Assignment: Image Upload API with AWS LocalStack, DynamoDB, S3, and Testing

---

1. Overview

This project implements an image management system using AWS services:

- **S3**: Stores image files.
- **DynamoDB**: Stores image metadata (file name, unique ID, created_at).
- **Lambda Functions**: Handle CRUD operations (upload, list, get, delete).
- **LocalStack**: Emulates AWS services locally for development.
- **Pytest + Moto**: Unit and integration testing.

The API supports:

- Uploading images (POST)
- Listing images (GET)
- Fetching images by ID (GET)
- Deleting images (DELETE)

---

2. Project Structure

```
monty-cloud-assignment/
│
├── Dockerfile.localstack
├── docker-compose.yml
├── handler.py              # Lambda handlers for CRUD operations
├── setup-localstack.sh     # Script to setup LocalStack resources
```

```
├── requirements.txt       # Python dependencies (boto3, pytest, moto, etc.)
├── tests/
│   └── test_handler.py    # Pytest test cases
├── logs/                  # LocalStack logs
└── handler.py             # Mounted handler code inside container
```

---

3. Lambda Handlers Overview

The Lambda functions perform CRUD operations with the following key considerations:

- **Upload**:
    - Accepts file name, base64-encoded file content, and optional metadata.
    - Generates a unique ID for each upload.
    - Saves the file to S3 and metadata to DynamoDB.
- **List Images**:
    - Scans the DynamoDB table and returns all image metadata.
    - Supports optional filtering by file name or creation date.
- **Get Image**:
    - Retrieves the image metadata from DynamoDB using the unique ID.
    - Fetches the corresponding file content from S3.
- **Delete Image**:
    - Deletes the file from S3 and metadata from DynamoDB.
    - Handles repeated deletions gracefully.
- **Error Handling**:
    - Returns proper HTTP codes: 2xx for success, 4xx for client errors, and 5xx for server errors.

---

4. Docker Setup

docker-compose.yml:

- Runs LocalStack in a container.
- Exposes the edge port for API requests.
- Mounts local files for handler code, setup script, and tests.
- Includes Docker socket to allow Lambda to run inside LocalStack.
- Installs dependencies (boto3, pytest, moto, awscli-local).

setup-localstack.sh:

- Creates the S3 bucket.
- Creates the DynamoDB table with a unique ID as the primary key.
- Ensures LocalStack resources are ready for Lambda invocation.

Ensure the setup script has execution permissions.

---

5. Testing with Pytest

- **Fixtures**:
    - Sets up environment variables (`BUCKET_NAME`, `TABLE_NAME`).
    - Mocks AWS resources using Moto (`mock_aws`).
- **Test Coverage**:
    - **Success Cases**: Upload, list, get, delete.

    - **Failure Cases**: Invalid HTTP method, missing fields, invalid base64, non-existent IDs.
    - **Edge Cases**: Large payloads, double deletion, DynamoDB or S3 failures.
    - **HTTP Codes**: Validates 2xx, 4xx, and 5xx responses.
- **Notes on Moto**:
    - Moto's `mock_aws` decorator mocks multiple services in one context.
    - Always specify a region to avoid `NoRegionError`.

---

6. Running the Project

Start LocalStack

- Use Docker Compose to start LocalStack in detached mode.

Invoke Lambda Functions Locally

- Use `docker exec -it localstack sh /app/setup-localstack.sh` to test Lambda functions directly.

Run Tests Inside Container

- Use `docker exec -it localstack pytest -v /app/tests/test_handler.py` to execute test cases inside the LocalStack container.

---

7. Notes & Recommendations

1. **Unique IDs**:
    - Use UUIDs to avoid collisions for parallel requests.

    - Guarantees uniqueness for high-concurrency uploads.

2. **Error Handling**:
    - Return clear HTTP 4xx/5xx messages.
    - Handle missing files, missing IDs, and invalid payloads.
3. **Testing**:
    - Cover success, failure, edge, and large payload scenarios.
    - Always mock AWS services for unit testing to avoid hitting real endpoints.
4. **LocalStack Considerations**:
    - Mount `/var/run/docker.sock` to allow Lambda Docker execution.
    - Ensure endpoint URLs are correctly set when calling S3/DynamoDB.
5. **Environment Variables**:
    - Use `BUCKET_NAME`, `TABLE_NAME`, `LOCALSTACK_ENDPOINT` in code and tests.
6. **Performance**:
    - DynamoDB primary key is `id` to ensure efficient lookups.
    - Use `scan` carefully; for large datasets consider `query` with indexes.