

The task is to build a simple weather API service using FastAPI that fetches weather data from an external public API like OpenWeatherMap. The service should handle high traffic using asynchronous programming and interact with AWS-like services to store and retrieve weather data. If AWS services are not available, local equivalents can be used, ensuring that these interfaces can be easily replaced with any other provider.

Requirements:

1. **FastAPI Setup:** Create a FastAPI application with a single endpoint `/weather` that accepts a `GET` request with a `city` query parameter.
2. **Asynchronous Data Fetching:** Use Python's `asyncio` to fetch the current weather data asynchronously from the external API based on the `city` parameter. Implement proper error handling for potential API failures or invalid city names.
3. **AWS S3 (or Local equivalent) Integration:** Store each fetched weather response as a JSON file in an S3 bucket or a local equivalent. The filename should be structured as `{city}_{timestamp}.json`. Use asynchronous methods to upload the data.
4. **AWS DynamoDB (or Local equivalent) Integration:** Log the event (with city name, timestamp, and S3 URL/local path) into a DynamoDB table or a local equivalent asynchronously after storing the JSON file. Ensure that database interactions are performed using async methods.
5. **Caching with S3/Local Equivalent:** Before fetching the weather data from the external API, check if the data for the requested city (fetched within the last 5 minutes) already exists in S3 or the local equivalent. If it exists, retrieve it directly without calling the external API. Implement a mechanism to expire the cache after 5 minutes.
6. **Deployment:** Provide deployment scripts (like `Dockerfile`, `docker-compose.yml`, etc.) in the repository.

Evaluation Criteria:

- Correctness: The solution should work as specified.
- Code Quality: Clean, readable, and maintainable code.
- Design Patterns: Appropriate application of design patterns.
- Asynchronous Implementation: Proper use of asynchronous features in Python.
- AWS/Local equivalent Integration: Effective use of AWS S3/DynamoDB or local equivalents.
- Error Handling: Robust error handling and logging.
- Deployment/Local Setup: Ability to run the service locally or deploy to AWS.

Submission:

- Provide a GitHub repository link with the source code, including a README file with setup instructions.
- Ensure the README includes instructions on how to run the application locally or deploy it to AWS

If you need any further details or assistance, feel free to ask!