Cloud Computing Project

Car Rental Website as a serverless application

Atul Vidyarthi
S20170010016
atul.v17@iiits.in

Aniket Raj
S20170010012
aniket.r17@iiits.in

D Naga Pranav
S20170010038
nagapranav.d17@iiits.in

Kiran Sankar E J
S20170010073
kiransankar.e17@iiits.in

***Abstract*—Our project is a car-rental web application which is implemented using flask where users can book a specific car from a specified place and for a specific duration. It is implemented as a serverless application with the help of AWS making use of services like Amazon Cognito, Dynamodb, Cloudfront, Amazon lambda, Amazon lex, Amazon S3 and AWS IAM. The implementation of these services will be discussed in detail in later sections. Deployment is done using Zappa, a tool for deploying serverless python applications on AWS Lambda and API Gateway.**

I. Introduction - Serverless Applications

Serverless applications are ones which make use of third party back-end services. For instance, instead of making our own backend module for login and signup, we could make use of AWS Cognito for a cloud-based authentication and registration and simply call the APIs. This lets us focus more on the UI and the other important aspects of our application. Apart from this, there are also a number of other benefits of serverless websites listed as follow:

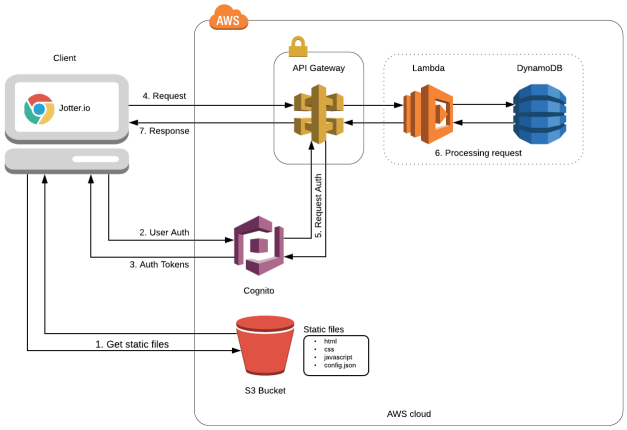

Fig. 1. Serverless application architecture

A. No server management

Although the serverless computing takes place on servers, they are not managed by the developers. These servers are managed by the vendors. This greatly helps the developers to focus on the business logic rather than the infrastructure.

B. Low cost deployment

Developers are only charged for the what they use, reducing the overall cost as the code runs only when the back-end functions are needed by the serverless application.

C. Better Scalability

Applications built with a serverless infrastructure will scale automatically as the user base grows or usage increases.As a result, a serverless application will be able to handle an unusually high number of requests just as well as it can process a single request from a single user within a specific period of time.

D. Low latency

Because the application is not hosted on an origin server, its code can be run from anywhere. It is therefore possible, depending on the vendor used, to run application functions on servers that are close to the end user. This reduces latency because requests from the user no longer have to travel all the way to an origin server.

E. Faster development

Lastly, since the services are already developed and implemented by the vendors, an application using these services for deployment can be deployed with minimal setup time.

II. Major Modules

A. Login-Signup

Amazon provides authentication and registration support through Amazon Cognito. The profiles of the users are stored in the user pools. Profile attributes can be chosen from the existing list or can be created manually. It also provides authentication of the users through external identity providers. We have implemented the forgot password, email verification and Facebook and Google OAuth alongside the general login and registration. For email verification, Cognito lets us choose between the code-based or the link-based verification out of which we have implemented the link-based verification. Cognito default email service has been used in the email verification. Forgot password, however, has been implemented using the code-based method.

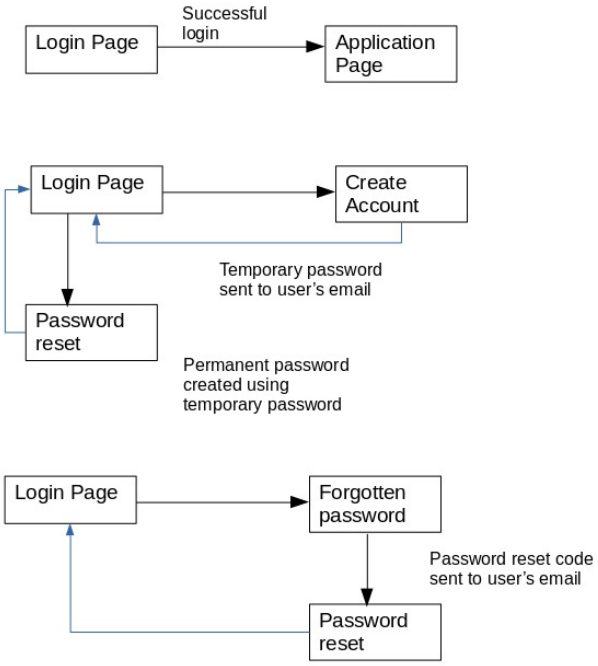

Fig. 2. Authentication and registration flow

B. Bookings Module

This module is the core of our project. This module has been implemented through the exhaustive usage of DynamoDB and Amazon S3. To let the users choose from the options, we need to display the vehicles. These vehicles are stored in the dynamodb database. The static files(like the images) are stored in S3 bucket and the link of those files are stored as the image source in the database. CRUD operations are implemented for bookings, that is, the users can book, modify, delete and see their previous bookings.

C. Chatbot

A chatbot has been added to the website to facilitate ease of booking. This has been done using Amazon Lex, and Amazon Lambda. Amazon Lex takes care of the dialogue processing and lambda functions were created to process and inject data into the database as and when necessary. The chatbot has been integrated with the website using a web-service called Kommunicate, which was selected owing to its inbuilt chat widget and the fact that it has a 30-day free trial period.
This helped negate the efforts needed to create a new chat widget as well as ways to enable bot to human handoff.

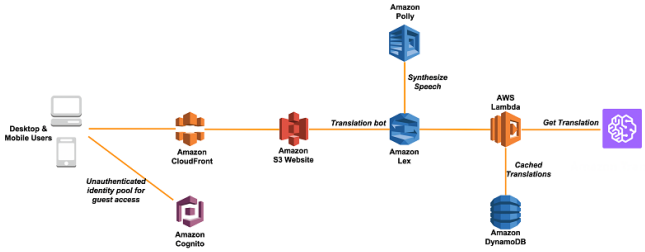

Fig. 3. Amazon Lex Implementation

III. Other Features

A. CloudFront

Amazon CloudFront is a fast content delivery network (CDN) service that securely delivers data, videos, applications, and APIs to customers globally with low latency, high transfer speeds, all within a developer-friendly environment.

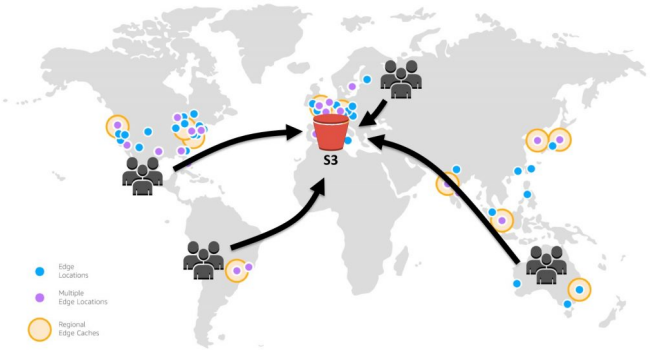

Fig. 4. CloudFront

Earlier, the static files used to be fetched from the origin every time they were requested by an end user. This increased the response time of the website and lead to long loading times. CloudFront caches frequently requested static files in the servers located in edge locations. This allows AWS to reduce latency by providing users frequently used content faster through the aforementioned edge locations. In our project, we have used CloudFront to display the images of the vehicles and to render the static webpages.

IV. Implementation

Flask, a python micro-framework for building web applications, was chosen owing to its simplicity and flexibility. The integration of the AWS services with flask application is done by Boto3, the Amazon Web Services(AWS) SDK for Python. It enables Python developers to create, configure, and manage AWS services. Boto3 provides an easy to use, object-oriented API, as well as low-level access to AWS services. Boto3 has been used to perform CRUD operations on the database, to access Lex responses, and so on. We chose boto3 from all the various other choices due to our familiarity with the python language.

V. Deployment

Out of all the options available, we chose Zappa to deploy our serverless application, which decreases a lot of the workload that deployment is usually associated with by itself. Zappa is a tool for deploying serverless python applications (Flask or Django) on AWS Lambda and API Gateway. It automatically packages up the application and local virtual environment into a lambda-compatible archive, replaces any dependencies with versions precompiled for lambda, sets up the function handler and necessary WSGI Middleware, uploads the archive to S3, creates and manages the necessary Amazon IAM policies and roles, registers it as a new Lambda function, creates a new API Gateway resource, creates WSGI-compatible routes for it, links it to the new Lambda function, and finally deletes the archives from your S3 bucket.

VI. Helpful Links

Click on the links listed below for the documentations:

- AWS Fundamentals: Building Serverless Applications - Coursera
- Why use serverless computing?
- Serverless Computing advantages and disadvantages
- Ten amazing benefits of serverless technology
- Zappa Documentation
- Flask Documentation
- Lex Documentation
- Cognito Documentation
- Boto3 documentation
- Boto3 with Cognito
- Boto3 Integration with DynamoDB