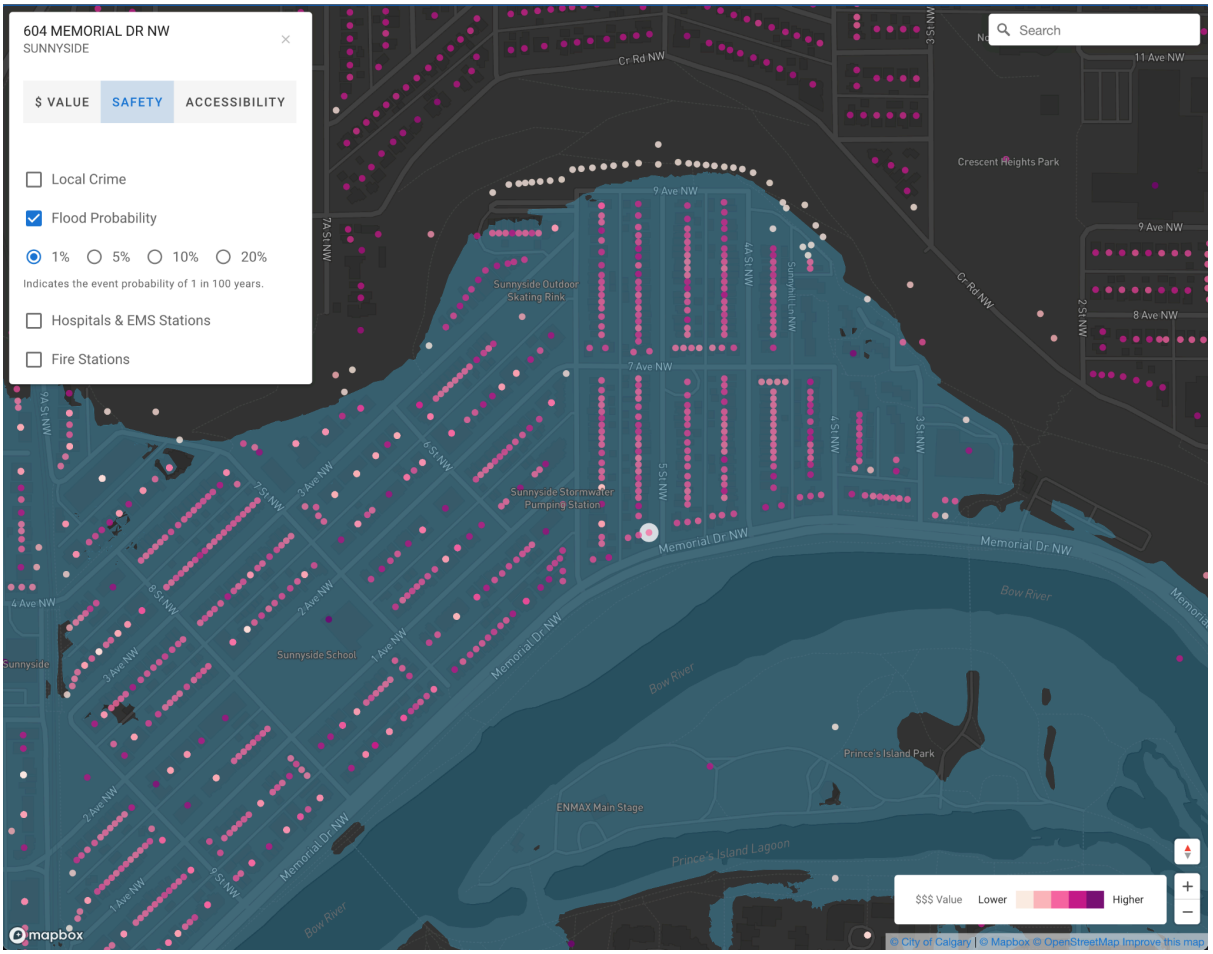
PickYourPlace

Pick a place to live based on property value, safety, and accessibility



Choosing a place to live is an important decision that will have a huge impact on everybody's life. So how do we choose our home? There are many factors; in fact the final decision is a trade-off of many factors and, since everyone or every family has their own priorities, that can be unique to them.

Location is a main component among all those factors. PickYourPlace is a location intelligence platform that allows users to visualize and navigate the world surrounding properties and make spatially-aware decisions. It enables users to understand property value trends in different geographic areas, evaluate a number of safety factors in a localized view, and compare accessibility to nearby amenities using different travel modes (walking, biking, riding (on public transit), and driving).

Challenge

There are two main challenges when building a location intelligence platform. The first challenge relates to user experience; how do we present the information in a way that drives user engagement and awareness? Some real estate applications focus solely on the

“interior” of properties like the type (detached house vs. townhouse vs. apartment), number of bedrooms, garage capacity, etc. and overlook the surroundings. Some other platforms do provide metrics about the “exterior” but present them using some indices that are not always easy to interpret and understand.

The second issue is about performance; how do we make sure that a data-centric platform performs well and does not become a bottleneck when dealing with a lot of “data layers”? Every property has a coordinate that does not change much (ignoring small changes in local and global coordinates because of the constant movement of tectonic plates) but its surrounding infrastructure and nearby amenities and services may change from time to time. This requires a dynamic, up-to-date system that is able to run spatial queries and aggregations on-demand.

PickYourPlace is an effort to address these challenges — it aims to present a set of “useful” location-based indicators to help users pick a place to live based on (1) their budget, (2) safety, and (3) the quality of amenities and services in their neighborhood. It augments data from different open data sources including the City of Calgary’s open data portal and OpenStreetMap to display all properties within the City of Calgary on an interactive map, along with their assessed value, crime statistics, and areas susceptible to flooding. It also uses the city’s road network along with walking, biking, and public transit infrastructure to highlight the areas and POIs (points of interest) that are reachable within a 15-minute walk, bike, ride (on public transit), and drive — the POIs include hospitals, emergency and fire stations, schools, parks, trails, and bikeways.

The following section presents the technical architecture and describes how the system works.

Solution

PickYourPlace is built using open source software, open standards, and open data and hosted on Amazon’s AWS infrastructure — see the PickYourPlace’s overall architecture below.

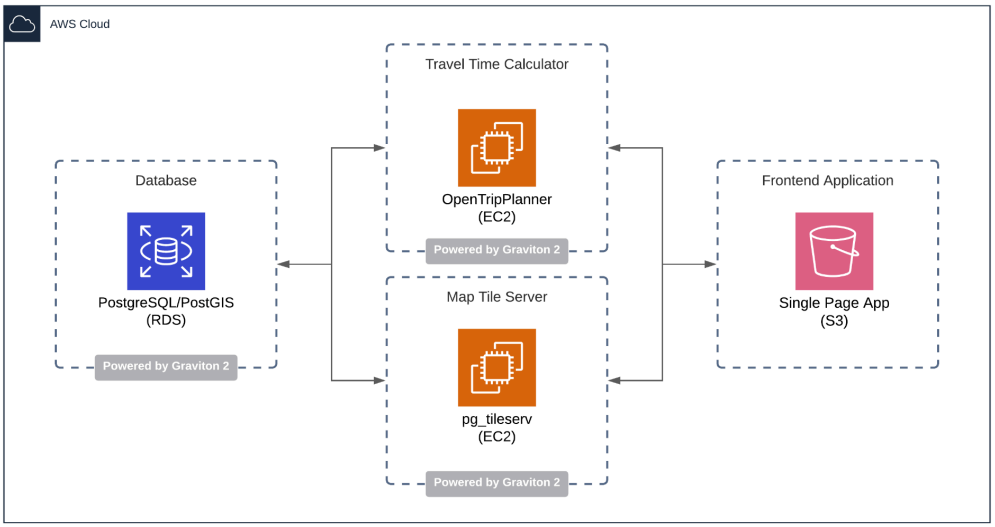


Figure 1. PickYourPlace’s overall architecture

PostgreSQL/PostGIS is the database — it stores spatial “data layers”, executes queries, and generates vector tiles. The data layers include bikeways, crime statistics, hospitals and EMS stations, fire stations, flood probability maps, parks, property values, schools, and trails. A Graviton-powered RDS is used to create the database for this project.

pg_tileserv is the map tile server — it is a lightweight tile server, written in Go, that takes in HTTP requests, forms SQL queries and sends them to a PostgreSQL/PostGIS database for execution, and returns the results as vector tiles. The tile server is running on a Graviton-powered EC2 instance.

OpenTripPlanner is the travel time calculator — it is a multimodal trip planning tool, written in Java, that takes in HTTP requests, computes isochrones (areas that are reachable within a specified amount of time from a location), and returns the results as GeoJSON. It uses OpenStreetMap data, GTFS (General Transit Feed Specification) feed, and a digital elevation model (DEM) to build a transportation network that supports different travel modes including walking, biking, riding (on public transit), and driving. The OpenTripPlanner server is running on a Graviton-powered EC2 instance.

The frontend application is an SPA (single page app), built using Vue.js, Vuetify, and Mapbox GL JS. The SPA is hosted on S3 as a static website using AWS Amplify.

The frontend application visualizes all residential and non-residential properties in Calgary on a map. When a user selects a property, it will show the data layers associated with the location of interest under three categories:

1. Property value, which lists the assessed property value and its trend over the past three years, plus some information about the property class, its construction year, and square footage;
2. Safety, which provides local crime statistics (population adjusted and aggregated based on different types of crime including assault, break and enter, robbery, theft from vehicle, theft of vehicle, and violence), flood risk maps with event probability ranging from 1% to 20%, emergency stations and hospitals, and fire stations within a 15-minute drive;
3. Accessibility, which shows schools and universities, parks, trails, and bikeways that are reachable within a 15-minute walk, bike, ride (on public transit), and drive.

This provides an interactive experience for users, empowering them to explore their options in an intuitive and performant way and make their decision with more location context. I encourage you to play with the web app live here, and explore its GitHub repository to take a deeper dive into the code base.

The next section presents the results of a performance evaluation between Graviton-powered instances used in this project and their x86-based counterparts.

Performance Evaluation

As mentioned in the previous section, a Graviton-powered RDS PostgreSQL and two Graviton-powered EC2 instances run the database and backend services for this project. The database stores the data in 15 tables and handles the spatial queries sent by pg_tileserv. To execute the queries, four spatial operations need to be performed: bounding box calculation, coordinate transformation, overlay analysis, and vector tile generation.

To evaluate the performance of Graviton-powered vs. x86-based RDS PostgreSQL instances, the following query was executed on two different instances (db.t4g.micro vs. db.t3.micro) and the response time was recorded. The experiment was repeated 30 times to get representative results.

```
EXPLAIN ANALYZE
WITH bounds AS (
    SELECT ST_TileEnvelope(15, 5995, 10956) AS geom
),
     mvtgeom AS (
         SELECT ST_AsMVTGeom(ST_Transform(t.geom, 3857), bounds.geom) AS geom,
                t.address
         FROM property_value t,
              bounds
         WHERE ST_Intersects(t.geom, ST_Transform(bounds.geom, 4326))
     )
SELECT ST_AsMVT(mvtgeom, 'public.property_value')
FROM mvtgeom;
```

Figure 2. The SQL query for vector tile generation used in this test

Instance	CPU	RAM	Storage	Software
db.t4g.micro	2	1 GB	20 GB General Purpose SSD (gp2)	PostgreSQL 13.4 R1 PostGIS 3.1.4
db.t3.micro	2	1 GB	20 GB General Purpose SSD (gp2)	PostgreSQL 13.4 R1 PostGIS 3.1.4

Table 1. Configuration of the RDS instances

The results suggest that the Graviton-based RDS instance provides more than 46% better performance over the x86-based instance — see the results in the table below.

RDS Instance	Response Time (ms)
db.t4g.micro	2.44 ± 0.07
db.t3.micro	3.57 ± 2.60

Table 2. Performance test results

The map tile server publishes all the available tables in the database (15 tables in this project) as vector tile layers. It accepts HTTP tile requests, generates SQL queries and sends them to the database, and returns the response.

To test the map tile server, pg_tileserv was compiled for arm64 and x86 architectures and deployed on two different instances (t4g.micro vs t3.micro). ApacheBench was used to run the test and measure the response time — a vector tile request was used for this experiment. The experiment was repeated 30 times to get representative results.

```
ab -n 30 -c 1 '${IP_ADDRESS}/public.property_value/16/11992/21914.pbf'
```

Figure 3. The ApacheBench command used in this test

Instance	CPU	RAM	Storage	Software
t4g.micro	2 (2.5 GHz)	1 GB	30 GB General Purpose SSD (gp2)	Ubuntu Server 20.04 LTS Go 1.17.2 pg_tileserv 1.0.8 nginx 1.18.0
t3.micro	2 (2.5 GHz)	1 GB	30 GB General Purpose SSD (gp2)	Ubuntu Server 20.04 LTS Go 1.17.2 pg_tileserv 1.0.8 nginx 1.18.0

Table 3. Configuration of the EC2 instances

The results suggest that the Graviton-based EC2 instance provides more than 68% better performance over the x86-based instance — see the results in the table below.

EC2 Instance	Response Time (ms)
t4g.micro	112 ± 6.7
t3.micro	189 ± 105.3

Table 4. Performance test results

The travel time calculator generates isochrones from a specified location and returns the results as GeoJSON polygons. To do so, the OpenTripPlanner server is first used to build a

graph object (i.e., a transportation network) and store it on the disk. Then, it loads the graph in memory, exposes an API, and handles isochrone HTTP requests.

To test the travel time calculator, OpenTripPlanner was deployed on two different instances (t4g.small vs t3.small) and two test scenarios were designed. In the first scenario, the graph build time was measured — the build process is composed of three steps: (1) building the transportation network from OpenStreetMap data, (2) building the transit network using GTFS data, (3) and analyzing elevation profiles using DEM data. The result of this process was a graph with 203,546 nodes and 617,268 edges for the City of Calgary.

In the second scenario, ApacheBench was used to measure the response time to isochrone requests. Both experiments were repeated 30 times to get representative results.

```
ab -n 30 -c 1 '${IP_ADDRESS}/isochrone?fromPlace=51.085699,-114.122344&mode=WALK,TRANSIT&cutoffSec=900'
```

Figure 4. The ApacheBench command used in the second experiment of this test

Instance	CPU	RAM	Storage	Software
t4g.small	2 (2.5 GHz)	2 GB	30 GB General Purpose SSD (gp2)	Ubuntu Server 20.04 LTS Java 8 (OpenJDK) OpenTripPlanner 1.2.0 nginx 1.18.0
t3.small	2 (2.5 GHz)	2 GB	30 GB General Purpose SSD (gp2)	Ubuntu Server 20.04 LTS Java 8 (OpenJDK) OpenTripPlanner 1.2.0 nginx 1.18.0

Table 5. Configuration of the EC2 instances

The results suggest that the Graviton-based EC2 instance provides 20% and more than 71% better performance over the x86-based instance when building the graph and handling isochrone HTTP requests, respectively — see the results in the table below.

EC2 Instance	Build Time (min)	Response Time (ms)
t4g.small	2.0 ± 0.3	103 ± 4.4
t3.small	2.4 ± 0.8	177 ± 101.4

Table 6. Performance test results