Docker Compose

WRITE ONE AND MULTIPLE MANY

Agenda

Intro / Prep Environments

Day 1: Docker Deep Dive

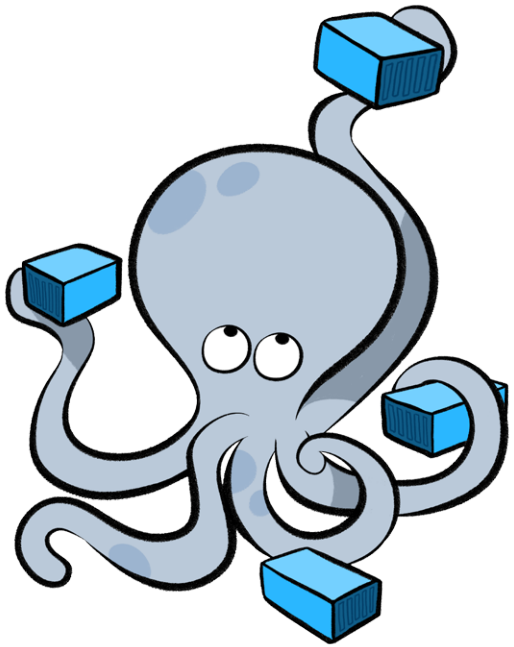
Day 2: Docker Advanced Deep Dive

Day 3: Kubernetes Deep Dive

Day 4: Advanced Kubernetes: Concepts, Management, Middleware

Compose Introduction

- When working with **multiple** containers, it can be difficult to manage starting, along with the configuration, of variables and links.
- To solve this problem, Docker has a tool called Docker Compose, to manage the **orchestration** and launching of containers.



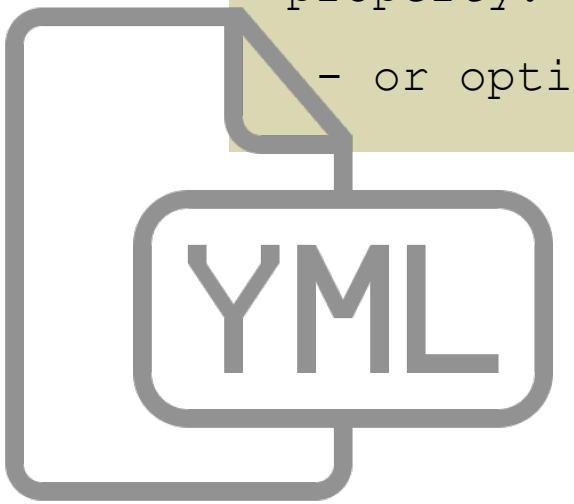
Compose Introduction

- Docker Compose is **based** on a `docker-compose.yml` file.
- This file **defines** all of the containers and settings you need to launch your set of clusters.
- The properties **map** onto how you use the `docker run` commands, however, are now stored in source control and shared along with your code.

YAML Format

- The **format** of YAML (Yet Another Markup Language), looks like this:

```
container_name:
 property: value
  - or options
```



Defining Our First Container

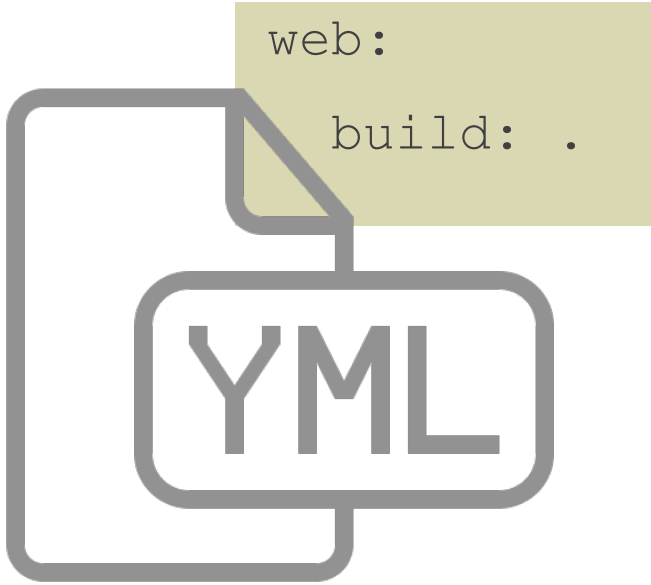
- In this **example**, let's say we have a `Node.js` application, which requires connecting to `Redis`.
- To start, we need to **define** our `docker-compose.yml` file to launch the `Node.js` application.

Current Directory (.)

```
./
Dockerfile
Makerfile
docker-compose.yaml
node_modules/
  redis/
    .npmignore
    README.md
    connection_breaker.js
    index.js
    lib/
    package.json
package.json
server.js
```

Formatting Basic

- Given the **format** shared previously, the file needs to name the container '`web`' and set the build property to the current directory.

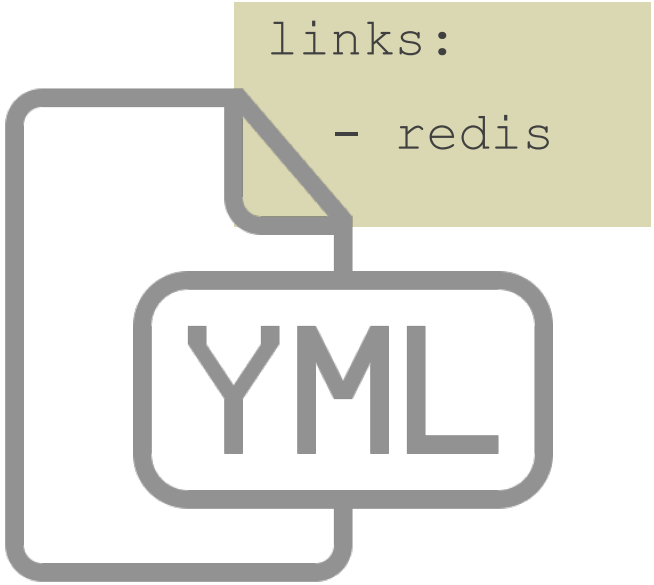


- Copying the following code into our YAML file, will **define** a container called web, which is based on the build of the current directory.

  - NOTE: Remember we have a `Node.js` app in our current directory.

Defining Container Properties

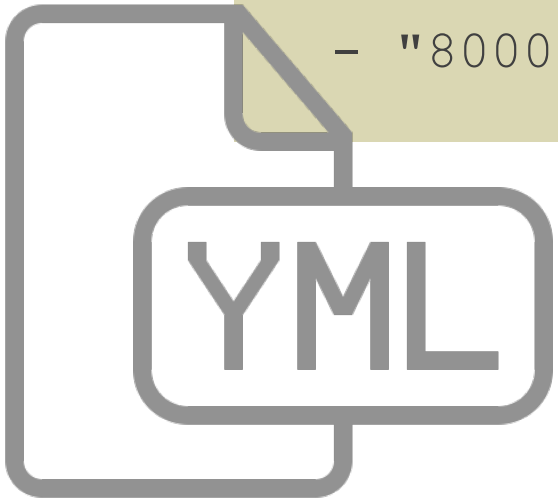
- Docker Compose supports all of the properties which can be defined using `docker run`.
- To link two containers together to specify a `links` property and list required connections.
- For example, the following would link to the `redis` source container defined in the same file and assign the same name to the alias.



Defining Container Properties

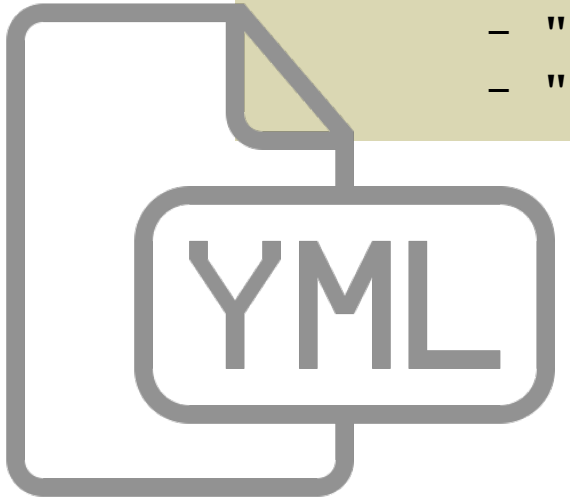
- The same format is used for other properties, such as **ports**:

```
ports:
  - "3000”
  - “3001”
  - "8000"
```



First Container Overview

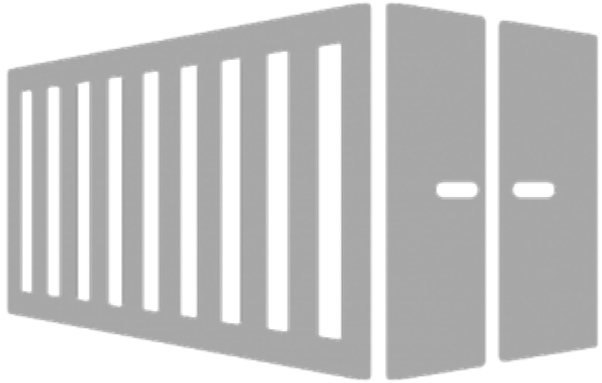
```
web:
      build: .

      links:
        - redis

      ports:
        - "3000"
        - "3001"
        - "8000"
```



- So, after all that, our yaml file would look something like this.

Defining a Second Container

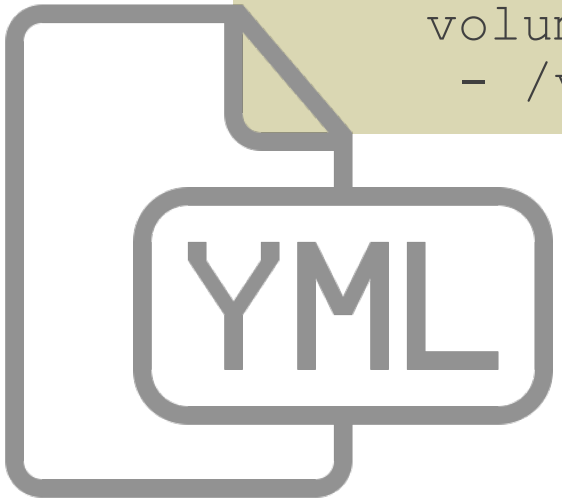
- In the previous step, we used the `Dockerfile` in the current directory (.) as the **base** for our container.
- In this step, we want to use an existing **image** from Docker Hub as a second container.
- To define the second container you simply use the same **format** as before on a new line.
- The YAML format is flexible enough to define multiple **containers** within the same file.



Second Container Properties

- Defining the **second** container with the name `redis`, which will use the image `redis`. Following the YAML format, the container details would be:

```
redis:
      image: redis:alpine
      volumes:
       - /var/redis/data:/data
```



Starting Compose Up

- With the `docker-compose.yml` file we created in place, you can launch all the applications with a single command of `up`, like this:

```
docker compose up
```

- If you wanted to bring up a single container, then we could use `up <name>`.

```
docker compose up <name>
```

Docker Compose Management

- Docker Compose starts containers, but it also provides a commands to **manage** all the containers.
- To see the details of the **launched** containers we could use:

```
docker-compose ps
```

- To access all the **logs** via a single snapshot stream we could use:

```
docker-compose logs
```

- **Other** commands follow the same pattern, and as always can be seen by typing the command itself: `docker-compose`.

Scaling With Compose

- Let’s talk about how Compose can also be used to **scale** up the number of running containers.
- The scale option allows you to specify the service and then the **number** of instances you want.
- If the number is **greater** than the instances already running, it will launch additional containers.
- And if the number is **less**, it will stop the unrequired containers.

Scaling With Compose

- To **scale** the number of web containers we're running we'd use this command:

```
docker-compose scale web=3
```

- And we could scale it back **down** using:

```
docker-compose scale web=1
```

Stop and Remove

- As when we launch an application with start, to **stop** a set of containers we can use the command:

```
docker-compose stop
```

- And to **remove** all the containers we can use the command:

```
docker-compose rm
```

Result Summary

- In this lesson we explored how you can use Docker **Compose** to manage the orchestration of multi-container applications.

Lab

End of Chapter