Compose and Swarm

Virtualisation

Otago Polytechnic
Dunedin, New Zealand

Part 1: Docker swarm

We've just been working on one Docker server. However, Docker is a service that listens on network interfaces. New versions have the capability to be joined with other docker servers over a network to form a *swarm*.

Forming a swarm

1. Initialise the swarm on one server. This server becomes a *manager*.
2. Optionally, join additional managers to the swarm[1].
3. Join *workers* to the swarm.
4. On a manager node we may issue commands to control and monitor the swarm.

[1] Best practice is to limit the number of managers to 3-7.

Part 2: Compose and swarm

There are a few options for deploying containerised services on a swarm. We will see how to use a Docker Compose file for this.

To some extent, it just works

Make a `docker-compose.yml` file, then run

```
$ docker-compose up
```

There are some issues, however[2].

[2] You don't actually do it this way, though.

First issue, building

- We saw that you can specify that Docker build an image in a Compose file.
- But swarm needs images to be accessible in a registry.
- So, building in a compose file used for swarm deployment is right out.
- Really, when you're deploying to a production setting, you shouldn't be building on the fly anyway.

Next issue, volumes

- We use volumes to let containers work directly with the host file system.
- But we don't always know what host our containers will run upon.
- If our container creates volumes, use named ones.
- If our containers read from volumes, have other containers that populate them, then use a `volumes_from` directive.

Issue three: dependencies

The `depends_on` directive in a compose file has a different meaning in a swarm context.

- We can create dependencies between containers
  - Explicitly: `depends_on`
  - Implicitly: `volumes_from`
- In either case, Docker interprets this to mean that a container must be deployed on the same node as its dependencies.
- In simple cases this works fine.
- We need to be aware of this when scaling services. When scaling a service, we may need to scale its dependencies.

The tricky case: multiple dependencies

```
version: "3.7"
services:
    svc_a:
        image: image_a
        depends_on:
            - svc_b
            - svc_c
    svc_b:
        image: image_b
    svc_c:
        image: image_c
```

In this case, `svc_a` must be deployed on a host that is already running `svc_b` and `svc_c`. But we aren't guaranteed that `svc_b` and `svc_c` will be placed together on the same host. There is a `constraints` directive available in Compose that helps deal with this.

Replicas

We will see in the lab that we can run multiple replicas of a container on a swarm. We control this by setting a `replicas` value in a `deploy` section.

```
services:
  foo:
    image: tclark/foo
    deploy:
      replicas: 3
```

Final issue: networks

- We've seen that Docker can set up internal networks that allow containers to communicate and find each other by name.
- What happens when the containers are on seperate hosts?
- We can get the same result by creating *overlay networks*.

In compose

```
networks:
    foo:
        driver: overlay
```

Running your services

```
docker stack deploy --compose-file <path to file> stack-name
```

Final caveats

- This stuff is highly version-dependent.
- Documentation is sometimes lacking and what's there can be confusing.
- Stack Overflow does not care about you.

Besides the Docker CLI and Compose File references on `https://docs.docker.com/`, the docs at `https://docs.docker.com/engine/swarm/swarm-tutorial/` are useful for the information we looked at here as well as other topics.