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

This environment has the latest Docker Compose installed. Visit https://docs.docker.com/compose/install/ for instructions on how to install this into your local environment.

Step 1 - Defining First Container
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.

The format of the file is based on YAML (Yet Another Markup Language).

container_name:
  property: value
    - or options

Task: Defining First Container
In this scenario, 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.

Given the format above, the file needs to name the container 'web' and set the build property to the current directory. We'll cover the other properties in future steps.

Copy the following yaml into the editor. This will define a container called web, which is based on the build of the current directory.

```
web:
  build: .
```

Step 2 - Defining Settings
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.

```
  links:
    - redis
```

The same format is used for other properties such as ports

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

Additional documentation on the options can be found at https://docs.docker.com/compose/compose-file/

Task
Update our web container to expose the port 3001 and create a link to our Redis container.

Step 3 - Defining 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 find 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.

Task: Define Second Container
Define the second container with the name redis which uses the image redis. Following the YAML format, the container details would be:

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

- /var/redis/data:/data: This line defines a volume mount, where the data stored inside the Redis container at /data will be persisted on the host machine at /var/redis/data. This ensures that the Redis data is retained even if the container is stopped or restarted.

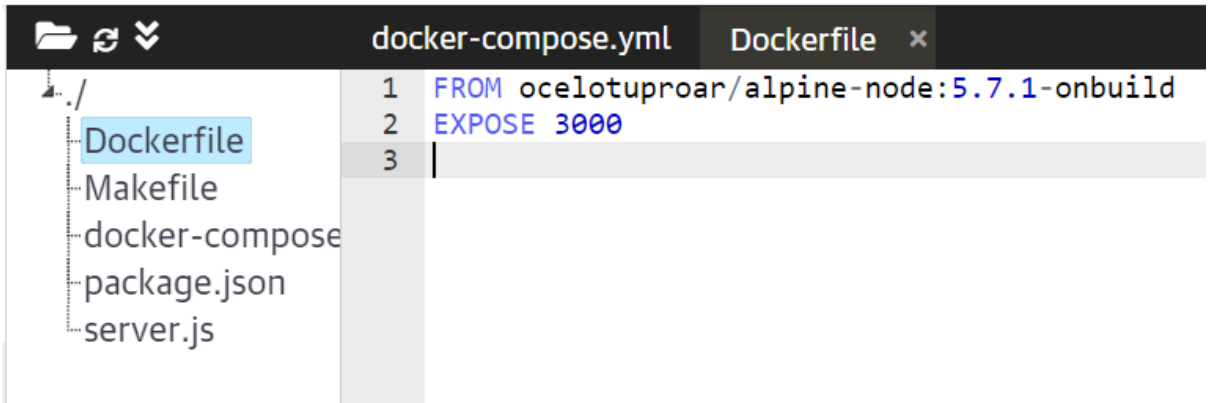
Step 4 - Docker Up
With the created docker-compose.yml file in place, you can launch all the applications with a single command of up. If you wanted to bring up a single container, then you can use up <name>.

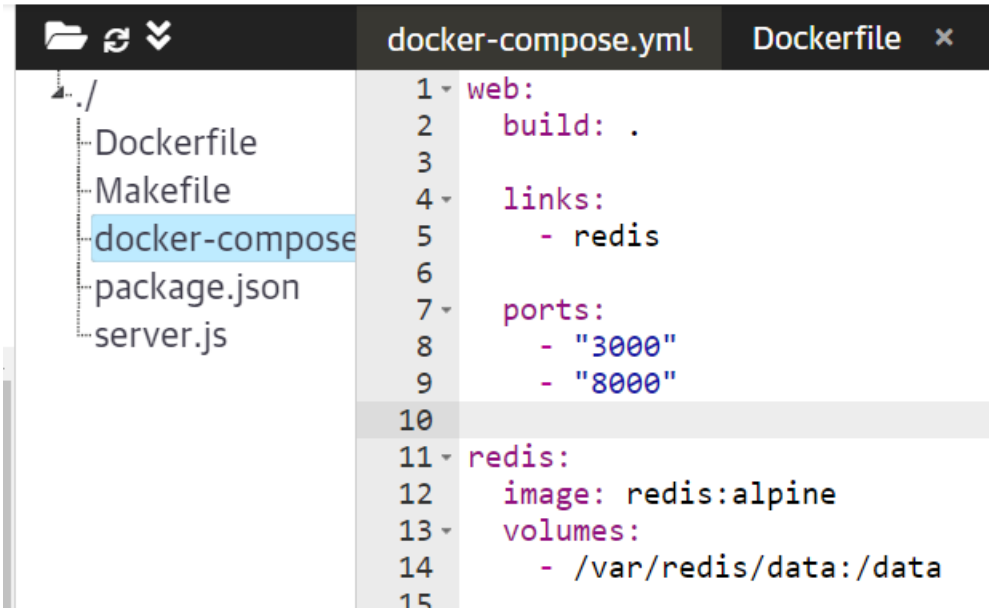
The -d argument states to run the containers in the background, similar to when used with docker run.

Task
Launch your application using
docker-compose up -d

```
$ docker-compose up -d
Building web
Step 1/2 : FROM ocelotuproar/alpine-node:5.7.1-onbuild
# Executing 3 build triggers
 ---> Running in cbfe3b444ea6
scrapbook-redis-node-docker-example@0.0.0 /usr/src/app
`-- redis@0.12.1

Removing intermediate container cbfe3b444ea6
 ---> 759920ddbb94
Step 2/2 : EXPOSE 3000
 ---> Running in 25c23305a04c
Removing intermediate container 25c23305a04c
 ---> 8678a0fbe007
```





Step 5 - Docker Management
Not only can Docker Compose manage starting containers but it also provides a way manage all the containers using a single command.

For example, to see the details of the launched containers you can use docker-compose ps

To access all the logs via a single stream you use docker-compose logs

Other commands follow the same pattern. Discover them by typing docker-compose.

```
$ docker-compose ps
      Name                     Command               State                        Ports
-------------------------------------------------------------------------------------------------------
tutorial_redis_1   docker-entrypoint.sh redis ...   Up      6379/tcp
tutorial_web_1     npm start                        Up      0.0.0.0:32769->3000/tcp, 0.0.0.0:32768->8000/tcp
$ docker-compose logs
Attaching to tutorial_web_1, tutorial_redis_1
redis_1  | 1:C 29 May 2023 06:23:44.372 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
redis_1  | 1:C 29 May 2023 06:23:44.377 # Redis version=7.0.11, bits=64, commit=00000000, modified=0, pid=1, ju
redis_1  | 1:C 29 May 2023 06:23:44.377 # Warning: no config file specified, using the default config. In order
nf
redis_1  | 1:M 29 May 2023 06:23:44.377 * monotonic clock: POSIX clock_gettime
redis_1  | 1:M 29 May 2023 06:23:44.378 * Running mode=standalone, port=6379.
redis_1  | 1:M 29 May 2023 06:23:44.378 # WARNING: The TCP backlog setting of 511 cannot be enforced because /p
redis_1  | 1:M 29 May 2023 06:23:44.378 # Server initialized
redis_1  | 1:M 29 May 2023 06:23:44.378 # WARNING Memory overcommit must be enabled! Without it, a background s
g disabled, it can can also cause failures without low memory condition, see https://github.com/jemalloc/jemall
= 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take ef
```

Step 6 - Docker Scale
As Docker Compose understands how to launch your application containers, it can also be used to scale the number of containers running.

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 then, it will launch additional containers. If the number is less, then it will stop the unrequired containers.

Task
Scale the number of web containers you're running using the command docker-compose scale web=3

You can scale it back down using docker-compose scale web=1

```
$ docker-compose scale web=3
WARNING: The scale command is deprecated. Use t
Desired container number already achieved
$ docker-compose scale web=1
WARNING: The scale command is deprecated. Use t
Stopping and removing tutorial_web_2 ... done
Stopping and removing tutorial_web_3 ... done
```

Step 7 - Docker Stop
As when we launched the application, to stop a set of containers you can use the command
docker-compose stop

To remove all the containers use the command
docker-compose rm

```
$ docker-compose stop
Stopping tutorial_web_1   ... done
Stopping tutorial_redis_1 ... done
$ docker-compose rm
Going to remove tutorial_web_1, tutorial_redis_1
Are you sure? [yN] y
Removing tutorial_web_1   ... done
Removing tutorial_redis_1 ... done
```

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