PROJECT 1: Docker and Containers

Name: Patrick Keogh
Student ID: 19321326

Exercise One - Images & Containers

Task 1: Create a Docker volume named ex1_vol

Command:

```
docker volume create ex1_vol
```

Result: A Docker volume named ex1_vol was successfully created.

Task 2: Start an ubuntu:22.04 container named sender

Command:

```
docker run -it --name sender -v ex1_vol:/data ubuntu:22.04
```

Result: An interactive session with the sender container was initiated. The volume ex1_vol was mounted to the container's /data directory.

Task 3: Create a message inside the sender container and read it in receiver

Commands and Steps:

1. Inside the sender container:

```
  cd /data
  echo "Patrick Keogh" > name.txt
```

Result: A file named message.txt was created in /data with the content "Hello from sender".

2. Exiting the sender container:

```
  exit
```

Result: The interactive session with the sender container was closed.

3. Starting receiver container with read-only volume:

```
  docker run -it --name receiver -v ex1_vol:/data:ro ubuntu:22.04
```

4. Inside the receiver container:

```
  cd /data
  cat name.txt
```

Result: The content "Patrick Keogh" was displayed, indicating that the volume retains its data across different containers.

Exercise Two - Dockerfile

Task 1: Crafting the Dockerfile

Dockerfile:

```
FROM ubuntu

COPY install.sh /

RUN chmod +x /install.sh
RUN /install.sh

CMD ["/bin/sh"]
```

Command:

```
docker build -t ex2_ubuntu:latest .
```

Result:

A Docker image named ` ex2_ubuntu` was created, which, when instantiated as a container, would have the configurations and installations from the `install.sh` script. This command is run from the directory containing the Dockerfile.

Task 2: Engaging the Shell Application (`sh`)

Description:

There are primarily two methods to auto-initiate the shell application (`sh`):

1. Using CMD directive in Dockerfile:
Incorporate the CMD directive in the Dockerfile:

```
CMD ["/bin/sh"]
```

This method allows the container to run the shell application by default upon initiation.

2. Command Override during container run:
Specify the shell application when starting the container:

```
docker run -it ex2_ubuntu:latest /bin/sh
```

This approach lets users manually override the default behaviour and start the shell application.

Discussion:

- CMD directive: This approach is advantageous when a specific, consistent behaviour is desired each time the container is started. It abstracts the complexities for users who might not be aware of specific commands. This is the option I chose as the behaviour of this container is very consistent.
- Command Override: It offers flexibility, especially beneficial when a container serves multiple purposes or when developers intend to debug or inspect it differently than its typical operation.

Exercise Three - Dockerize a Web Application using docker-compose

Task 1: MySQL Server Definition

Steps:

1. Service Definition: The service was named `db`.

2. Image Selection: The `mysql` image from the public repository was selected.

3. Restart Policy: Restart policy was defined as `always` to ensure the container restarts if any issues arise or after a system reboot.

4. Port Mapping: Ports were mapped such that 9906 on the host corresponds to 3306 on the container.

5. Environment Variables:
   - `MYSQL_ROOT_PASSWORD`: Defined as ` rootPassword12`.
   - `MYSQL_DATABASE`: Set to ` mydb`.
   - `MYSQL_USER`: Designated as ` dbuser`.
   - `MYSQL_PASSWORD`: Marked as ` sqlPassword12`.

Task 2: phpMyAdmin Service Update:

1. Service Definition: The service was named `phpmyadmin`.

2. Image Selection: The `phpmyadmin/phpmyadmin` image was selected.

3. Restart Policy: Much like the database, a restart policy of `always` was chosen for resilience.

4. Port Export: A unique port different from the PHP server was chosen for exposure. In this specific instance, the port was selected as `8080` (as deduced from provided hints).

5. Dependencies: The phpMyAdmin service was made dependent on the `db` service, ensuring that the database container starts first.

6. Environment Variables:
   a. `PMA_HOST`: Defined as `db` to link to the database service.

Test & Validation:

1. Started the app using the command ‘docker-compose up -d’
2. Identified the creation of the `php/src` folder.
3. Copied PHP source codes manually from the `ex3` folder to `php/src` using the command ‘cp insert.php index.php php/src/’
4. Accessed the web application through `http://localhost:8000` successfully.
5. Entered the database admin interface via `http://localhost:8080` (as hinted).
6. Used `root` as the username and ` rootPassword12` as the password for entry.

Exercise Four - Kubernetes: Scale-Up Your App

Task 1: Describe the Deployment Configuration

First step:
Run ‘minikube start –-nodes 2’
Check nodes with ‘kubectl get nodes’

kind:

This field defines the type of Kubernetes resource to create. Deployment was specified, which means that this configuration is designed to ensure a specific number of pod replicas are running at all times.

spec.replicas:

This field specifies the desired number of pod replicas. It was set to 2, indicating two replicas of the pod should always be maintained.

spec.strategy:

The strategy selected was RollingUpdate, an approach that incrementally updates pods with minimal downtime.
The sub-field maxUnavailable was set to 100%, a notable configuration as it allows all old pods to be terminated before spinning up any new ones.

spec.template.spec.affinity:

This configuration is crucial for scheduling. The given podAntiAffinity settings ensured that the pods with the label app: hello would not be co-located on the same node.
The key "kubernetes.io/hostname" ensures these pods are spread across different nodes.

spec.template.spec.containers:

This section defines the properties of the container within the pod. The image pbitty/hello-from:latest was specified, and it was set to expose port 80.

Task 3: Modify Deployment Configuration

Steps:

1. Deploy using Configuration File:

```
kubectl apply -f hello-deployment.yaml
```

2. View Running Pods:

```
kubectl get pods -o wide
```

3. Delete Previous Deployment:

```
kubectl delete -f hello-deployment.yaml
```

Ensure No Other Pod is Running:

```
kubectl get pods
```

If there are any pods still running, delete them using:

```
kubectl delete pod <POD_NAME>
```

4. Label preferred Node:

```
kubectl label nodes <your-node-name> disktype=<new-label>
(Chosen: kubectl label nodes minikube disktype=ssd)
```

5. View node labels:

```
kubectl get nodes --show-labels
```

6. Update configuration file:

```
nodeSelector:
    disktype: ssd
```

7. Apply the updated configuration:

```
kubectl apply -f hello-deployment_updated.yaml
```