Simple HTTP Relay Server with a WebApp and CI/CD Design HLD Part 3

Server CI/CD Pipeline

1. Introduction

We want to have a simple user friendly web application that allows us to access messages over a web server. This design has three components:

1. A backend server for sending and getting messages on the cloud
2. A web-application with an interface to the backend server
3. ***(this document) A CI/CD pipeline that automatically deploys after development workflow***

This document will focus only on the design and considerations of the pipeline and deployment environments.

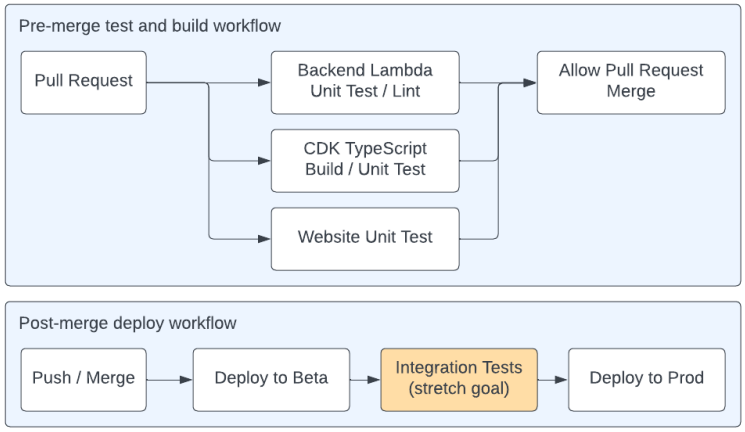
1.1 Glossary

Term	Definition
Continuous Integration (CI)	A continuous integration pipeline is a series of steps that automates the process of building and testing code. This helps to ensure that code is of high quality and that it can be deployed quickly and reliably.
Continuous Deployment (CD)	A process that automatically deploys code to production after it has been successfully built and tested.
NixOS	A package manager that boasts highly reliable and cross platform uniformity that installs packages in isolation.
AWS	Amazon Web Services, a cloud computing platform that offers a wide range of services, including computing, storage, networking, databases, analytics, and machine learning.
CDK	A set of tools that allow you to define AWS resources using code.

2. Pipeline Design

2.1 High Level Design

Our full continuous integration pipeline will consist of two workflows: a pre-merge workflow and a post-merge workflow. The pre-merge workflow is responsible for testing and linting the code before it is merged. The post-merge workflow is responsible for deploying the approved code to the corresponding environments.



The deployment workflow will include two AWS environments, both managed by the same account. The development environment will be deployed to the region us-west-2, and the production environment will be deployed to us-east-1.

2.2 Alternatives / Considerations

1. **Github Workflows over Code Pipeline:** While all the server assets are defined and deployed within the AWS ecosystem, the repository is currently managed in github. Using a code pipeline would break up the development workflow unnecessarily.
2. **Disallowing Merge unless pipeline tests succeed:** While this can sometimes get in the way of a rapid feature push, we can go out of the way to disable it for a specific commit. Disallowing will make broken merges less common.
3. **Using the same AWS account for each stage:** Normally it is better to have a separate AWS account for each stage because it's easier to manage permissions for prod and beta accounts separately. In this case we're making a demo so there's no need to have two accounts. If we choose to, we can swap the AWS credentials in git to deploy to a different account and change client api endpoints and there shouldn't be any problems.
4. **us-west-2 and us-east-1 as deployment regions:** Amazon internally has regular practices around beta and production environments, and therefore AWS has some hidden preference for deployment regions. Amazon internally uses us-west-2 for nearly all dev/beta accounts and us-east-1 for north american production environments.

2.3 Detailed Design

2.3.1 Cloud Formation Stack

2.3.1 New Variables

The cloud formation stack will need to take in environment variables on deployment to specify which account, region, and stage the resources are being deployed to:

- AWS_ACCOUNT
- AWS_REGION
- AWS_STAGE

2.3.2 Pre-merge Workflow

2.3.2.1 Commands

The pre-merge workflow will run three main jobs that get their primary functionality from the following commands:

1. Python linting and testing
    a. `pylint ./lambda/src ./lambda/tests`
    b. `python -m unittest {tests}`
2. TypeScript CDK Build / Test
    a. `tsc`
    b. `jest`
3. Website Unit tests
    a. `jest ./website`

We'll use NixOS to install the commands so that the pipeline is virtually identical to the development build environment. Nix is a package builder that boasts highly reliable and cross platform uniformity that installs packages in isolation, which means whatever happens in the pipeline will be identical to what happens locally. I'll include instructions on setting up nix after I have deployed the pipeline and all aspects work as expected.

- **NixOS Website:** **https://nixos.org/**
- **NixOS Package Search:** **https://search.nixos.org/packages**

2.3.2.2 Github Actions / Workflows

Note that since we're using nix to download packages we won't be using other actions that install the packages themselves.

1. actions/checkout@v3
2. workflow/nix-shell-action@v3
3. (...Possibly more)

2.3.2.3 Alternatives / Considerations

1. **Using Nix instead of simple commands :** While using NixOS will add some overhead to pipeline development, it'll make replicating build results locally trivial.

2.3.3 Continuous Deployment Workflow

2.3.3.1 Commands

The continuous deployment workflow will use a nix curated version of `cdk deploy` within the deployment job to deploy to AWS. Github will store manually created credentials that will allow it to launch a deployment against the AWS account and region, and these credentials will only need to be set up once.

2.3.3.2 Github Actions / Workflows

Same as pre-merge pipeline.

2.3.3.3 Alternatives / Considerations

1. **Using cdk deploy as a command instead of CodeDeploy:** CodeDeploy runs configurable deployment rollouts of assets stored in S3. We aren't interested in **just** rolling out an executable. We're interested in deploying AWS resources, S3 Website files, and lambda assets all while publishing changed assets to the cloud. Using cdk deploy will not only handle everything, but it will better recreate what we test locally.

3. Workflow Development Timeline

Task	Estimated Dev Time	Loose Deadline
Research and Document Plan	(this document, already done) 5 hours	Completed around: 06/02 11:00 AM EST
Update cloud formation stack to take environment variables.	45 minutes	06/02 12:00 AM EST
Add jest tests for website javascript	1 hour	06/02 1:30 PM EST
Locally use Nix for all build requirements and record build process to replicate in github workflows	2 hours	06/02 3:30 PM EST
Create mini workflow in private repo using some of the nix commands	2 hours	06/02 5:30 PM EST
Update any code that doesn't pass linter requirements.	1 hour	06/02 6:30 PM EST
Launch pre-merge pipeline	3 hours	06/03 11:00 AM EST
Push post-merge pipeline.	2 hours	06/03 1:00 PM EST

Appendix

Project Requirements

In a single mono repo:

1. Develop a simple HTTP relay server using a programming language of your choice. The server should have three functionalities:
    - Receive a message, assign a unique number to it, and return the number.
    - Accept a unique number and return the corresponding message.
    - Display the total number of messages stored on the server.
2. Create a user-friendly web app for the server, enabling users to:
   View the total number of messages on the server.
    - Submit a new message and receive a unique number for it.
    - Enter a unique message number to retrieve the corresponding message.
3. Implement Continuous Integration (CI) and Continuous Deployment (CD) for the project. Ensure that the CI tests the applications.
4. Thoroughly document the deployment process, detailing how developers can deploy, debug, and test the project using local or isolated environments.

Assume that this repo will serve as a starting point for a team of developers and independent contributors. **Focus on providing the basic functionalities without incorporating additional features.** However, consider various aspects of the project's future development and best practices. Make provisions for the project's potential evolution and scalability, as its direction remains uncertain. Include these considerations in the documentation.