**CS 470 Module Two Assignment Two Guide**

**Introduction**
This lesson will complete your work with containers by introducing Docker Compose to allow orchestrating multiple containers as a logical unit. You will bring up the entire stack in three containers working together to serve a full stack application.

**Summary Steps**
- Create a shared network:
  - o   docker network create --driver bridge lafs-net
- Create a Docker Compose script for the backend containers:
  - o   Create docker-compose.yml file in top-level lafs-api directory.
  - o   docker-compose up
  - o   Edit lafs-api/server/datasources.development.js to add default local values.
  - o   docker-compose up
  - o   http://localhost:3000/explorer to verify and add data
  - o   Use Mongo shell or Robo 3T to verify the data stored in the Mongo container.
- Create a Docker Compose script for the frontend container:
  - o   Create docker-compose.yml file in top-level lafs-web directory.
  - o   Edit lafs-web/src/environments/environment.ts to use localhost:3000.
  - o   docker-compose up
  - o   http://localhost:4200 to verify and add data
  - o   Use Mongo shell or Robo 3T to verify the data stored in the Mongo container.

**Detailed Steps**

**Docker Network**
As you discovered in the previous assignment, every container is isolated. There are three common ways for the application(s) in the container to interact with resources outside of the container:

1. Use port mapping to allow network traffic from the host computer into the container.
2. Use Docker networking to build a virtual network between containers.
3. Use Docker volume management to mount storage volumes into the container.

You have already practiced using port mapping in the previous assignment. Volume management will not be covered in this exercise. You will build a bridge network for the containers to communicate with each other.

Create a new bridge network using the Docker network commands:

> docker network create --driver bridge lafs-net
> docker network list

```
> docker network create --driver bridge lafs-net
7a8db2af9ac5817ae29aede2ec8ee22c0288d3677d99720312ec850a4d953ea0

> docker network list
NETWORK ID      NAME              DRIVER          SCOPE
b4ff8b47f38d    bridge            bridge          local
f194fde17772    host              host            local
7a8db2af9ac5    lafs-net          bridge          local
da240ef3df71    none              null            local

>
```

*Note: The network list command will show the Docker networks including the new one just created.*

You have created a virtual network within Docker and given it the name of "lafs-net", which you can now refer to in your containers.

**Backend**

Docker Compose uses a YAML file to define the services, network, and storage volumes that an application will use. Like the Python programming language, YAML files rely on indentation to group values together. A single compose file can configure one or more containers. For the backend application you will define two services – one for Node JS and the other for MongoDB.

Refer to the documentation for compose and study the proposed file below:

```yaml
docker-compose.yml

version: '3.7'
services:
# REST API running on Node JS container
 app:
   container_name: lafs-api
   restart: always
   build: .
   ports:
    - '3000:3000'
  # link this container to the Mongo DB container
   links:
    - mongo
  # pass in environment variables for database host and name
   environment:
    - DB_HOST=mongo
    - DB_NAME=lafs-db
# Mongo DB storage container
 mongo:
   container_name: lafs-db
   image: 'mongo:4'
   ports:
    - '27017:27017'
# Attach the external network to these containers
networks:
```

        default:
         external:
          name: lafs-net

Create a "docker-compose.yaml" file in the top of the project directory and input the above information.

To ensure that the REST API application will default to running against a local Mongo DB instance, update the **server/datasources.development.js** file to specify default values if not overridden by environment values:

```
    module.exports = {
     mongodb: {
      connector: 'mongodb',
      hostname: process.env.DB_HOST || 'localhost',
      port: process.env.DB_PORT || 27017,
      user: process.env.DB_USER || '',
      password: process.env.DB_PASSWORD || '',
      database: process.env.DB_NAME || 'lafs',
      url: process.env.DB_URL
     }
    };
```

Edit the file by adding the values shown highlighted above.

Now issue the Docker Compose command in your PowerShell window to bring up both containers:

        > docker-compose up



        Note: Two containers started and attached to the virtual network.

First, open a command shell inside the Mongo container and see the status of the database after initial startup:

1.       On the Docker Dashboard app, hover the mouse over the Mongo container entry and click the
         Shell/CLI icon:
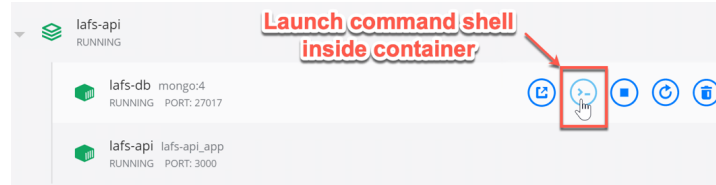
2. Now input the follow commands:

> mongo
> show dbs
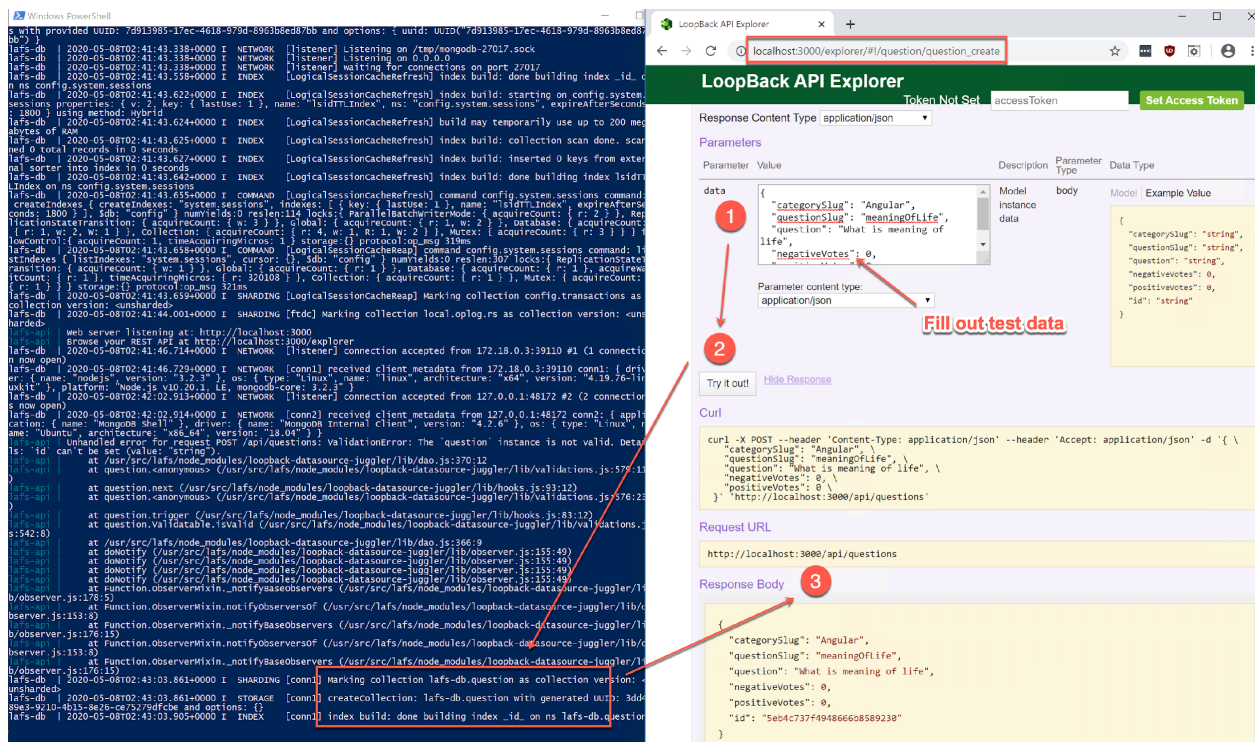


To test these containers, open a web browser and navigate to the LoopBack Explorer test page:

> http://localhost:3000/explorer

To test the REST API running in the container, do the following:

1. Input a test question into the text box labeled "Data" using information like this:

```
{
    "categorySlug": "angular",
    "questionSlug": "what-is-the-meaning-of-life",
    "question": "What is the meaning of life?",
    "negativeVotes": 0,
    "positiveVotes": 0
}
```

2. Click the **Try it out!** button.
3. An HTTP request will be sent to the REST endpoint. You can see in the console window that it was received and processed.
4. A successful response is returned with the "id" value populated by Mongo.

Now go back to the Mongo command shell and see what has been stored using the following commands:

```
> show dbs
> use lafs_db
> show collections
> db.getCollection('question').find({})
```

Running a browser on your computer, you tested sending a question to the REST API endpoint running in one container. That Node JS code in turn connected to the second container running Mongo DB, where the question was finally stored.

**Frontend Development**
Because the frontend Angular site is a separate code repository, it is a separate directory on your computer. Therefore, a separate docker-compose YML file needs to be created there to load it into a container. By specifying the same external virtual network, you will communicate with the other two containers and you will see previously entered questions and answers shown on the frontend web application.

```
docker-compose.yml

version: '3.7'
services:
# Angular frontend application
 app:
  container_name: lafs-web
  restart: always
  build: .
  ports:
   - '4200:4200'
  command: >
    bash -c "npm install && ng serve --host 0.0.0.0 --port 4200"
# Attach the external network to these containers
networks:
 default:
  external:
   name: lafs-net
```

The Angular tutorial code is written to run against a Heroku cloud instance, so you will need to change the **api_url** value in **src/environments/environment.ts**.
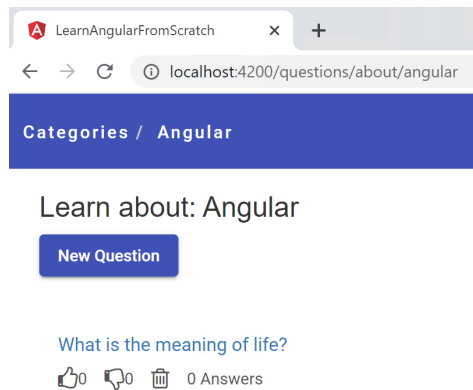
```
export const environment = {
 production: false,
 api_url: 'http://localhost:3000'
};
```

Start the container as you did for the backend application, open a browser and navigate to
http://localhost:4200, choose the Angular category, and you will see the following:



You now have three Docker containers, each of them isolated from the others yet communicating
together. Use the Docker ps command to see all the container instances running and their configured
state:

> docker ps -a

```
> docker ps -a
CONTAINER ID    IMAGE           COMMAND                 CREATED         STATUS                  PORTS                       NAMES
d096d411de4c    lafs-web_app    "docker-entrypoint.s…"  2 hours ago     Up 2 hours              0.0.0.0:4200->4200/tcp      lafs-web
c2e95ae05b83    lafs-api_app    "docker-entrypoint.s…"  11 hours ago    Up 11 hours             0.0.0.0:3000->3000/tcp      lafs-api
44668e532c72    mongo:4         "docker-entrypoint.s…"  11 hours ago    Up 11 hours             0.0.0.0:27017->27017/tcp    lafs-db
e1d5ffe938fb    mongo           "docker-entrypoint.s…"  4 days ago      Exited (0) 2 days ago                               mongodb
> _
```

These container images can be deployed to a server in a data center or to a cloud provider and will run
there exactly as you have configured them on your computer!