QA Consulting

# Docker Compose CLI Commands

TRAINING MATERIALS - MODULE HANDOUT

**Contacts**

**robert.crutchley@qa.com**

*team.qac.all.trainers@qa.com*

*www.consulting.qa.com*

# Contents

## Overview

The CLI (Command Line Interface) for Docker Compose is how we are going to be able to run our configurations, manage images and containers. Docker Compose commands always start with docker-compose, use the --help to see a high level view of the available commands you can execute. This handout aims to help you understand the most commonly used commands for the Compose CLI.

```
docker-compose --help
```

## UP

### Usage

This command if for creating all of your services  defined in your configurations.

```
docker-compose up [OPTIONS]
```

```
docker-compose up
```

### Detaching Containers

When you are running this command you may wish to use detached mode with -d or --detach, which run the containers in the background, otherwise you won't be able to run any other commands.

```
docker-compose up -d
```

### Scaling

Scaling is a powerful option for the "up" command that allows you to create replicas of the containers that are running by specifying the service name and the amount containers that you would like. Its advised that you used the detached mode for this, otherwise the command with just 'hang'.

```
docker-compose up [OPTIONS] --scale [SERVICE]=[SCALE_COUNT]
```

```
docker-compose up -d --scale web_server=5
```

```
docker-compose up -d --scale web_server=5 backend_server=5
```

## DOWN

### Usage

For removing  all of the services that you have defined in your configurations you can use the down command.

```
docker-compose down [OPTIONS]
```

```
docker-compose down
```

### Removing Service Images

Conveniently, Docker Compose will build the images for our services if they don't exist, however when you take down the containers you might not want them to be there afterwards. The --rmi option takes two types;

- **all**
  Remove all images that have been used by any service.
- **local**
  Remove only the images that have been built by Docker Compose, images like nginx won't be removed with this type for example.

```
docker-compose down --rmi [TYPE]
```

```
docker-compose down --rmi all
```

```
docker-compose down --rmi local
```

## BUILD

### Usage

The build command can be used for just building the Docker images separately, without running any containers. It only makes sense to use this command if you are creating your own Docker images as well, as opposed to just downloading them.

```
docker-compose build [OPTIONS]
```

```
docker-compose build
```

### Building in Parallel

To save build time you can build all the images for your defined services at the same time, instead of one at a time. When building in parallel, make sure that the image builds do not depend on each other at any point.

```
docker-compose build --parallel
```

## PS

### Usage

All the running containers for your configurations can be viewed with the ps command. The advantage of using ps with Docker Compose over just the Docker ps command is that you can only see the containers specific to the services that you defined in your configurations.

```
docker-compose ps
```

## EXEC

### Usage

Similar to the docker exec command, exec allows us to running commands on the containers that are running.

```
docker-compose exec [OPTIONS] [SERVICE] [COMMAND]
```

```
docker-compose exec nginx bash
```

### Connect to a Specific Container

With Docker Compose we may have several containers that are running for a service. By default exec will connect to the first container, if we want to connect to another one then we can provide the index as an option.

```
docker-compose exec [OPTIONS] [SERVICE] [COMMAND]
```

```
docker-compose exec --index 2 nginx bash
```

## LOGS

### Usage

Display the log outputs from all of the services that are running.

```
docker-compose logs [OPTIONS] [SERVICE]
```

```
docker-compose logs
```

### Follow the Logs

Having to run the logs command over and over again to see the most up to date logs would be annoying os there is an option that allows you see the logs update in realtime.

```
docker-compose logs -f
```

### Logs for One or More Services

If there are many services that are running the we might want to only see the logs for a couple of services or even just the one service.

```
docker-compose logs my_service_1 my_service_2
```

## Tasks

This exercise will reference the OLS case study, addressing some of the issues the company is facing by utilizing the Docker Compose CLI. Compose and Docker configurations will be provided in the exercise for the project.

### Download and Configure the Project

Before we can get started we need to download the OSL project, we can access it from the public GitHub repository (https://github.com/original-lizard-studios/website).

```
cd ~; git clone https://github.com/original-lizard-studios/website; cd
website
```

At the root of the project we need to make a Dockerfile.

~/website/Dockerfile

```
FROM node:10 as client-build
WORKDIR /build
COPY client .
RUN npm install
RUN npm run build
FROM maven as server-build
WORKDIR /build
COPY server .
COPY --from=client-build /build/build src/main/resources/static
RUN mvn clean package
FROM java:8
WORKDIR /opt/app
COPY --from=server-build /build/target/app-0.0.1-SNAPSHOT.jar app.jar
EXPOSE 8080
ENTRYPOINT ["/usr/bin/java", "-jar", "app.jar"]
```

Also at the root of the project we need to make a docker-compose.yaml file.

~/website/docker-compose.yaml

```
version: '3.7'
services:
 app:
   image: app:${APP_VERSION}
```

```
    build: .
    ports:
    - target: 8080
      protocol: tcp
  mongo:
    image: mongo:latest
    container_name: mongo
```

The Compose file we have created relies on a host environment variable, which is the version of the application, we will set this now.

```
export APP_VERSION=v1
```

### Deploying an Environment Efficiently

OLS's concern is that it will take a long time to get a new environment setup for a developer and that deployments are tedious and slow. Just by configuring the project for Docker and Compose, we have now addressed these issues. To get a complete environment running, no software or database need to be setup because it's all handle by Docker and Compose. Go ahead and deploy the application with a single Compose command.

```
docker-compose up -d
```

The application will accessible from http://localhost:[PORT], use docker ps to see which port has been published.

### Updating a Service

Currently the version deployed will be v1, if we change the environment variable and deploy again the new version will be running.

```
# set the environment variable
export APP_VERSION=v2
# build the new version first (optional)
docker-compose build --no-cache app
# redeploy the app service
docker-compose up -d app
# view the container running as a different version
docker ps
```

Notice we are building the application here before running it, this ties in with OLS's issues with build times. Without looking into how to change the application architecture, we can reduce the amount of time it takes to get the application running by building separately. For instance if there was a deployment scheduled towards the end of the day, the container image could be built well in advance of that, then it takes a fraction of a second to simply run the container images.

### Rolling Back Versions Effectively

When deployments of a new version go wrong for OLS, they often suffer from downtime because there isn’t an effective roll back strategy in place. We would be able to easily roll back versions the same way we updated to them. Let’s rollback to v1 again to see how easy it is.

```
# set the environment variable
export APP_VERSION=v1
# redeploy the app service
docker-compose up -d app
# view the container running as a different version
docker ps
```

### Service Scaling & HIgh Availability

To help with down time issues on OLS, if there was a load balancer in place, we could scale the application to allow for high availability. If one of the applications fail then there will be another one or more still running.

```
docker-compose up -d --scale app=3
```

### Log Aggregation For Multiple Services

Scaling is all well and good, however when there are 3 times as many applications running, there are 3 times as many places we need to go to looks at logs. With Compose, we can view them all in one place.

```
docker-compose logs app
```

### Delete the Environment

We can start from scratch by deleting all containers and images used if need be, giving us a clean slate.

```
docker-compose down --rmi all
```