# Local-Only AWS Development Strategy

This document summarizes a practical approach for running and testing a full AWS-style web application stack entirely on your local machine using Docker containers. It allows you to mirror a production AWS architecture (with Cognito, DynamoDB, S3, and RDS) using local, containerized equivalents — ideal for offline development, teaching, or CI environments.

## 1. Goals

1 Develop and test a full-stack AWS-style system locally without incurring cloud costs.
2 Maintain architectural parity with production (Auth, storage, databases, messaging).
3 Keep setup simple and reproducible for students or teammates.

## 2. Local Service Equivalents

- **Cognito → Dex**: Use Dex as a local OIDC identity provider. Runs as a container exposing /auth endpoints compatible with Cognito flows. - **RDS (PostgreSQL) → PostgreSQL container**: Standard `postgres` image for relational data. - **DynamoDB → DynamoDB Local**: Official AWS image `amazon/dynamodb-local`. Fully API-compatible, accessed via `http://localhost:8000`. - **S3 → MinIO**: S3-compatible local object storage, accessible at `http://localhost:9000` (API) and `http://localhost:9001` (UI). - **SNS/SQS → Direct HTTP calls**: Replace asynchronous message passing with synchronous REST calls between services for simplicity.

## 3. Architecture Overview

Your local environment mirrors the cloud structure but runs entirely via Docker Compose: - **Login Service** authenticates via Dex and writes to its own DynamoDB instance. - It then sends user creation/update messages directly (HTTP POST) to the User Service. - **User Service** receives these events, stores them in PostgreSQL, and may upload files to MinIO. This pattern maintains the same logical separation and data flow as the cloud version, only with simpler, containerized connections.

## 4. Docker Compose Components

A typical `docker-compose.yml` would include: - dex (auth) - postgres (relational DB) - dynamodb-local (NoSQL DB) - minio (object storage) - login-service (Node/Express) - user-service (Node/Express) All services communicate via Docker network using environment variables such as: ``` USER_SERVICE_URL=http://user-service:4000/internal/user-update DYNAMODB_ENDPOINT=http://dynamodb:8000 S3_ENDPOINT=http://minio:9000 ```

## 5. Local Messaging (Option A: Direct HTTP)

In local mode, replace SNS/SQS messaging with a direct HTTP call from the Login Service to the User Service. Example: ```ts if (process.env.LOCAL_MODE === 'true') { await axios.post(process.env.USER_SERVICE_URL, eventData); } else { await snsClient.publish(...); } ``` This preserves the event-driven logic without introducing extra infrastructure. It’s simple, transparent, and perfect for local teaching environments.

## 6. Advantages of This Approach

1. Complete offline development capability.
2. Identical API surfaces to AWS (DynamoDB Local, MinIO).
3. No additional costs or IAM setup.
4. Simple, reproducible with Docker Compose.
5. Straightforward transition to real AWS in production.

## 7. Next Steps

Once local development is stable, students can transition seamlessly to AWS by: - Switching Dex → Cognito - DynamoDB Local → DynamoDB - PostgreSQL container → RDS - MinIO → S3 - HTTP calls → SNS + SQS This gradual migration reinforces best practices in cloud architecture and Infrastructure-as-Code.