Assignments 1 and 2

1. We started by creating the "Dockerfile" file for the frontend and backend:

```
FROM node

WORKDIR /app

COPY package.json .

RUN npm install

COPY . .

EXPOSE 80

CMD ["node", "app.js"]
```

Image 1 - Dockerfile for backend

```
FROM node

WORKDIR /app

COPY package.json .

RUN npm install

COPY . .

EXPOSE 3000

CMD [ "npm", "start" ]
```

Image 2 - Dockerfile for frontend

2. Then, we created the network with the name ‘projectdb’ using the following command, so that the different containers can comunicate:
   a. `docker network create projectdb`

3. After the creation of the network, to run the database we needed the following command:
   a. `docker run --name mongo --network projectdb mongo`

4. Since the database has the name 'mongo', we had to change the connection in the 'app.js' file in the backend folder:
   a. `‘mongodb://mongo:27017/course-goals’`

5. After that, we created the images for the backend and frontend with the following command, in each directory:
   a. `docker build -t backend_image .`
   b. `docker build -t frontend_image .`

6. To run the frontend and backend, it is necessary to ensure that the database is running (point 3) and then run the following commands. If mongo is not running, it can be started with ‘docker start mongo’:
   a. `docker run --name backend --network projectdb -p 80:80 backend_image`
   b. `docker run -it --name frontend --network projectdb -p 3000:3000 frontend_image`

7. To persist data from the database and NodeJS (backend), it was necessary to create volumes and run both parts with the respective volumes.
   a. Database:
      i. `docker volume create mongodb_data`
      ii. `docker run --name mongo --network projectdb -v mongodb_data:/data/db mongo` (If the mongo container already exists, it should be deleted, command: `docker container rm mongo`)
   b. NodeJS (in the folder backend)
      i. `docker volume create app_logs`
      ii. `docker run --name backend --network projectdb -p 80:80 -v app_logs:/logs backend_image` (If the backend container already exists, it should be deleted, command: `docker container rm backend`)

Duarte Meneses 2019216949
Patricia Costa 2019213995

Assignment 3

1. Publish the containers in Docker Hub
    a. The Mongo image is an official Docker image and is therefore already published on Docker Hub;
    b. In Docker Hub, two repositories were created to place the frontend and backend images. The repositories created were 'project_iacd_frontend' and 'project_iacd_backend' for the respective images;
    c. Tag for Docker Hub: By default, the images created have a tag called "latest", however, to test the various commands, the tags of the images created have been changed (the username was changed depending on who used the command):
        i. Backend: `docker tag backend_image patii01/project_iacd_backend:backend_image`
        ii. Frontend: `docker tag frontend_image patii01/project_iacd_frontend:frontend_image`
    d. The following commands were used to publish the images in Docker Hub to the respective repositories:
        i. Backend: `docker push patii01/project_iacd_backend:backend_image`
        ii. Frontend: `docker push patii01/project_iacd_frontend:frontend_image`

2. Docker Compose
    a. To configure Docker Compose we needed to create the file 'docker-compose.yaml';

```yaml
version: '3.8'
services:

  mongo:
    image: mongo
    volumes:
    - mongodb_data:/data/db
    ports:
    - 27017:27017

  backend:
    image: patii01/project_iacd_backend:backend_image
    #build: ./backend/
    ports:
      - 80:80
    volumes:
    - app_logs:/backend/logs
    depends_on:
      - 'mongo'

  frontend:
    image: patii01/project_iacd_frontend:frontend_image
    #build: ./frontend/
    ports:
      - 3000:3000
    stdin_open: true
    tty: true

volumes:
  mongodb_data:
  app_logs:
```

i. Services: list of containers to be deployed - Mongo, Backend and Frontend;
ii. Image: Fetch the images for the backend and frontend from the Docker Hub repositories;
iii. Ports: Port Mapping between the local machine and the containers network;
iv. Depends-on: The backend depend on the mongo since the mongo container must be launched before the backend container;
v. Volumes: The same names were used as in the previous task and taking into account the path in the container to mirror the data;
vi. Interactive Mode: Interactive mode was used for the frontend, from 'stdin_open' and 'tty', so that the container started waiting for input information in the terminal;
vii. Build: Although not used in this case, it is utilised for the image to be created taking into account the path where the respective Dockerfile is located. To use it, you need to remove the images part.

Image 3 - Code for Docker Compose

    b. To run the App with Docker Compose the following command was used: `docker-compose up`

Duarte Meneses 2019216949
Patricia Costa 2019213995

Assignments 4 and 5

1. Deploy the multi-container application using Kubernetes **Imperative** Approach
    a. Start minikube: `minikube start`
    b. Deploy containers on kubernetes:
        i. Database: `kubectl create deployment kubs-mongo-proj-depl --image=mongo`
        ii. Backend: `kubectl create deployment kubs-backend-proj-depl --image=patii01/project_iacd_backend:backend_image --port=80 --replicas=2`
        iii. Frontend: `kubectl create deployment kubs-frontend-proj-depl --image=patii01/project_iacd_frontend:frontend_image --port=3000 --replicas=3`
    c. Expose the Kubernetes Deployment Object
        i. Database: `kubectl expose deployment kubs-mongo-proj-depl --type=ClusterIP --port=27017`
        ii. Backend: `kubectl expose deployment kubs-backend-proj-depl --type=LoadBalancer --port=80`
        iii. Frontend: `kubectl expose deployment kubs-frontend-proj-depl --type=LoadBalancer --port=3000`
    d. At the end, to switch off the minikube: `minikube stop`

2. Deploy the multi-container application using Kubernetes **Declarative** Approach
    a. Change the connection in the backend code to: `` `mongodb://${process.env.DB_ADDRESS}:27017/course-goals` ``
    b. Because of this we had to create the image again. To create it with the same name (not mandatory) you need to delete the old image and push it back to Docker Hub.
        i. Build the image (in the folder backend): `docker build -t backend_image .`
        ii. Push to Docker Hub: `docker tag backend_image patii01/project_iacd_backend:backend_image | docker push patii01/project_iacd_backend:backend_image`
    c. To use the declarative approach, you need to create two files yaml, deployment and service, for each of the parts (database, backend and frontend)
        i. Deployment: the files are attached at 'assignments#4&#5'
        ii. Service: the files are attached at 'assignments#4&#5'
    d. Start minikube: `minikube start`
    e. To Persist the data, we needed 2 files for each volume (for Database and Backend), one for the Persistent Volume and another for Persistent Volume Claim: the files are attached at 'assignments#4&#5'
    f. To create the volumes we needed to run the following command:
        i. Persistent Volume Mongo: `kubectl apply -f=mongo-pv.yaml`
        ii. Persistent Volume Claim Mongo: `kubectl apply -f=mongo-pvc.yaml`
        iii. Persistent Volume Backend: `kubectl apply -f=backend-pv.yaml`
        iv. Persistent Volume Claim Backend: `kubectl apply -f=backend-pvc.yaml`
    g. To create the deployments we needed to run the following command:
        i. Deployment mongo: `kubectl apply -f=mongo-deployment.yaml`
        ii. Deployment backend: `kubectl apply -f=backend-deployment.yaml`
        iii. Deployment frontend: `kubectl apply -f=frontend-deployment.yaml`

Duarte Meneses 2019216949
Patricia Costa 2019213995

h. To create the services we needed to run the following command:
   i. Service mongo: `kubectl apply -f=mongo-service.yaml`
   ii. Service backend: `kubectl apply -f=backend-service.yaml`
   iii. Service frontend: `kubectl apply -f=frontend-service.yaml`

i. Since, according to the files provided, it is not possible to connect to the frontend, Postman was used to check the platform. With this, we tested through the backend with the following command: `minikube service backend`

j. The result of this command is an address to use in Postman. This address changes every time the command is used, so the following address is just an example.

k. Get: `http://127.0.0.1:46107/goals`

   Post: `http://127.0.0.1:46107/goals`

   Body, JSON:

```
{
   "text": "teste"
}
```

   Delete: `http://127.0.0.1:46107/goals/{id}` (The "id" will be changed depending on what we want to eliminate)

l. At the end, to switch off the minikube: `minikube stop`