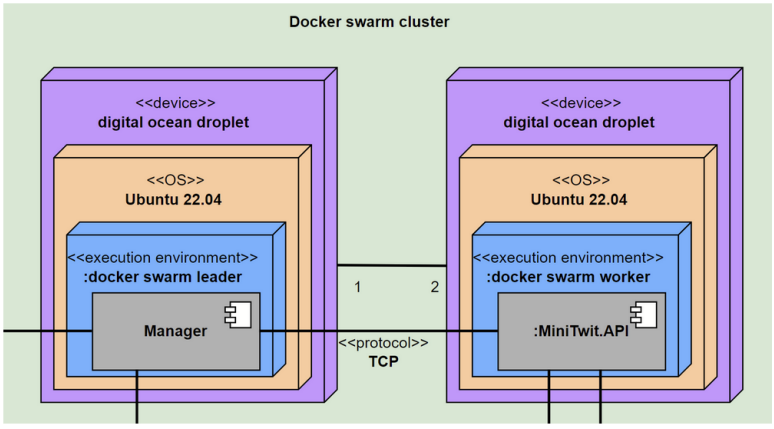
Why did we use docker swarm? So far in the project, we have used Docker for Conterization, and amassed knowledge. By using Docker Swarm, we would not need to change our technology stack that.

Additionally, docker swarm is easy to use (and easier than other alternatives) and has a streamlined deployment process. It is also good for medium sized projects, and team sizes such as ours.

Alternatives could have been Kubernetes, but have a heavier setup, but provide more flexibility and options, but have been warned that it is difficult to use and a nightmare.

How Docker Swarm?

- Docker Swarm Setup
- Responsibilities
- Benefits
  - Load Balancing
  - Service Scaling
  - Fault Tolerance



On the picture, you can see our Docker Swarm Cluster Architecture.

We are running a three-node Swarm Cluster. It consists of a single leader node, acting as a Swarm Manager, and two workers that host our MiniTwit API containers.

By spreading replicas across our multiple DigitalOcean, we avoid a single point of failure - if one VM or container goes down, the others can keep serving traffic.

The manager handles database migrations and orchestration; by scheduling replicas and detecting failures through health checks.

Each worker runs the minitwit-api containers as directed by the manager (which can be seen in docker-compose.yml for swarm) Where they pull images from Docker Hub.

This provides us the following key benefits:

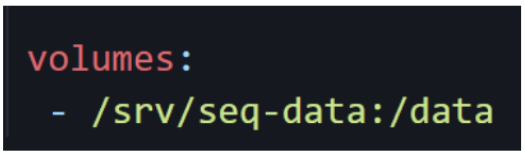
Load Balancing: by evenly distributing incoming requests to healthy replicas/worker nodes

Service Scaling: can easily declare the desired replica/worker count in our compose file

Fault tolerance: if a worker fails, the manager automatically reschedules tasks onto the remaining node, keeping our service available.

Persisting Seq Data

- Dependencies (Depandabot)
- Container Data



Dependabot might bump our Seq docker image to a newer version, meaning that if we pulled that update without persistence in place, we'd wipe our existing logs.

Right now our Seq server is running in a container with no persistent storage; meaning in the event that the container restarts, or the host goes down (i.e. our droplet), we lose our entire log history; preventing us from debugging an outage or past trends.

Further reflecting on this (after handing in), we found out that we can mount Seq's Data folder (containing logs and graphs) to either a Docker volume (as can be seen in the picture), or a DigitalOcean block volume (attached to the droplet), allowing our Seq data to persist even if the droplet dies.