Simple HTTP Relay Server with a WebApp and CI/CD Design HLD Part 2

Frontend Design

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. ***(this document) A web-application with an interface to the backend server***
3. A CI/CD pipeline and development environment that makes development easier

This document will focus only on the design and considerations of the frontend.

2. Frontend Design

2.1 High Level Design

To access the webapp we'll use an S3 bucket to host the core website in a single index.html file. This website will have permissions based on limited iam User credentials stored in code. We will use these credentials to get an access token for the api calls and show the api call results in text boxes next to buttons on the website.

2.2 Alternatives / Considerations

1. **Using an S3 Bucket for the Web App Host instead of the root of the API Gateway:** Some of the core cdk constructs for hosting a website from S3 are not yet supported idiomatically in CDK v2. It's very possible to link an API to an S3 structure, but for now that's not necessary and would bloat the final solution. This can be a stretch goal.
2. **Using an iam User with api permissions for a first draft instead of SaaS linked to a domain name:** The correct way to do permissions for an api with a web app is to use the SaaS credential system offered by AWS. For the purposes of this demo I'll use an iam User with hardcoded credentials that are stored within code. These credentials will allow the user to read and write to the api gateway api, which is relatively low risk since the website already allows that. *In the worst case the api will throttle if someone misuses these credentials.*

2.3 Detailed Design

2.3.1 Front End Visuals

The website will include three buttons on a gray background one after the other.

1. Create a message
2. Read a message
3. Show the number of messages.

When the buttons are pressed or an input is put in, the website will call the message api and fill in the details next to the button.

2.3.2 Front End Credentials

As mentioned above, the cloud formation stack will generate a user that can only access the api gateway calls for the message api. The public and private key of this user will be hardcoded into the index file as a first-draft. The website will use these to get an access token from api gateway so that it can make the api calls.

2.4 Follow-up Work

1. **Use SaaS specific credentials for the website:** While leaking credentials with limited access is relatively low risk, it would be better to use SaaS specific credentials.
2. **Host the website in a more robust environment:** Using an S3 bucket directly only allows for a single index.html file to represent the website. If we want to grow the website to include subpages we should link the bucket to ApiGateway.

3. Frontend Development Timeline

Because this is an interview I've included conservative estimates for the time it will take for each stage of development.

Task	**Estimated Dev Time**	**Loose Deadline**
Research Project and Document Plan	4 hours	05/31 3:00 PM EST
Locally create basic website	1 hour	05/31 4:00 PM EST
Update cloud formation file to publish website and create credentials	3 hours	05/31 Midnight EST
Push cloud formation changes with basic index.html	1 hour	06/01 11:00 AM EST
Push full web app	2 hours	06/01 11:00 AM 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.