Dockerize Node.js Express and MySQL example – Docker Compose

Last modified: September 2, 2021 (https://www.bezkoder.com/docker-compose-nodejs-mysql/) bezkoder (https://www.bezkoder.com/author/bezkoder/) Deployment (https://www.bezkoder.com/category/deployment/), Docker (https://www.bezkoder.com/category/docker/), Node.js (https://www.bezkoder.com/category/node-js/)

Docker (https://www.docker.com/) provides lightweight containers to run services in isolation from our infrastructure so we can deliver software quickly. In this tutorial, I will show you how to dockerize Nodejs Express and MySQL example using Docker Compose (https://docs.docker.com/compose/).

Related Posts:
– Build Node.js Rest APIs with Express & MySQL (https://www.bezkoder.com/node-js-rest-api-express-mysql/)
– Build Node.js Rest APIs with Express, Sequelize & MySQL (https://www.bezkoder.com/node-js-express-sequelize-mysql/)
– Upload/store images in MySQL using Node.js, Express & Multer (https://www.bezkoder.com/node-js-upload-image-mysql/)
– Node.js: Upload CSV file data into Database with Express (https://bezkoder.com/node-js-upload-csv-file-database/)
– Node.js: Upload Excel file data into Database with Express (https://www.bezkoder.com/node-js-upload-excel-file-database/)
– Node.js Express: Token Based Authentication & Authorization (https://www.bezkoder.com/node-js-jwt-authentication-mysql/)

Dockerize the fullstack:
– Docker Compose: React, Node.js, MySQL example (https://www.bezkoder.com/docker-compose-react-nodejs-mysql/)

Contents [hide]

Node.js and MySQL with Docker Overview
Create Nodejs App
Create Dockerfile for Nodejs App
Write Docker Compose configurations
Docker Compose Environment variables with MySQL
Run Nodejs MySQL with Docker Compose
Stop the Application
Conclusion
Source Code

Node.js and MySQL with Docker Overview

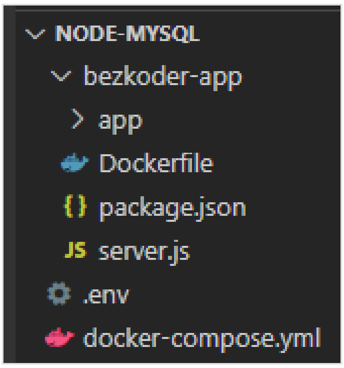
Assume that we have a Nodejs Application working with MySQL database.
The problem is to containerize a system that requires more than one Docker container:

- Node.js Express for API
- MySQL for database

Docker Compose helps us setup the system more easily and efficiently than with only Docker. We're gonna following these steps:

- Create Nodejs App working with MySQL database.
- Create Dockerfile for Nodejs App.
- Write Docker Compose configurations in YAML file.
- Set Environment variables for Docker Compose
- Run the system.

Directory Structure:



```
NODE-MYSQL
  bezkoder-app
    app
    Dockerfile
    package.json
    server.js
  .env
  docker-compose.yml
```

Create Nodejs App

You can read and get Github source code from one of following tutorials:
– Build Node.js Rest APIs with Express & MySQL (https://www.bezkoder.com/node-js-rest-api-express-mysql/)
– Build Node.js Rest APIs with Express, Sequelize & MySQL (https://www.bezkoder.com/node-js-express-sequelize-mysql/)
– Upload/store images in MySQL using Node.js, Express & Multer (https://www.bezkoder.com/node-js-upload-image-mysql/)
– Node.js: Upload CSV file data into Database with Express (https://bezkoder.com/node-js-upload-csv-file-database/)
– Node.js: Upload Excel file data into Database with Express (https://www.bezkoder.com/node-js-upload-excel-file-database/)
– Node.js Express: Token Based Authentication & Authorization (https://www.bezkoder.com/node-js-jwt-authentication-mysql/)

Using the code base above, we put the Nodejs project in **bezkoder-app** folder and modify some files to work with environment variables.

Firstly, let's add `dotenv` module into *package.json*.

```
{
  ...
  "dependencies": {
    "dotenv": "^10.0.0",
    ...
  }
}
```

Next we import `dotenv` in *server.js* and use `process.env` for setting port.

```
require("dotenv").config();
..
// set port, listen for requests
const PORT = process.env.NODE_DOCKER_PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

Then we change modify database configuration and initialization.

app/**config**/*db.config.js*

```
module.exports = {
  HOST: process.env.DB_HOST,
  USER: process.env.DB_USER,
  PASSWORD: process.env.DB_PASSWORD,
  DB: process.env.DB_NAME,
  port: process.env.DB_PORT,
  dialect: "mysql",
  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000
  }
};
```

app/**models**/*index.js*

```
const dbConfig = require("../config/db.config.js");

const Sequelize = require("sequelize");
const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER, dbConfig.PASSWORD, {
  host: dbConfig.HOST,
  dialect: dbConfig.dialect,
  port: dbConfig.port,
  operatorsAliases: false,

  pool: {
    max: dbConfig.pool.max,
    min: dbConfig.pool.min,
    acquire: dbConfig.pool.acquire,
    idle: dbConfig.pool.idle
  }
});

...
```

We also need to make a *.env* sample file that shows all necessary arguments.

bezkoder-app/*.env.sample*

```
DB_HOST=localhost
DB_USER=root
DB_PASSWORD=123456
DB_NAME=bezkoder_db
DB_PORT=3306

NODE_DOCKER_PORT=8080
```

Create Dockerfile for Nodejs App

Dockerfile defines a list of commands that Docker uses for setting up the Node.js application environment. So we put the file in **bezkoder-app** folder.

Because we will use Docker Compose, we won't define all the configuration commands in this Dockerfile.

bezkoder-app/*Dockerfile*

```
FROM node:14

WORKDIR /bezkoder-app
COPY package.json .
RUN npm install
COPY . .
CMD npm start
```

Let me explain some points:

- `FROM` : install the image of the Node.js version.
- `WORKDIR` : path of the working directory.
- `COPY` : copy *package.json* file to the container, then the second one copies all the files inside the project directory.
- `RUN` : execute a command-line inside the container: `npm install` to install the dependencies in *package.json*.
- `CMD` : run script `npm start` after the image is built.

Write Docker Compose configurations

On the root of the project directory, we're gonna create the *docker-compose.yml* file. Follow version 3 (https://docs.docker.com/compose/compose-file/compose-file-v3/) syntax defined by Docker:

```
version: '3.8'

services:
    mysqldb:
    app:

volumes:
```

- `version` : Docker Compose file format version will be used.
- `services` : individual services in isolated containers. Our application has two services: `app` (Nodejs) and `mysqldb` (MySQL database).
- `volumes (https://docs.docker.com/storage/volumes/)` : named volumes that keeps our data alive after restart.

Let's implement the details.

docker-compose.yml

```
version: '3.8'

services:
  mysqldb:
    image: mysql:5.7
    restart: unless-stopped
    env_file: ./.env
    environment:
      - MYSQL_ROOT_PASSWORD=$MYSQLDB_ROOT_PASSWORD
      - MYSQL_DATABASE=$MYSQLDB_DATABASE
    ports:
      - $MYSQLDB_LOCAL_PORT:$MYSQLDB_DOCKER_PORT
    volumes:
      - db:/var/lib/mysql
  app:
    depends_on:
      - mysqldb
    build: ./bezkoder-app
    restart: unless-stopped
    env_file: ./.env
    ports:
      - $NODE_LOCAL_PORT:$NODE_DOCKER_PORT
    environment:
      - DB_HOST=mysqldb
      - DB_USER=$MYSQLDB_USER
      - DB_PASSWORD=$MYSQLDB_ROOT_PASSWORD
      - DB_NAME=$MYSQLDB_DATABASE
      - DB_PORT=$MYSQLDB_DOCKER_PORT
    stdin_open: true
    tty: true

volumes:
  db:
```

– **mysqldb**:

- `image` : official Docker image
- `restart` : configure the restart policy (https://docs.docker.com/config/containers/start-containers-automatically/#use-a-restart-policy)
- `env_file` : specify our *.env* path that we will create later
- `environment` : provide setting using environment variables
- `ports` : specify ports will be used
- `volumes` : map volume folders

– **app**:

- `depends_on (https://docs.docker.com/compose/compose-file/compose-file-v3/#depends_on)` : dependency order, **mysqldb** is started before **app**

- `build` : configuration options that are applied at build time that we defined in the *Dockerfile* with relative path
- `environment` : environmental variables that Node application uses
- `stdin_open` and `tty` : keep open the terminal after building container

You should note that the host port (`LOCAL_PORT`) and the container port (`DOCKER_PORT`) is different. Networked service-to-service communication uses the container port, and the outside uses the host port.

Docker Compose Environment variables with MySQL

In the service configuration, we used environmental variables defined inside the *.env* file. Now we start writing it.

.env

```
MYSQLDB_USER=root
MYSQLDB_ROOT_PASSWORD=123456
MYSQLDB_DATABASE=bezkoder_db
MYSQLDB_LOCAL_PORT=3307
MYSQLDB_DOCKER_PORT=3306

NODE_LOCAL_PORT=6868
NODE_DOCKER_PORT=8080
```

Run Nodejs MySQL with Docker Compose

We can easily run the whole with only a single command:
```
docker-compose up
```

Docker will pull the MySQL and Node.js images (if our machine does not have it before).

The services can be run on the background with command:
```
docker-compose up -d
```

```
$ docker-compose up -d
Creating network "node-mysql_default" with the default driver
Creating volume "node-mysql_db" with default driver
Pulling mysqldb (mysql:5.7)...
5.7: Pulling from library/mysql
33847f680f63: Pull complete
5cb67864e624: Pull complete
1a2b594783f5: Pull complete
b30e406dd925: Pull complete
48901e306e4c: Pull complete
603d2b7147fd: Pull complete
802aa684c1c4: Pull complete
5b5a19178915: Pull complete
f9ce7411c6e4: Pull complete
f51f6977d9b2: Pull complete
aeb6b16ce012: Pull complete
Digest: sha256:be70d18aedc37927293e7947c8de41ae6490ecd4c79df1db40d1b5b5af7d9596
Status: Downloaded newer image for mysql:5.7
Building app
Sending build context to Docker daemon  17.41kB
Step 1/6 : FROM node:14
14: Pulling from library/node
08224db8ce18: Pull complete
abd3caf86f5b: Pull complete
71c316554a55: Pull complete
721081de66bf: Pull complete
239fb482263d: Pull complete
26d24e5f0efd: Pull complete
4a43fffd53dd: Pull complete
4e10c266ec1a: Pull complete
6c4e1d6ce241: Pull complete
Digest: sha256:adbbb61dab70ea6e5a6c2ad7fba60e4d1047ba98ad1afcd631c15553163b22b7
Status: Downloaded newer image for node:14
 ---> e0ab58ea4a4f
Step 2/6 : WORKDIR /bezkoder-app
 ---> Running in 6ab4079d2f00
Removing intermediate container 6ab4079d2f00
 ---> 59e985358175
Step 3/6 : COPY package.json .
 ---> cc619b75b822
Step 4/6 : RUN npm install
 ---> Running in 90bccd42e0d7

added 88 packages from 144 contributors and audited 88 packages in 7.655s
found 0 vulnerabilities

Removing intermediate container 90bccd42e0d7
 ---> c9f5592ab65a
Step 5/6 : COPY . .
```

```
 ---> 65d7d8927e3c
Step 6/6 : CMD npm start
 ---> Running in 2b7b5fe7dbb3
Removing intermediate container 2b7b5fe7dbb3
 ---> 9d0109ff706c
Successfully built 9d0109ff706c
Successfully tagged node-mysql_app:latest
WARNING: Image for service app was built because it did not already exist. To rebuild thi
Creating node-mysql_mysqldb_1 ... done
Creating node-mysql_app_1     ... done
```

Now you can check the current working containers:

```
$ docker ps
CONTAINER ID   IMAGE            COMMAND                  CREATED         STATUS
b8b12819d371   node-mysql_app   "docker-entrypoint.s…"   2 minutes ago   Up About a minut
b0d665c00073   mysql:5.7        "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes
```

And Docker images:

```
$ docker images
REPOSITORY           TAG            IMAGE ID       CREATED         SIZE
node-mysql_app       latest         9d0109ff706c   5 minutes ago   965MB
node                 14             e0ab58ea4a4f   6 minutes ago   944MB
mysql                5.7            8cf625070931   6 minutes ago   448MB
```

Stop the Application

Stopping all the running containers is also simple with a single command:
`docker-compose down`

If you need to stop and remove all containers, networks, and all images used by any service in *docker-compose.yml* file, use the command:
`docker-compose down --rmi all`

Conclusion

Today we've successfully created Docker Compose file for MySQL and Nodejs application. Now we can deploy Nodejs Express and MySQL with Docker on a very simple way: *docker-compose.yml*.

You can apply this way to one of following project:
– Build Node.js Rest APIs with Express & MySQL (https://www.bezkoder.com/node-js-rest-api-express-mysql/)
– Build Node.js Rest APIs with Express, Sequelize & MySQL (https://www.bezkoder.com/node-js-express-sequelize-mysql/)
– Upload/store images in MySQL using Node.js, Express & Multer (https://www.bezkoder.com/node-js-upload-

image-mysql/)
– Node.js: Upload CSV file data into Database with Express (https://bezkoder.com/node-js-upload-csv-file-database/)
– Node.js: Upload Excel file data into Database with Express (https://www.bezkoder.com/node-js-upload-excel-file-database/)
– Node.js Express: Token Based Authentication & Authorization (https://www.bezkoder.com/node-js-jwt-authentication-mysql/)

Or Heroku instead: Deploying/Hosting Node.js app on Heroku with MySQL database (https://bezkoder.com/deploy-node-js-app-heroku-cleardb-mysql/)

Dockerize the fullstack:
– Docker Compose: React, Node.js, MySQL example (https://www.bezkoder.com/docker-compose-react-nodejs-mysql/)

Happy Learning! See you again.

Source Code

The source code for this tutorial can be found at Github (https://github.com/bezkoder/docker-compose-nodejs-mysql).

You can deploy the container on Digital Ocean (https://www.digitalocean.com/?refcode=560b7a03275b&utm_campaign=Referral_Invite&utm_medium=Referral_Program&utm_source=CopyPaste) with very small budget: **5$**/month.
Using referral link below, you will have **100$** in credit over **60** days. After that, you can stop the VPS with no cost.

Powered by DigitalOcean (https://www.digitalocean.com/?

refcode=560b7a03275b&utm_campaign=Referral_Invite&utm_medium=Referral_Program&utm_source=badge)

deploy (https://www.bezkoder.com/tag/deploy/) deployment (https://www.bezkoder.com/tag/deployment/)
docker (https://www.bezkoder.com/tag/docker/)
docker compose (https://www.bezkoder.com/tag/docker-compose/)
dockerize (https://www.bezkoder.com/tag/dockerize/) express (https://www.bezkoder.com/tag/express/)
mysql (https://www.bezkoder.com/tag/mysql/) node.js (https://www.bezkoder.com/tag/node-js/)
rest api (https://www.bezkoder.com/tag/rest-api/)

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

☐

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Docker MERN stack with Nginx example – Docker Compose (https://www.bezkoder.com/docker-mern/)

Spring Boot + React Redux example: Build a CRUD App (https://www.bezkoder.com/spring-boot-react-redux-example/)

Search...

FOLLOW US

(https://facebook.com/bezkoder)

(https://www.youtube.com/channel/UCp0mx9RH0JxaFsmMvKXA86Q)

(https://github.com/bezkoder)

TOOLS

Json Formatter (https://www.bezkoder.com/json-formatter/)