Fast Path to AWS Serverless Applications

Sakina Shaikh

Jim Ladd

May 12, 2021

Contents

Introduction

Serverless technologies represent a relatively new and very exciting paradigm in software development. While the name is misleading…Yes, Virginia there is a server…the software development team is no longer burden with the provisioning and management of servers, containers, etc. With serverless technologies, the team can focus on what is most important…adding business value.

As SOFWERX began the transformation of its public-facing web site from WordPress to serverless, the challenge of staff self-training arose. After some online research, we found several tutorials available for serverless development, ranging from the trivial to the overwhelming. The ideal tutorial had to be straightforward but complete and include our specific technologies. While no examples were perfect, we did select Serverless Stack, an open-source guide for building full-stack serverless applications [1]. The scope of the problem domain was perfect. The guide creates a notes management application that includes the full lifecycle of a simple object. We decide to use this guide but alter it to suit our needs.

This project uses popular technologies and common AWS services. The client project uses Nodejs, React, Amplify, and CloudFront while the service project utilizes Cognito, API Gateway, Lambda, DynamoDB, and Python. The two projects are maintained in GitLab and uses its CI/CD pipeline feature. This project deviated from the Serverless Stack guide in the use of GitLab instead of GitHub and Python instead of Nodejs for the Lambda language.

Background

The Serverless Stack tutorial features a simple but effective notes management solution. A user is able to sign up for the service with an email verification step. Once registered, the user is presented with a list of existing notes. These notes represent text data along with the option of an attachment file. The user may create a new note, edit an existing note, and delete a note. The home page of the application appears as:

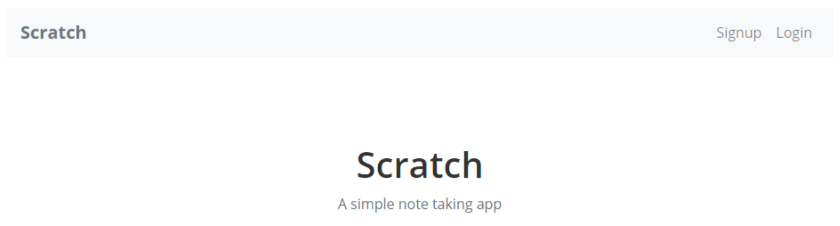


Figure 1 - Home page

The application incorporates user authentication as a core feature. To gain access beyond the home page, the user must first register.

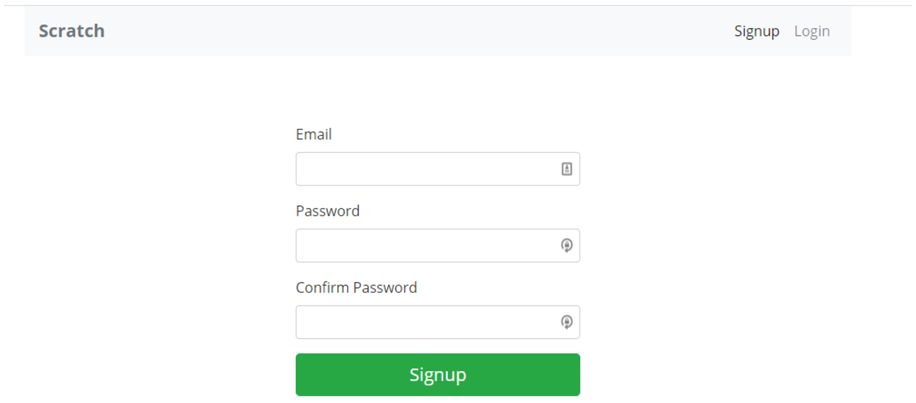


Figure 2 - Registration page

AWS provides email and/or text confirmation of the user's information. This application uses the email option.

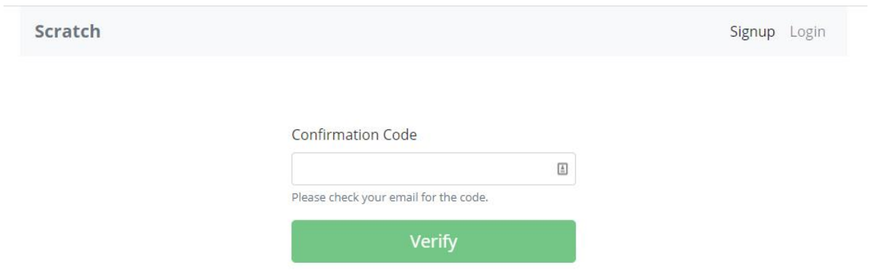


Figure 3 - Email confirmation

Once confirmed, the user may access the core functions of the application.

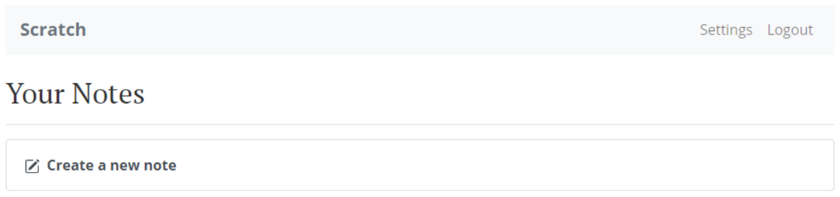


Figure 4 - Main page

The user may create a new note and attach a file. The text for the note is saved to a DynamoDB table and the attached file is uploaded to an S3 bucket.

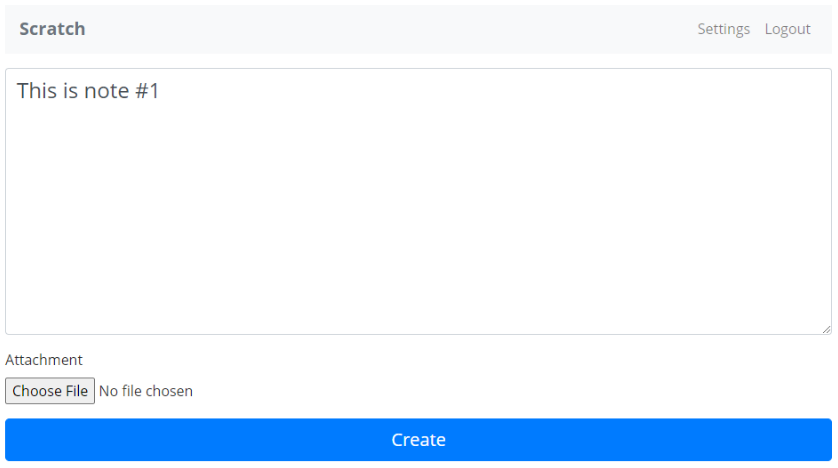
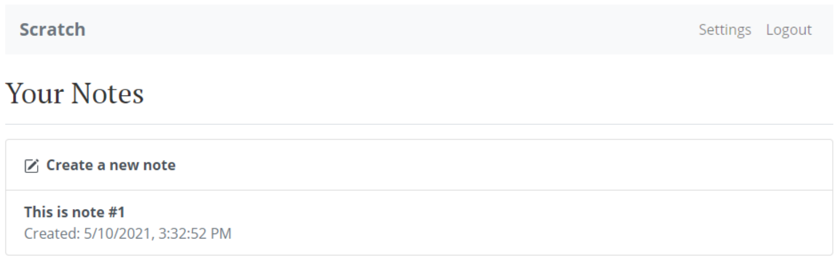


Figure 5 - Create page



This is note #1
Created: 5/10/2021, 3:32:52 PM

Figure 6 - Newly created note

The user may also delete a note.

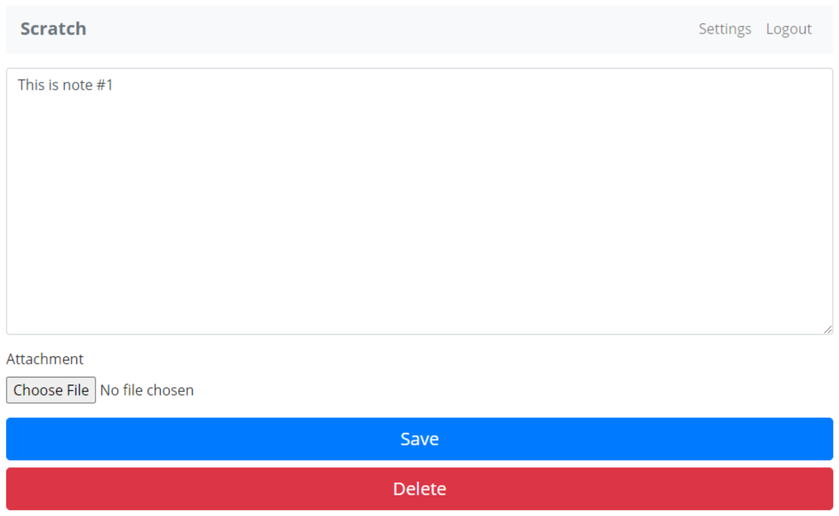


Figure 7 - Delete note page

Architecture

One of the driving factors of this project is to utilize the relevant AWS serverless technologies within a simple but useful architecture. The major components of this architecture include:

- ***React*** – A common JavaScript framework [2].
- ***Amplify*** – This is a set of tools to ease the development of mobile and web applications [3]. This project uses Amplify to send the HTTPS requests to the API Gateway.
- ***Cognito*** – This service facilitates the process of user sign-up, sign-in, and access control [4].
- ***S3*** – The Simple Storage Service is AWS's object storage service [5].
- ***CloudFront*** – This service provides a fast content delivery network [6].
- ***API Gateway*** – The gateway service provides a "entry point" to the other AWS services [7].
- ***Lambda*** – This service allows code to be executed on demand and without managing servers [8]
- ***DynamoDB*** – This is a key-value and document high performance database [9].

A diagram of this architecture is shown below:

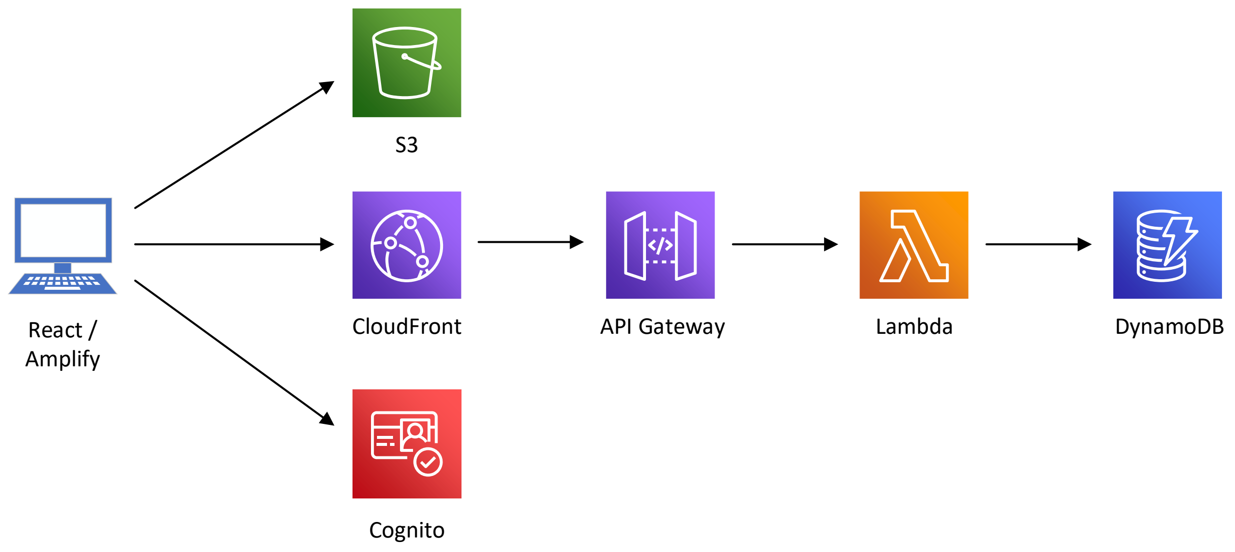


Figure 8 - Architecture components

This architecture is partitioned into two projects. The *serverless-stack-api* (or API) project is the "backend" that provides different services. The *serverless-stack-client* (or Client) project is responsible for providing a user interface for human operators. The artifacts for these projects are maintained in GitLab repositories. Additional information is provided in the *Repositories* section of this document.

API Project

The API project provides services to support the life-cycle of the "notes" object. These services include creating, updating, deleting, and retrieving note instances for a given user. The artifacts for this project are maintained in GitLab. The project relies on GitLab's CI/CD service to build and deploy the project when a push is executed. The build and deploy processing utilizes the Serverless Stack Toolkit (SST) which is an extension of the AWS Cloud Development Kit (CDK).

When the API repository is cloned from GitLab, the file structure will appear as below:

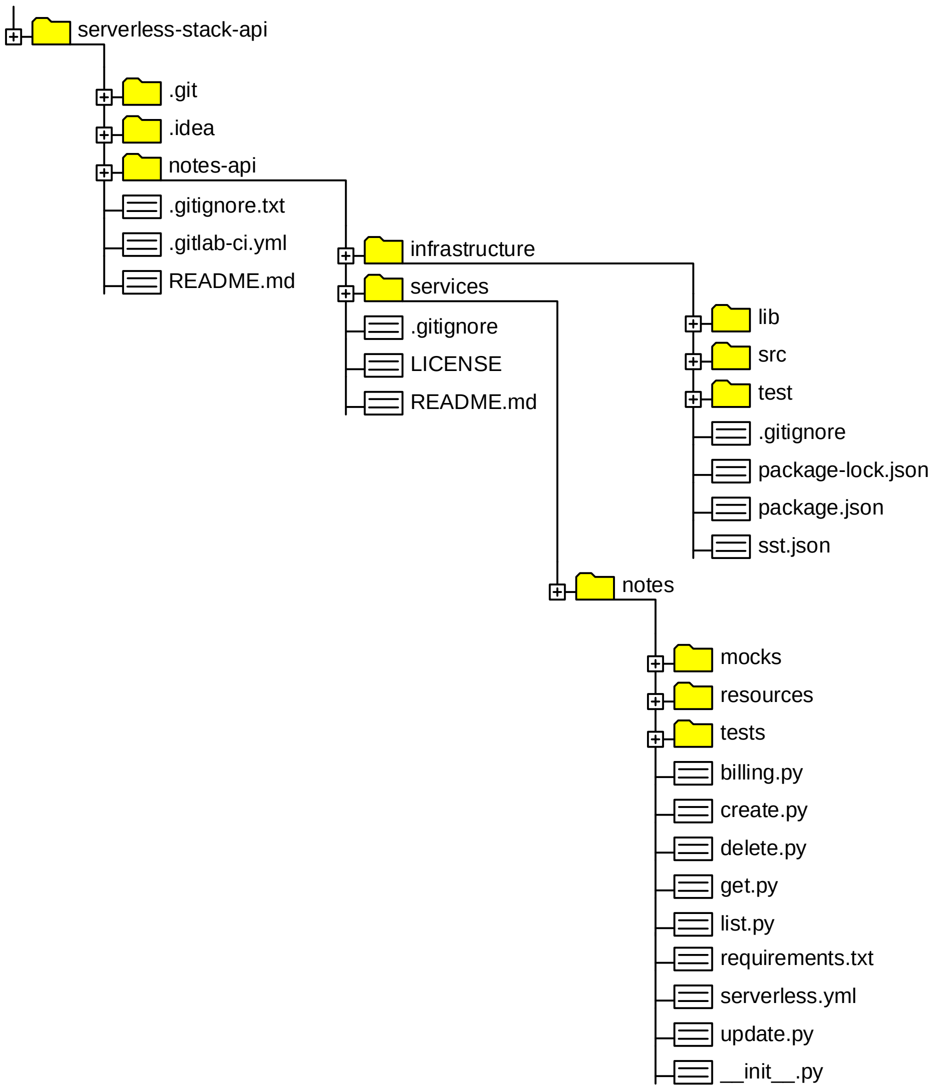


Figure 9 - severless-stack-api file structure

The key files within this API project are described below:

- *.gitlab-ci.yml* – This is the script for the GitLab CI/CD pipeline. It will be executed every time a push is made to the repository.
- *sst.json* – This file configures the high level information for the project including the name of the application, the default stage, and the target AWS region.
- *serverless.yml* – This file contains the configuration information for the infrastructure.
- *create.py, get.py, etc.* – The Python files represent the “logic” that will be executed as Lambda expressions when a request is received.

Only the *sst.json* and *serverless.xml* files need to be modified before the GitLab pipeline is executed. The region value in these files need to reflect the target AWS region. The default value is "us-east-1". All of the other configuration for the API project is performed programmatically.

One area of the API project that should be highlighted is the configuration files for the different infrastructure components. While they do not need to be modified for this project to be built and deployed, they will probably be changed as this project is used for other, real-world applications and more human friendly names and ids are desired. These configuration files are show in the diagram below:

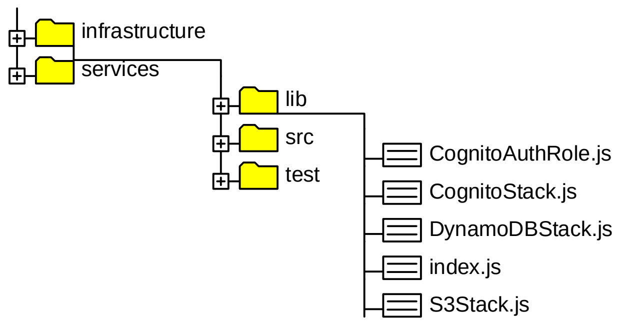


Figure 10 - Infrastructure configuration files

Client Project

The Client project provides an interface for humans to interact with the notes objects. The project is a React-based application that invokes the REST-based services provided by the API project. It relies on the common *npm* utility for package installation, version management, dependency management. *npm* relies on AWS's API for building and deploying the client artifacts.

When the Client repository is cloned from GitLab, the file structure will appear as below:

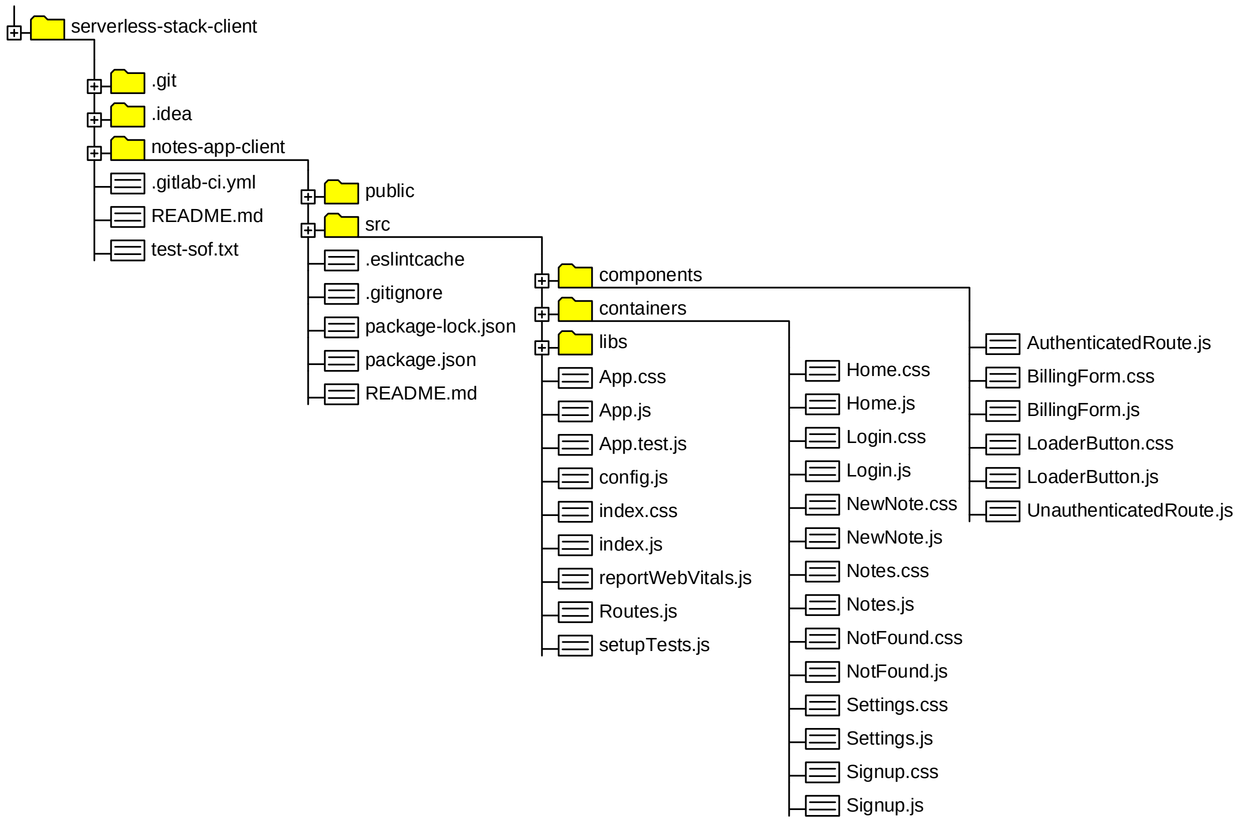


Figure 11 - severless-stack-client file structure

The Client project is structured like most React applications. The key files that are relevant to the serverless stack are described below:

- *.gitlab-ci.yml* – This is the script for the GitLab CI/CD pipeline. It will be executed every time a push is made to the repository.
- *config.js* – This file stores the information for the backend resources like the API Gateway, Cognito, and S3.
- *package.json* – The module dependencies are maintained by this file along with instructions for building and deploying the application.

The *config.js* and *package.json* files will be updated as needed during the manual configuration steps described in the next section.

Deployment

One of the goals of this effort is to create an easy build and deploy process. For the *serverless-stack-api* project, the configuration consists of two manual steps. First, an IAM user must be created in AWS that has the permissions to perform the build and deployment steps. The second step is to enter the access keys of this user into the GitLab CI/CD pipeline. This user should be given the following permissions:

- AmazonS3FullAccess
- CloudFrontFullAccess
- AmazonDynamoDBFullAccess
- AdministratorAccess
- AmazonAPIGatewayAdministrator
- AmazonSQSFullAccess

The values of the AWS Access Key and the AWS Secret Access key should be saved so they can be entered into GitLab.

When the *serverless-stack-api* codebase is checked into a new GitLab project, the CI/CD pipeline will execute the *.gitlab-ci.yml* file. It will fail due to the lack of the IAM user access keys. To rectify this condition, follow these steps to manually configure the CI/CD pipeline:

1. In GitLab, navigate to the *serverless-stack-api* -> *Settings* -> *CI/CD* -> *Variables* section.
2. Create a new variable with the name of AWS_ACCESS_KEY_ID and enter the value for the IAM user.
3. Create a new variable with the name of AWS_SECRET_ACCESS_KEY and enter the value for the IAM user.
4. Navigate to the *serverless-stack-api* -> *Pipelines* page and restart the failed pipeline. This time it should run successfully.

The default AWS region is set to “us-east-1”. If you are targeting another region, the *serverless-stack-api/notes-api/infrastructure/sst.json* file and the *serverless-stack-api/notes-api/services/notes/serverless.yml* file must be updated with the desired value. The remainder of this example uses the “us-west-2” region so these files should be modified.

The rest of the manual configuration involves the *serverless-stack-client* project. The first task is the create a S3 bucket in AWS that will host the static content of the web app for the AWS CloudFront service.

1. Navigate to the S3 service in the AWS console.
2. Create a new S3 bucket.
3. Copy the ARN value for the bucket, for example *arn:aws:s3:::sofwerx-serverless-client*
4. In the *Permissions* section, ensure the **Block public access to buckets and objects granted through *new* public bucket or access point policies** is unchecked.
5. Ensure the **Block public and cross-account access to buckets and objects through *any* public bucket or access point policies** is unchecked.
6. Edit the *Bucket policy* and add the following JSON data:

```
{
  "Version":"2012-10-17",
  "Statement":[{
      "Sid":"PublicReadForGetBucketObjects",
            "Effect":"Allow",
       "Principal": "*",
      "Action":["s3:GetObject"],
      "Resource":["arn:aws:s3:::sofwerx-serverless-client/*"]
    }
  ]
}
```

7. Navigate to the *Properties* configuration for the bucket.
8. Scroll to the *Static website hosting* section.
9. Enable static web hosting.
10. Enter *index.html* for the Index Document and Error Document fields.
11. Copy the URL in the *Endpoint* field:
    (i.e., http://sofwerx-serverless-client.s3-website-us-west-2.amazonaws.com)

The next task is to create a content distribution within the AWS CloudFront service.

1. Navigate to the CloudFront service.
2. Click on the *Create distribution* button.
3. Enter the Endpoint URL of the S3 bucket into the *Origin Domain Name* field.
4. Scroll down to the *Compress Objects Automatically* field and select the *Yes* option.
5. In the *Default Root Object* field, enter "index.html".
6. Click the Create Distribution button at the bottom of the page.
7. Navigate to the details of the newly created CloudFront Distribution.
8. The *Domain Name* is the URL for the application.

The next step is to update the *package.json* in the client application.

1. Edit the *serverless-stack-client/notes-app-client/package.json* file.
2. Change the following line to include the name of the newly created S3 bucket:

```
"deploy" : "aws s3 sync build/ s3://sofwerx-serverless-client --delete",
```

3. Change the following line to include the id of the newly created CloudFront Distribution:

```
"postdeploy": "aws cloudfront create-invalidation --distribution-id
E2ES9ODPKCJ5Z4 --paths '/*'",
```

4. Save the file.

Now commit and push the code to GitLab. The CI/CD pipeline will create a S3 bucket that will be used for file uploads by the application. A Cognito User Pool and Identity Pool will be created. Execute the following steps to complete the Cognito configuration:

1. Navigate to the Cognito service.
2. Edit the newly created User Pool.
3. Select *App integration* -> *Domain name*
4. Enter a domain name:
   (https://sofwerx-serverless-notes.auth.us-west-2.amazoncognito.com)
5. Save the changes.
6. Get the *App client id* by selecting *App integration* -> *App client settings.*
7. Save the id (i.e., ch6hqdn616k6ppe3vr4rrbqod).

The next task is to retrieve the Invoke URL value of the API Gateway created for this project.

1. Navigate to the API Gateway service.
2. Click on the Stages link.
3. Copy the value for the Invoke URL at the top of the page.
   (i.e., https://2oeddyowbl.execute-api.us-west-2.amazonaws.com/dev)

The last task is to update the config.js file with the id and URL values.

1. Edit the *serverless-stack-client/src/config.js* file.
2. Update the cognito section in the dev configuration with the appropriate values.

```
const dev = {
    STRIPE_KEY:
"pk_test_51IEFAcFGqZEtoGFEkLv5z3z7RyGPrSnMu7m8yGls5t6xIY3Xum07tqHu56mPU
sWqTP3Ded5qpjhHVBKcebrhn7QR001AbfarFf",
    s3: {
        REGION: "us-west-2",
        BUCKET: "dev-notes-infra-s3-uploads4f6eb0fd-ixk7qhny72r1",
    },
    apiGateway: {
        REGION: "us-west-2",
        URL: "https://2oeddyowbl.execute-api.us-west-
2.amazonaws.com/dev",
    },
    cognito: {
        REGION: "us-west-2",
        USER_POOL_ID: "us-east-1_KgcncdnFB",
        APP_CLIENT_ID: "ch6hqdn616k6ppe3vr4rrbqod",
        IDENTITY_POOL_ID: "us-west-2:552cbec8-5b1e-45fc-99a1-
d892ac39838b",
    }
};
```

When the code is pushed to GitLab, the pipeline should build the client and deploy it. It should be accessible via the value of *Domain Name* in the CloudFront Distribution.

Summary

We began this journey with a desire to have a template for developing internal web applications using AWS serverless technologies. Our initial research for tutorials led to the Serverless Stack guide which is an incredible resource. After going through the guide, we altered the example to fit our requirements which included using GitLab instead of GitHub and Python for the Lambda expressions instead of Nodejs. We now have a template that can be used to quickly create a simple but working web application. These artifacts can then be evolved into the desired web application.

Who We Are

SOFWERX is a non-profit entity that accelerates evolution of the Warfighter through technology discovery, engagement, development, and transition. SOFWERX was created under a Partnership Intermediary Agreement, established in September of 2015, between DEFENSEWERX and the United States Special Operations Command (USSOCOM).

Sakina Shaikh is a candidate for Bachelor of Science in Computer Science at the University of South Florida. At USF, Sakina is a member of Society of Women Engineers, Woman in Computer Science and Engineering, and Software Developers Network. She contributed to this project during the Spring 2021 internship at SOFWERX. Her LinkedIn profile link is https://www.linkedin.com/in/sakina-shaikh-2019661a9/

Jim Ladd is a Senior Software Architect and IT Manager at SOFWERX where he leads the software engineering, web development, and system administration teams. Jim has been developing software solutions for over 35 years. Before joining SOFWERX in 2019, Jim was CEO and Principal Consultant at Wazee Group, a niche consulting company, for 20 years. His LinkedIn profile link is https://www.linkedin.com/in/jim-ladd/

Repositories

The two projects that make up this web application are hosted in GitLab at the following URLs:

https://gitlab.com/swxadmin/serverless-stack-api

https://gitlab.com/swxadmin/serverless-stack-client

References

[1] S. Stack, "Get Started With SST," [Online]. Available: https://docs.serverless-stack.com/. [Accessed 12 5 2021].

[2] "React - A JavaScript library for building user interfaces," [Online]. Available: https://reactjs.org/. [Accessed 12 5 2021].

[3] "AWS Amplify - Fastest, easiest way to build mobile and web apps that scale," [Online]. Available: https://aws.amazon.com/amplify/. [Accessed 12 5 2021].

[4] "Amazon Cognito - Simple and Secure User Sign-Up, Sign-In, and Access Control," [Online]. Available: https://aws.amazon.com/cognito/. [Accessed 12 5 2021].

[5] "Amazon S3 - Object storage built to store and retrieve any amount of data from anywhere," [Online]. Available: https://aws.amazon.com/s3/. [Accessed 12 5 2021].

[6] "Amazon CloudFront - Fast, highly secure and programmable content delivery network (CDN)," [Online]. Available: https://aws.amazon.com/cloudfront/. [Accessed 12 5 2021].

[7] "Amazon API Gateway - Create, maintain, and secure APIs at any scale," [Online]. Available: https://aws.amazon.com/api-gateway/. [Accessed 12 5 2021].

[8] A. L. -. R. c. w. t. a. s. o. clusters.. [Online]. Available: https://aws.amazon.com/lambda/. [Accessed 12 5 2021].

[9] "Amazon DynamoDB - Fast and flexible NoSQL database service for any scale," [Online]. Available: https://aws.amazon.com/dynamodb/. [Accessed 12 5 2021].