CODELABS ACADEMY

Backend Deployment

In this assignment, you're going to deploy your backend services (express and nestjs) using Docker.

1. Initial Setup

To complete this assignment, you need to have a docker engine installed on your system. For more details about how to install it please check this link.

2. Deploy Your Expressjs Application

- Create a `Dockerfile` for the application.
- As a base image, use `node:lts-alpine3.19` as it has a small size compared to other images.
- Create a directory named `app` inside the container's filesystem and set it as the working directory for subsequent commands. This is where your application code will be copied.
- Copy the `package.json` file.
- Run the command that installs the packages.
- Copy the code files host machine to the `app` directory in the Docker container.
- Expose the application port.
- Run the command that starts the server
- Build the docker image and give it a tag of `codecla-express:v1` for example.
- Run the container
- Before running the container, start a MongoDB server using the official MongoDB Docker image.
- Ensure that you publish the necessary ports and pass the required environment variables like port number, database URL, secret keys, etc.

Note: If you face errors in accessing the database from the express app container, you can run the containers within the `host` network (`--network host`), publish the ports, and use

`mongodb://localhost:27017` as a database URL for the express app container.

- Make sure that all endpoints of your Express.js application work as expected.

3. Deploy Your NestJS Application

To deploy your NestJS application, we are going to use a **two-stage build** as it optimizes nest js builds to end up with a small image size.

First, you need to create a `Dockerfile`, then follow these steps:

Development stage

For the development stage:

- Use the same nodejs base image, `node:lts-alpine3.19`.
- Create a directory named `app` inside the container's filesystem and set it as the working directory
- Copy the `package.json` file.
- Run the command that installs the packages.
- Copy the code files host machine to the `app` directory in the Docker container.
- Run the build command.

Production stage

For the production stage:

- Use the same nodejs base image, `node:lts-alpine3.19`.
- Set the working dir to `app`.
- Copy the `package.json` file.
- Install **production-only** packages, using `npm install --only=production`.
- Copy the code files host machine to the `app` directory in the Docker container.

- Copy the generated `dist` directory from the `development stage` to this container.
- Execute the main script using `node`.
- Build the docker image and give a tag of `codecla-nestjs:v1` for example.
- Run the container
- Before running the container, start a MongoDB server using the official MongoDB Docker image.
- Ensure that you publish the necessary ports and pass the required environment variables like port number and database URL.
- Make sure that all endpoints of your Nest.js application work as expected.

Important: Make sure to run your services on different ports to avoid errors.

Note: if you list the images and inspect your nest js image you'll see that it has a relatively small size (around 350MB). It would've been around 1.5GB if we used the traditional build! Hence, the power of multi-stage builds.