COLLEGE OF ENGINEERING

Electrical Engineering and Computer Science

CLOUD-BASED APPLICATION FOR FIRE DEPT. RESPONSE TIME ANALYSIS

Gregory Melsby, Jason Rash, Clay Vander Werff, Lok Wai Wong

PROJECT DESCRIPTION

Levrum Data Technologies has developed a desktop application which helps fire departments analyze the coverage areas of their fire station network by range or time. Levrum would like expand the compatibility of this tool with more devices, which involves converting portions of the functionality, such as the coverage calculations, to a cloud-based solution.

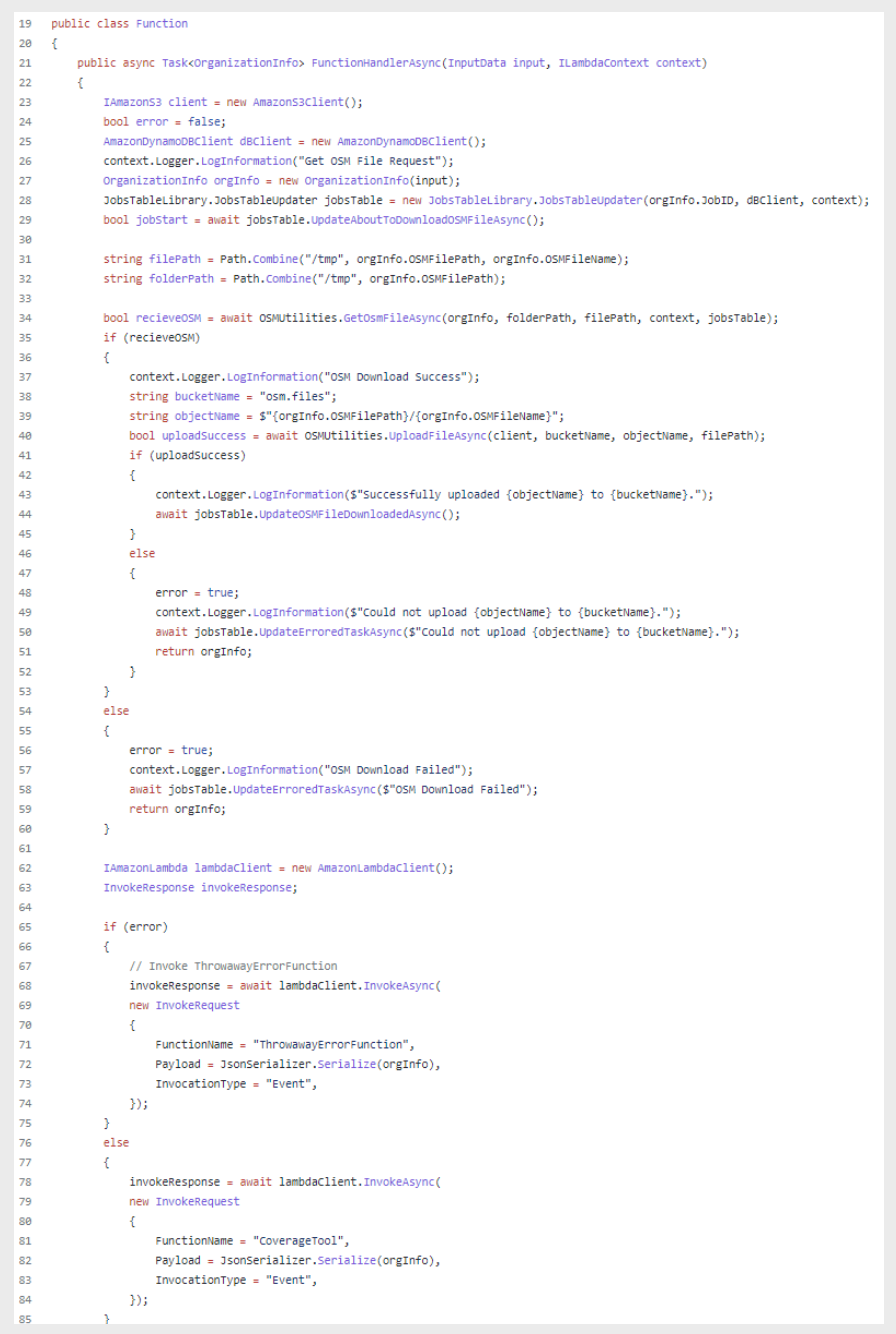
Our team was tasked with converting the primary calculation engine that allows the station coverage maps to be built from a desktop application to a cloud-based one hosted on AWS. This involved breaking the functionality into Lambda functions, which we can conceptually group as the following microservices:

- A service that takes a state and provides a list of detailed location data for each municipality
- A service that takes location data and provides a list of applicable fire stations and station coordinates
- A service that takes location data then downloads an OSM file and save it for later use
- A service that performs the existing calculations to create a coverage map for an area and selected fire stations and caches the results
- A service that keeps the client app updated on the status of the coverage run
- A service that fetches the results from a completed client run

These microservices were then connected within the AWS ecosystem to provide a comparable functionality to the desktop application, but now hosted on AWS and callable from anywhere.

Our team also improved upon an existing web-based frontend to provide a demonstration of the capacities of our API.

Oregon State University



```
public class Function
{
    public async Task<OrganizationInfo> FunctionHandlerAsync(InputData input, ILambdaContext context)
    {
        IAmazonS3 client = new AmazonS3Client();
        bool error = false;
        AmazonDynamoDBClient dBClient = new AmazonDynamoDBClient();
        context.Logger.LogInformation("Get OSM File Request");
        OrganizationInfo orgInfo = new OrganizationInfo(input);
        JobsTableLibrary.JobsTableUpdater jobsTable = new JobsTableLibrary.JobsTableUpdater(orgInfo.JobID, dBClient, context);
        bool jobStart = await jobsTable.UpdateAboutToDownloadOSMFileAsync();

        string filePath = Path.Combine("/tmp", orgInfo.OSMFilePath, orgInfo.OSMFileName);
        string folderPath = Path.Combine("/tmp", orgInfo.OSMFilePath);

        bool recieveOSM = await OSMUtilities.GetOsmFileAsync(orgInfo, folderPath, filePath, context, jobsTable);
        if (recieveOSM)
        {
            context.Logger.LogInformation("OSM Download Success");
            string bucketName = "osm.files";
            string objectName = $"{orgInfo.OSMFilePath}/{orgInfo.OSMFileName}";
            bool uploadSuccess = await OSMUtilities.UploadFileAsync(client, bucketName, objectName, filePath);
            if (uploadSuccess)
            {
                context.Logger.LogInformation($"Successfully uploaded {objectName} to {bucketName}.");
                await jobsTable.UpdateOSMFileDownloadedAsync();
            }
            else
            {
                error = true;
                context.Logger.LogInformation($"Could not upload {objectName} to {bucketName}.");
                await jobsTable.UpdateErroredTaskAsync($"Could not upload {objectName} to {bucketName}.");
                return orgInfo;
            }
        }
        else
        {
            error = true;
            context.Logger.LogInformation("OSM Download Failed");
            await jobsTable.UpdateErroredTaskAsync($"OSM Download Failed");
            return orgInfo;
        }

        IAmazonLambda lambdaClient = new AmazonLambdaClient();
        InvokeResponse invokeResponse;

        if (error)
        {
            // Invoke ThrowawayErrorFunction
            invokeResponse = await lambdaClient.InvokeAsync(
            new InvokeRequest
            {
                FunctionName = "ThrowawayErrorFunction",
                Payload = JsonSerializer.Serialize(orgInfo),
                InvocationType = "Event",
            });
        }
        else
        {
            invokeResponse = await lambdaClient.InvokeAsync(
            new InvokeRequest
            {
                FunctionName = "CoverageTool",
                Payload = JsonSerializer.Serialize(orgInfo),
                InvocationType = "Event",
            });
        }
```

Figure 1: Microservice Code to Download and Save OSM File

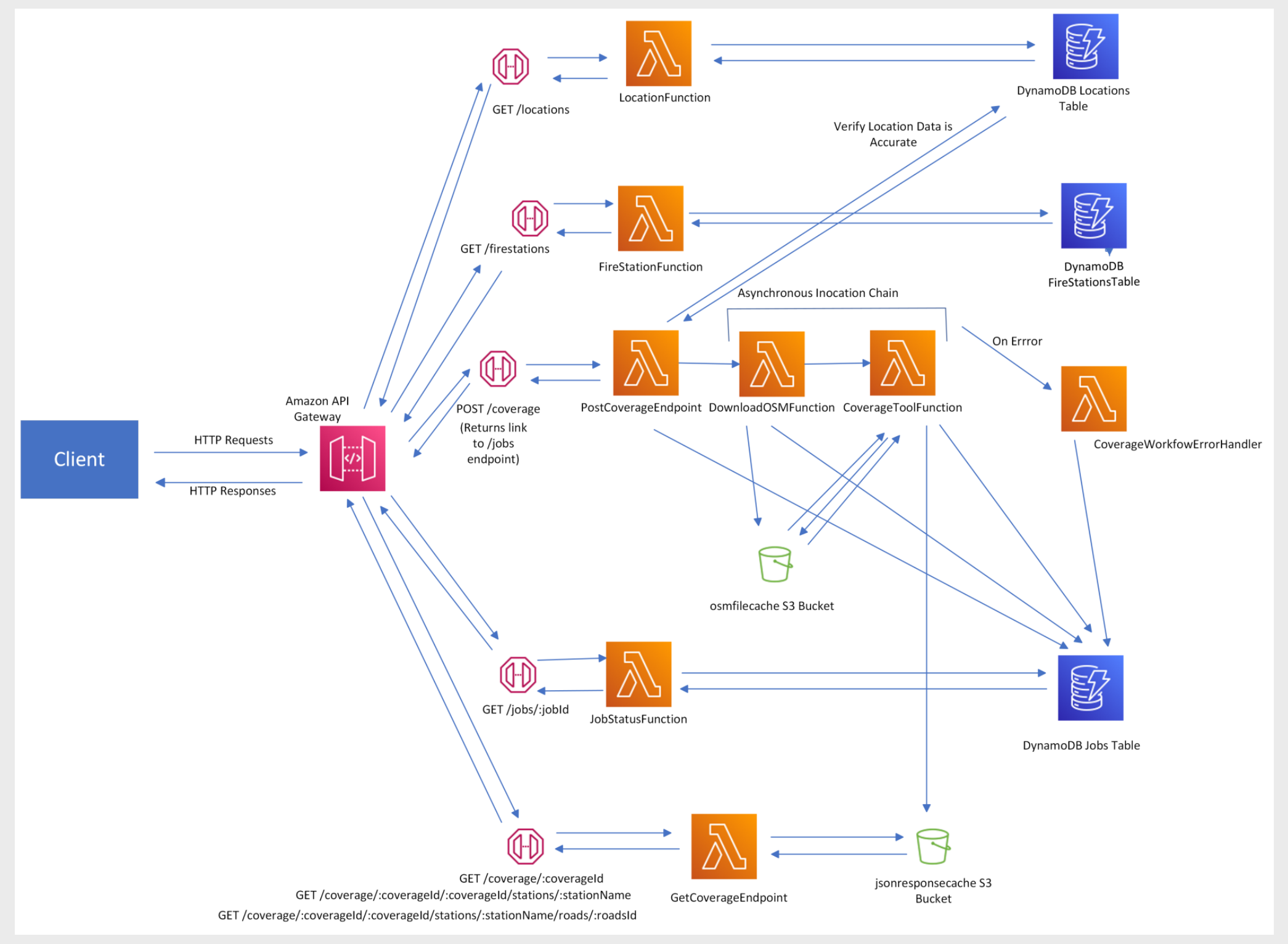

GET /locations
LocationFunction
DynamoDB Locations Table
Verify Location Data is Accurate
GET /firestations
FireStationFunction
DynamoDB FireStationsTable
Asynchronous Inocation Chain
On Errror
Amazon API Gateway
Client
HTTP Requests
HTTP Responses
POST /coverage (Returns link to /jobs endpoint)
PostCoverageEndpoint
DownloadOSMFunction
CoverageToolFunction
CoverageWorkfowErrorHandler
osmfilecache S3 Bucket
GET /jobs/:jobId
JobStatusFunction
DynamoDB Jobs Table
GET /coverage/:coverageId
GET /coverage/:coverageId/:coverageId/stations/:stationName
GET /coverage/:coverageId/:coverageId/stations/:stationName/roads/:roadsId
GetCoverageEndpoint
jsonresponsecache S3 Bucket

Figure 3: AWS Process Flow

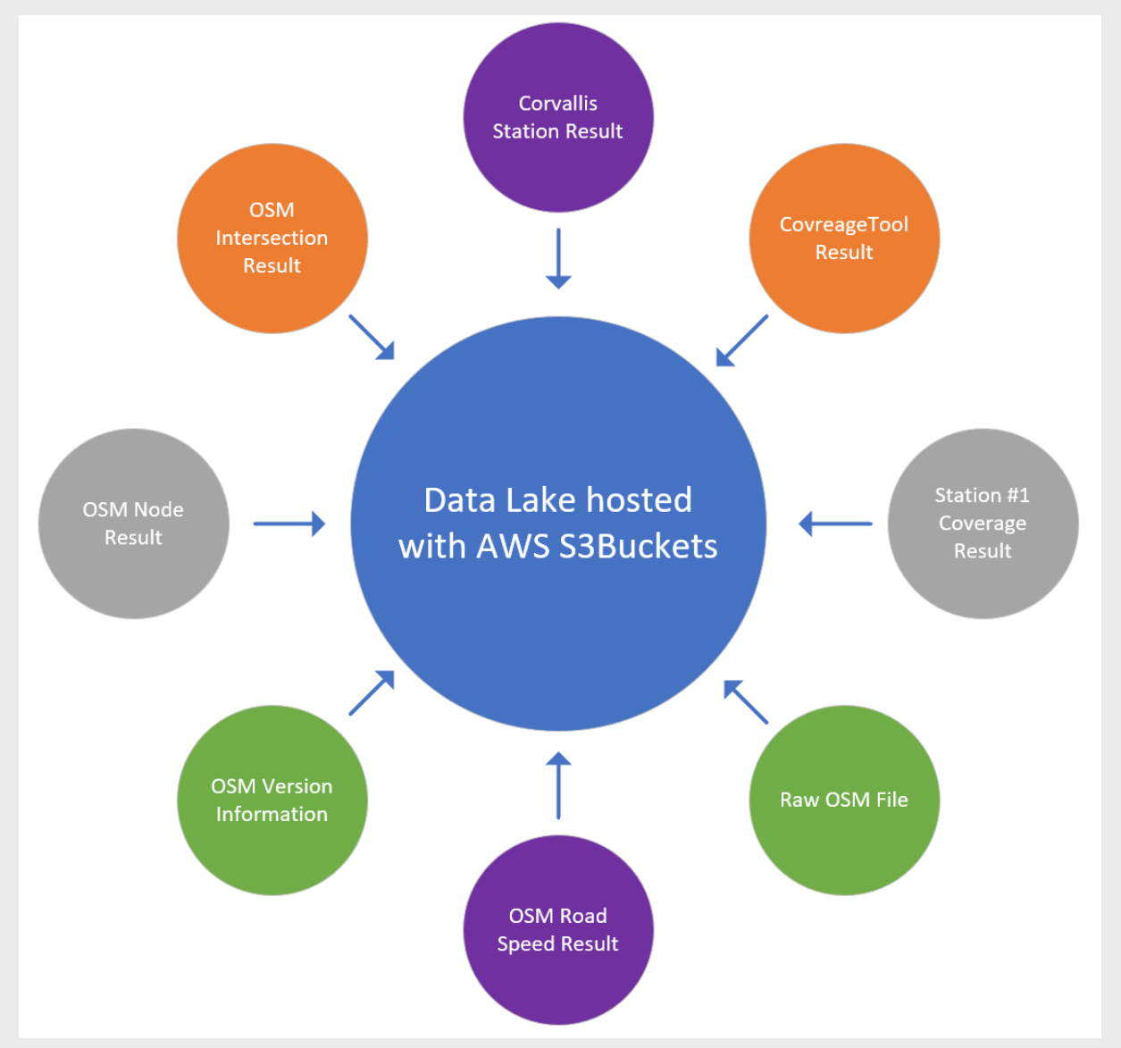
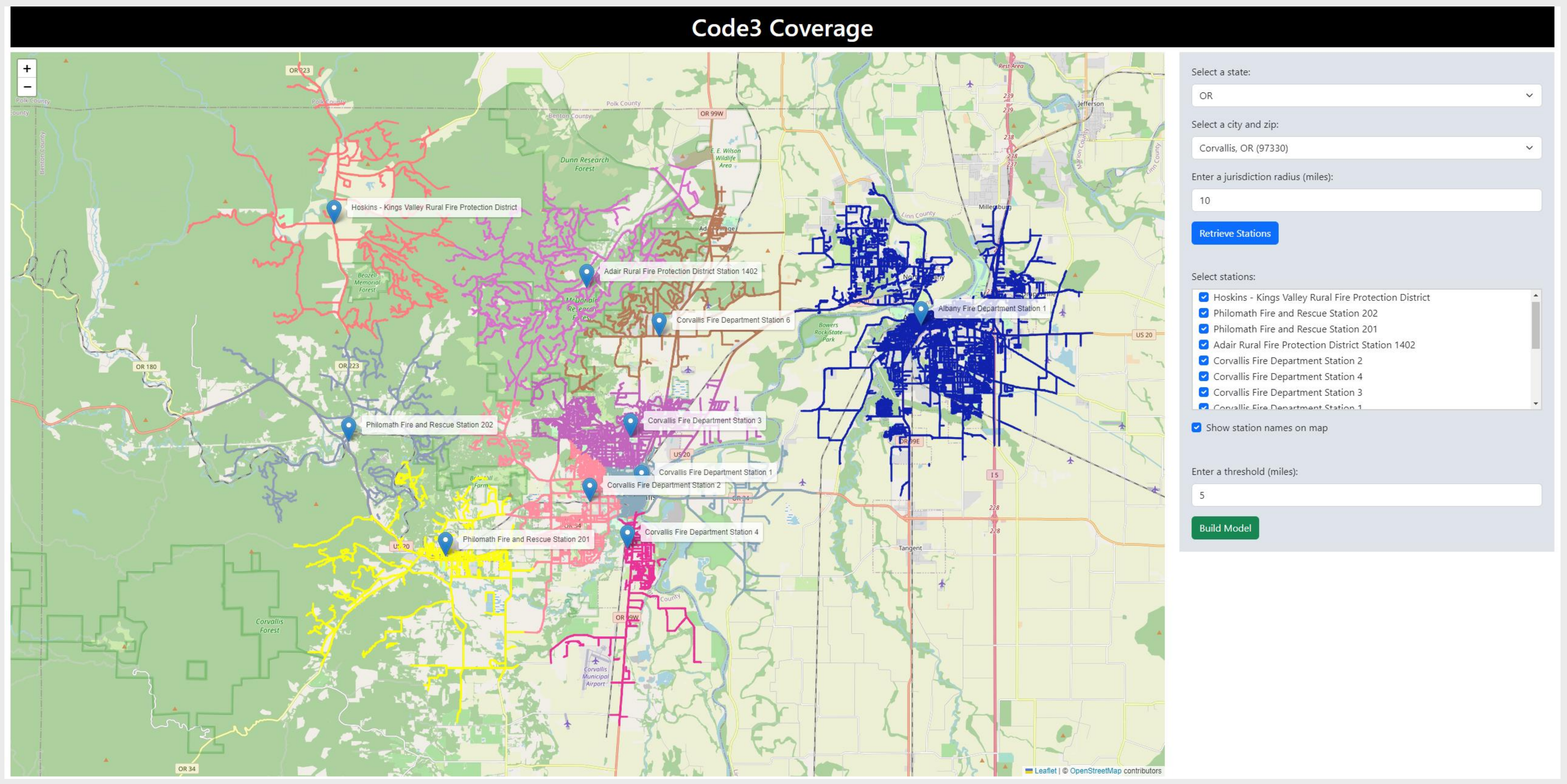


Figure 4: S3Bucket Usage



Code3 Coverage

Select a state: OR

Select a city and zip: Corvallis, OR (97330)

Enter a jurisdiction radius (miles): 10

Retrieve Stations

Select stations:

- ✓ Hoskins - Kings Valley Rural Fire Protection District
- ✓ Philomath Fire and Rescue Station 202
- ✓ Philomath Fire and Rescue Station 201
- ✓ Adair Rural Fire Protection District Station 1402
- ✓ Corvallis Fire Department Station 2
- ✓ Corvallis Fire Department Station 4
- ✓ Corvallis Fire Department Station 3
- ✓ Corvallis Fire Department Station 1

✓ Show station names on map

Enter a threshold (miles): 5

Build Model

Figure 2: Web App Interface with Coverage Map Built By Calculations

AWS SERVICES AND TECHNOLOGY STACK

- Desktop functionality was broken down into AWS Lambda Functions, which are serverless cloud functions that only run when called.
- Existing codebase for desktop application was written in C# using Microsoft's .NET framework. The team chose to use C# for our backend Lambda functions to avoid complicating the codebase with multiple languages and technology stacks.
- AWS API Gateway was used to handle requests/responses with the client and to initiate the appropriate functions required by the client request.
- Chains of Lambda functions are asynchronously invoked, resulting in efficient use of cloud services.
- Due to the computationally intensive nature of this program, there is a result caching functionality to improve performance when calculation runs have been processed already. The team implemented a file caching system utilizing AWS S3 Buckets which allows us to create a data lake for various .csv files.
- Our frontend is written in JavaScript and uses React and Leaflet.

DATA PERSISTENCE

- We are using a comparable file caching system to the desktop application where previous runs are cached in a .csv file within cloud storage.
- AWS S3Buckets are Amazon's simple, stagnant storage solution where we could use a similar folder/file structure to the desktop application. Although likely not optimal, this strategy was selected primary due to time constraints. There are many internal data validation functions and other functionality that would need to be rewritten which the team was not sure could be completed with the remaining time this quarter. Using a similar file/file structure for caching allowed us to use the existing data validation functions and checks. Additionally, we cache the JSON response to the /coverage endpoint, allowing for faster responses to the client.
- We also use DynamoDB to store info about municipalities, fire station locations, and the status of each coverage run.