

**GEORGIA INSTITUTE OF TECHNOLOGY  
SCHOOL OF COMPUTER SCIENCE**

**ECE-4150 Spring 2024  
Lab03 : Containerization with Docker**

**Professor:** Dr. Vijay Madiseti (vkm@gatech.edu)

---

**References:**

- [1] A. Bahga, V. Madiseti, "Cloud Computing Solutions Architect: A Hands-On Approach", ISBN: 978-0996025591
- [2] <https://docs.docker.com/>
- [3] <https://docs.min.io/docs/minio-docker-quickstart-guide>
- [4] <https://www.digitalocean.com/community/tutorials/how-to-create-your-first-web-application-using-flask-and-python-3>
- [5] <https://www.twilio.com/docs/usage/tutorials/how-to-set-up-your-python-and-flask-development-environment>

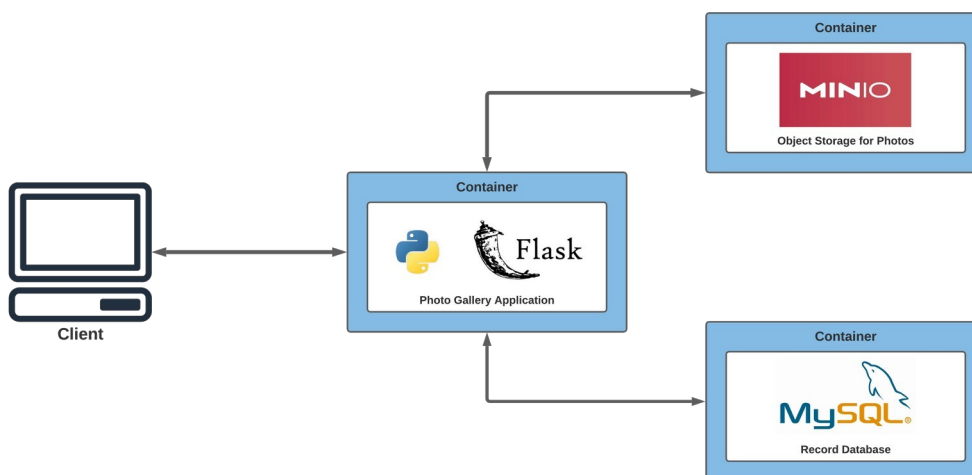
**Due Date:**

The lab report will be **due on Mar 04 11:59PM, 2024**

---

The purpose of this lab is to create a Photo Gallery application using Docker containers. The Photo Gallery application allows users to upload photos, view the details of the images, and search for photos uploaded within the application. The application is implemented in Python and uses Flask Web Framework. There are three main components in this application:

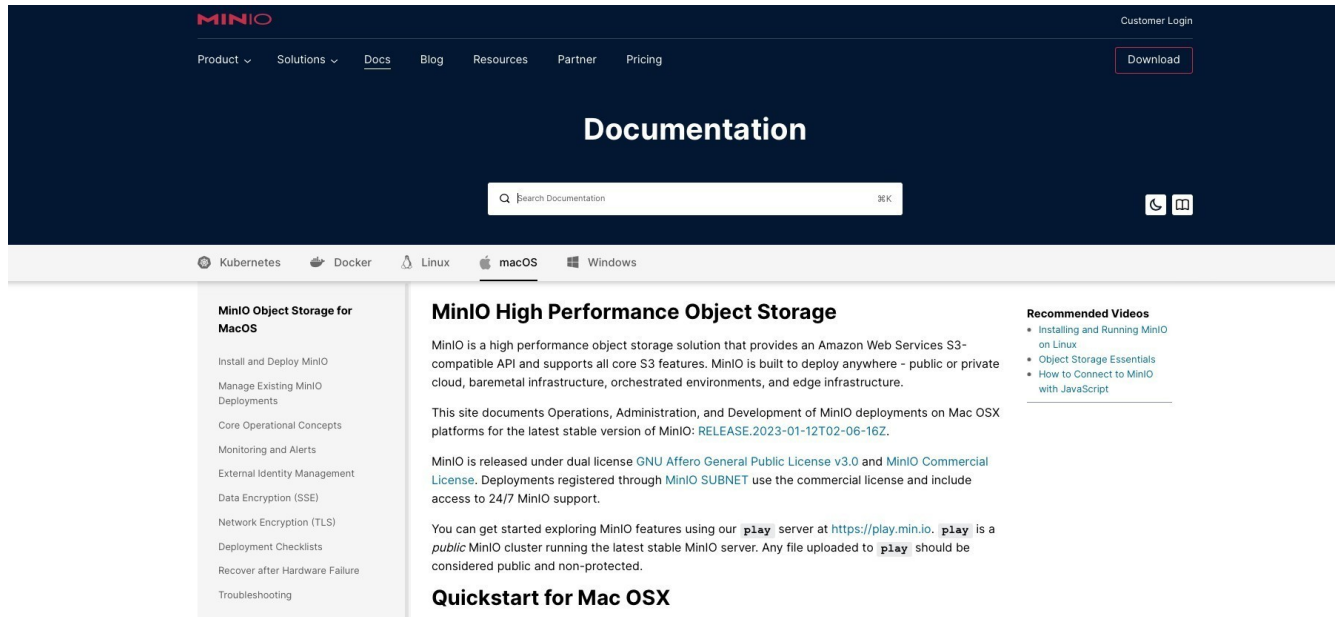
1. Object storage called MinIO uses by the application to store the photos.
2. A relational database using MySQL is used to keep the records of the images.
3. The Python-based application using Flask.



The Flask application, MySQL database, and MinIO storage will be set up in three separate Docker containers. Follow the steps below to set up the Photo Gallery application.

## 1. Install MinIO Server and Client

- To use MinIO, download and install MinIO Server and Client on your computer. Instructions to install these components MacOS, Windows, and Linux can be found in <https://min.io/docs>.



## 2. Run Server MinIO Server and Client

- In a terminal window, create a new local directory `~/minio/data` in your user home directory. Then, run the start the MinIO Server as shown below.

```
$ mkdir -p ~/minio/data

$ minio server ~/minio/data --console-address :9001
```

- In a second terminal window, connect to the MinIO Server using the MinIO Client to create an alias to authenticate and quickly connect to the MinIO Server. Using that alias, add a new user with a set of credentials.
- Tip:** use your complete name + the code and for the password, reverse the order
  - username: JOHNDOECE4150
  - password: ECE4150JOHNDOE

- Create a new group and add the new user to this group. Change the group's policy to allow read and write access for any user in the group.

```
# Create alias to connect to local MinIO Server
$ mc alias set local http://127.0.0.1:9000 minioadmin minioadmin

# Add new user with a set of credentials
$ mc admin user add local JOHNDOECS4150 ECE4150JOHNDOE

# Create a new group and include the new user to the group
mc admin group add local internal-services JOHNDOECS4150

# Change the policy to allow read and write access for users in this group
mc admin policy attach local readwrite --group=internal-services
```

- In a second terminal window, connect to the MinIO Server using the MinIO Client to create an alias to quickly authenticate and connect to the MinIO Server.

### 3. MinIO Storage Container

- Use the following commands to create a Docker container running MinIO object storage. To create a MinIO container with persistent storage, you must map a local directory from the host OS within the container config.
- Use the `~/minio/data` directory created before. Then, start the MinIO container with the `-v` argument to map the local path (`~/minio/data`) to the virtual container directory (`/data`). When MinIO writes data to `/data`, that data is actually written to the local path `~/minio/data` where it can persist between container restarts.

Note: use **sudo** to run the docker command

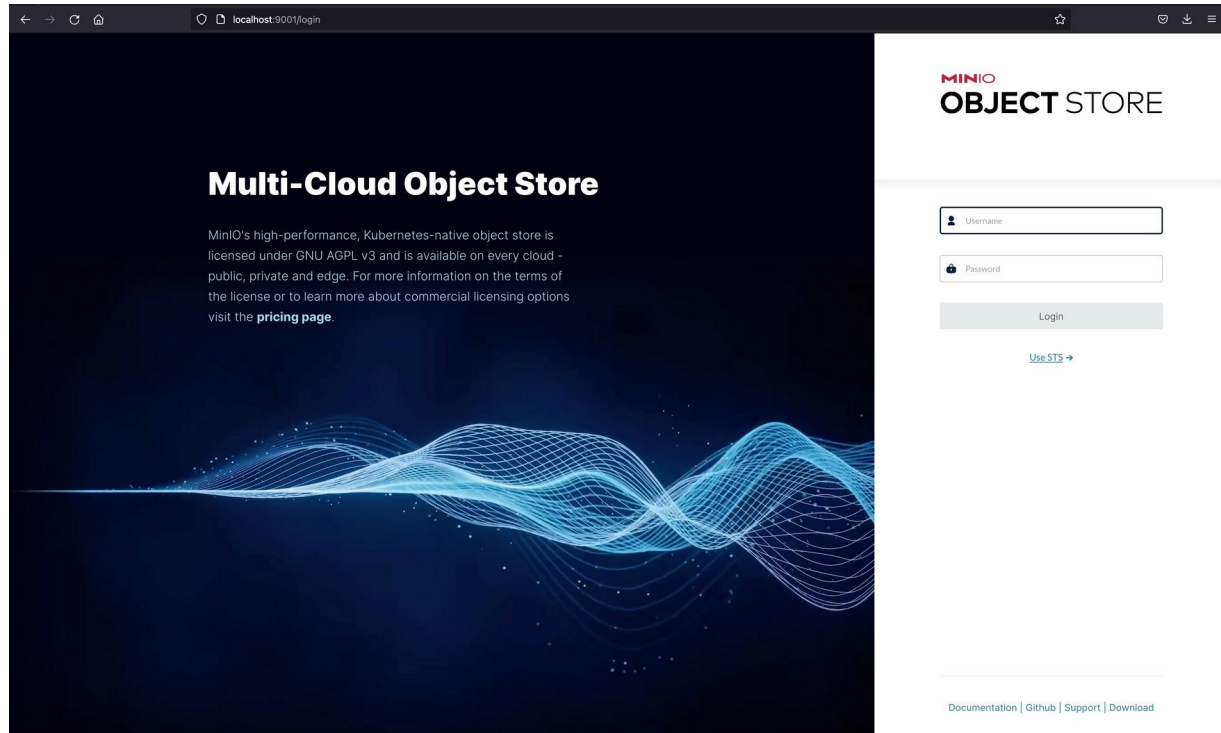
```
$ docker run \
--detach \
-p 9000:9000 \
-p 9001:9001 \
--name minio1 \
-v ~/minio/data:/data \
-e "MINIO_ROOT_USER=JOHNDOECS4150" \
-e "MINIO_ROOT_PASSWORD=ECE4150JOHNDOE" \
quay.io/minio/minio server /data --console-address ":9001"
```

**\*\*\*You cannot run this command with multiple lines for some command line Terminals. If that is the case for your computer, please take each command to one line.**

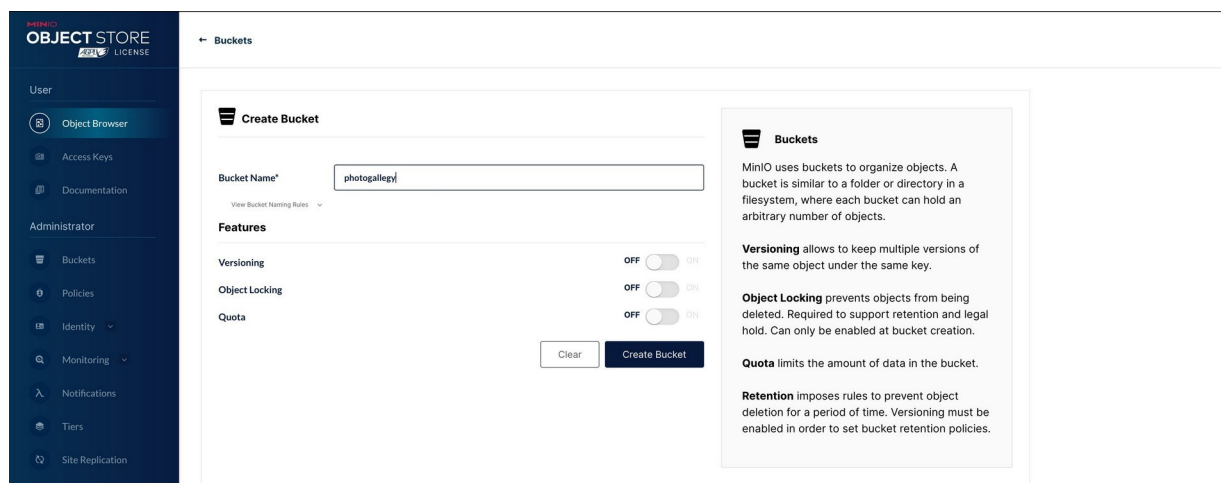
- To see the container running, use the “`docker ps`” command to list the containers. You should see the MinIO container running as shown below

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
c5225cacc2f	quay.io/minio/minio	"/usr/bin/docker-ent..."	About a minute ago	Up About a minute	0.0.0.0:9000-9001->9000-9001/tcp	minio1

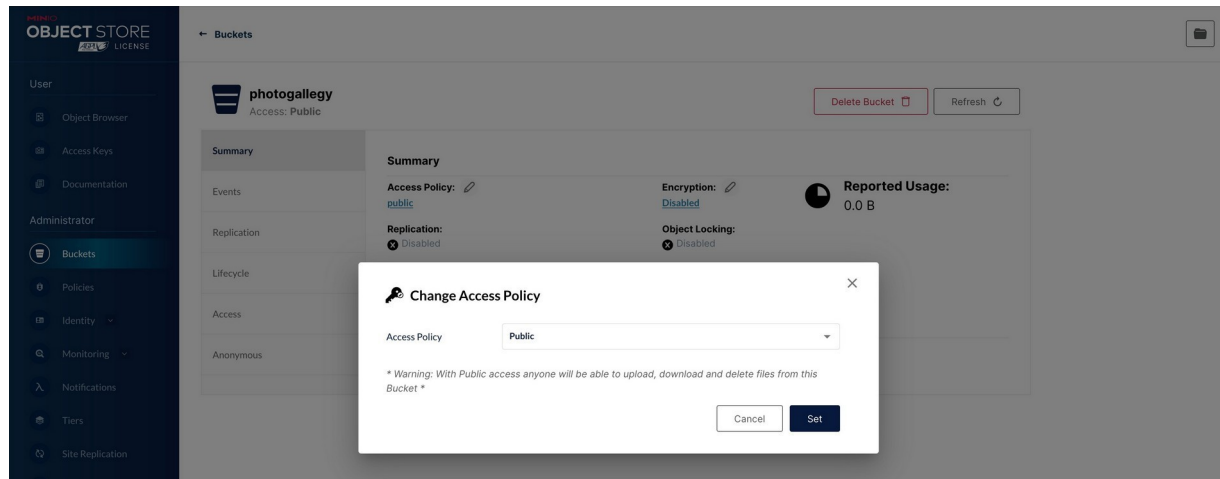
- Browse to URL: <http://localhost:9001/> to access the MinIO console.



- Use the same username and password allocated as environment variables to run the containers.
  - Username: JOHNDOECE4150
  - Password: ECE4150JOHNDOE
- From the MinIO console create a bucket named **"photogallery"**.



- Go to the Buckets page (**click on photogallery**) and under the **Summary** section, edit the **“Access Policy”** by clicking the pencil icon. Change the access policy of the bucket to **public**.



## 2. MySQL database container

- Create a directory for MySQL data on the host (`~/mysql/database/mysql-data`) to map to the container directory (`/var/lib/mysql`). Then, launch the MySQL container using the following command.

```
$ mkdir -p ~/mysql/database/mysql-data

$ docker run \
  --detach \
  --name=database \
  -e "MYSQL_ROOT_PASSWORD=photo123" \
  -e "MYSQL_DATABASE=photodbs" \
  -p 6603:3306 \
  -v ~/mysql/database/mysql-data:/var/lib/mysql \
  mysql
```

**\*\*\*You cannot run this command with multiple lines for some command line Terminals. If that is the case for your computer, please take each command to one line.**

- To see the container running, use the `“docker ps”` command to list the containers. You should see both the MinIO and MySQL containers running as shown below

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
6603e272cf85	mysql	"docker-entrypoint.s..."	About a minute ago	Up About a minute	33060/tcp, 0.0.0.0:6603->3306/tcp	database
c5225cacc2f	quay.io/minio/minio	"/usr/bin/docker-ent..."	29 minutes ago	Up 29 minutes	0.0.0.0:9000-9001->9000-9001/tcp	minio1

### 3. Create virtual environment using virtualenv

- Use the following commands to setup the Python virtual environment and install required packages.

```
# Create python virtual environment with virtualenv utility
$ virtualenv venv

# Activate the virtual environment
$ source venv/bin/activate

# Install the required packages
$ pip3 install -r requirements.txt
```

### 4. Create new database table to store the photo records (**env.py** has ur configuration details)

- Create a new database table to store the photo records using the python scripts below.

```
# Run the flask application
$ python3 utils/photo-table.py
```

### 5. Modify code and test Photogallery Flask application

- Navigate to the util directory and add the username and password created for the MinIO. Then, use the following commands to run the flask application.

```
# Run the flask application
$ python3 main.py
```

- Browse to <http://localhost:5000/>
- You will see the photo gallery application. Add photos and try browsing and searching for photos.

### 6. Containerizing the flask photo gallery application

- Till now, we have created docker containers for MinIO storage and MySQL database and tested the flask photo gallery application running on the host OS, where the flask application connects to the containers running the storage and

database backends.

Next, we will build a Docker image from the flask application source. But before you create the Docker image, browse to <https://hub.docker.com/> and register for a Docker Hub account.

```
# Login into docker
$ docker login

# Browse to the flask app directory and build the docker image
$ docker build -f Dockerfile -t <DOCKERUSERNAME>/photogallery:v1 .

# Now tag the locally created image to the docker hub. This means we have
to tag the image with the docker hub username.
$ docker tag <DOCKERUSERNAME>/photogallery:v1 <DOCKERUSERNAME>/photogallery:v1

#Push the image to docker hub
$ docker push <DOCKERUSERNAME>/photogallery:v1
```

## 7. Launch a container using the photo gallery docker image

- Launch a container using the docker image created in the previous step.

```
$ docker run \
  --detach \
  --name=photogalleryapp \
  --publish 5000:5000 \
  <DOCKERUSERNAME>/photogallery:v1
```

- Use the “docker ps” command to list the containers. You will see three containers running (MinIO, MySQL and Flask photo gallery application).

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
4e5e233b4aea	quay.io/minio/minio	"/usr/bin/docker-ent..."	31 seconds ago	Up 30 seconds	0.0.0.0:9000->9000/tcp	minio1
d8cf9cb63066	mysql	"docker-entrypoint.s..."	40 seconds ago	Up 39 seconds	33060/tcp, 0.0.0.0:6603->3306/tcp	database
1d3e3a3302fb	corporan/photogallery:v1	"python3 /app/main.py"	52 seconds ago	Up 51 seconds	0.0.0.0:5000->5000/tcp	photogalleryapp

- Browse to the URL <http://localhost:5000/> to check if the containerized photo gallery application works and you will notice the following error similar to the image below.

### OperationalError

pymysql.err.OperationalError: (2003, "Can't connect to MySQL server on 'localhost' ([Errno 99] Cannot assign requested address)")

#### Traceback (most recent call last)

```
File "/usr/local/lib/python3.9/site-packages/pymysql/connections.py", line 613, in connect
    sock = socket.create_connection(
File "/usr/local/lib/python3.9/socket.py", line 844, in create_connection
    raise err
File "/usr/local/lib/python3.9/socket.py", line 832, in create_connection
    sock.connect(sa)
```

- Although we have successfully launched the containers for the photo gallery application, storage, and database, we haven't linked the containers. The photo gallery app container doesn't know how to connect to the database and storage containers. We will address this issue in the next step, but before, remove all the container allocated by running the following command:

```
# Remove all running containers
$ docker rm -f $(docker ps -aq)
```

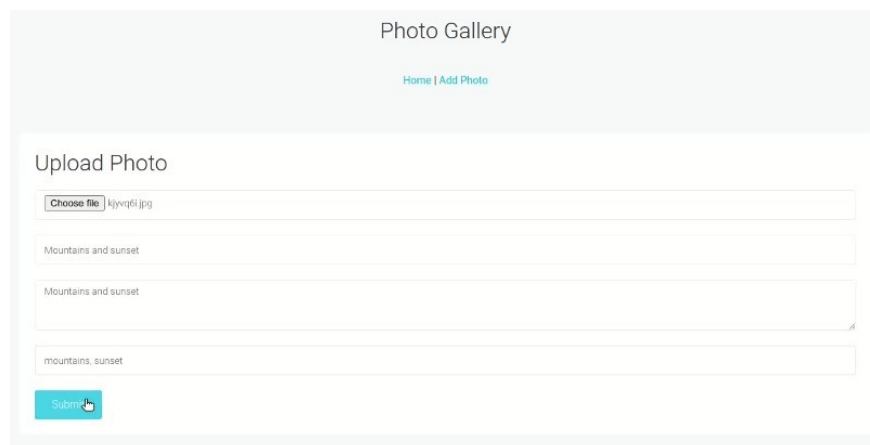
## 8. Docker Compose

- Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, you create and start all the services from your configuration with a single command.
- The Compose file (docker-compose.yml) provided with the lab defines the services that make up the application. Modify the compile provided to included the required credentials.
- This file provides a way to document and configure all of the application's service dependencies (such as database and storage backend). Using the Compose command line tool, you can create and start one or more containers for each dependency with a single command (docker-compose up).

```
# Run the application with singe command
$ docker-compose up

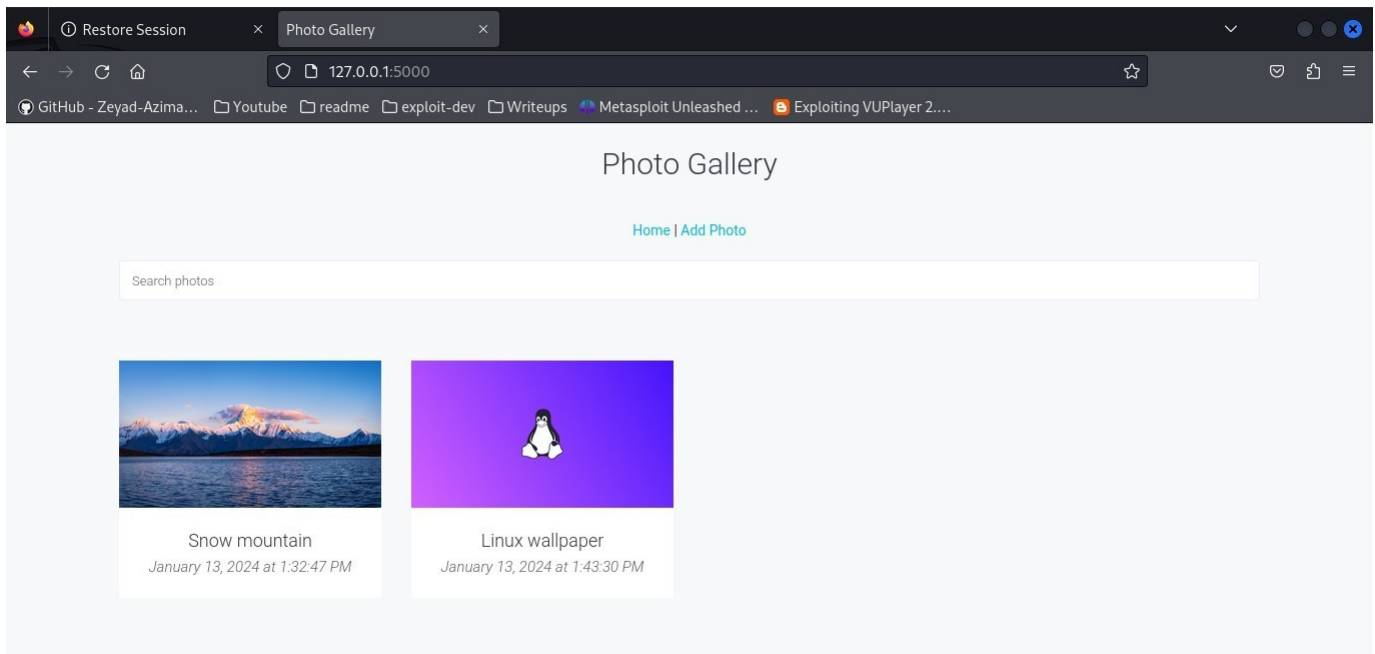
# Stop the application
$ docker-compose down
```

- Browse to the URL <http://localhost:5000/> to check if the containerized photo gallery application works. Add photos and try browsing and searching for images.



The screenshot shows a web application titled "Photo Gallery". At the top, there are links for "Home" and "Add Photo". Below this is a section titled "Upload Photo" which contains a form with four input fields: a file selection field (showing "Choose file" and "klyxq6l.jpg"), a text field (containing "Mountains and sunset"), another text field (also containing "Mountains and sunset"), and a text field (containing "mountains, sunset"). At the bottom of the form is a blue "Submit" button.





**Congratulation! You successfully completed this lab.**

### **Deliverables:**

A proof **video (duration 3-6 minutes. Not more than 6 minutes)** that shows the deployment/shipment of the containers and functions of your website: upload a picture, view the photo, and search for it using the search bar. Also, include a copy of your **docker-compose.yaml** file completed.