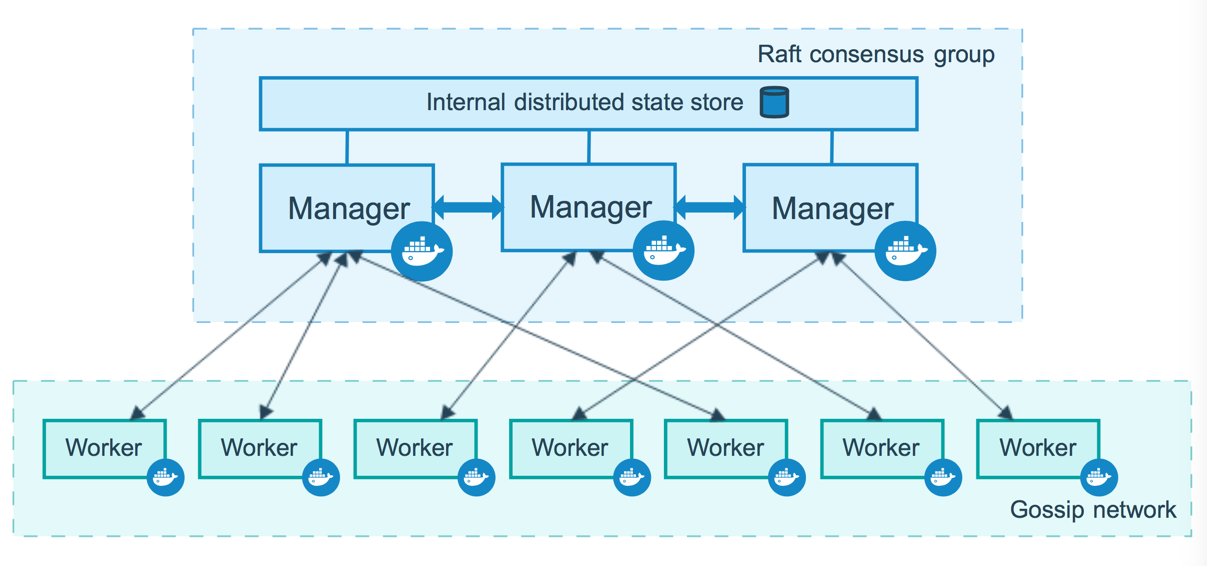
**Deploy Your Application Using Docker Swarm**

How to deploy a simple application within a cluster?

Introduction

Docker Swarm is a leader-workers model based on Raft algorithm. The nodes within the cluster can have either managers or workers roles. The managers have administration and functional responsibilities. They are responsible for launching the services whereas the workers have to execute these services. We must always have a leader node in our cluster. Due to consistency issues, it is recommended to have an odd number of managers and workers.



The goal is to develop an application for managing movies in a library. A movie is characterized by the following attributes: id (primary key), title, genre, budget, release date and imdb score.

**Create the DB**

We will use postgres for the database, so we need an init configuration file for our movies table.

CREATE TABLE IF NOT EXISTS movies (

id serial PRIMARY KEY,

title VARCHAR NOT NULL,

genre VARCHAR NOT NULL,

budget INTEGER NOT NULL,

release\_date TIMESTAMP NOT NULL,

imdb\_score FLOAT NOT NULL

);

We will have two routes, one for getting all the movies from the database, and another one for adding a new movie.

For our server, we are going to use the well known express package.

Paste the following code in a file index.js:

**const express = require("express");**

**const app = express();**

**const { query } = require("./data.js");**

**app.use(express.json());**

**app.post("/api/movies", async(req, res) => {**

**const { title, genre, budget, release\_date, imdb\_score } = req.body;**

**console.info(`Adding movie with title ${title}`);**

**const result = await query(**

**"INSERT INTO movies (title, genre, budget, release\_date, imdb\_score) VALUES ($1, $2, $3, $4, $5) RETURNING id", [title, genre, budget, release\_date, imdb\_score]**

**);**

**res.json({ id: result[0].id });**

**});**

**app.get("/api/movies", async(req, res) => {**

**console.info('Getting all movies');**

**const movies = await query("SELECT \* FROM movies");**

**res.json(movies);**

**});**

**app.listen(80, () => {**

**console.info(`Server started listening on port 80`);**

**});**

To make our work easier, we will use pg package for parsing our SQL queries. Paste the following in a file named data.js:

**const { Pool } = require("pg");**

**const options = {**

**host: process.env.PGHOST,**

**database: process.env.PGDATABASE,**

**port: process.env.PGPORT,**

**user: process.env.PGUSER,**

**password: process.env.PGPASSWORD**

**};**

**const pool = new Pool(options);**

**const query = async(text, params) => {**

**const { rows } = await pool.query(text, params);**

**return rows;**

**};**

**module.exports = {**

**query**

**};**

**Build and Push Image to Docker Hub**

Docker Swarm uses services, so we need to build and push our API image to Docker Hub first.

In order to be able to do that, we need a Dockerfile to build the image.

**FROM node:14-alpine**

**WORKDIR /usr/src/app**

**COPY package\*.json ./**

**RUN ["npm", "ci"]**

**COPY . .**

**EXPOSE 80**

**CMD ["npm", "run", "start"]**

From the folder where Dockerfile is stored, we have to run the following commands:

**docker build -t movies-api .**

**docker tag movies-api <docker-hub-id>/docker-swarm-demo:movies-api**

**docker push <docker-hub-id>/docker-swarm-demo:movies-api**

**Configure Docker Swarm Cluster**

In order to set our virtual machines as a swarm, we need to run the following command from the manager node:

**docker swarm init --advertise-addr <eth1-interface-addr>**

After the command is executed, the manager node is set as a leader. To add a worker to the swarm, we have to run the following command from every worker:

**docker swarm join --token <token> <eth1-interface-addr>:<port>**

**Docker Swarm YAML File**

In our docker swarm YAML file, we have four services:

api (the image is pulled from Docker Hub repository)

db, based on the official postgres:latest image

adminer, based on the official adminer image

visualizer, based on the official dockersamples/visualizer:stable image

Copy and paste following code in a file named docker-swarm.yml:

**version: "3.8"**

**services:**

**api:**

**image: mihaiconstantin98/docker-swarm-demo:movies-api**

**environment:**

**PGUSER: admin**

**PGPASSWORD: admin**

**PGDATABASE: movies**

**PGHOST: db**

**PGPORT: 5432**

**deploy:**

**replicas: 3**

**resources:**

**limits:**

**cpus: "0.2"**

**memory: 50M**

**restart\_policy:**

**condition: on-failure**

**placement:**

**max\_replicas\_per\_node: 2**

**ports:**

**- "5555:80"**

**networks:**

**- db-network**

**depends\_on:**

**- db**

**db:**

**image: postgres:latest**

**environment:**

**POSTGRES\_USER: admin**

**POSTGRES\_PASSWORD: admin**

**POSTGRES\_DB: movies**

**volumes:**

**- db-persistent-volume:/var/lib/postgresql/data**

**- "${PWD}/database/init-db.sql:/docker-entrypoint-initdb.d/init-db.sql"**

**deploy:**

**placement:**

**constraints:**

**- "node.role==manager"**

**networks:**

**- db-network**

**- db-adminer-network**

**adminer:**

**image: adminer**

**ports:**

**- "8080:8080"**

**networks:**

**- db-adminer-network**

**depends\_on:**

**- db**

**visualizer:**

**image: dockersamples/visualizer:stable**

**ports:**

**- 8081:8080**

**deploy:**

**placement:**

**constraints:**

**- "node.role==manager"**

**volumes:**

**- "/var/run/docker.sock:/var/run/docker.sock"**

**volumes:**

**db-persistent-volume:**

**networks:**

**db-network:**

**db-adminer-network:**

**Deploy the Application Within the Cluster**

We need to send the docker-swarm.yml file and the database init configuration file to our manager machine. From your local environment, execute the following commands or use WinSCP:

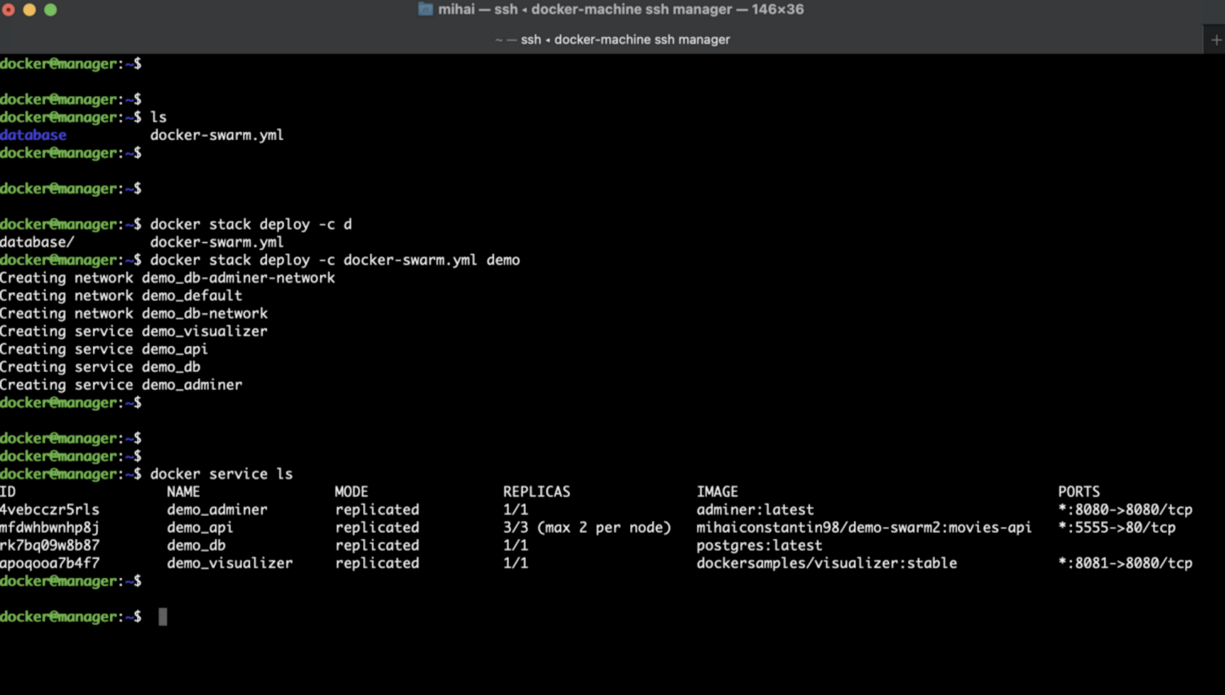
**docker-machine scp -r database manager:.**

**docker-machine scp docker-swarm.yml manager:.**

**docker stack deploy -c docker-swarm.yml demo**

To check if all the services are up and running, we can run:

**docker service ls**



**Test the Application**

In order to test our application, we can use Postman.

To get all the movies from the database, we need to send a GET request to docker-machine-ip:5555/api/movies. To add a movie in the database, we send a POST request to the same endpoint.

We can check in the browser at Master-ip:8080 if the database contains all the data we have expected.

**Visualize the Services Within the Cluster**

We can check how the services are distributed within the cluster, if we go to docker-machine-ip:8081 in our browser.

