- **Assignment 1 - Container Basics**

  In this Assignment, it was supposed to build images for the backend, frontend and run the Mongo image. Besides the images, I had to create Dockerfiles for both the backend and frontend.

  The commands used to build and run the images were as follows:

  (mongo) - docker pull mongo

  (backend) - docker build -t backend backend/.

  (frontend) - docker build -t frontend frontend/.

  Afterwards, I ran the containers without connecting them to make sure they were properly made.

  The Dockerfiles created are:

```
FROM node

WORKDIR /app

COPY package.json .

RUN npm install

COPY . .

EXPOSE 80

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

```
FROM node

WORKDIR /app

COPY package.json .

RUN npm install

COPY . .

EXPOSE 3000

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

  The first image corresponds to the backend Dockerfile and the second one to the frontend Dockerfile.

  To create the containers, the following commands were used:

  (mongo) - docker run -d –name mongo_container mongo

  (backend) - docker run -d –name back_container backend

  (frontend) - docker run -d –name front_container frontend

  For these commands, I used -d to detach the container from the terminal in order for it to be running in the background and just gave the containers a name and the required image.

- **Assignment 2 - Container Connectivity & Volumes**

  In this second Assignment, I had to make connections between the components (backend with the database, frontend with backend). Following those connections, I had to make sure the data was persisted from the database and the Node.js Web App.

  In order to have the connection between backend and the mongo database running, I had to create a network, using the following command:

  (network) - docker network create net

  This command creates a Docker network called net.

  To use this network properly, I had to change the Mongo database URL in the backend source code changing it from ***localhost*** to the name of the container, in this case ***mongo_container***.

  After this step, I had to build a new image for the backend with the following command, using a new tag to be able to identify the version of the image:

  (backend) - docker build -t backend:network backend/.

  For this task, the creation of volumes was also needed so, I used the following commands:

  (mongo) - docker volume create mongo_vol

  (backend) - docker volume create backend_vol

  After this step, I ran the containers using the following commands:

(mongo) - docker run -d –name mongo_container –network net -v mongo_vol:/data/db mongo
(backend) - docker run -d –name back_container –network net -v backend_vol:/logs -p 80:80 backend:network
(frontend) - docker run -d –name front_container -p 3000:3000 frontend

With these commands, the communication between the mongo container and the backend container is made through the network. And, the communication between the browser and the backend and frontend is made through port mapping. For the backend, the requests come through port 80 and, on the frontend, they come through port 3000.

- **Assignment 3 - Docker Compose**
  For this Assignment, the expected output was the deployment of the App using Docker Compose, but first I needed to publish the created containers on Docker Hub.

  To push the images into my Docker Hub profile after logging in to my account, the following commands were used:
  (login) - docker login -u paulinomariana

  (backend) - docker push paulinomariana/iacd_backend:network
  (frontend) - docker push paulinomariana/iacd_frontend

  The mongo image didn't have to be pushed since it is public on Docker Hub and when we try to use it in projects it is accessible just by using the **docker pull mongo command**.

  After pushing the images into my Docker Hub profile, I created the docker-compose.yaml file where all the needed containers are declared. In this file, I had to declare which image is needed and the volumes, networks and the port mapping that are going to be used.

```yaml
version: "3.8"

services:

  mongo_container:
    image: mongo
    volumes:
      - mongo_vol:/data/db
    networks:
      - net

  back_container:
    image: paulinomariana/iacd_backend:network
    ports:
      - "80:80"
    depends_on:
      - mongo_container
    networks:
      - net
    volumes:
      - backend_vol:/logs
    stdin_open: true
    tty: true

  front_container:
    image: paulinomariana/iacd_frontend
    ports:
      - "3000:3000"
    depends_on:
      - back_container

volumes:
  mongo_vol:
  backend_vol:

networks:
  net:
```

To run the application using the ***docker-compose*** file, the command needed is:
(up) - docker compose up -d
And, when we have to remove all the components created by the docker-compose file, the command that needs to be run is:
(down) - docker compose down -d

- **Assignment 4 - Kubernetes Deployment**
  For Assignment number four, I had to install Kubernetes (***minikube*** and ***kubectl***)
  Firstly I did the deployment of the multi-container application using the **Kubernetes Imperative** Approach.
  This approach starts by initiating the minikube with the command:
  (minikube) - mikikube start

  After the minikube starts, the following step is to create the deployment objects:
  (mongo) - kubectl create deployment mongo-pod --image=mongo --port=27017
  (backend) - kubectl create deployment backend-deployment --image=paulinomariana/iacd_backend --port=80 --replicas=2
  (frontend) - kubectl create deployment frontend-deployment --image=paulinomariana/iacd_frontend --port=3000 --replicas=3

  After the deployments were created, the following commands were executed to create the services:
  (mongo) - kubectl expose deployment mongo-deployment --type=ClusterIP --port=27017
  (backend) - kubectl expose deployment backend-deployment --type=LoadBalancer --port=80
  (frontend) - kubectl expose deployment frontend-deployment --type=LoadBalancer --port=3000

  Since in this assignment, it wasn't supposed to have Data Persistence, a part of this assignment is also terminated. To make sure the minikube stops properly, I ran the following command:
  (minikube) - minikube stop

- **Assignment 5 - Kubernetes Connectivity & Volumes**
  In Assignment 4 it was asked to create a deployment using Imperative and Declarative Approaches. The Imperative one was demonstrated above in the prior section. In Assignment 5, it is asked to implement and show the **Declarative Approach** using Volumes and Networks, so that the data becomes persistent.
  This approach starts by initiating the minikube with the command:
  (minikube) - mikikube start

  After the minikube starts, I had to run the apply command for the deployments, services, and volumes (persistent volume and the persistent volume claims) in this order.
  Deployments
  (mongo) - kubectl apply f=mongo-deployment.yaml
  (backend) - kubectl apply f=backend-deployment.yaml
  (frontend) - kubectl apply f=frontend-deployment.yaml

Advanced Infrastructures for Data Science

Mariana Lopes Paulino, 2020190448

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: iacd-app
  template:
    metadata:
      labels:
        app: iacd-app
    spec:
      containers:
        - name: backend-container
          image: paulinomariana/iacd_backend
          ports:
            - containerPort: 80
          volumeMounts:
            - name: backend-persistent-volume
              mountPath: /logs
      volumes:
        - name: backend-persistent-volume
          persistentVolumeClaim:
            claimName: backend-pvc
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: iacd-app
  template:
    metadata:
      labels:
        app: iacd-app
    spec:
      containers:
        - name: mongo-container
          image: mongo
          ports:
            - containerPort: 80
          volumeMounts:
            - name: mongo-persistent-volume
              mountPath: /data/db
      volumes:
        - name: mongo-persistent-volume
          persistentVolumeClaim:
            claimName: mongo-pvc
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: iacd-app
  template:
    metadata:
      labels:
        app: iacd-app
    spec:
      containers:
        - name: frontend-container
          image: paulinomariana/iacd_frontend
          ports:
            - containerPort: 3000
```

Services

(mongo) - kubectl apply f=mongo-service.yaml
(backend) - kubectl apply f=backend-service.yaml
(frontend) - kubectl apply f=frontend-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mongo-service
spec:
  selector:
    app: iacd-app
  type: ClusterIP
  ports:
    - protocol: TCP
      port: 27017
      targetPort: 27017
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  selector:
    app: iacd-app
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  type: LoadBalancer
  selector:
    app: iacd-app
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
```

Volumes

Persistent Volume

(mongo) - kubectl apply f=mongo-pv.yaml
(backend) - kubectl apply f=backend-pv.yaml

Persistent Volume Claim

(mongo) - kubectl apply f=mongo-pvc.yaml
(backend) - kubectl apply f=backend-pvc.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mongo-pv
spec:
  capacity:
    storage: 1Gi
  volumeMode: Filesystem
  storageClassName: standard
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /data/mongo
    type: DirectoryOrCreate
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongo-pvc
spec:
  volumeName: mongo-pv
  accessModes:
    - ReadWriteOnce
  storageClassName: standard
  resources:
    requests:
      storage: 1Gi
```

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: backend-pv
spec:
  capacity:
    storage: 1Gi
  volumeMode: Filesystem
  storageClassName: standard
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /data/backend
    type: DirectoryOrCreate
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: backend-pvc
spec:
  volumeName: backend-pv
  accessModes:
    - ReadWriteOnce
  storageClassName: standard
  resources:
    requests:
      storage: 1Gi
```

To stop the minikube the command is the same as before **minikube stop**.