FREE eBook

LEARNING docker-compose

Free unaffiliated eBook created from
Stack Overflow contributors.

#docker-compose

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: docker-compose

It is an unofficial and free docker-compose ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official docker-compose.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with docker-compose

Remarks

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a Compose file to configure your application's services. Then, using a single command, you create and start all the services from your configuration. To learn more about all the features of Compose see the list of features.

Using Compose is basically a three-step process.

1. Define your app's environment with a `Dockerfile` so it can be reproduced anywhere.
2. Define the services that make up your app in `docker-compose.yml` so they can be run together in an isolated environment.
3. Lastly, run `docker-compose up` and Compose will start and run your entire app.

Examples

Installation

If you are running Docker on OS X or Windows, docker-compose should be included in your Docker for Windows or Docker Toolbox installation.

On Linux you can get the latest binaries straight from the GitHub release page: https://github.com/docker/compose/releases

You can install the specific release with the following commands:

```
curl -L https://github.com/docker/compose/releases/download/1.7.1/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
```

For more info please refer to documentation page

Create a simple application

This example comes from the official document. Suppose you have a python application using redis as backend. After writing `Dockerfile`, create a `docker-compose.yml` file like this:

```
version: '2'
services:
  web:
    build: .
    ports:
     - "5000:5000"
```

```
    volumes:
     - .:/code
    depends_on:
     - redis
  redis:
    image: redis
```

Then run `docker-compose up` will setup the entire application includes: python app and redis.

- `version: '2'` is the version of the docker-compose file syntax
- `services:` is a section that describes the services to run
- `web:` and `redis:` are the names of the services to start, their contents describe how docker should start containers for those services
- `depends_on` implies a dependency of web to redis and therefor docker-compose first starts the `redis` container and then the web container. Nevertheless does `docker-compose` not wait until the `redis` container is ready before starting the `web` container. To achieve this you have to use a script that delays the start of the application server or whatever until the `redis` container can perform requests.

A volumes and networks section can be added as well. Using the volumes section allows for disconnected volume that can live independently of the docker compose services section. The networks section has a similar result.

The redis section of services would have to adjusted like so:

```
redis:
  image: redis
  volumes:
    - redis-data:/code
  networks:
    -back-tier
```

Next, add the following sections to the bottom of the docker compose version 2 file.

```
volumes:
  # Named volume
  redis-data:
    driver: local
networks:
  back-tier:
    driver: bridge
```

`redis-data` provides an accessible label from the services section. `driver:local` sets the volume to the local file system.

`back-tier` sets the networks section label to be accessible in the services section as bridged.

Run command in docker-compose service

```
docker-compose run service-name command
```

If, for example, you wanted to run `rake db:create` in your `web` service, you'd use the following command:

```
docker-compose run web rake db:create
```

Install Docker Compose

1. Install Docker Engine.

   If you get a `Permission denied` error, Run `sudo -i` before the two commands below, then exit.

2. Pull Docker Compose to `/usr/local/bin/docker-compose`.

```
curl -L https://github.com/docker/compose/releases/download/1.7.1/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose
```

   You can change version `1.7.1` to match your desired version. Try get version from https://github.com/docker/compose/releases

3. Apply executable permissions to the binary.

```
chmod +x /usr/local/bin/docker-compose
```

4. Test the installation.

```
docker-compose --version
```

   Expected docker-compose version 1.7.1, build 0a9ab35

Docker Compose hello world

A very basic `docker-compose.yml` looks like this:

```
version: '2'
services:
  hello_world:
    image: ubuntu
    command: [/bin/echo, 'Hello world']
```

This file is making it so that there's a `hello_world` service, that's initialized from the `ubuntu:latest` image and that, when it's run, it just runs `echo 'Hello world'`

If you're on the `folder` directory (and it contains this `docker-compose.yml` file), you can do `docker-compose up` and you should see

```
Creating folder_hello_world_1
Attaching to folder_hello_world_1
hello_world_1 | Hello world
folder_hello_world_1 exited with code 0
```

This created the container from the ubuntu image, and ran the command that was specified on the `docker-compose.yml`

`Docker-Compose` uses the folder name as the project name to prefix containers and networks. To set another project name, you can either call `docker-compose --project-name NAME {up|down|...}` or you suppy a file called `.env` next to your `docker-compose.yml` and write `COMPOSE_PROJECT_NAME=name` in it. Better avoid long project names with hyphens (-) because docker compose bahaves strange with this kind of names.

Note: docker-compose allows you to run multiple docker containers on a single host. If you want to run multiple containers on more than one node, please refer to solution such as swarm / kubernetes.

Ruby on Rails with docker-compose

If you want to use docker for rails app, and use database, you need to know that all the data in the docker container will be destroyed (unless you configure the container specifically for keeping data) Sometimes, you need to create a docker container with an application and attach it to an old container with a database.

As an example of rails application, I used a simple app. You can create it from command:

```
rails new compose-app --database=postgresql
```

Of course, you need to install rails, ruby, etc. beforehand.

Then, create Dockerfile in your project, and set this data to it:

```
FROM ruby:2.3.1
RUN apt-get update -qq && apt-get install -y build-essential libpq-dev nodejs
RUN mkdir /compose-app
WORKDIR /compose-app
ADD Gemfile /compose-app/Gemfile
ADD Gemfile.lock /compose-app/Gemfile.lock
RUN bundle install
ADD . /compose-app
```

Next step - create docker-compose.yml with the data:

```
version: '2'
services:
  db:
    image: postgres
  web:
    build: .
    command: bundle exec rails s -e development -p 80 -b '0.0.0.0'
    volumes:
      - .:/compose-app
    ports:
      - "80:80"
    depends_on:
      - db
```

You can replace 80 port (-p 80) with another.

Develop section of database.yml config must be changed to:

```
development: &default
  adapter: postgresql
  encoding: unicode
  database: postgres
  pool: 5
  username: postgres
  password:
  host: db
```

Now you can build images from command:

```
docker-compose build
```

(Run this in project directory)

And start all from:

```
docker-compose up
```

If everything is done correctly, you will be able to see logs from rails in the console.

Close console. It will be working.

If you want to delete only the container with the rails application without the database, you need to run then in project directory:

```
docker-compose stop web
docker-compose build web
docker-compose up -d --no-deps web
```

New container with rails app will be created and launched.

Read Getting started with docker-compose online: https://riptutorial.com/docker-compose/topic/1266/getting-started-with-docker-compose

Chapter 2: Docker-compose multi-container example with default network

Remarks

By default Compose sets up a single network for your app. Each container for a service joins the default network and is both reachable by other containers on that network, and discoverable by them at a hostname identical to the container name.

Links allow you to define extra aliases by which a service is reachable from another service. They are not required to enable services to communicate – by default, any service can reach any other service at that service's name.

https://docs.docker.com/compose/networking/

Examples

How to create a basic LAMP environment with default networking

docker-compose.yml

```
version: '2'
services:
  php:
     image: phpmyadmin/phpmyadmin
     links:
       - mysql:db
     depends_on:
       - mysql

  mysql:
    image: k0st/alpine-mariadb
    volumes:
      - ./data/mysql:/var/lib/mysql
    environment:
      - MYSQL_DATABASE=mydb
      - MYSQL_USER=myuser
      - MYSQL_PASSWORD=mypass

  nginx:
    image: nginx:stable-alpine
    ports:
      - "81:80"
    volumes:
      - ./nginx/log:/var/log/nginx
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
    depends_on:
      - php
```

nginx/nginx.conf

```
worker_processes  1;
events {
  worker_connections  1024;
}
http {
  sendfile  off;
  server {
    listen 80;

    location / {
      proxy_pass  http://php;
      proxy_set_header Host $host;
      proxy_redirect     off;
    }
  }
}
```

Note the nginx config is simplified but above should work for testing — basically all it's doing is proxying the php app. Maps to port 81 to avoid conflicts on the host - adjust as needed.

Regarding linking, you can see that if you run: `docker-compose exec mysql ping -c2 nginx` to ping *from* the mysql container *to* the nginx container, you will succeed even though there are *no links specified between these containers*. Docker Compose will maintain those links in the default network for you.

Read Docker-compose multi-container example with default network online: https://riptutorial.com/docker-compose/topic/3226/docker-compose-multi-container-example-with-default-network

Chapter 3: Environment variables external file

Introduction

There a number of ways to include environment variables into the docker application. Here are some examples:

Examples

Via External File

docker-composer.yml

```
  web:
    ...
    env_file:
     - ./filename
```

filename

```
variable=value
```

within the docker-compose itself

```
app:
  ...
  environment:
    - var=value
```

Read Environment variables external file online: https://riptutorial.com/docker-compose/topic/10598/environment-variables-external-file

Credits

S. No	Chapters	Contributors
1	Getting started with docker-compose	cizixs, Community, Flamine, g3rv4, granthbr, ItayB, katopz, Kevin Wittek, Nadya Knyazeva, Nate Todd, Ohmen, Thomasleveil, Wolfgang
2	Docker-compose multi-container example with default network	ldg
3	Environment variables external file	Simon Tsang