Cloud Ready Hack

Challenge Description

Challenge 1: Pack it and send it to the dock

As a newly formed team, your job is to improve how the new application versions are deployed to production. Ultimately, you wish to have an automated continuous integration and deployment process, but it'll take time to get there. First, you want to choose the technology carefully so you don't depend on the physical or virtual hardware infrastructure. The only technology that ticks all your boxes are containers; of course, you decide to give it a go with Docker.

You got access to the source code. The Git repo has been zipped and can be downloaded from this link: https://storage.googleapis.com/cloud-ready-hackathon/apis/odh-containers.tar.gz

Your first task is to familiarize yourselves with the codebase and verify that everything works correctly. Your first step towards that is to build and test everything locally, and you will start doing just that by building and running the User API container and a PostgreSQL Server container.

Luckily again, you don't have to build everything from scratch. You can use the Share&Care source code and many references available on the internet that explain the process of Dockerfile creation. As a team, you'll have to create a docker file for all parts of your solution: a web application and 4 APIs.

What is an app without the data? Therefore, you are going to set up a PostgreSQL Server and import databases for each of the APIs. Luckily again, the data team already prepared the .sql scripts for you, ready to be imported, and you can find them on this link: https://nephoshacks.blob.core.windows.net/cloud-ready-hack/dbscripts.tar.gz

To set up a PostgreSQL Server container running locally, you can use the following command, which helps you add sample data to the database PostgreSQL will generate automatically. Keep in mind that you need to adapt this command to your environment:

```
docker run -d --name=postgres --rm --env=POSTGRES_PASSWORD=foo \
--volume=$(pwd)/schema.sql:/docker-entrypoint-initdb.d/schema.sql \
postgres:latest -c log_statement=all
```

Your goal is to configure the User API application to connect to the PostgreSQL server so you can test that the application works. You can use curl commands to test the application's endpoints. You can find a set of methods that User API serves in the following table.

HTTP verb	URI	Scope	Semantics
GET	/users/	User	Retrieves all users
GET	/users/<user_id>	User	Retrieve a specific user
POST	/users/<user_id>	User	Create a new user
PUT	/users/<user_id>	User	Update an existing user
DELETE	/users/<user_id>	User	Delete an existing user

When each of you tests the User API and it works properly, the team should proceed to create Dockerfiles for the rest of the APIs (and the web application). You'll need to build Docker images and push them to the team's Container Registry in Microsoft Azure, which you'll have to create.

Don’t forget to push your changes to the team's GitHub repository when you complete everything.

Definition of done

- Show your coach a locally running User API container connected to a running PostreSQL container. For local environment, use the the team’s VM.
- Container images for all the Share&Care components are pushed to the team's Container Repository (ACR).

Reading materials

- Docker basics
- Docker Networking
- Dockerfile reference
- Nuxt - The env property (nuxtjs.org)
- Docker Command Line Interface reference
- Docker Registry
- Managed container registries - Azure Container Registry
- Quickstart - Create registry in portal and push your first image - Azure Container Registry