CSAA Bootcamp – June 2019

Problem Scenario #3

Table of Contents

1 DOCUMENT DEFINITION

The purpose of this document is to present a solution to the Problem Scenario #3.

2 PROBLEM STATEMENT

PROBLEM SCENARIO

Problem scenario 3

Overview:

Design a simple serverless web application that enables users to request KALESA rides from ABER fleet. The application will present users with an HTML based user interface for indicating the location where they would like to be picked up and will interface on the backend with a RESTful web service to submit the request and dispatch a nearby KALESA. The application will also provide facilities for users to register with the service and log in before requesting rides.

Application Architecture – supplemental information

- Hosts static web resources including HTML, CSS, JavaScript, and image files which are loaded in the user's browser.
- Provides user management and authentication functions to secure the backend API.
- Provides a persistence layer where data can be stored by the API's Lambda function.
- Script executed in the browser will send and received data from a public backend API built using Lambda and API Gateway.

3 SOLUTION

3.1 Key Considerations

The following are the key considerations for this solution:

1. Serverless web application
2. HTML-based user interface
3. With location API
4. Restful Web Services
5. With user management and authentication
6. With persistence layer

3.2 Technology Assumptions

The following are the technology assumptions for this solution:

1. Operational expense is not constrained.
2. User-friendly FQDN is required.
3. The application is a Progressive Web Application.

4. User identity is the responsibility of third-party application (e.g., Google, Facebook).

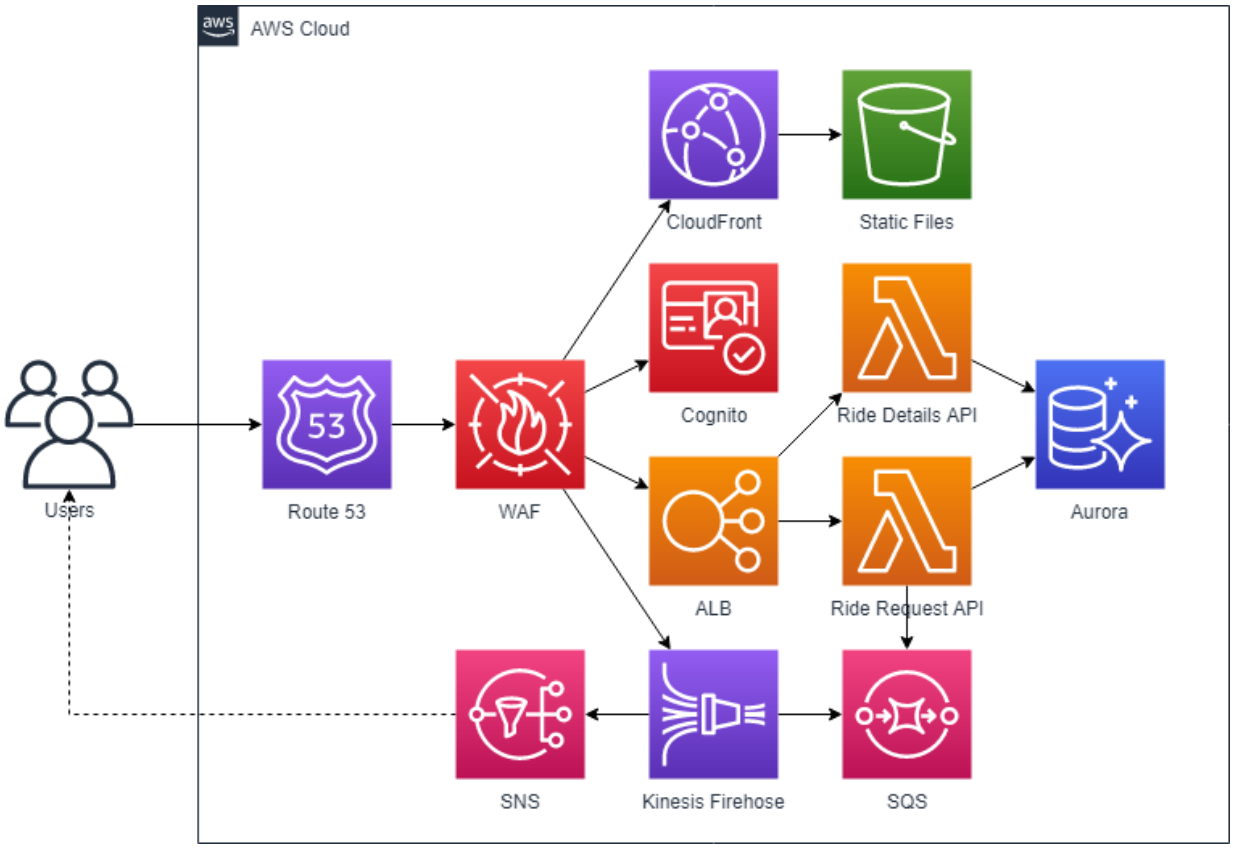
3.3 AWS Services

To achieve the considerations defined, the following AWS Services will be required:

- Route 53 – serves as domain name provider to utilize user-friendly FQDNs
- WAF – rule-based service to block common SQL Injection and Cross-side scripting attacks
- CloudFront – caches static contents in edge locations
- S3 – serves static files which build up the web application
- Cognito – processes identity from third-party users for user management and authentication
- ALB – regulates the flow of requests across the different serverless APIs
- Lambda – backbone of our serverless APIs
- Aurora – serverless RDS instance
- Kinesis Firehose – responsible for ETL of data streams
- SQS – queueing service
- SNS – notification service

3.4 Architecture Design

Below is the architecture designed to achieve the key considerations:



A progressive web application is an HTML-based application with native mobile support through a service worker. This type of application enables offlline-first and mobile-first native user experience.

The client-side of the application will be served to the user through a user-friendly FQDN using Route 53. Route 53 will request the static files from edge locations through CloudFront. These static files are stored in an S3 bucket.

Since our application will not store user credentials natively, it instead employs risk transference to other third-party services. These services will simply provide our application with identity tokens through JWT which our Cognito service uses to authenticate our user. Simply put, an SSO-based authentication is leveraged. If Cognito does not recognize the user, the application will prompt the user to register as a rider or as a driver. This selection will serve as the registration mechanism of the application wherein data will be stored together with the user's identity in Cognito.

For users with rider attribute, the application will enable them to select locations as their origin and destination. This functionality will again be served by third-party library. As for the GPS, since the application is a PWA, client-side can directly call the Location API of the mobile device after the user provided access to the application. The selected values will be sent to AWS through OAuth v2.0 HTTPS request. The request will pass through Route 53, WAF, ALB to Ride Request API wherein the data will be processed through and saved to the Aurora instance. A message about the ride will be queued to SQS.

For users with driver attribute, the application will continually send GPS details to the Kinesis Firehose. Firehose will assess each queued message if the driver is nearby. If not, the message will be returned to the queue and the process will continue with the next message. If yes, an SNS notification will be sent to the driver wherein the application will respond accordingly. The driver will be informed and await confirmation. If the driver declined, the application will return the message back to the queue. If the driver agreed, the driver's and the ride's status attributes will be updated accordingly.

Users can retrieve the details about the ride through the application making a request to the Ride Details API. If during the process, before the rider and driver meet, a user cancelled; both the driver's and the ride's status attributes will be updated accordingly.