aws

Developer Guide

Amazon Location Service

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Location Service: Developer Guide

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

Welcome 1
What is Amazon Location Service? 1
Key features 2
Related services 3
Quick start 5
Creating a web app 5
Create resources 6
Setup authentication 7
Creating HTML 8
Adding the map 12
Adding search 16
Final application 20
What's next 25
Creating an Android app 25
Creating Amazon Location resources for your app 26
Setup authentication 27
Create the app 30
Add the map 31
Add search 34
Add tracking 44
What's next 53
Creating an iOS app 53
Create resources 54
Setup authentication 55
Creating the app 58
Initial code 59
Add a map 61
Add search 66
Add tracking 67
What's next 79
Amazon Location concepts 80
Overview 81
Maps 82
Map styles 83

Political views ... 83
Custom Layers ... 84
Map rendering ... 84
Maps terminology ... 85
Places search ... 87
Geocoding concepts ... 88
Search results ... 89
Multiple results and relevance ... 89
Address results ... 90
Storing geocode results ... 92
Places terminology ... 92
Routes ... 93
Route calculator resources ... 94
Calculating a route ... 94
Planning routes ... 96
Route terminology ... 96
Geofences and Trackers ... 98
Geofences ... 98
Trackers ... 100
Geofence terminology ... 104
Tracker terminology ... 105
Common use cases ... 106
User engagement and geomarketing applications ... 107
Asset tracking applications ... 109
Delivery applications ... 110
Data providers ... 112
Data provider coverage and features ... 112
Map styles ... 114
More details ... 114
Esri ... 114
GrabMaps ... 123
HERE Technologies ... 128
Open Data ... 135
Features by data provider ... 144
Terms of use and data attribution ... 149
Regions and endpoints ... 149

Regions ... 149
Endpoints ... 151
API operation Endpoints ... 152
Service quotas ... 153
Managing your Amazon Location service quotas ... 167
Developing with Amazon Location ... 169
Scenarios and use cases ... 169
SDKs and tools ... 170
SDKs by language ... 171
MapLibre ... 175
Amazon Location SDK ... 180
Amazon Location APIs ... 204
Using Amazon Location with an AWS SDK ... 204
Error message updates ... 204
Code examples ... 237
Amazon Location Demo site ... 239
Tutorial: Quick start ... 239
Tutorial: Database enrichment ... 240
Example: Explore app ... 241
Example: Style a map ... 242
Example: Draw markers ... 242
Example: Draw clustered points ... 243
Example: Draw a polygon ... 243
Example: Change the map language ... 244
Blog: Estimated delivery time notifications ... 244
Example: Stream Position Updates ... 245
Example: Geofencing and Tracking mobile application ... 246
How to use Amazon Location ... 247
Account prerequisites ... 248
Sign up for an AWS account ... 248
Create a user with administrative access ... 249
Grant access to Amazon Location Service ... 250
Using maps ... 251
Prerequisites ... 253
Displaying maps ... 256
Drawing on a map ... 310

Settings extents for a map ... 311
Managing map resources ... 312
Places search ... 316
Prerequisites ... 317
Geocoding ... 320
Reverse geocoding ... 327
Autocomplete ... 331
Using place IDs ... 338
Categories and filtering ... 339
Tutorial: Database enrichment ... 344
Managing place index resources ... 359
Calculating routes ... 362
Prerequisites ... 363
Calculate route ... 366
Route planning ... 371
Positions not located on a road ... 377
Departure time ... 379
Travel mode ... 380
Managing route resources ... 381
Geofencing and tracking ... 385
Step 1: Add geofences ... 386
Step 2: Start tracking ... 393
Step 3: Link a tracker to a geofence collection ... 407
Step 4: Evaluate device positions against geofences ... 408
Verify device positions ... 411
Reacting to events with EventBridge ... 413
Tracking using AWS IoT and MQTT ... 418
Managing geofence resources ... 426
Managing tracker resources ... 434
Sample Geofencing and Tracking mobile application ... 438
Tagging your resources ... 457
Restrictions ... 457
Grant permission to tag ... 458
Add a tag to a resource ... 459
Track cost by tag ... 459
Control access to resources using tags ... 460

Learn more ... 461
Grant access to Amazon Location ... 461
Using API keys ... 462
Using Amazon Cognito ... 468
Monitoring Amazon Location Service ... 478
Monitoring with CloudWatch ... 479
Using CloudTrail with Amazon Location ... 484
Using AWS CloudFormation to create resources ... 488
Amazon Location and AWS CloudFormation templates ... 488
Learn more about AWS CloudFormation ... 489
Security ... 490
Data protection ... 491
Data privacy ... 492
Data retention ... 492
Data at rest encryption ... 492
Data in transit encryption ... 505
Identity and Access Management ... 505
Audience ... 506
Authenticating with identities ... 506
Managing access using policies ... 510
How Amazon Location Service works with IAM ... 512
How Amazon Location Service works with unauthenticated users ... 520
Identity-based policy examples ... 521
Troubleshooting ... 533
Incident response ... 535
Logging and Monitoring ... 535
Compliance validation ... 536
Resilience ... 537
Infrastructure security ... 537
Configuration and vulnerability analysis ... 538
Confused deputy prevention ... 538
Security best practices ... 538
Detective best practices ... 539
Preventive best practices ... 539
Best practices ... 540
Security ... 541

Resource management 541
Billing and cost management 542
Quotas and usage 542
Document history 543
AWS Glossary 552

Welcome to Amazon Location Service

Welcome to the *Amazon Location Service Developer Guide*.

The following topics can help you get started in the documentation, based on what you are trying to do.

Get an overview of Amazon Location

- Learn about the concepts in Amazon Location.
- Dive deeper into the functionality in the How to use Amazon Location Service chapter.
- See demo apps in the Amazon Location demo site.
- If you already have an AWS account, you can use the Amazon Location Service console to explore the functionality first-hand.

Use Amazon Location as a developer

- Build your first app with the Quick start.
- Learn how the various Amazon Location Service features work in the How to use Amazon Location Service chapter.
- See the SDKs and tools available to you in the Developing with Amazon Location chapter.
- See code examples and tutorials that you can use in your own apps. You can also visit the Amazon Location demo site samples page to find samples, filterable by feature, language, or platform.
- Get information about Amazon Location APIs in the API Reference guide.

What is Amazon Location Service?

Amazon Location Service lets you add location data and functionality to applications, which includes capabilities such as maps, points of interest, geocoding, routing, geofences, and tracking. Amazon Location provides location-based services (LBS) using high-quality data from global, trusted providers Esri, Grab, and HERE. With affordable data, tracking and geofencing capabilities, and built-in metrics for health monitoring, you can build sophisticated location-enabled applications.

With Amazon Location, you retain control of your organization's data. Amazon Location anonymizes all queries sent to data providers by removing customer metadata and account

information. Additionally, sensitive tracking and geofencing location information, such as facility, asset, and personnel locations, does not leave your AWS account at all. This helps you protect sensitive information from third parties, protect user privacy, and reduce your application's security risks. With Amazon Location, Amazon and third parties do not have rights to sell your data or use it for advertising.

Amazon Location is fully integrated with services such as AWS CloudTrail, Amazon CloudWatch, Amazon EventBridge, and AWS Identity and Access Management (IAM). Amazon Location simplifies your development workflow with data integration, and fast tracks apps to production with built-in monitoring, security, and compliance features.

For highlights, product details, and pricing, see the service page for Amazon Location Service.

Key features in Amazon Location

Amazon Location provides the following features:

Maps

Amazon Location Service Maps lets you visualize location information and is the foundations of many location-based service capabilities. Amazon Location Service provides map tiles of different styles sourced from global location data providers Esri, Grab, and HERE, as well Open data maps.

Places

Amazon Location Service Places lets you integrate search functionality into your application, convert addresses into geographic coordinates in latitude and longitude (geocoding), and convert a coordinate into a street address (reverse geocoding). Amazon Location Service sources high-quality geospatial data from Esri, Grab, and HERE to support Places functions.

Routing

Amazon Location Service Routes lets you find routes and estimate travel time based on up-to-date roadway and live traffic information. Build features that allow your application to request the travel time, distance, and directions between any two locations. Calculate the time and distance for a matrix of routes to use in route planning.

Geofencing

Amazon Location Service Geofences lets you give your application the ability to detect and act when a device enters or exits a defined geographical boundary known as a geofence. Automatically send an entry or exit event to Amazon EventBridge when a geofence breach is detected. This lets you initiate downstream actions such as sending a notification to a target.

Trackers

Amazon Location Service Trackers lets you retrieve the current and historical location of devices that are running your tracking-enabled application. You can also link trackers with Amazon Location Service geofences to evaluate location updates from your devices against your geofences automatically. Trackers can help you reduce costs by filtering position updates that haven't moved before storing or evaluating them against geofences.

When you use trackers, sensitive location information on your tracked devices does not leave your AWS account. This helps protect sensitive information from third parties, protect user privacy, and reduce security risks.

Services you can use with Amazon Location

Use the following services along with Amazon Location Service.

Integrated monitoring and management

Amazon Location Service is integrated with Amazon CloudWatch, AWS CloudTrail, and Amazon EventBridge for efficient monitoring and data management:

- **Amazon CloudWatch** – View metrics on service usage and health, including requests, latency, faults, and logs. For more information, see the section called “Monitoring with CloudWatch”.
- **AWS CloudTrail** – Log and monitor your API calls, which include actions taken by a user, role or an AWS service. For more information, see the section called “Using CloudTrail with Amazon Location”.
- **Amazon EventBridge** – Enable an event-driven application architecture so you can use AWS Lambda functions to activate other parts of your application and work flows. For more information, see the section called “Reacting to events with EventBridge”.

Developer tools

Amazon Location Service offers a variety of tools for developers to build location-enabled applications. These include the standard AWS SDKs, mobile and web SDKs, and sample code to combine them with open source libraries such as MapLibre. Use the Amazon Location Service console to learn about resources, and to get started with a visual and interactive learning tool.

Quick start with Amazon Location Service

The most efficient way to get started with Amazon Location Service is to use the Amazon Location console. You can create and manage your resources and try the Amazon Location functionality using the Explore page.

> **Note**
>
> To use the Amazon Location Service console, or following the rest of this tutorial, requires that you first complete the Prerequisites for using Amazon Location Service, including creating an AWS account, and allowing access to Amazon Location.

To begin learning about the Amazon Location APIs, use the following tutorial to create a simple application that displays an interactive map and uses search functionality. There are three versions of the tutorial: one shows you how to create a simple webpage using JavaScript, the second shows the same for an Android application using Kotlin, and the third shows the same for an iOS application using Swift.

Topics

- Creating a web app
- Creating an Android app
- Creating an iOS app

Creating a web app

In this section, you will create a static webpage with a map and the ability to search at a location. First you will create your Amazon Location resources and create an API key for your application.

Topics

- Creating Amazon Location resources for your app
- Setting up authentication for your application
- Creating the HTML for your application
- Adding an interactive map to your application
- Adding search to your application

- Seeing the final application
- What's next

Creating Amazon Location resources for your app

If you do not already have them, you must create the Amazon Location resources that your application will use. Here, you create a map resource to display maps in your application, and a place index to search for locations on the map.

To add location resources to your application

1. Choose the map style that you want to use.

   a. In the Amazon Location console, on the Maps page, choose **Create map** to preview map styles.

   b. Add a **Name** and **Description** for the new map resource. Make a note of the name that you use for the map resource. You will need it when creating your script file later in the tutorial.

   c. Choose a map.

   > **Note**
   >
   > Choosing a map style also chooses which map data provider that you will use. If your application is tracking or routing assets that you use in your business, such as delivery vehicles or employees, you may only use HERE as your geolocation provider. For more information, see section 82 of the AWS service terms.

   d. Agree to the **Amazon Location Terms and Conditions**, then choose **Create map**. You can interact with the map that you've chosen: zoom in, zoom out, or pan in any direction.

   e. Make a note of the Amazon Resource Name (ARN) that is shown for your new map resource. You'll use it to create the correct authentication later in this tutorial.

2. Choose the place index that you want to use.

   a. In the Amazon Location console on the **Place indexes** page, choose **Create place index**.

   b. Add a **Name** and **Description** for the new place index resource. Make a note of the name that you use for the place index resource. You will need it when creating your script file later in the tutorial.

c. Choose a data provider.

> **Note**
>
> In most cases, choose the data provider that matches the map provider that you already chose. This helps to ensure that the searches will match the maps.
> If your application is tracking or routing assets that you use in your business, such as delivery vehicles or employees, you may only use HERE as your geolocation provider. For more information, see section 82 of the AWS service terms.

d. Choose the **Data storage option**. For this tutorial, the results are not stored, so you can choose **No, single use only**.

e. Agree to the **Amazon Location Terms and Conditions**, then choose **Create place index**.

f. Make a note of the ARN that is shown for your new place index resource. You'll use it to create the correct authentication in the next section of this tutorial.

Setting up authentication for your application

The application that you create in this tutorial has anonymous usage, meaning that your users are not required to sign into AWS to use the application. However, by default, the Amazon Location Service APIs require authentication to use. You can use either Amazon Cognito or API keys to provide authentication and authorization for anonymous users. In this tutorial, you will create API keys for use in the sample application.

> **Note**
>
> For more information about using API keys or Amazon Cognito with Amazon Location Service, see Granting access to Amazon Location Service.

To set up authentication for your application

1. Go to the Amazon Location console, and choose **API keys** from the left menu.
2. Choose **Create API key**.

> **Important**
>
> The API key that you create must be in the same AWS account and AWS Region as the Amazon Location Service resources that you created in the previous section.

3. One the **Create API key** page, fill in the following information.

   - **Name** – A name for your API key, such as `MyWebAppKey`.
   - **Resources** – Choose the Amazon Location Map and Place index resources that you created in the previous section. You can add more than one resource by choosing **Add resource**. This will allow the API key to be used with those resources.
   - **Actions** – Specify the actions you want to authorize with this API key. You must select at least **geo:GetMap*** and **geo:SearchPlaceIndexfForPosition** so that the tutorial will work as expected.
   - You can optionally add a **Description**, **Expiration time**, or **Tags** to your API key. You can also add a referer (such as `*.example.com`), to limit the key to only being used from a particular domain. This will mean that the tutorial will only work from that domain.

   > **Note**
   >
   > It is recommended that you protect your API key usage by setting either an expiration time or a referer, if not both.

4. Choose **Create API key** to create the API key.
5. Choose **Show API key**, and copy the key value for use later in the tutorial. It will be in the form `v1.public.a1b2c3d4...`.

   > **Important**
   >
   > You will need this key when writing the code for your application later in this tutorial.

Creating the HTML for your application

In this tutorial, you will create a static HTML page that embeds a map, and allows the user to find what's at a location on the map. The app will consist of three files: an HTML file and CSS file for

the webpage, and a JavaScript (.js) file for the code that creates the map and responds to the user's interactions and map events.

First, let's create the HTML and CSS framework that will be used for the application. This will be a simple page with a `<div>` element to hold the map container and a <pre> element to show the JSON responses to your queries.

To create the HTML for your quick start application

1. Create a new file called `quickstart.html`.
2. Edit the file in the text editor or environment of your choice. Add the following HTML to the file.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Quick start tutorial</title>

    <!-- Styles -->
    <link href="main.css" rel="stylesheet" />
  </head>

  <body>
    <header>
      <h1>Quick start tutorial</h1>
    </header>
    <main>
      <div id="map"></div>
      <aside>
        <h2>JSON Response</h2>
        <pre id="response"></pre>
      </aside>
    </main>
    <footer>This is a simple Amazon Location Service app. Pan and zoom. Click to
 see details about entities close to a point.</footer>

  </body>
</html>
```

This HTML has a pointer to the CSS file that you will create in the next step, some placeholder elements for the application, and some explanatory text.

There are two placeholder elements that you will use later in this tutorial. The first is the `<div id="map>` element, which will hold the map control. The second is the `<pre id="response">` element, which will show the results of searching on the map.

3. Save your file.

Now add the CSS for the webpage. This will set the style of the text and placeholder elements for the application.

To create the CSS for your quick start application

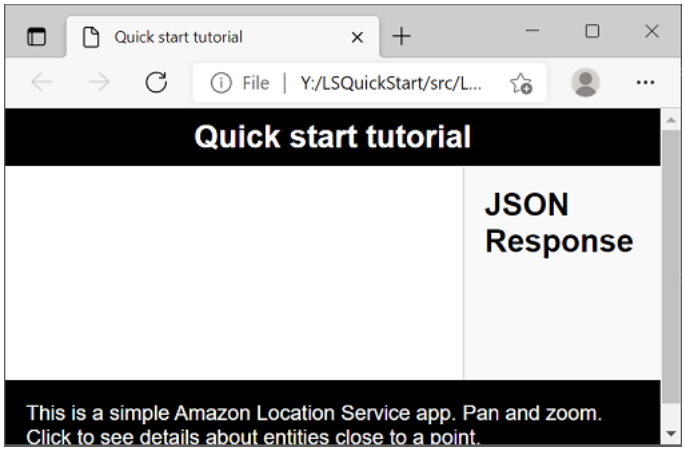
1. Create a new file called `main.css`, in the same folder as the quickstart.html file created in the previous procedure.
2. Edit the file in whatever editor that you want to use. Add the following text to the file.

```
* {
  box-sizing: border-box;
  font-family: Arial, Helvetica, sans-serif;
}

body {
  margin: 0;
}

header {
  background: #000000;
  padding: 0.5rem;
}

h1 {
  margin: 0;
  text-align: center;
  font-size: 1.5rem;
  color: #ffffff;
}

main {
  display: flex;
  min-height: calc(100vh - 94px);
}
```

```
#map {
  flex: 1;
}

aside {
  overflow-y: auto;
  flex: 0 0 30%;
  max-height: calc(100vh - 94px);
  box-shadow: 0 1px 1px 0 #001c244d, 1px 1px 1px 0 #001c2426, -1px 1px 1px 0
 #001c2426;
  background: #f9f9f9;
  padding: 1rem;
}

h2 {
  margin: 0;
}

pre {
  white-space: pre-wrap;
  font-family: monospace;
  color: #16191f;
}

footer {
  background: #000000;
  padding: 1rem;
  color: #ffffff;
}
```

This sets the map to fill the space not used by anything else, sets the area for our responses to take up 30% of the width of the app, and sets color and styles for the title and explanatory text.

3. Save the file.
4. You can now view the `quickstart.html` file in a browser to see the layout of the application.



Next, you will add the map control to the application.

Adding an interactive map to your application

Now that you have a framework and a div placeholder, you can add the map control to your application. This tutorial uses MapLibre GL JS as a map control, getting data from Amazon Location Service. You will also use the JavaScript Authentication helper to facilitate signing of calls to the Amazon Location APIs with your API key.

To add an interactive map to your application

1. Open the `quickstart.html` file that you created in the previous section.
2. Add references to the needed libraries, and the script file that you will create. The changes you need to make are shown in **`green`**.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Quick start tutorial</title>

    <!-- Styles -->
    <link href="https://unpkg.com/maplibre-gl@3.x/dist/maplibre-gl.css"
 rel="stylesheet" />
    <link href="main.css" rel="stylesheet" />
  </head>

  <body>
    ...
```

```
    <footer>This is a simple Amazon Location Service app. Pan and zoom. Click to
 see details about entities close to a point.</footer>

    <!-- JavaScript dependencies -->
    <script src="https://unpkg.com/maplibre-gl@3.x/dist/maplibre-gl.js"></script>
    <script src="https://unpkg.com/@aws/amazon-location-client@1.x/dist/
amazonLocationClient.js"></script>
    <script src="https://unpkg.com/@aws/amazon-location-utilities-auth-helper@1.x/
dist/amazonLocationAuthHelper.js"></script>

    <!-- JavaScript for the app -->
    <script src="main.js"></script>
  </body>
</html>
```

This adds the following dependencies to your app:

- **MapLibre GL JS**. This library and stylesheet include a map control that displays map tiles and includes interactivity, such as pan and zoom. The control also allows extensions, such as drawing your own features on the map.
- **Amazon Location client**. This provides interfaces for the Amazon Location functionality needed to get map data, and to search for places on the map. The Amazon Location client is based on the AWS SDK for JavaScript v3.
- **Amazon Location Authentication Helper**. This provides helpful functions for authenticating Amazon Location Service with API keys or Amazon Cognito.

This step also adds a reference to `main.js`, which you will create next.

3. Save the `quickstart.html` file.
4. Create a new file called `main.js` in the same folder as your HTML and CSS files, and open it for editing.
5. Add the following script to your file. The text in *`red`* should be replaced with the API key value, map resource name, and place resource name that you created earlier, as well as the region identifier for your region (such as `us-east-1`).

```
// Amazon Location Service resource names:
const mapName = "explore.map";
const placesName = "explore.place";
const region = "your_region";
```

```
const apiKey = "v1.public.a1b2c3d4...

// Initialize a map
async function initializeMap() {
  const mlglMap = new maplibregl.Map({
    container: "map", // HTML element ID of map element
    center: [-77.03674, 38.891602], // Initial map centerpoint
    zoom: 16, // Initial map zoom
    style: 'https://maps.geo.${region}.amazonaws.com/maps/v0/maps/${mapName}/style-
descriptor?key=${apiKey}', // Defines the appearance of the map and authenticates
 using an API key
  });

  // Add navigation control to the top left of the map
  mlglMap.addControl(new maplibregl.NavigationControl(), "top-left");

  return mlglMap;
}

async function main() {
  // Initialize map and Amazon Location SDK client:
  const map = await initializeMap();
}

main();
```

This code sets up Amazon Location resources, then configures and initializes a MapLibre GL JS map control and places it in your element with the id map.

The initializeMap() function is important to understand. It creates a new MapLibre map control (called mlglMap locally, but called map in the rest of the code) that is used to render the map in your application.

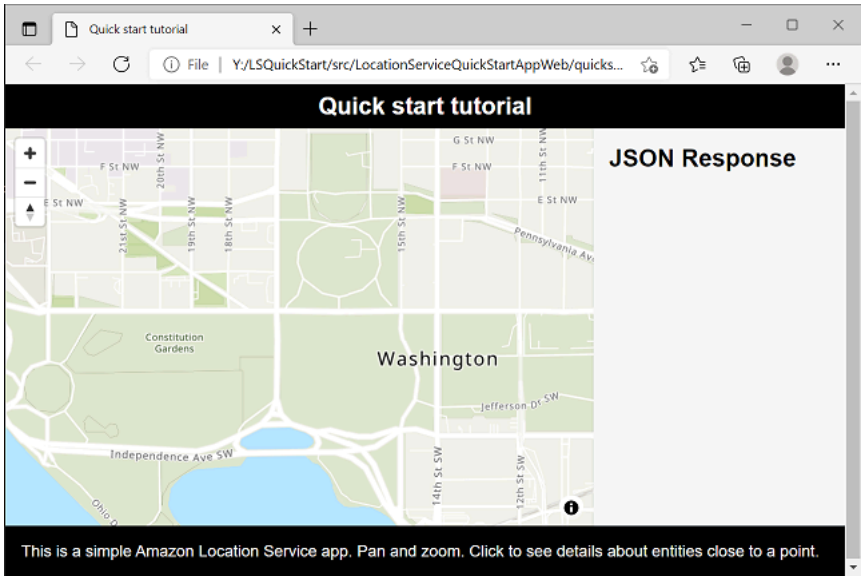
```
  // Initialize the map
  const mlglMap = new maplibregl.Map({
    container: "map", // HTML element ID of map element
    center: [-77.03674, 38.891602], // Initial map centerpoint
    zoom: 16, // Initial map zoom
    style: 'https://maps.geo.${region}.amazonaws.com/maps/v0/maps/${mapName}/style-
descriptor?key=${apiKey}`, // Defines the appearance of the map and authenticates
 using an API key
  });
```

When you create a new MapLibre map control, the parameters that you pass indicate the initial state of the map control. Here, we set the following parameters.

- HTML container, which uses the map div element in our HTML.
- The initial center of the map to a point in Washington, DC.
- The zoom level to 16 (zoomed into a neighborhood or block level).
- The style to use for the map, which gives MapLibre a URL to use to get the map tiles and other information to render the map. Notice that this URL includes your API key for authentication.

6. Save your JavaScript file, and open it with a browser. You now have a map on your page, where you can use pan and zoom actions.

> **Note**
>
> You can use this app to see how the MapLibre map control behaves. You can try using Ctrl or Shift while using a dragging operation, to see other ways to interact with the map. All of this functionality is customizable.



Your app is nearly complete. In the next section, you will handle choosing a location on the map, and show the address of the location chosen. You will also show the resulting JSON on the page, to see the full results.

Adding search to your application

The last step for your application is to add searching on the map. In this case, you will add a reverse geocoding search, where you find the items at a location.

> **Note**
>
> Amazon Location Service also provides the ability to search by name or address to find the locations of places on the map.

To add search functionality to your application

1. Open the `main.js` file that you created in the previous section.
2. Modify the `main` function, as shown. The changes you need to make are shown in **`green`**.

```
async function main() {
  // Create an authentication helper instance using an API key
  const authHelper = await amazonLocationAuthHelper.withAPIKey(apiKey);


  // Initialize map and Amazon Location SDK client:
  const map = await initializeMap();

  const client = new amazonLocationClient.LocationClient({
    region,
    ...authHelper.getLocationClientConfig(), // Provides configuration required to
 make requests to Amazon Location
  });


  // On mouse click, display marker and get results:
  map.on("click", async function (e) {
    // Set up parameters for search call
    let params = {
      IndexName: placesName,
      Position: [e.lngLat.lng, e.lngLat.lat],
      Language: "en",
      MaxResults: "5",
    };

    // Set up command to search for results around clicked point
```

```
    const searchCommand = new
 amazonLocationClient.SearchPlaceIndexForPositionCommand(params);

    try {
      // Make request to search for results around clicked point
      const data = await client.send(searchCommand);

      // Write JSON response data to HTML
      document.querySelector("#response").textContent = JSON.stringify(data,
 undefined, 2);

      // Display place label in an alert box
      alert(data.Results[0].Place.Label);
    } catch (error) {
      // Write JSON response error to HTML
      document.querySelector("#response").textContent = JSON.stringify(error,
 undefined, 2);

      // Display error in an alert box
      alert("There was an error searching.");
    }
  });
}
```

This code starts by setting up the Amazon Location authentication helper to use your API key.

```
const authHelper = await amazonLocationAuthHelper.withAPIKey(apiKey);
```

Then it uses that authentication helper, and the region you are using to create a new Amazon Location client.

```
const client = new amazonLocationClient.LocationClient({
    region,
    ...authHelper.getLocationClientConfig(),
  });
```

Next, the code responds to the user choosing a spot on the map control. It does this by catching a MapLibre provided event for `click`.

```
map.on("click", async function(e) {
    ...
```

```
  });
```

The MapLibre `click` event provides parameters that include the latitude and longitude that the user chose (`e.lngLat`). Within the `click` event, the code creates the `searchPlaceIndexForPositionCommand` to find the entities at the given latitude and longitude.

```
    // Set up parameters for search call
    let params = {
      IndexName: placesName,
      Position: [e.lngLat.lng, e.lngLat.lat],
      Language: "en",
      MaxResults: "5"
    };

    // Set up command to search for results around clicked point
    const searchCommand = new
 amazonLocationClient.SearchPlaceIndexForPositionCommand(params);

    try {
      // Make request to search for results around clicked point
      const data = await client.send(searchCommand);
      ...
    });
```

Here, the `IndexName` is the name of the Place Index resource that you created earlier, the `Position` is the latitude and longitude to search for, `Language` is the preferred language for results, and `MaxResults` tells Amazon Location to return only a maximum of five results.

The remaining code checks for an error, and then displays the results of the search in the `<pre>` element called `response`, and shows the top result in an alert box.

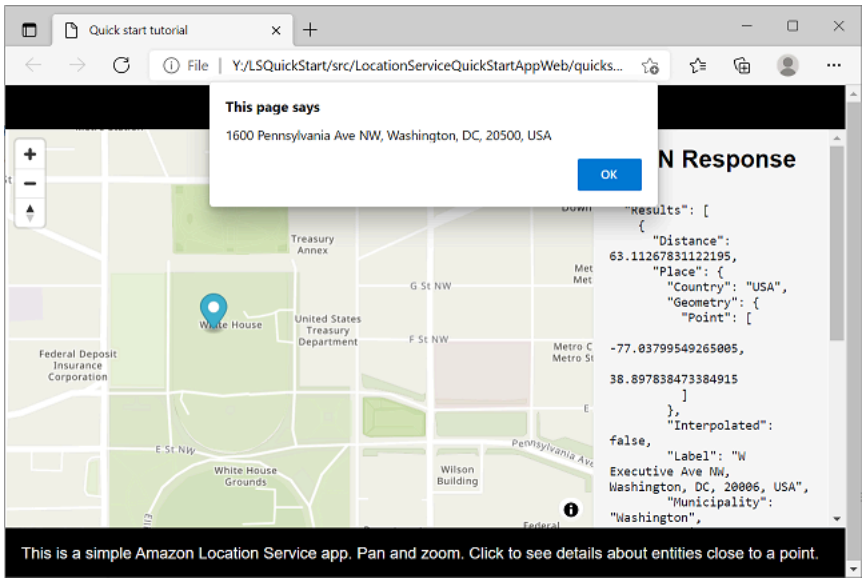
3. (Optional) If you save and open the `quickstart.html` file in a browser now, choosing a location on the map will show you the name or address of the place that you chose.
4. The final step in the application is to use the MapLibre functionality to add a marker on the spot that the user selected. Modify the `main` function as follows. The changes you need to make are shown in **`green`**.

```
async function main() {
  // Create an authentication helper instance using an API key
  const authHelper = await amazonLocationAuthHelper.withAPIKey(apiKey);
```

```
 // Initialize map and Amazon Location SDK client
 const map = await initializeMap();
 const client = new amazonLocationClient.LocationClient({
   region,
   ...authHelper.getLocationClientConfig(), // Provides configuration required to
make requests to Amazon Location
 });

 // Variable to hold marker that will be rendered on click
 let marker;

 // On mouse click, display marker and get results:
 map.on("click", async function (e) {
   // Remove any existing marker
   if (marker) {
     marker.remove();
   }

   // Render a marker on clicked point
   marker = new maplibregl.Marker().setLngLat([e.lngLat.lng,
e.lngLat.lat]).addTo(map);

   // Set up parameters for search call
   let params = {
     IndexName: placesName,
     Position: [e.lngLat.lng, e.lngLat.lat],
     Language: "en",
     MaxResults: "5",
   };

   // Set up command to search for results around clicked point
   const searchCommand = new
amazonLocationClient.SearchPlaceIndexForPositionCommand(params);

...
```

This code declares a `marker` variable, that is populated each time the user selects a location, showing where they selected. The marker is automatically rendered by the map control, once it's added to the map with `.addTo(map);`. The code also checks for a previous marker, and removes it, so that there is only 1 marker on the screen at a time.

5. Save the `main.js` file, and open the `quickstart.html` file in a browser. You can pan and zoom on the map, as before, but now if you choose a location, you will see details about the location that you chose.



Your quick start application is complete. This tutorial has shown you how to create a static HTML application that:

- Creates a map that users can interact with.
- Handles a map event (`click`).
- Calls an Amazon Location Service API, specifically to search the map at a location, using `searchPlaceIndexForPosition`.
- Uses the MapLibre map control to add a marker.

Seeing the final application

The final source code for this application is included in this section. You can also find the final project on GitHub.

You can also find a version of the application that uses Amazon Cognito instead of API keys on GitHub.

Overview

Select each tab to view the final source code of the files in this quick start tutorial.

The files are:

- **quickstart.html** — the framework for your application, including the HTML element holders for the map and search results.
- **main.css** — the stylesheet for the application.
- **main.js** — the script for your application that authenticates the user, creates the map, and searches on a `click` event.

quickstart.html

The HTML framework for the quick start application.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Quick start tutorial</title>

    <!-- Styles -->
    <link href="https://unpkg.com/maplibre-gl@3.x/dist/maplibre-gl.css"
 rel="stylesheet" />
    <link href="main.css" rel="stylesheet" />
  </head>

  <body>
    ...
    <footer>This is a simple Amazon Location Service app. Pan and zoom. Click to see
 details about entities close to a point.</footer>

    <!-- JavaScript dependencies -->
    <script src="https://unpkg.com/maplibre-gl@3.x/dist/maplibre-gl.js"></script>
    <script src="https://unpkg.com/@aws/amazon-location-client@1.x/dist/
amazonLocationClient.js"></script>
    <script src="https://unpkg.com/@aws/amazon-location-utilities-auth-helper@1.x/
dist/amazonLocationAuthHelper.js"></script>

    <!-- JavaScript for the app -->
    <script src="main.js"></script>
  </body>
</html>
```

main.css

The stylesheet for the quick start application.

```
* {
  box-sizing: border-box;
  font-family: Arial, Helvetica, sans-serif;
}

body {
  margin: 0;
}

header {
  background: #000000;
  padding: 0.5rem;
}

h1 {
  margin: 0;
  text-align: center;
  font-size: 1.5rem;
  color: #ffffff;
}

main {
  display: flex;
  min-height: calc(100vh - 94px);
}

#map {
  flex: 1;
}

aside {
  overflow-y: auto;
  flex: 0 0 30%;
  max-height: calc(100vh - 94px);
  box-shadow: 0 1px 1px 0 #001c244d, 1px 1px 1px 0 #001c2426, -1px 1px 1px 0
 #001c2426;
  background: #f9f9f9;
  padding: 1rem;
}
```

```
h2 {
  margin: 0;
}

pre {
  white-space: pre-wrap;
  font-family: monospace;
  color: #16191f;
}

footer {
  background: #000000;
  padding: 1rem;
  color: #ffffff;
}
```

main.js

The code for the quick start application. The text in *red* should be replaced with the appropriate Amazon Location object names.

```
// Amazon Location Service resource names:
const mapName = "explore.map";
const placesName = "explore.place";
const region = "your_region";
const apiKey = "v1.public.a1b2c3d4...

// Initialize a map
async function initializeMap() {
  // Initialize the map
  const mlglMap = new maplibregl.Map({
    container: "map", // HTML element ID of map element
    center: [-77.03674, 38.891602], // Initial map centerpoint
    zoom: 16, // Initial map zoom
    style: `https://maps.geo.${region}.amazonaws.com/maps/v0/maps/${mapName}/style-
descriptor?key=${apiKey}`, // Defines the appearance of the map and authenticates
 using an API key
  });

  // Add navigation control to the top left of the map
  mlglMap.addControl(new maplibregl.NavigationControl(), "top-left");

  return mlglMap;
```

```
}

async function main() {
  // Create an authentication helper instance using an API key
  const authHelper = await amazonLocationAuthHelper.withAPIKey(apiKey);

  // Initialize map and Amazon Location SDK client
  const map = await initializeMap();
  const client = new amazonLocationClient.LocationClient({
    region,
    ...authHelper.getLocationClientConfig(), // Provides configuration required to
 make requests to Amazon Location
  });

  // Variable to hold marker that will be rendered on click
  let marker;

  // On mouse click, display marker and get results:
  map.on("click", async function (e) {
    // Remove any existing marker
    if (marker) {
      marker.remove();
    }

    // Render a marker on clicked point
    marker = new maplibregl.Marker().setLngLat([e.lngLat.lng,
 e.lngLat.lat]).addTo(map);

    // Set up parameters for search call
    let params = {
      IndexName: placesName,
      Position: [e.lngLat.lng, e.lngLat.lat],
      Language: "en",
      MaxResults: "5",
    };

    // Set up command to search for results around clicked point
    const searchCommand = new
 amazonLocationClient.SearchPlaceIndexForPositionCommand(params);

    try {
      // Make request to search for results around clicked point
      const data = await client.send(searchCommand);
```

```
      // Write JSON response data to HTML
      document.querySelector("#response").textContent = JSON.stringify(data,
 undefined, 2);

      // Display place label in an alert box
      alert(data.Results[0].Place.Label);
    } catch (error) {
      // Write JSON response error to HTML
      document.querySelector("#response").textContent = JSON.stringify(error,
 undefined, 2);

      // Display error in an alert box
      alert("There was an error searching.");
    }
  });
}

main();
```

What's next

You have completed the quick start tutorial, and should have an idea of how Amazon Location Service is used to build applications. To get more out of Amazon Location, you can check out the following resources:

- Dive deeper into the concepts of Amazon Location Service
- Get more information about how to use Amazon Location features and functionality
- See how to expand on this sample and build more complex applications by looking at code examples using Amazon Location

Creating an Android app

In this section, you will create an Android application with a map, the ability to search at a location and tracking in the foreground. First, you will create your Amazon Location resources, an Amazon Cognito identity and an API key for your application.

Topics

- Creating Amazon Location resources for your app

- Setting up authentication for your application
- Creating the base Android application
- Adding an interactive map to your application
- Adding reverse geocoding search to your application
- Adding tracking to your application
- What's next

Creating Amazon Location resources for your app

If you do not already have them, you must create the Amazon Location resources that your application will use. Here, you create a map resource to display maps in your application, a place index to search for locations on the map, and a tracker to track an object across the map.

To add location resources to your application

1. Choose the map style that you want to use.

   a. In the Amazon Location console, on the Maps page, choose **Create map** to preview map styles.

   b. Add a **Name** and **Description** for the new map resource. Make a note of the name that you use for the map resource. You will need it when creating your script file later in the tutorial.

   c. We recommend you choose the HERE map stlye for your map.

   > **Note**
   >
   > Choosing a map style also chooses which map data provider that you will use. If your application is tracking or routing assets that you use in your business, such as delivery vehicles or employees, you may only use HERE as your geolocation provider. For more information, see section 82 of the AWS service terms.

   d. Agree to the **Amazon Location Terms and Conditions**, then choose **Create map**. You can interact with the map that you've chosen: zoom in, zoom out, or pan in any direction.

   e. Make a note of the Amazon Resource Name (ARN) that is shown for your new map resource. You'll use it to create the correct authentication later in this tutorial.

2. Choose the place index that you want to use.

a. In the Amazon Location console on the **Place indexes** page, choose **Create place index**.

b. Add a **Name** and **Description** for the new place index resource. Make a note of the name that you use for the place index resource. You will need it when creating your script file later in the tutorial.

c. Choose a data provider.

> **Note**
>
> In most cases, choose the data provider that matches the map provider that you already chose. This helps to ensure that the searches will match the maps.
> If your application is tracking or routing assets that you use in your business, such as delivery vehicles or employees, you may only use HERE as your geolocation provider. For more information, see section 82 of the AWS service terms.

d. Choose the **Data storage option**. For this tutorial, the results are not stored, so you can choose **No, single use only**.

e. Agree to the **Amazon Location Terms and Conditions**, then choose **Create place index**.

f. Make a note of the ARN that is shown for your new place index resource. You'll use it to create the correct authentication in the next section of this tutorial.

3. To create a tracker using the Amazon Location console.

a. Open the Amazon Location Service console.

b. In the left navigation pane, choose **Trackers**.

c. Choose **Create tracker**.

d. Fill in the all the required fields.

e. Under **Position filtering**, we recommened you use the default setting: **TimeBased**.

f. Choose **Create tracker** to finish.

Setting up authentication for your application

The application that you create in this tutorial has anonymous usage, meaning that your users are not required to sign into AWS to use the application. However, the Amazon Location Service APIs require authentication to use. You can use either API keys or Amazon Cognito to provide

authentication and authorization for anonymous users. This tutorial will use Amazon Cognito and API keys to authenticate your application.

> **Note**
>
> For more information about using Amazon Cognito or API keys with Amazon Location Service, see Granting access to Amazon Location Service.

The following tutorials show you how to set up authentication for the map, the place index, and tracker you created in as well setting up permissions for Amazon Location.

Set up authentication

1. Navigate to the Amazon Location console and select **API keys** from the left-hand menu.
2. Click on 'Create API key'. Remember that the API key must be in the same AWS account and region as the previously created Amazon Location Service resources.
3. Fill in the required details on the 'Create API key' page:
   - Name: Provide a name for your API key, like `MyAppKey`.
   - Resources: Choose the Amazon Location Service Map and Place index resources created earlier. You can add multiple resources by selecting 'Add Resource'. This allows the API key to be used with specified resources.
   - Actions: Specify authorized actions for this API key. At a minimum, select `geo:GetMap` and `geo:SearchPlaceIndexForPosition` to ensure the tutorial functions as intended.
   - Optional you can add a Description, Expiration time, Tags, or a referer for example `https://www.example.com` to limit the key's usage to a specific domain, enabling the tutorial to function only within that domain.
4. Click **Create API Key** to generate the API key.
5. Select **Show API Key** and copy the key value for example `v1.public.a1b2c3d4` for later use in the tutorial.

Create an IAM policy for tracking

1. Sign in to the IAM console at https://console.aws.amazon.com/iam/ with your user that has administrator permissions.

2. In the navigation pane, choose Policies.
3. In the content pane, choose Create policy.
4. Choose the **JSON** option, then copy and paste this JSON policy into the JSON text box.

```
{
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "geo:GetMapTile",
                    "geo:GetMapStyleDescriptor",
                    "geo:GetMapSprites",
                    "geo:GetMapGlyphs",
                    "geo:SearchPlaceIndexForPosition",
                    "geo:GetDevicePositionHistory",
                    "geo:BatchUpdateDevicePosition"
                ],
                "Resource": [
                    "arn:aws:geo:{Region}:{Account}:map/{MapName}",
                    "arn:aws:geo:{Region}:{Account}:place-index/{IndexName}",
                    "arn:aws:geo:{Region}:{Account}:tracker/{TrackerName}"
                ]
            }
        ]
    }
```

This is a policy example for Tracking. To use the example for your own policy, replace the `Region`, `Account`, and **TrackerName** placeholders.

> **Note**
>
> While unauthenticated identity pools are intended for exposure on unsecured internet sites, note that they will be exchanged for standard, time-limited AWS credentials. It's important to scope the IAM roles associated with unauthenticated identity pools appropriately. For more information about using and appropriately scoping policies in Amazon Cognito with Amazon Location Service, see Granting access to Amazon Location Service.

5. On the Review and Create page, provide a name for the policy name field. Review the permissions granted by your policy, and then choose Create Policy to save your work.

The new policy appears in the list of managed policies and is ready to attach.

Set up authentication for your tracking

1. Set up authentication for your map application in the Amazon Cognito console.
2. Open the **Identity pools** page.

> **Note**
>
> The pool that you create must be in the same AWS account and AWS Region as the Amazon Location Service resources that you created in the previous section.

3. Choose **Create Identity pool**.
4. Starting with the **Configure identity pool trust** step. For user access authentication, select **Guest access**, and press next.
5. On the **Configure permissions** page select the **Use an existing IAM role** and enter the name of the IAM role you created in the previous step. When ready press next to move on to the next step.
6. On the **Configure properties** page, provide a name for your identity pool. Then press **Next**.
7. On the **Review and create** page, review all the information present then press **Create identity pool**.
8. Open the **Identity pools** page, and select the identity pool you just created. Then copy or write down the IdentityPoolId that you will use later in your browser script.

Creating the base Android application

In this tutorial, you will create an Android application that embeds a map and allows the user to find what's at a location on the map.

First, create an empty Kotlin application using Android Studio's new project wizard.

To create an empty application (AndroidStudio)

1. Start AndroidStudio. Open the menu, and choose **File**, **New**, **New Project**.

2. From the **Phone and Tablet** tab, select **Empty Activity**, and then choose **Next**.
3. Choose a **Name**, **Package name**, and **Save location** for your application.
4. In the dropdown list for **Language**, select **Kotlin**.
5. Choose **Finish** to create your blank application.

Adding an interactive map to your application

Now that you have created a basic application, you can add map control to your application. This tutorial uses API keys for managing the map view. The map control itself is part of the MapLibre Native library, with the API key and MapLibre, and the map data comes from Amazon Location.

To add a map to your application you will have to perform the following actions:

- Add the MapLibre dependency to your project.
- Set up the map view code with compose.
- Write code to show the map.

Use the following procedure to add the map to your app:

1. Add the MapLibre dependency to your project

    a. In AndroidStudio, select the **View** menu, and choose **Tool Windows**, **Project**. This will open the Project window, which gives you access to all the files in your project.

    b. In the **Project** window, open **gradle** then open the `libs.versions.toml` file in the tree view. This will open the `libs.versions.toml` file for editing. Now add the below version and libraries data in the `libs.versions.toml` file.

```
[versions]
 ...
 auth = "0.2.4"
 tracking = "0.2.4"

 [libraries]
 ...
 auth = { group = "software.amazon.location", name = "auth", version.ref =
 "auth" }
 tracking = { module = "software.amazon.location:tracking", version.ref =
 "tracking" }
```

```
[plugins]
...
```

c. After you finish editing the `libs.versions.toml` file, AndroidStudio must re-sync the project. At the top of the `libs.versions.toml` editing window, AndroidStudio prompts you to sync. Select 'Sync Now' to sync your project before continuing.

d. In the Project window, open Gradle Scripts in the tree view and select the `build.gradle` file for your application module. This will open the `build.gradle` file for editing.

e. At the bottom of the file, in the dependencies section, add the following dependency.

```
dependencies {
    ...
    implementation(libs.org.maplibre.gl)
}
```

f. After you finish adding the Gradle dependencies, Android Studio must re-sync the project. At the top of the **build.gradle** editing window, Android Studio, select **Sync Now** to sync your project before continuing.

2. Now you will set up the map view code with compose. Use the following steps:

a. From the Project window, open App, Java, *`your package name`* in the tree view, and go to the **ui** folder, inside the **ui** folder create a **view** directory.

b. Inside **view** directory create a `MapLoadScreen.kt` file.

c. Add the following code to your `MapLoadScreen.kt` file.

```
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.viewinterop.AndroidView
import org.maplibre.android.maps.OnMapReadyCallback

 @Composable
 fun MapLoadScreen(
     mapReadyCallback: OnMapReadyCallback,
 ) {
     Box(
         modifier = Modifier
```

```
            .fillMaxWidth()
            .fillMaxHeight(),
    ) {
        MapView(mapReadyCallback)
    }
}

@Composable
fun MapView(mapReadyCallback: OnMapReadyCallback) {
    AndroidView(
        factory = { context ->
            val mapView = org.maplibre.android.maps.MapView(context)
            mapView.onCreate(null)
            mapView.getMapAsync(mapReadyCallback)
            mapView
        },
    )
}
```

3. Write code to show the map.

   a. Add the following code to your `MainActivity.kt` file.

```
// ...other imports
import org.maplibre.android.MapLibre
import org.maplibre.android.camera.CameraPosition
import org.maplibre.android.geometry.LatLng
import org.maplibre.android.maps.MapLibreMap
import org.maplibre.android.maps.OnMapReadyCallback
import org.maplibre.android.maps.Style

class MainActivity : ComponentActivity(), OnMapReadyCallback {
    private val region = "YOUR_AWS_REGION"
    private val mapName = "YOUR_AWS_MAP_NAME"
    private val apiKey = "YOUR_AWS_API_KEY"
    override fun onCreate(savedInstanceState: Bundle?) {
        MapLibre.getInstance(this)
        super.onCreate(savedInstanceState)
        setContent {
            TestMapAppTheme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
```

```
                    MapLoadScreen(this)
                }
            }
        }
    }

    override fun onMapReady(map: MapLibreMap) {
        map.setStyle(
            Style.Builder()
                .fromUri(
                    "https://maps.geo.$region.amazonaws.com/maps/v0/maps/
$mapName/style-descriptor?key=$apiKey"
                ),
        ) {
            map.uiSettings.isAttributionEnabled = true
            map.uiSettings.isLogoEnabled = false
            map.uiSettings.attributionGravity = Gravity.BOTTOM or Gravity.END
            val initialPosition = LatLng(47.6160281982247,
 -122.32642111977668)
            map.cameraPosition = CameraPosition.Builder()
                .target(initialPosition)
                .zoom(14.0)
                .build()
        }
    }
 }
```

b. Save the `MainActivity.kt` file. You can now build the application. To run it, you may have to set up a device to emulate it in AndroidStudio, or use the app on your device. Use this app to see how the map control behaves. You can pan by dragging it on the map and pinching it to zoom.

   In the next section, you will add a marker on the map and show the address of the location where the marker is as you move the map.

Adding reverse geocoding search to your application

You will now add reverse geocoding search to your application, where you find the items at a location. To simplify using an Android app, we will search the center of the screen. To find a new location, move the map to where you want to search. We will place a marker at the centre of the map to show where we are searching.

Adding a reverse geocoding search will consist of two parts.

- Add a marker at the centre of the screen to show the user where we are searching.
- Add a text box for results, then search for what is at the marker's location and show it in the text box.

To add a marker to your application

1. Save this image to your project in the `app/res/drawable` folder as `red_marker.png` (you can also access the image from GitHub. Alternatively, you can create your image. You can also use a .png file with transparency for the parts you don't want shown.
2. Add the following code to your **MapLoadScreen.kt** file.

```
// ...other imports
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.size
import androidx.compose.ui.Alignment
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.unit.dp
import com.amazon.testmapapp.R

@Composable
fun MapLoadScreen(
    mapReadyCallback: OnMapReadyCallback,
) {
    Box(
        modifier = Modifier
            .fillMaxWidth()
            .fillMaxHeight(),
    ) {
        MapView(mapReadyCallback)
        Box(
            modifier = Modifier
                .align(Alignment.Center),
        ) {
            Image(
                painter = painterResource(id = R.drawable.red_marker),
                contentDescription = "marker",
                modifier = Modifier
                    .size(40.dp)
```

```
                    .align(Alignment.Center),
            )
        }
    }
}

@Composable
fun MapView(mapReadyCallback: OnMapReadyCallback) {
    AndroidView(
        factory = { context ->
            val mapView = org.maplibre.android.maps.MapView(context)
            mapView.onCreate(null)
            mapView.getMapAsync(mapReadyCallback)
            mapView
        },
    )
}
```

3. Build and run your app to preview the functionality.

Your app now has a marker on the screen. In this case, it is a static image that doesn't move. It is used to show the centre of the map view, which is where we will search. In the following procedure, we will add the search at that location.

To add reverse geocoding search at a location to your app

1. In the **Project** window, open **gradle** to `libs.versions.toml` file in the tree view. This will open the `libs.versions.toml` file for editing. Now add the below version and libraries data in the `libs.versions.toml` file.

```
[versions]
 ...
 okhttp = "4.12.0"

 [libraries]
 ...
 com-squareup-okhttp3 = { group = "com.squareup.okhttp3", name = "okhttp",
 version.ref = "okhttp" }

 [plugins]
 ...
```

2. After you finish editing the `libs.versions.toml` file, AndroidStudio must re-sync the project. At the top of the `libs.versions.toml` editing window, AndroidStudio prompts you to sync. Select 'Sync Now' to sync your project before continuing.
3. In the Project window, open Gradle Scripts in the tree view and select the `build.gradle` file for your application module. This will open the `build.gradle` file for editing.
4. At the bottom of the file, in the dependencies section, add the following dependency.

```
dependencies {
    ...
    implementation(libs.com.squareup.okhttp3)
 }
```

5. After you finish editing the Gradle dependencies, AndroidStudio must re-sync the project. At the top of the `build.gradle` editing window, AndroidStudio prompts you to sync. Select **SyncNow** to sync your project before continuing.
6. Now in the tree view add the **data**, to the **request** directory, and create the `ReverseGeocodeRequest.kt` data class. Add the following code to the class.

```
import com.google.gson.annotations.SerializedName

data class ReverseGeocodeRequest(
    @SerializedName("Language")
    val language: String,
    @SerializedName("MaxResults")
    val maxResults: Int,
    @SerializedName("Position")
    val position: ListDouble
)
```

7. Now in the tree view add the **data** to **response** directory, and create the `ReverseGeocodeResponse.kt` data class. Add the following code inside it.

```
import com.google.gson.annotations.SerializedName

data class ReverseGeocodeResponse(
    @SerializedName("Results")
    val results: ListResult
)

data class Result(
```

```
    @SerializedName("Place")
    val place: Place
)

data class Place(
    @SerializedName("Label")
    val label: String
)
```

8. Now, From the Project window, open App, Java, *your package name* in the tree view, and go to the **ui** folder, inside **ui** folder create **viewModel** directory.
9. Inside **viewModel** directory create `MainViewModel.kt` file.
10. Add the following code to your `MainViewModel.kt` file.

```
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.setValue
import androidx.lifecycle.ViewModel
import com.amazon.testmapapp.data.request.ReverseGeocodeRequest
import com.amazon.testmapapp.data.response.ReverseGeocodeResponse
import com.google.gson.Gson
import java.io.IOException
import okhttp3.Call
import okhttp3.Callback
import okhttp3.MediaType.Companion.toMediaTypeOrNull
import okhttp3.OkHttpClient
import okhttp3.Request
import okhttp3.RequestBody.Companion.toRequestBody
import okhttp3.Response
import org.maplibre.android.geometry.LatLng
import org.maplibre.android.maps.MapLibreMap

class MainViewModel : ViewModel() {
    var label by mutableStateOf("")
    var isLabelAdded: Boolean by mutableStateOf(false)
    var client = OkHttpClient()
    var mapLibreMap: MapLibreMap? = null

    fun reverseGeocode(latLng: LatLng, apiKey: String) {
        val region = "YOUR_AWS_REGION"
        val indexName = "YOUR_AWS_PLACE_INDEX"
        val url =
```

```
            "https://places.geo.${region}.amazonaws.com/places/v0/indexes/
${indexName}/search/position?key=${apiKey}"

        val requestBody = ReverseGeocodeRequest(
            language = "en",
            maxResults = 1,
            position = listOf(latLng.longitude, latLng.latitude)
        )
        val json = Gson().toJson(requestBody)

        val mediaType = "application/json".toMediaTypeOrNull()
        val request = Request.Builder()
            .url(url)
            .post(json.toRequestBody(mediaType))
            .build()

        client.newCall(request).enqueue(object : Callback {
            override fun onFailure(call: Call, e: IOException) {
                e.printStackTrace()
            }

            override fun onResponse(call: Call, response: Response) {
                if (response.isSuccessful) {
                    val jsonResponse = response.body?.string()

                    val reverseGeocodeResponse =
                        Gson().fromJson(jsonResponse,
 ReverseGeocodeResponse::class.java)

                    val responseLabel =
 reverseGeocodeResponse.results.firstOrNull()?.place?.label

                    if (responseLabel != null) {
                        label = responseLabel
                        isLabelAdded = true
                    }
                }
            }
        })
    }
}
```

11. If it's not open already, open the `MapLoadScreen.kt` file, as in the previous procedure. Add the following code. This will create a compose Text view where you will see reverse geocoding search results at the location.

```
// ...other imports
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Text
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.testTag
import androidx.compose.ui.semantics.contentDescription
import androidx.compose.ui.semantics.semantics
import androidx.compose.ui.unit.sp
import com.amazon.testmapapp.ui.viewModel.MainViewModel

@Composable
fun MapLoadScreen(
    mapReadyCallback: OnMapReadyCallback,
    mainViewModel: MainViewModel,
) {
    Box(
        modifier = Modifier
            .fillMaxWidth()
            .fillMaxHeight(),
    ) {
        MapView(mapReadyCallback)
        Box(
            modifier = Modifier
                .align(Alignment.Center),
        ) {
            Image(
                painter = painterResource(id = R.drawable.red_marker),
                contentDescription = "marker",
                modifier = Modifier
                    .size(40.dp)
                    .align(Alignment.Center),
            )
```

```
        }
        if (mainViewModel.isLabelAdded) {
            Column(
                modifier = Modifier.fillMaxSize(),
                verticalArrangement = Arrangement.Bottom
            ) {
                Box(
                    modifier = Modifier
                        .fillMaxWidth()
                        .background(Color.White),
                ) {
                    Text(
                        text = mainViewModel.label,
                        modifier = Modifier
                            .padding(16.dp)
                            .align(Alignment.Center)
                            .testTag("label")
                            .semantics {
                                contentDescription = "label"
                            },
                        fontSize = 14.sp,
                    )
                }
                Spacer(modifier = Modifier.height(80.dp))
            }
        }
    }
}

@Composable
fun MapView(mapReadyCallback: OnMapReadyCallback) {
    AndroidView(
        factory = { context ->
            val mapView = org.maplibre.android.maps.MapView(context)
            mapView.onCreate(null)
            mapView.getMapAsync(mapReadyCallback)
            mapView
        },
    )
}
```

12. In the app, under java, in the **package name** folder in AndroidStudio, open the `MainActivity.kt` file. Modify the code as shown.

```
// ...other imports
import androidx.activity.viewModels
import com.amazon.testmapapp.ui.viewModel.MainViewModel

 class MainActivity : ComponentActivity(), OnMapReadyCallback,
 MapLibreMap.OnCameraMoveStartedListener, MapLibreMap.OnCameraIdleListener {

     private val mainViewModel: MainViewModel by viewModels()
     private val region = "YOUR_AWS_REGION"
     private val mapName = "YOUR_AWS_MAP_NAME"
     private val apiKey = "YOUR_AWS_API_KEY"
     override fun onCreate(savedInstanceState: Bundle?) {
         MapLibre.getInstance(this)
         super.onCreate(savedInstanceState)
         setContent {
             TestMapAppTheme {
                 Surface(
                     modifier = Modifier.fillMaxSize(),
                     color = MaterialTheme.colorScheme.background
                 ) {
                     MapLoadScreen(this, mainViewModel)
                 }
             }
         }
     }

     override fun onMapReady(map: MapLibreMap) {
         map.setStyle(
             Style.Builder()
                 .fromUri(
                     "https://maps.geo.$region.amazonaws.com/maps/v0/maps/$mapName/
style-descriptor?key=$apiKey"
                 ),
         ) {
             map.uiSettings.isAttributionEnabled = true
             map.uiSettings.isLogoEnabled = false
             map.uiSettings.attributionGravity = Gravity.BOTTOM or Gravity.END
             val initialPosition = LatLng(47.6160281982247, -122.32642111977668)
             map.cameraPosition = CameraPosition.Builder()
                 .target(initialPosition)
                 .zoom(14.0)
                 .build()
```

```
            map.addOnCameraMoveStartedListener(this)
            map.addOnCameraIdleListener(this)
            map.cameraPosition.target?.let { latLng ->
                mainViewModel.reverseGeocode(
                    LatLng(
                        latLng.latitude,
                        latLng.longitude
                    ), apiKey
                )
            }
        }
    }
    override fun onCameraMoveStarted(p0: Int) {
        mainViewModel.label = ""
        mainViewModel.isLabelAdded = false
    }

    override fun onCameraIdle() {
        if (!mainViewModel.isLabelAdded) {
            mainViewModel.mapLibreMap?.cameraPosition?.target?.let { latLng ->
                mainViewModel.reverseGeocode(
                    LatLng(
                        latLng.latitude,
                        latLng.longitude
                    ), apiKey
                )
            }
        }
    }
}
```

This code works with the map view. A virtual camera position defines the map view in MapLibre. Moving the map can be thought of as moving that virtual camera.

- ViewModel has a label variable: This variable sets data in compose text view.
- **onMapReady**:This function is updated to register two new events.
- The **onCameraMove** event happens whenever the user is moving the map. In general, when moving the map, we want to hide the search until the user is done moving the map.
- The **onCameraIdle** event occurs when the user pauses moving the map. This event calls our reverse geocode function to search at the centre of the map.

- `reverseGeocode(latLng: LatLng, apiKey: String)`: This function, called in the event onCameraIdle, searches at the centre of the map for a location and updates the label to show the results. It uses the camera target, which defines the centre of the map (where the camera is looking).

13. Save your files, and build and run your app to see if it works.

Your quick-start application with search functionality is complete.

Adding tracking to your application

To add tracking to your sample application, follow these steps:

1. Add tracking and auth SDK dependencies to your project.
2. Include permission and service entries in your AndroidManifest.xml file.
3. Set up the start/stop tracking button code with compose.
4. Add code for creating a LocationTracker object and start and stop tracking.
5. Create a test route with Android Emulator.

1. **Add tracking and auth SDK dependencies to your project.**

   a. In the **Project** window, open **gradle** then open the `libs.versions.toml` file in the tree view. This will open the `libs.versions.toml` file for editing. Now add the below version and libraries data in the `libs.versions.toml` file.

```
[versions]
 ...
 auth = "0.0.1"
 tracking = "0.0.1"

 [libraries]
 ...
 auth = { group = "software.amazon.location", name = "auth", version.ref =
 "auth" }
 tracking = { module = "software.amazon.location:tracking", version.ref =
 "tracking" }

 [plugins]
 ...
```

b. After you finish editing the `libs.versions.toml` file, AndroidStudio must re-sync the project. At the top of the `libs.versions.toml` editing window, AndroidStudio prompts you to sync. Select 'Sync Now' to sync your project before continuing.

c. In the Project window, open Gradle Scripts in the tree view and select the `build.gradle` file for your application module. This will open the `build.gradle` file for editing.

d. At the bottom of the file, in the **dependencies** section, add the following dependency.

```
dependencies {
    ...
    implementation(libs.auth)
    implementation(libs.tracking)
 }
```

e. After you finish editing the Gradle dependencies, AndroidStudio must re-sync the project. At the top of the **build.gradle** editing window, AndroidStudio prompts you to sync. Select **SyncNow** to sync your project before continuing.

2. **Include permission and service entries in your AndroidManifest.xml file.**

- To include the correct permission and service entries in your `AndroidManifest.xml file`, update the file with the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.AndroidQuickStartApp"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
```

```
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.AndroidQuickStartApp">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

3. **Set up the start/stop tracking button code with compose.**

   a. Add two images of Play and Pause in **res** under **drawable** named as **ic_pause** and **ic_play**. You can also access the image from GitHub.

   b. If it's not open already, open the `MapLoadScreen.kt` file, as in the previous procedure. Add the following code. This will create a compose Button view where we can click on it to **start** and **stop** tracking.

```
// ...other imports
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults

@Composable
fun MapLoadScreen(
    mapReadyCallback: OnMapReadyCallback,
    mainViewModel: MainViewModel,
    onStartStopTrackingClick: () -> Unit
) {
    Box(
        modifier = Modifier
            .fillMaxWidth()
            .fillMaxHeight(),
    ) {
        MapView(mapReadyCallback)
        Box(
            modifier = Modifier
                .align(Alignment.Center),
        ) {
            Image(
                painter = painterResource(id = R.drawable.red_marker),
```

```
                contentDescription = "marker",
                modifier = Modifier
                    .size(40.dp)
                    .align(Alignment.Center),
            )
        }
        if (mainViewModel.isLabelAdded) {
            Column(
                modifier = Modifier.fillMaxSize(),
                verticalArrangement = Arrangement.Bottom
            ) {
                Box(
                    modifier = Modifier
                        .fillMaxWidth()
                        .background(Color.White),
                ) {
                    Text(
                        text = mainViewModel.label,
                        modifier = Modifier
                            .padding(16.dp)
                            .align(Alignment.Center)
                            .testTag("label")
                            .semantics {
                                contentDescription = "label"
                            },
                        fontSize = 14.sp,
                    )
                }
                Spacer(modifier = Modifier.height(80.dp))
            }
        }
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(bottom = 16.dp),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Bottom,
        ) {
            Button(
                onClick = onStartStopTrackingClick,
                modifier = Modifier
                    .padding(horizontal = 16.dp)
            ) {
                Text(
```

```
                    text = if
 (mainViewModel.isLocationTrackingForegroundActive) "Stop tracking" else "Start
 tracking",
                    color = Color.Black
                )
                Spacer(modifier = Modifier.size(ButtonDefaults.IconSpacing))
                Image(
                    painter = painterResource(id = if
 (mainViewModel.isLocationTrackingForegroundActive) R.drawable.ic_pause else
 R.drawable.ic_play),
                    contentDescription = if
 (mainViewModel.isLocationTrackingForegroundActive) "stop_tracking" else
 "start_tracking"
                )
            }
        }
    }
}

@Composable
fun MapView(mapReadyCallback: OnMapReadyCallback) {
    AndroidView(
        factory = { context ->
            val mapView = org.maplibre.android.maps.MapView(context)
            mapView.onCreate(null)
            mapView.getMapAsync(mapReadyCallback)
            mapView
        },
    )
}
```

4. **Add code for creating a LocationTracker object and start and stop tracking.**

   a. Add the following code inside the `MainViewModel.kt` file.

```
...
var isLocationTrackingForegroundActive: Boolean by mutableStateOf(false)
var locationTracker: LocationTracker? = null
```

   b. Add the following code to your `MainActivity.kt` file.

```
// ...other imports
import software.amazon.location.auth.AuthHelper
```

```
import software.amazon.location.auth.LocationCredentialsProvider
import software.amazon.location.tracking.LocationTracker
import software.amazon.location.tracking.aws.LocationTrackingCallback
import software.amazon.location.tracking.config.LocationTrackerConfig
import software.amazon.location.tracking.database.LocationEntry
import software.amazon.location.tracking.filters.DistanceLocationFilter
import software.amazon.location.tracking.filters.TimeLocationFilter
import software.amazon.location.tracking.util.TrackingSdkLogLevel

class MainActivity : ComponentActivity(), OnMapReadyCallback,
    MapLibreMap.OnCameraMoveStartedListener, MapLibreMap.OnCameraIdleListener {

    private val mainViewModel: MainViewModel by viewModels()
    private val poolId = "YOUR_AWS_IDENTITY_POOL_ID"
    private val trackerName = "YOUR_AWS_TRACKER_NAME"
    private val region = "YOUR_AWS_REGION"
    private val mapName = "YOUR_AWS_MAP_NAME"
    private val apiKey = "YOUR_AWS_API_KEY"
    private val coroutineScope = MainScope()
    private lateinit var locationCredentialsProvider:
 LocationCredentialsProvider
    private lateinit var authHelper: AuthHelper

    override fun onCreate(savedInstanceState: Bundle?) {
        MapLibre.getInstance(this)
        super.onCreate(savedInstanceState)
        setContent {
            TestMapAppTheme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    MapLoadScreen(this, mainViewModel, onStartStopTrackingClick
 = {
                        if (mainViewModel.isLocationTrackingForegroundActive) {
                            mainViewModel.isLocationTrackingForegroundActive =
 false
                            mainViewModel.locationTracker?.stop()
                        } else {
                            if (checkLocationPermission(this))
 return@MapLoadScreen
                            mainViewModel.isLocationTrackingForegroundActive =
 true
```

```
mainViewModel.locationTracker?.start(locationTrackingCallback = object :
                                LocationTrackingCallback {
                                override fun
onLocationAvailabilityChanged(locationAvailable: Boolean) {
                                }

                                override fun onLocationReceived(location:
LocationEntry) {
                                }

                                override fun onUploadSkipped(entries:
LocationEntry) {
                                }

                                override fun onUploadStarted(entries:
ListLocationEntry) {
                                }

                                override fun onUploaded(entries:
ListLocationEntry) {
                                }

                            })
                        }
                    })
                }
            }
        }
        authenticateUser()
    }

    private fun authenticateUser() {
        coroutineScope.launch {
            authHelper = AuthHelper(applicationContext)
            locationCredentialsProvider =
authHelper.authenticateWithCognitoIdentityPool(
                poolId,
            )
            locationCredentialsProvider.let {
                val config = LocationTrackerConfig(
                    trackerName = trackerName,
                    logLevel = TrackingSdkLogLevel.DEBUG,
                    latency = 1000,
```

```
                    frequency = 5000,
                    waitForAccurateLocation = false,
                    minUpdateIntervalMillis = 5000,
                )
                mainViewModel.locationTracker = LocationTracker(
                    applicationContext,
                    it,
                    config,
                )

 mainViewModel.locationTracker?.enableFilter(TimeLocationFilter())

 mainViewModel.locationTracker?.enableFilter(DistanceLocationFilter())
            }
        }
    }

    private fun checkLocationPermission(context: Context) =
 ActivityCompat.checkSelfPermission(
        context,
        Manifest.permission.ACCESS_FINE_LOCATION,
    ) != PackageManager.PERMISSION_GRANTED &&
 ActivityCompat.checkSelfPermission(
        context,
        Manifest.permission.ACCESS_COARSE_LOCATION,
    ) != PackageManager.PERMISSION_GRANTED

    override fun onMapReady(map: MapLibreMap) {
        map.setStyle(
            Style.Builder()
                .fromUri(
                    "https://maps.geo.$region.amazonaws.com/maps/v0/maps/
$mapName/style-descriptor?key=$apiKey"
                ),
        ) {
            mainViewModel.mapLibreMap = map
            map.uiSettings.isAttributionEnabled = true
            map.uiSettings.isLogoEnabled = false
            map.uiSettings.attributionGravity = Gravity.BOTTOM or Gravity.END
            val initialPosition = LatLng(47.6160281982247, -122.32642111977668)
            map.cameraPosition = CameraPosition.Builder()
                .target(initialPosition)
                .zoom(14.0)
                .build()
```

```
            map.addOnCameraMoveStartedListener(this)
            map.addOnCameraIdleListener(this)
            map.cameraPosition.target?.let { latLng ->
                mainViewModel.reverseGeocode(
                    LatLng(
                        latLng.latitude,
                        latLng.longitude
                    ), apiKey
                )
            }
        }
    }

    override fun onCameraMoveStarted(p0: Int) {
        mainViewModel.label = ""
        mainViewModel.isLabelAdded = false
    }

    override fun onCameraIdle() {
        if (!mainViewModel.isLabelAdded) {
            mainViewModel.mapLibreMap?.cameraPosition?.target?.let { latLng ->
                mainViewModel.reverseGeocode(
                    LatLng(
                        latLng.latitude,
                        latLng.longitude
                    ), apiKey
                )
            }
        }
    }
}
```

The above code shows how to create a `LocationTracker` object with `AuthHelper` and how to start and stop tracking with LocationTracker.

- `authenticateUser()`: This method creates AuthHelper and LocationTracker objects.
- on`StartStopTrackingClick`: This callback is triggered when the user clicks on the start/stop tracking button, which will start/stop tracking with Tracking SDK.

5. **Create a test route with Android Emulator.**

   a. **Open Emulator** by launching the AVD using Android Studio.

b. **Open Extended Controls** by clicking on the **More** (three dots) icon in the emulator toolbar.
c. **Open Location** by selecting **Location** from the sidebar.
d. **Create route** with GPX data or by clicking on the map and choosing source and destination data.
e. **Start Simulation** by clicking on **PLAY ROUTE** to begin simulating the GPS route.
f. **Test Application** by running your application and observing how it handles the simulated route.

This is the full demo of the Android Quick Start application.

What's next

The source code for this application is available on GitHub.

To get more out of Amazon Location, you can check out the following resources:

- Dive deeper into the concepts of Amazon Location Service
- Get more information about how to use Amazon Location features and functionality
- See how to expand on this sample and build more complex applications by looking at code examples using Amazon Location

Creating an iOS app

In this section, you will create an iOS application with the ability to search at a location and tracking in the foreground. First, you will create your Amazon Location resources, and an Amazon Cognito identity for your application.

Topics

- Creating Amazon Location resources for your app
- Setting up authentication for your application
- Creating the base iOS application
- Setting up the initial code
- Adding an interactive map to your application
- Adding search to your application

- Adding tracking to your application
- What's next

Creating Amazon Location resources for your app

If you do not already have them, you must create the Amazon Location resources that your application will use. You will create a map resource to display maps in your application, a place index to search for locations on the map, and a tracker to track an object across the map.

To add location resources to your application

1. Choose the map style that you want to use.

   a. In the Amazon Location console, on the Maps page, choose **Create map** to preview map styles.

   b. Add a **Name** and **Description** for the new map resource. Make a note of the name that you use for the map resource. You will need it when creating your script file later in the tutorial.

   c. Choose a map.

   > **Note**
   >
   > Choosing a map style also chooses which map data provider that you will use. If your application is tracking or routing assets that you use in your business, such as delivery vehicles or employees, you may only use HERE as your geolocation provider. For more information, see section 82 of the AWS service terms.

   d. Agree to the **Amazon Location Terms and Conditions**, then choose **Create map**. You can interact with the map that you've chosen: zoom in, zoom out, or pan in any direction.

   e. Make a note of the Amazon Resource Name (ARN) that is shown for your new map resource. You'll use it to create the correct authentication later in this tutorial.

2. Choose the place index that you want to use.

   a. In the Amazon Location console on the **Place indexes** page, choose **Create place index**.

   b. Add a **Name** and **Description** for the new place index resource. Make a note of the name that you use for the place index resource. You will need it when creating your script file later in the tutorial.

c. Choose a data provider.

> **Note**
>
> In most cases, choose the data provider that matches the map provider that you already chose. This helps to ensure that the searches will match the maps.
> If your application is tracking or routing assets that you use in your business, such as delivery vehicles or employees, you may only use HERE as your geolocation provider. For more information, see section 82 of the AWS service terms.

d. Choose the **Data storage option**. For this tutorial, the results are not stored, so you can choose **No, single use only**.

e. Agree to the **Amazon Location Terms and Conditions**, then choose **Create place index**.

f. Make a note of the ARN that is shown for your new place index resource. You'll use it to create the correct authentication in the next section of this tutorial.

3. To create a tracker using the Amazon Location console.

a. Open the Amazon Location Service console.

b. In the left navigation pane, choose **Trackers**.

c. Choose **Create tracker**.

d. Fill in the all the required fields.

e. Under **Position filtering**, we recommened you use the default setting: **TimeBased**.

f. Choose **Create tracker** to finish.

Setting up authentication for your application

The application that you create in this tutorial has anonymous usage, meaning that your users are not required to sign into AWS to use the application. However, the Amazon Location Service APIs require authentication to use. You will use Amazon Cognito to provide authentication and authorization for anonymous users. This tutorial will use Amazon Cognito to authenticate your application.

> **Note**
>
> For more information about using Amazon Cognito with Amazon Location Service, see Granting access to Amazon Location Service.

The following tutorials show you how to set up authentication for the map, the place index, and tracker you created in as well setting up permissions for Amazon Location.

Create an IAM policy for tracking

1. Sign in to the IAM console at https://console.aws.amazon.com/iam/ with your user that has administrator permissions.
2. In the navigation pane, choose Policies.
3. In the content pane, choose Create policy.
4. Choose the **JSON** option, then copy and paste this JSON policy into the JSON text box.

```
{
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "geo:GetMapTile",
                    "geo:GetMapStyleDescriptor",
                    "geo:GetMapSprites",
                    "geo:GetMapGlyphs",
                    "geo:SearchPlaceIndexForPosition",
                    "geo:GetDevicePositionHistory",
                    "geo:BatchUpdateDevicePosition"
                ],
                "Resource": [
                    "arn:aws:geo:{Region}:{Account}:map/{MapName}",
                    "arn:aws:geo:{Region}:{Account}:place-index/{IndexName}",
                    "arn:aws:geo:{Region}:{Account}:tracker/{TrackerName}"
                ]
            }
        ]
    }
```

This is a policy example for Tracking. To use the example for your own policy, replace the `Region`, `Account`, `IndexName`, `MapName` and **TrackerName** placeholders.

> **Note**
>
> While unauthenticated identity pools are intended for exposure on unsecured internet sites, note that they will be exchanged for standard, time-limited AWS credentials. It's important to scope the IAM roles associated with unauthenticated identity pools appropriately. For more information about using and appropriately scoping policies in Amazon Cognito with Amazon Location Service, see Granting access to Amazon Location Service.

5. On the Review and Create page, provide a name for the policy name field. Review the permissions granted by your policy, and then choose Create Policy to save your work.

The new policy appears in the list of managed policies and is ready to attach.

Set up authentication for your tracking

1. Set up authentication for your map application in the Amazon Cognito console.
2. Open the **Identity pools** page.

> **Note**
>
> The pool that you create must be in the same AWS account and AWS Region as the Amazon Location Service resources that you created in the previous section.

3. Choose **Create Identity pool**.
4. Starting with the **Configure identity pool trust** step. For user access authentication, select **Guest access**, and press next.
5. On the **Configure permissions** page select the **Use an existing IAM role** and enter the name of the IAM role you created in the previous step. When ready press next to move on to the next step.
6. On the **Configure properties** page, provide a name for your identity pool. Then press **Next**.
7. On the **Review and create** page, review all the information present then press **Create identity pool**.

8. Open the **Identity pools** page, and select the identity pool you just created. Then copy or write down the IdentityPoolId that you will use later in your browser script.

Creating the base iOS application

In this tutorial, you will create an iOS application that embeds a map, and allows the user to find what's at a location on the map.

First, let's create a Swift application using Xcode's project wizard.

To create an empty application (Xcode)

1. Open Xcode, and from the menu, choose **File**, **New**, **New Project**.
2. From the **iOS** tab, select **App**, and then choose **Next**.
3. Provide a **Product Name**, an **Organization Identifier**, and in the **Interface** field input `SwiftUI`. Choose **Next** to finalize the selection.
4. Select a location where you will save your project and press **create** button to create the empty application.

Once you have creating the base application, you will need to install the required packages for the sample app.

Installing required dependencies

1. In Xcode, right-click on the **project** and choose **Add Packages...**. This will open the Packages window, where you can add packages to your project.
2. In the Packages window, add the following packages:

   - For the Maplibre native package, use this URL: https://github.com/maplibre/maplibre-gl-native-distribution. From the URL, add these packages: `maplibre-gl-native-distribution`, and `Mapbox`.
   - For the Amazon Location authentication iOS SDK, use this URL: https://github.com/aws-geospatial/amazon-location-mobile-auth-sdk-ios. From the URL, add these packages: `amazon-location-mobile-auth-sdk-ios`, and `AmazonLocationiOSAuthSDK`.
   - For the Amazon Location tracking iOS SDK, use this URL: https://github.com/aws-geospatial/amazon-location-mobile-tracking-sdk-ios. From the URL,

add these packages: `amazon-location-mobile-tracking-sdk-ios`, and `AmazonLocationiOSTrackingSDK`.

Setting up the initial code

Enable Location permissions in your app

1. Open your Xcode project.
2. Locate the project's `Info.plist` file.
3. Add the necessary keys for location permissions based on your app's requirements. Here are the keys:
   - `NSLocationWhenInUseUsageDescription`: Description of why your app needs location access when it's in use.
   - `NSLocationAlwaysAndWhenInUseUsageDescription`: Description of why your app needs continuous location access.

Now you will need to configure resource values in your app. Add a new file named `Config.xcconfig` and fill out the values that you had created previously in the Amazon console.

```
REGION =
INDEX_NAME =
MAP_NAME =
IDENTITY_POOL_ID =
TRACKER_NAME =
```

1. From the left side navigator section, select the project.
2. Under the targets section, select your app and click on the info tab.
3. Add info properties with values like the below:
4. Add the `Config.swift` file with the contents below, which will read config values from the Bundle info file.

```
import Foundation

enum Config {
```

```
    static let region = Bundle.main.object(forInfoDictionaryKey: "Region") as!
 String
    static let mapName = Bundle.main.object(forInfoDictionaryKey: "MapName") as!
 String
    static let indexName = Bundle.main.object(forInfoDictionaryKey: "IndexName")
 as! String
    static let identityPoolId = Bundle.main.object(forInfoDictionaryKey:
 "IdentityPoolId") as! String
    static let trackerName = Bundle.main.object(forInfoDictionaryKey:
 "TrackerName") as! String
}
```

5. Create a new folder with the name `ViewModel` and add a `TrackingViewModel.swift` file inside it.

```
import SwiftUI
import AmazonLocationiOSAuthSDK
import MapLibre

final class TrackingViewModel : ObservableObject {
    @Published var trackingButtonText = NSLocalizedString("StartTrackingLabel",
 comment: "")
    @Published var trackingButtonColor = Color.blue
    @Published var trackingButtonIcon = "play.circle"
    @Published var region : String
    @Published var mapName : String
    @Published var indexName : String
    @Published var identityPoolId : String
    @Published var trackerName : String
    @Published var showAlert = false
    @Published var alertTitle = ""
    @Published var alertMessage = ""
    @Published var centerLabel = ""

    var clientIntialised: Bool
    var client: LocationTracker!
    var authHelper: AuthHelper
    var credentialsProvider: LocationCredentialsProvider?
    var mlnMapView: MLNMapView?
    var mapViewDelegate: MapViewDelegate?
    var lastGetTrackingTime: Date?
    var trackingActive: Bool
```

```
    init(region: String, mapName: String, indexName: String, identityPoolId:
 String, trackerName: String) {
        self.region = region
        self.mapName = mapName
        self.indexName = indexName
        self.identityPoolId = identityPoolId
        self.trackerName = trackerName
        self.authHelper = AuthHelper()
        self.trackingActive = false
        self.clientIntialised = false
    }

    func authWithCognito(identityPoolId: String?) {
        guard let identityPoolId =
 identityPoolId?.trimmingCharacters(in: .whitespacesAndNewlines)
        else {
            alertTitle = NSLocalizedString("Error", comment: "")
            alertMessage = NSLocalizedString("NotAllFieldsAreConfigured", comment:
 "")
            showAlert = true
            return
        }
        credentialsProvider =
 authHelper.authenticateWithCognitoUserPool(identityPoolId: identityPoolId)
        initializeClient()
    }

    func initializeClient() {
        client = LocationTracker(provider: credentialsProvider!, trackerName:
 trackerName)
        clientIntialised = true
    }
}
```

Adding an interactive map to your application

You will now add the map control to your application. This tutorial uses MapLibre and the AWS API for managing the map view in the application. The map control itself is part of the MapLibre GL Native iOS library.

1. Add `MapView.swift` file under the **Views** folder with the following code:

```
import SwiftUI
import MapLibre

struct MapView: UIViewRepresentable {
    var onMapViewAvailable: ((MLNMapView) -> Void)?
    var mlnMapView: MLNMapView?
    var trackingViewModel: TrackingViewModel

    func makeCoordinator() -> MapView.Coordinator {
        return Coordinator(self, trackingViewModel: trackingViewModel)
    }

    func makeUIView(context: Context) -> MLNMapView {
        let styleURL = URL(string: "https://maps.geo.
\(trackingViewModel.region).amazonaws.com/maps/v0/maps/
\(trackingViewModel.mapName)/style-descriptor")
        let mapView = MLNMapView(frame: .zero, styleURL: styleURL)
        mapView.autoresizingMask = [.flexibleWidth, .flexibleHeight]
        mapView.setZoomLevel(15, animated: true)
        mapView.showsUserLocation = true
        mapView.userTrackingMode = .follow
        context.coordinator.mlnMapView = mapView
        mapView.delegate = context.coordinator

        mapView.logoView.isHidden = true
        context.coordinator.addCenterMarker()

        onMapViewAvailable?(mapView)
        trackingViewModel.mlnMapView = mapView
        return mapView
    }

    func updateUIView(_ uiView: MLNMapView, context: Context) {
    }

    class Coordinator: NSObject, MLNMapViewDelegate, MapViewDelegate {
        var control: MapView
        var mlnMapView: MLNMapView?
        var trackingViewModel: TrackingViewModel
        var centerMarker: MLNPointAnnotation?

        public init(_ control: MapView, trackingViewModel: TrackingViewModel) {
            self.control = control
```

```
            self.trackingViewModel = trackingViewModel
            super.init()
            self.trackingViewModel.mapViewDelegate = self
        }

        func mapViewDidFinishRenderingMap(_ mapView: MLNMapView, fullyRendered:
Bool) {
            if(fullyRendered) {
                mapView.accessibilityIdentifier = "MapView"
                mapView.isAccessibilityElement = false
            }
        }

        func addCenterMarker() {
            guard let mlnMapView = mlnMapView else {
                return
            }

            let centerCoordinate = mlnMapView.centerCoordinate
            let marker = MLNPointAnnotation()
            marker.coordinate = centerCoordinate
            marker.accessibilityLabel = "CenterMarker"
            mlnMapView.addAnnotation(marker)
            centerMarker = marker

            trackingViewModel.reverseGeocodeCenter(centerCoordinate:
centerCoordinate, marker: marker)
        }

        func mapView(_ mapView: MLNMapView, regionDidChangeAnimated animated: Bool)
{
            if let marker = centerMarker {
                                DispatchQueue.main.asyncAfter(deadline: .now() + 1.0)
{
                    mapView.deselectAnnotation(marker, animated: false)
                    marker.coordinate = mapView.centerCoordinate
                    let centerCoordinate = mapView.centerCoordinate
                    self.trackingViewModel.reverseGeocodeCenter(centerCoordinate:
centerCoordinate, marker: marker)
                }
            }
        }
    }
```

```
}
```

2. Add `AWSSignatureV4Delegate` file under the **ViewModel** folder. This file is used to sign with all the MapView http requests to render the map:

```
import MapLibre
import AmazonLocationiOSAuthSDK

class AWSSignatureV4Delegate : NSObject, MLNOfflineStorageDelegate {
    private let awsSigner: AWSSigner

    init(credentialsProvider: LocationCredentialsProvider) {
        self.awsSigner = DENY LIST ERROR , serviceName: "geo")
        super.init()
    }

    func offlineStorage(_ storage: MLNOfflineStorage, urlForResourceOf kind:
 MLNResourceKind, with url: URL) -> URL {
        if url.host?.contains("amazonaws.com") != true {
            return url
        }
        let signedURL = awsSigner.signURL(url: url, expires: .hours(1))

        return signedURL
    }
}
```

3. Add `UserLocationView.swift` file under **Views** folder. This adds a button which centers the map to the user's location

```
import SwiftUI

struct UserLocationView: View {
    @ObservedObject var trackingViewModel: TrackingViewModel
    var body: some View {
            Button(action: {
                trackingViewModel.locateMe()
            }) {
                Image(systemName: "scope")
                    .resizable()
                    .frame(width: 24, height: 24)
                    .padding(5)
                    .background(Color.white)
```

```
                    .foregroundColor(.blue)
                    .clipShape(RoundedRectangle(cornerRadius: 8))
                    .shadow(color: Color.black.opacity(0.3), radius: 3, x: 0, y: 2)
            }
            .accessibility(identifier: "LocateMeButton")
            .padding(.trailing, 10)
            .padding(.bottom, 10)
            .frame(maxWidth: .infinity, alignment: .trailing)
    }
}
```

4. Add the `TrackingView.swift` file with the following code:

```
import SwiftUI

struct TrackingView: View {
    @ObservedObject var trackingViewModel: TrackingViewModel
    var body: some View {
        ZStack(alignment: .bottom) {
            MapView(trackingViewModel: trackingViewModel)
            VStack {
                UserLocationView(trackingViewModel: trackingViewModel)
            }
        }
        .onAppear() {
            if !trackingViewModel.identityPoolId.isEmpty {
                trackingViewModel.authWithCognito(identityPoolId:
 trackingViewModel.identityPoolId)
            }
        }
    }
}
```

You can now build the application. To run it, you may have to set up a device to emulate it in Xcode or use the app on your device. Use this app to see how the map control behaves. You can pan by dragging on the map and pinch to zoom. On your own, you can change how the map control works to customize it to the needs of your application.

Adding search to your application

You now will add reverse geocoding search to the application, where you find the items at a location. To simplify the use of an iOS app, we will search the center of the screen. To find a new location, move the map to where you want to search. We will place a marker at the center of the map to show where we are searching.

1. Add the following code in `TrackingViewModel.swift` file which is related to the reverse geocoding search

```
func reverseGeocodeCenter(centerCoordinate: CLLocationCoordinate2D, marker:
 MLNPointAnnotation) {
    let position = [NSNumber(value: centerCoordinate.longitude), NSNumber(value:
 centerCoordinate.latitude)]
    searchPositionAPI(position: position, marker: marker)
}

func searchPositionAPI(position: [Double], marker: MLNPointAnnotation) {
    if let amazonClient = authHelper.getLocationClient() {
        Task {
            let searchRequest = SearchPlaceIndexForPositionInput(indexName:
 indexName, language: "en" , maxResults: 10, position: position)
            let searchResponse = try? await amazonClient.searchPosition(indexName:
 indexName, input: searchRequest)
            DispatchQueue.main.async {
                self.centerLabel = searchResponse?.results?.first?.place?.label ??
 ""
                self.mlnMapView?.selectAnnotation(marker, animated: true,
 completionHandler: {})
            }
        }
    }
}
```

2. Update `TrackingView.swift` file with the following code which will show the mapview's centered location's address

```
import SwiftUI

struct TrackingView: View {
    @ObservedObject var trackingViewModel: TrackingViewModel
    var body: some View {
```

```
        ZStack(alignment: .bottom) {
            if trackingViewModel.mapSigningIntialised {
                MapView(trackingViewModel: trackingViewModel)
                VStack {
                    UserLocationView(trackingViewModel: trackingViewModel)
                    CenterAddressView(trackingViewModel: trackingViewModel)
                }
            }
            else {
                Text("Loading...")
            }
        }
        .onAppear() {
            if !trackingViewModel.identityPoolId.isEmpty {
                Task {
                    do {
                        try await trackingViewModel.authWithCognito(identityPoolId:
 trackingViewModel.identityPoolId)
                    }
                    catch {
                        print(error)
                    }
                }
            }
        }
    }
}
```

Adding tracking to your application

The last step for your application is to add tracking functionality to your app. In this case, you will add start tracking, stop tracking, fetch and display tracker points on your app.

1. Add the `TrackingBottomView.swift` file in your project. Which has a button that starts and stops tracking user locations and shows tracking points on the map.

```
import SwiftUI

struct TrackingBottomView: View {
    @ObservedObject var trackingViewModel: TrackingViewModel
    var body: some View {
           Button(action: {
```

```
                Task {
                    if(trackingViewModel.trackingButtonText ==
 NSLocalizedString("StartTrackingLabel", comment: "")) {
                        trackingViewModel.startTracking()
                    } else {
                        trackingViewModel.stopTracking()
                    }
                }
            }) {
                HStack {
                    Spacer()
                    Text("Tracking")
                        .foregroundColor(trackingViewModel.trackingButtonColor)
                        .background(.white)
                        .cornerRadius(15.0)

                    Image(systemName: trackingViewModel.trackingButtonIcon)
                        .resizable()
                        .frame(width: 24, height: 24)
                        .padding(5)
                        .background(.white)
                        .foregroundColor(trackingViewModel.trackingButtonColor)

                }
            }
            .accessibility(identifier: "TrackingButton")
            .background(.white)
            .clipShape(RoundedRectangle(cornerRadius: 8))
            .padding(.trailing, 10)
            .padding(.bottom, 40)
            .frame(width: 130, alignment: .trailing)
            .shadow(color: Color.black.opacity(0.3), radius: 3, x: 0, y: 2)
        }
}
```

2. Update `TrackingView.swift` file with the following code

```
import SwiftUI

struct TrackingView: View {
    @ObservedObject var trackingViewModel: TrackingViewModel
    var body: some View {
        ZStack(alignment: .bottom) {
            if trackingViewModel.mapSigningIntialised {
```

```
                MapView(trackingViewModel: trackingViewModel)
                VStack {
                    UserLocationView(trackingViewModel: trackingViewModel)
                    CenterAddressView(trackingViewModel: trackingViewModel)
                    TrackingBottomView(trackingViewModel: trackingViewModel)
                }
            }
            else {
                Text("Loading...")
            }
        }
        .onAppear() {
            if !trackingViewModel.identityPoolId.isEmpty {
                Task {
                    do {
                        try await trackingViewModel.authWithCognito(identityPoolId:
 trackingViewModel.identityPoolId)
                    }
                    catch {
                        print(error)
                    }
                }
            }
        }
    }
}
```

3. Add the following code in `TrackingViewModel.swift` file. These functions are responsible for start and stop tracking. It will also show an error alert if user location permission is denied.
4. To implement foreground tracking copy paste the following code example:

```
func showLocationDeniedRationale() {
        alertTitle = NSLocalizedString("locationManagerAlertTitle", comment: "")
        alertMessage = NSLocalizedString("locationManagerAlertText", comment: "")
        showAlert = true
    }

    // Required in info.plist: Privacy - Location When In Use Usage Description
    func startTracking() {
        do {
            print("Tracking Started...")
            if(client == nil) {
                initializeClient()
```

```
            }
            try client.startTracking()
            DispatchQueue.main.async { [self] in
                self.trackingButtonText = NSLocalizedString("StopTrackingLabel",
comment: "")
                self.trackingButtonColor = .red
                self.trackingButtonIcon = "pause.circle"
                trackingActive = true
            }
        } catch TrackingLocationError.permissionDenied {
            showLocationDeniedRationale()
        } catch {
            print("error in tracking")
        }
    }

    func stopTracking() {
        print("Tracking Stopped...")
        client.stopTracking()
        trackingButtonText = NSLocalizedString("StartTrackingLabel", comment: "")
        trackingButtonColor = .blue
        trackingButtonIcon = "play.circle"
        trackingActive = false
    }
```

Note

The `startTracking` will ask for the user's location permission. The application must use **When In Use** or **Only Once** permissions. Otherwise, the application will throw a permission denied error.

To get and display tracking locations, follow this procedure:

1. To get the locations from the user's device, you need to provide the start and end date and time. A single call returns a maximum of 100 tracking locations, but if there are more than 100 tracking locations, it will return a `nextToken` value. You will need to call subsequent `getTrackerDeviceLocation` calls with `nextToken` to load more tracking points for the given start and end time.

```
func getTrackingPoints(nextToken: String? = nil) async throws {
```

```
        guard trackingActive else {
            return
        }
        // Initialize startTime to 24 hours ago from the current date and time.
        let startTime: Date = Date().addingTimeInterval(-86400)
        var endTime: Date = Date()
        if lastGetTrackingTime != nil {
            endTime = lastGetTrackingTime!
        }
        let result = try await client?.getTrackerDeviceLocation(nextToken:
nextToken, startTime: startTime, endTime: endTime)
        if let trackingData = result {

            lastGetTrackingTime = Date()
            let devicePositions = trackingData.devicePositions

            let positions = devicePositions!.sorted { (pos1:
LocationClientTypes.DevicePosition, pos2: LocationClientTypes.DevicePosition) ->
Bool in
                guard let date1 = pos1.sampleTime,
                      let date2 = pos2.sampleTime else {
                    return false
                }
                return date1 < date2
            }

            let trackingPoints = positions.compactMap { position ->
CLLocationCoordinate2D? in
                guard let latitude = position.position!.last, let longitude =
position.position!.first else {
                    return nil
                }
                return CLLocationCoordinate2D(latitude: latitude, longitude:
longitude)
            }
            DispatchQueue.main.async {
                self.mapViewDelegate!.drawTrackingPoints( trackingPoints:
trackingPoints)
            }
            if let nextToken = trackingData.nextToken {
                try await getTrackingPoints(nextToken: nextToken)
            }
        }
```

```
    }
```

2. Now replace the code in the `MapView.swift` file with the following code:

```
import SwiftUI
import MapLibre

struct MapView: UIViewRepresentable {
    var onMapViewAvailable: ((MLNMapView) -> Void)?
    var mlnMapView: MLNMapView?
    var trackingViewModel: TrackingViewModel

    func makeCoordinator() -> MapView.Coordinator {
        return Coordinator(self, trackingViewModel: trackingViewModel)
    }

    func makeUIView(context: Context) -> MLNMapView {
        let styleURL = URL(string: "https://maps.geo.
\(trackingViewModel.region).amazonaws.com/maps/v0/maps/
\(trackingViewModel.mapName)/style-descriptor")
        let mapView = MLNMapView(frame: .zero, styleURL: styleURL)
        mapView.autoresizingMask = [.flexibleWidth, .flexibleHeight]
        mapView.setZoomLevel(15, animated: true)
        mapView.showsUserLocation = true
        mapView.userTrackingMode = .follow
        context.coordinator.mlnMapView = mapView
        mapView.delegate = context.coordinator

        mapView.logoView.isHidden = true
        context.coordinator.addCenterMarker()

        onMapViewAvailable?(mapView)
        trackingViewModel.mlnMapView = mapView
        return mapView
    }

    func updateUIView(_ uiView: MLNMapView, context: Context) {
    }

    class Coordinator: NSObject, MLNMapViewDelegate, MapViewDelegate {
        var control: MapView
        var mlnMapView: MLNMapView?
        var trackingViewModel: TrackingViewModel
        var centerMarker: MLNPointAnnotation?
```

```
        public init(_ control: MapView, trackingViewModel: TrackingViewModel) {
            self.control = control
            self.trackingViewModel = trackingViewModel
            super.init()
            self.trackingViewModel.mapViewDelegate = self
        }

        func mapViewDidFinishRenderingMap(_ mapView: MLNMapView, fullyRendered:
Bool) {
            if(fullyRendered) {
                mapView.accessibilityIdentifier = "MapView"
                mapView.isAccessibilityElement = false
            }
        }

        func addCenterMarker() {
            guard let mlnMapView = mlnMapView else {
                return
            }

            let centerCoordinate = mlnMapView.centerCoordinate
            let marker = MLNPointAnnotation()
            marker.coordinate = centerCoordinate
            marker.accessibilityLabel = "CenterMarker"
            mlnMapView.addAnnotation(marker)
            centerMarker = marker

            trackingViewModel.reverseGeocodeCenter(centerCoordinate:
centerCoordinate, marker: marker)
        }

        func mapView(_ mapView: MLNMapView, regionDidChangeAnimated animated: Bool)
{
            if let marker = centerMarker {
                DispatchQueue.main.asyncAfter(deadline: .now() + 1.0) {
                    mapView.deselectAnnotation(marker, animated: false)
                    marker.coordinate = mapView.centerCoordinate
                    let centerCoordinate = mapView.centerCoordinate
                    self.trackingViewModel.reverseGeocodeCenter(centerCoordinate:
centerCoordinate, marker: marker)
                }
            }
        }
```

```
       func mapView(_ mapView: MLNMapView, viewFor annotation: MLNAnnotation) ->
MLNAnnotationView? {
           guard let pointAnnotation = annotation as? MLNPointAnnotation else {
               return nil
           }

           let reuseIdentifier: String
           var color: UIColor = .black
           if pointAnnotation.accessibilityLabel == "Tracking" {
               reuseIdentifier = "TrackingAnnotation"
               color = UIColor(red: 0.00784313725, green: 0.50588235294, blue:
0.58039215686, alpha: 1)
           } else if pointAnnotation.accessibilityLabel == "LocationChange" {
               reuseIdentifier = "LocationChange"
               color = .gray
           } else {
               reuseIdentifier = "DefaultAnnotationView"
           }

           var annotationView =
mapView.dequeueReusableAnnotationView(withIdentifier: reuseIdentifier)

           if annotationView == nil {
               if reuseIdentifier != "DefaultAnnotationView" {
                   annotationView = MLNAnnotationView(annotation: annotation,
reuseIdentifier: reuseIdentifier)
                   //If point annotation is an uploaded Tracking point the radius
is 20 and color is blue, otherwise radius is 10 and color is gray
                   let radius = pointAnnotation.accessibilityLabel == "Tracking" ?
20:10
                   annotationView?.frame = CGRect(x: 0, y: 0, width: radius,
height: radius)
                   annotationView?.backgroundColor = color
                   annotationView?.layer.cornerRadius = 10

                   if pointAnnotation.accessibilityLabel == "Tracking" {
                       annotationView?.layer.borderColor = UIColor.white.cgColor
                       annotationView?.layer.borderWidth = 2.0
                       annotationView?.layer.shadowColor = UIColor.black.cgColor
                       annotationView?.layer.shadowOffset = CGSize(width: 0,
height: 2)
                       annotationView?.layer.shadowRadius = 3
                       annotationView?.layer.shadowOpacity = 0.2
```

```
                        annotationView?.clipsToBounds = false
                    }
                }
                else {
                    return nil
                }
            }

            return annotationView
        }

        func mapView(_ mapView: MLNMapView, didUpdate userLocation:
MLNUserLocation?) {
            if (userLocation?.location) != nil {
                if trackingViewModel.trackingActive {
                    let point = MLNPointAnnotation()
                    point.coordinate = (userLocation?.location!.coordinate)!
                    point.accessibilityLabel = "LocationChange"
                    mapView.addAnnotation(point)
                    Task {
                        do {
                            try await trackingViewModel.getTrackingPoints()
                        }
                        catch {
                            print(error)
                        }
                    }
                }
            }
        }

        func checkIfTrackingAnnotationExists(on mapView: MLNMapView, at
coordinates: CLLocationCoordinate2D) -> Bool {
            let existingAnnotation = mapView.annotations?.first(where: { annotation
in
                guard let annotation = annotation as? MLNPointAnnotation else
{ return false }
                return annotation.coordinate.latitude == coordinates.latitude &&
                annotation.coordinate.longitude == coordinates.longitude &&
annotation.accessibilityLabel == "Tracking" })
            return existingAnnotation != nil
        }

        public func drawTrackingPoints(trackingPoints: [CLLocationCoordinate2D]?) {
```

```
            guard let mapView = mlnMapView, let newTrackingPoints =
 trackingPoints, !newTrackingPoints.isEmpty else {
                return
            }

            let uniqueCoordinates = newTrackingPoints.filter { coordinate in
                !checkIfTrackingAnnotationExists(on: mapView, at: coordinate)
            }

            let points = uniqueCoordinates.map { coordinate -> MLNPointAnnotation
 in
                let point = MLNPointAnnotation()
                point.coordinate = coordinate
                point.accessibilityLabel = "Tracking"
                return point
            }
            mapView.addAnnotations(points)
        }
    }
}

protocol MapViewDelegate: AnyObject {
    func drawTrackingPoints(trackingPoints: [CLLocationCoordinate2D]?)
}
```

To localize string values , use the following procedure.

1. Create and add a new file called `Localizable.xcstrings`.
2. Right-click on the `Localizable.xcstrings` file and open it as **Source Code**.
3. Replace its content with the following:

```
{
  "sourceLanguage" : "en",
  "strings" : {
    "Cancel" : {
      "extractionState" : "manual",
      "localizations" : {
        "en" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "Cancel"
```

```
        }
      }
    }
  },
  "Error" : {
    "extractionState" : "manual",
    "localizations" : {
      "en" : {
        "stringUnit" : {
          "state" : "translated",
          "value" : "Error"
        }
      }
    }
  },
  "Loading..." : {

  },
  "locationManagerAlertText" : {
    "extractionState" : "manual",
    "localizations" : {
      "en" : {
        "stringUnit" : {
          "state" : "translated",
          "value" : "Allow \\\"Quick Start App\\\" to use your location"
        }
      }
    }
  },
  "locationManagerAlertTitle" : {
    "extractionState" : "manual",
    "localizations" : {
      "en" : {
        "stringUnit" : {
          "state" : "translated",
          "value" : "We need your location to detect your location in map"
        }
      }
    }
  },
  "NotAllFieldsAreConfigured" : {
    "extractionState" : "manual",
    "localizations" : {
      "en" : {
```

```
          "stringUnit" : {
            "state" : "translated",
            "value" : "Not all the fields are configured"
          }
        }
      }
    },
    "OK" : {
      "extractionState" : "manual",
      "localizations" : {
        "en" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "OK"
          }
        }
      }
    },
    "StartTrackingLabel" : {
      "localizations" : {
        "en" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "Start Tracking"
          }
        }
      }
    },
    "StopTrackingLabel" : {
      "localizations" : {
        "en" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "Stop Tracking"
          }
        }
      }
    },
    "Tracking" : {

    }
  },
  "version" : "1.0"
```

```
}
```

4. Save your files, and build and run your app to preview the functionality.
5. Allow the location permission and tap on the tracking button. The app will start uploading user locations and upload them to the Amazon Location tracker. It will also show user location changes, tracking points, and current address on the map.

Your quick-start application is complete. This tutorial has shown you how to create an iOS application that:

- Creates a map that users can interact with.
- Handles several map events associated with the user changing the map view.
- Calls an Amazon Location Service API, specifically to search the map at a location, using Amazon Location's searchByPosition API.

What's next

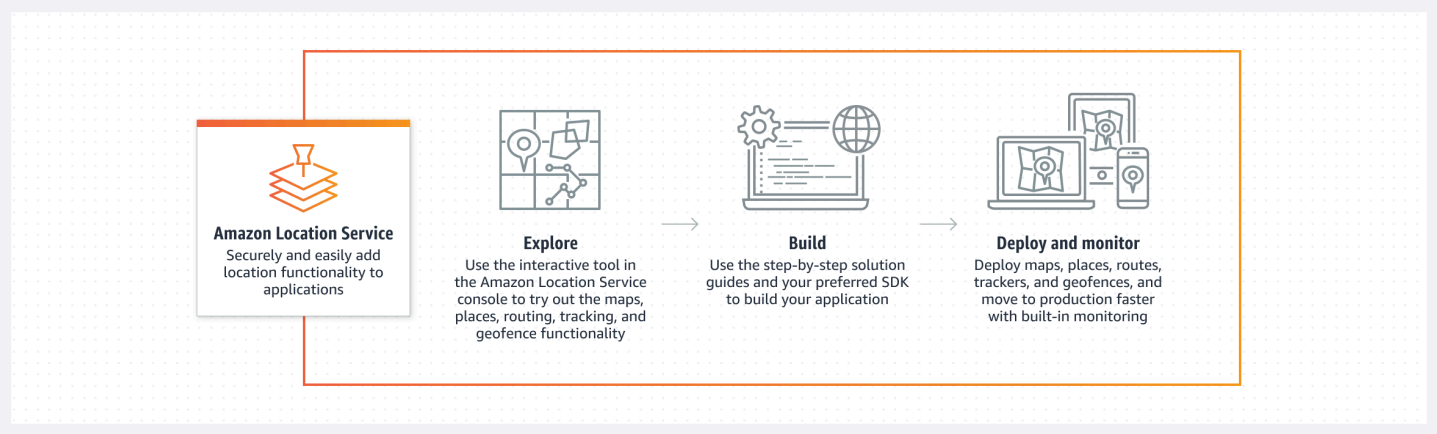
The source code for this application is available on GitHub.

To get more out of Amazon Location, you can check out the following resources:

- Dive deeper into the concepts of Amazon Location Service
- Get more information about how to use Amazon Location features and functionality
- See how to expand on this sample and build more complex applications by looking at code examples using Amazon Location

Amazon Location Service concepts

With Amazon Location Service, you can securely add location data to your application. Explore some of the capabilities by using the visual and interactive tool, available on the Amazon Location console. Using the explore tool, you can manipulate a default map, search for points of interest, draw geofences around areas of interest, and simulate sending device locations to a tracker.



When you are ready to build, create your resources and choose from a variety of map styles and data providers. Then you can install the SDK that matches your development environment, and use the Amazon Location APIs using the instructions in this guide. Additionally, you can integrate monitoring by using Amazon CloudWatch and AWS CloudTrail.

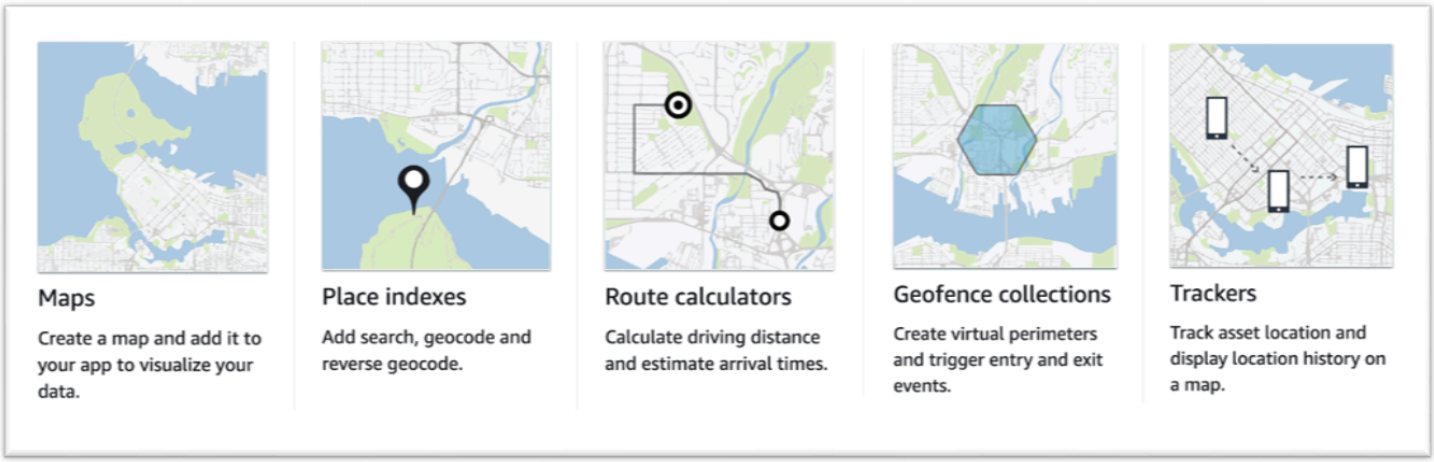
The topics in this section provide you an overview of the Amazon Location core concepts and prepare you to start working with location in your own applications.

Topics

- Amazon Location overview
- Maps
- Places search
- Routes
- Geofences and Trackers
- Common use cases for using Amazon Location Service
- What is a data provider?
- Amazon Location Regions and endpoints
- Amazon Location Service quotas

Amazon Location overview

Amazon Location Service provides access to location-based functionality and data providers through AWS resources. Amazon Location offers five types of AWS resources, depending on the type of functionality you need. Use the different resources together to create a full location-based application. You can create one or more of these resources by using the Amazon Location console, the Amazon Location APIs, or the SDKs.



Each resource defines the underlying data provider to be used (where applicable), and gives access to functionality related to its type.

For example:

- Amazon Location Service Maps lets you choose a map from a map provider to use on your mobile or web application.
- Amazon Location Service Places lets you choose a data provider for searching for points of interest, completing partial text, geocoding, and reverse geocoding.
- Amazon Location Service Routes lets you choose a data provider and find routes and estimate travel time based on up-to-date roadway and live traffic information.
- Amazon Location Service Geofences let you define areas of interest as a virtual boundary. You can then evaluate locations against them and get notifications of entry and exit events.
- Amazon Location Service Trackers receive location updates from your devices. You can link trackers to geofence collections so that all position updates are automatically evaluated against your geofences.

You can use IAM policies to manage and authorize access to your Amazon Location resources. You can also organize your resources into resource groups to manage and automate tasks as your resource numbers grow. For more information about managing AWS resources, see What are AWS Resource Groups? In the *AWS Resource Groups User Guide*.

Location is defined by using latitude and longitude coordinates that follow the World Geodetic System (WGS 84), commonly used as the standard coordinate reference system for Global Positioning System (GPS) services.

The following sections describe how the components of Amazon Location work.

Maps

The Amazon Location Service *Map* resource gives you access to the underlying basemap data for a map. You use the Map resource with a map rendering library to add an interactive map to your application. You can add other functionality to your map, such as markers (or pins), routes, and polygon areas, as needed for your application.

> **Note**
>
> For information about how to use map resources in practice, see Using Amazon Location Maps in your application.

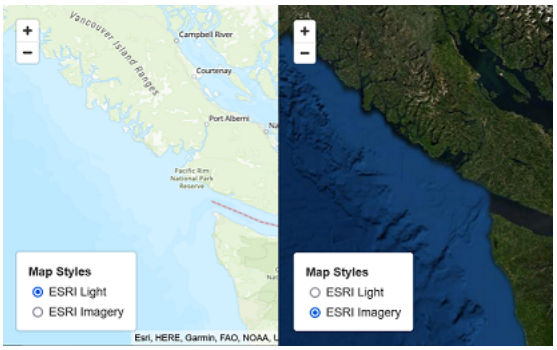
The following is an overview of how to create and use map resources:

1. You create a map resource in your AWS account by selecting a map style from a data provider.
2. You can then select and install the SDK that matches your development environment and applications. For more information about available options, see the topic about Accessing Amazon Location.
3. To display a map in your application, combine a map resource with a rendering library, such as Amplify, MapLibre, or Tangram. For more information. see Using maps in this guide.

4. You can then integrate monitoring by using services, such as Amazon CloudWatch and AWS CloudTrail with Amazon Location. For more information see, Monitoring Amazon Location Service with Amazon CloudWatch and Logging and monitoring with AWS CloudTrail.

Map styles

When you create a map resource, you must choose a map style for that resource. Map styles define the look of the rendered map. For example, the following image shows the same data provider with two different styles from different map resources in Amazon Location. One style is a typical road style, based on the vector data in the map. The other includes raster data showing satellite imagery. The style may change as you zoom in or out on the map, but typically styles have a consistent theme. It's possible to override parts or all of the style information before passing it to the map rendering library.



Political views

Certain maps styles in Amazon Location Service support additional political views.

> **Note**
>
> The political view must be used in compliance with applicable laws, including those laws about mapping of the country or region where the maps, images, and other data and third-party content which you access through Amazon Location Service is made available.

The following map styles support an India (IND) political view.

- Esri map styles:
  - Esri Navigation
  - Esri Light

  - Esri Street Map
  - Esri Dark Gray Canvas
  - Esri Light Gray Canvas
- Open Data map styles:
  - Open Data Standard Light
  - Open Data Standard Dark
  - Open Data Visualization Light
  - Open Data Visualization Dark

In the Amazon Location Service console, you can filter the styles shown to just show the styles that support the India political view.

Custom Layers

A custom layer is an additional layer you can enable for a map style. Currently only the VectorEsriNavigation map style supports the `POI` custom layer.

When you enable the `POI` custom layer it adds a richer set of places, such as shops, services, restaurants, attractions, and other points of interest to your map. By default, the custom layer is `unset`. For more information see, MapConfiguration in the Location API reference.

Map rendering

To render a map in your application, you will typically use a map rendering library. There are several common options for libraries to use:

- **MapLibre** – MapLibre is an open source library specifically for rendering interactive maps, and is the preferred method of rendering maps from Amazon Location Service. MapLibre includes the ability to render raster and vector data from a data source (such as an Amazon Location map resource). You can extend MapLibre to draw your own data on the map.
- **Amplify** – Amplify is an open source framework for building applications for the web, iOS, Android, and more. If your application uses Amplify, then you can extend it to include Amazon Location functionality. Amplify includes libraries specifically for creating Amazon Location based applications, including rendering maps. Amplify uses MapLibre to render the map, but provides additional functionality that is specific to Amazon Location Service to make it more efficient to use, and also adding search and other functionality.

- **Tangram** – Tangram is an alternative open source library that renders interactive maps, similar to MapLibre.

The map rendering library pulls data from Amazon Location Service at runtime, rendering the map data based on the map resource you select. The map resource defines the data provider and map style that will be used.

The following image shows how the map resource is used in Amazon Location Service along with a map rendering library to create the final map.

1. You create a map resource in Amazon Location Service, using the AWS Management Console or AWS CLI. This defines the data provider and the map style that you want to use.
2. Your application includes a map rendering library. You give the map rendering library the name of the map resource to use. The map rendering library pulls data and style information for that map resource from Amazon Location and renders the map on screen.

Maps terminology

Map resource

Allows you to access map data from a selected provider. Use the map resource to fetch map tiles that contain map data and a style descriptor to specify how features render on a map.

Basemap

Provides geographic context to your map, which is stored as vector tile layers. Tile layers include geographical context such as street names, buildings, and land use for visual reference.

Vector

Vector data is shape data made up of points, lines, and polygons. It is often used to store and display roads, locations, and areas on a map. A vector shape can also be used as icons for markers on a map.

Raster

Raster data is image data, made up of a grid, usually of colors. It is often used to store and display a representation of continuous data on maps, such as terrain, satellite imagery, or heat maps. Raster images can also be used as images or icons.

Map Style

Vector data does not inherently include information about how to draw the layers of data to create the final map. A map style defines color and other style information for the data to define how it will look when rendered. Map resources include style information for the map.

Amazon Location Service provides styles following the Mapbox GL style specification.

Vector tile

A tile format that stores map data using vector shapes. This data results in a map that can adjust to the display resolution, and selectively render features in a number of ways, while maintaining a small file size for optimal performance.

Supported vector file format: Mapbox Vector Tiles (MVT).

Glyph file

A binary file containing encoded Unicode characters. Used by a map renderer to display labels.

Sprite file

A Portable Network Graphic (PNG) image file that contains small raster images, with location descriptions in a JSON file. Used by a map renderer to render icons or textures on a map.

Places search

A key function of Amazon Location Service is the ability to search the geolocation information. Amazon Location provides this functionality via the *Place index* resource.

> **Note**
>
> For information about how to use place index resources to search in practice, see Searching place and geolocation data using Amazon Location.

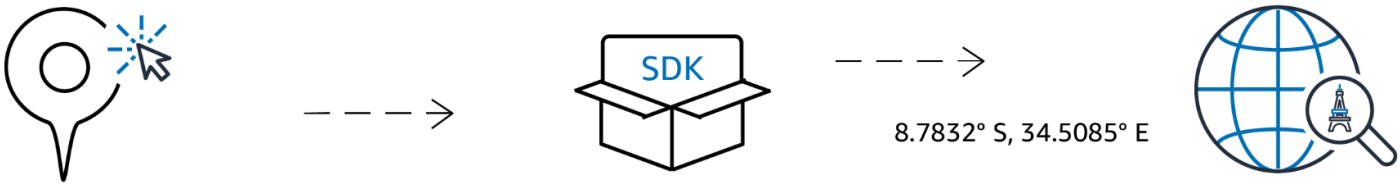
You can use the place index APIs to search for:

- Points of interest, such as restaurants and landmarks. Search by name, and optional location to search around, and receive a list of options ordered by relevance.
- A street address, receiving a latitude and longitude for that address. This is known as *geocoding*.
- A latitude and longitude position, receiving the associated street address or other information about the location. This is known as *reverse geocoding*.
- A partial or misspelled free-form text query, typically as a user types. This is known as *autocomplete*, autosuggest, or fuzzy matching.

The place index includes which data provider to use for the search.

> **Note**
>
> Map data and other geolocation information, including exact locations, can vary across data providers. As a best practice, use the same data provider for your place index, map, and other Amazon Location resources. For example, if the places returned by your place index do not match the location of the same places provided by your map resource, you can place a marker in what appears to be the wrong location on the map.

The following shows you how to create and use place index resources:



1. First, you create a place index resource in your AWS account by selecting a data provider.
2. You can then select and install the SDK that matches your development environment and applications. For more information about available options, see the topic about Accessing Amazon Location.
3. Start using the Amazon Location Places APIs . For more information, see the topic about using Places search.
4. You can then integrate monitoring using services such as Amazon CloudWatch and AWS CloudTrail. For more information see, the section called "Monitoring with CloudWatch" and the section called "Using CloudTrail with Amazon Location".

Geocoding concepts

An Amazon Location place index provides an action called `SearchPlaceIndexForText` that allows you to specify text to search. For example, you can search for:

- **Places** – a search for **`Paris`** could return the location of the city in France.
- **Businesses** – a search for **`coffee shop`** could return a list of coffee shops, including their names and locations. You can also specify a location to search around or a bounding box to search within, to make the results more relevant. In this case, providing a location in downtown Seattle, Washington, would return coffee shops in that area.
- **Addresses** – a search for **`1600 Pennsylvania Ave, Washington D.C.`** could return the location of the White House in the United States (which is at that address).

Searching for text in this way is generally referred to as **geocoding**, which involves finding a geographic location for the address or place.

Amazon Location Service also provides a **reverse geocoding** action called `SearchPlaceIndexForPosition`. This takes a geographic location and returns the address, business, or other information about what is at that location.

Search results

When you make a successful search request in Amazon Location Service, one or more results are returned. Each result includes a label, which is the name or description of the result. For example, a search for **`coffee shop`**, might return a result with the label `Hometown Cafe`, telling you that a coffee shop called "Hometown Cafe" was found. The search result will also typically include a structured address (with properties such as the address number, unit, street, and postal code). Depending on the data provider, it will include other meta data, as well, such as the country and time zone.

For a search on a business name or category (such as **`coffee shop`**), you might want to show all returned results on a map. For an address search, you might want to just use the first result automatically. See the next topic for information about relevance.

Multiple results and relevance

When searching by text, Amazon Location Service will often find more than a single result. For example, a search for **`Paris`** may return the city in France, but also the city in Texas. The results are sorted by the relevance, as determined by the data provider.

> **Note**
>
> Results are returned in relevance order from all providers. If you choose Esri or Grab as your data provider, the results include a relevance value that you can use to understand the relative relevance between the results of a single request.

Specifying additional information, such as a country name, or a location to search around, can change the order of results, reduce the number of results, or even change the set of results returned. For example, a search for **`Paris`** with a location in Texas to search around is more likely to return `Paris, Texas` as the first result than `Paris, France`.

In an interactive application, you can use relevance to help decide whether to accept the top result, or to ask a user to disambiguate between multiple returned results. If the first result has a high relevance, you might just accept it as the correct answer. If there are multiple high relevance results, or no high relevance results, you might want to list the results and let the user select the best result.

Address results

You can search for addresses with Amazon Location Service using the same `SearchPlaceIndexForText` action. The more information that you provide, the more likely the address returned will match the one given. For example, **`123 Main St`** is less likely to find a correct result than **`123 Main St, Anytown, California, 90210`**.

Addresses have multiple attributes, such as the street number, street, city, region, and postal code, etc. Those attributes are used to find an address in the place index that matches as many aspects as possible. The more attributes found, the more relevant the match is considered, and the more likely it will be returned.

> **Note**
>
> The relevance for address results is based on how closely the result matches the input. This could be the number of the attributes that matched, but also how closely the results match the input. For example, an input of **`123 Main St`** would have a higher relevance when `Main St` is found in the data, than if `Maine St` is the only result. `Maine St` will still be returned, but likely with a lower relevance value.

The search results include a label for the full address (`123 Main St, Anytown, California, 90210`), but also the individual structured attributes of the returned address. This is helpful, because you can use that, for example, to populate address fields in a database, or to examine the results and find the city, region, or postal code of the found location.

Interpolation

Addresses in the place index data includes exact address matches. For example, suppose that there is a street, `9th street` and one block has 2 houses, `220` and `240`, as in the following image.

The data provider creates the geolocation data with those two known addresses. You can search for those two addresses, and they are found. After the data provider creates the map data, let's suppose that a new house is added, between the first two addresses. This new house is given the address `230`. If you search for **`230 S 9th St`**, the data provider will still find a result. Instead of using a known address, it will interpolate between the already known addresses, and estimate the position of the new address from those. In this case, it might assume that 230 is halfway between 220 and 240 (and on the same side of the street), and return an approximate location based on that.

> **Note**
>
> Data providers periodically update their geolocation data with new addresses. In this case, `230 S 9th St` would get added to the data provider data, but there will typically be a period when a new address has been created but is not yet added to the data.

In this case, the data provider can't tell whether the new address exists in the world, as it is not yet in the data, but provides the best answer it can from the information it has. This result is called *interpolated*, and can be returned by the data provider in the results. If `interpolated` returns `false`, it is a known address. If it returns `true`, it's an approximated address. If it's not returned,

then the data provider did not provide the information about whether the result came from interpolation.

> **Important**
>
> The data provider may also return interpolated results for addresses that don't exist at all. For example in this case, if you entered **`232 S 9th St`**, the provider would find this nonexistent address, and return a location close to 230, but on the 240 side. Interpolated addresses are useful for getting you to the right location, but it is good to keep in mind that they are not known addresses.

Storing geocode results

When you create a place index resource, you must specify a **Data storage option** (called `IntendedUse` in the API). That can set to be either *single use* or *stored results*. This is asking about your intended use of the results. If you are going to store the results (even for caching purposes), you must choose the *storage* option, not the *single use* option.

> **Note**
>
> When you chose the stored option (labeled as **Yes, results will be stored** in the console, or choosing `storage` in the `CreatePlaceIndex` API), Amazon Location Service does not store the results for you. This is an indication that you are planning to store the results.

When looking at how you are going to use the results of your queries to Amazon Location Service, you should always be aware of the AWS Service Terms that apply.

Places terminology

Place index resource

Allows you to choose a data source to support search queries. For example, you can search for points of interest, addresses, or coordinates. When a search query is sent to a place index resource, it's fulfilled using the resource's configured data source.

Geocoding

Geocoding is the process of taking a text input, searching for it in the place index, and returning results with position.

Reverse geocoding

Reverse geocoding is the process of taking a position and returning information about that position from within the place index, such as the address, city, or business at that location.

Relevance

Relevance is how closely a result matches the input. It is not a measure of correctness.

Interpolation

Interpolation is the process of finding unknown addresses by using known address locations as guide points.

ISO 3166 country codes

Amazon Location Service Places uses the International Organization for Standardization (ISO) 3166 country codes to refer to countries or regions.

To find the code for a specific country or region, use the ISO Online Browsing Platform.

Routes

This section provides an overview of the concepts around routing using Amazon Location Service.

> **Note**
>
> For information about how to use route resources in practice, see Calculating routes using Amazon Location Service.

Route calculator resources

Route calculator resources allow you to find routes and estimate travel time based on up-to-date road network and live traffic information from your chosen data provider.

You can use the Routes APIs to build features that allow your application to request the travel time, distance, and geometry of the route between any two locations. You can also use the Routes API to request travel time and distance between a set of departures and destinations in a single request to calculate a matrix.

The following shows you how to create and use a route calculator resource:

1. First, you create a route calculator resource in your AWS account by selecting a data provider.
2. You can then select and install the SDK that matches your development environment and applications.
3. Start using the Amazon Location Routes APIs . For more information about how to use the routing APIs, see the topic on Calculating routes using Amazon Location Service.
4. You can then integrate monitoring using services such as Amazon CloudWatch and AWS CloudTrail. For more information see, Monitoring Amazon Location Service with Amazon CloudWatch and Logging and monitoring with AWS CloudTrail.

Calculating a route

An Amazon Location route calculator resource provides an action called `CalculateRoute` that you can use to create a route between two geographic locations (the *departure* and the *destination*). The calculated route includes the geometry for drawing the route on a map, plus the overall time and distance of the route.

Using waypoints

When you are creating your route request, you can add additional waypoints to the route. These are points between the departure and the destination that act as stops along the route. The route

will be calculated through each of the waypoints specified. The route from one point in the request to the next is called a Leg. Each leg includes a distance, time, and the geometry for that part of the route.

> **Note**
>
> The waypoints are routed in the order given in the request. They are not re-ordered for the shortest path. See the Planning routessection for information on finding the shortest path.

You can include up to 25 waypoints in a single request to calculate a route.

Traffic and departure time

The Amazon Location Service takes traffic into account when calculating a route. The trafffic that it considers is based on the time that you specify. You can specify to depart now, or you can provide a specific time that you want to leave, which will affect the route result by adjusting for traffic at the specified time.

> **Note**
>
> You can calculate the arrival time using the departure time and route response time, to estimate the arrival of a driver, for example.

If you want Amazon Location to not take traffic into account, then do not specify a departure time and do not specify depart now. This will calculate a route that assumes the best traffic conditions for the route.

Travel mode options

You can set the travel mode when calculating a route using Amazon Location Service. The default travel mode is *car*, but you can alternately select either *truck* or *walking*.

If you specify either car or truck mode, you can specific additional options, as well.

For **car mode**, you can specify that you wish to avoid toll roads or ferries. This will attempt to avoid ferries and toll roads, but will still route along them, if they are the only way to get to the destination.

For **truck mode**, you can also avoid ferries and toll roads, but additionally, you can specify the size and weight of the truck, to avoid routes that will not accommodate the truck.

Planning routes

You can use Amazon Location Service to create inputs to your route planning and optimization software. You can create route results, including travel time and travel distance, for routes between a set of departure positions and a set of destination positions. This is called creating a route *matrix*.

> **Note**
>
> There are many varying scenarios that route planning and optimization software can solve. For example, planning software can use the set of times and distances between points to calculate the shortest path that stops at each point, providing an efficient route for a single driver. Alternatively, planning software can be used to split stops between multiple trucks, providing efficiencies across a fleet, or to make sure that each customer is visited within the time frame that they require. Amazon Location provides the routing functions in an efficient way to allow the planning software to complete it's task.

For example, given departure positions A and B, and destination positions X and Y, Amazon Location Service will return travel time and travel distance for routes from A to X, A to Y, B to X, and B to Y.

As with calculating a single route, you can calculate the routes with different modes of transportation, avoidances, and traffic conditions. For example, you can specify that the vehicle is a truck that is 35 feet long, and the route calculated will use those restrictions to determine the travel time and travel distance. You can't include waypoints in a route matrix calculation.

The number of results returned (and routes calculated) is the number of departure positions multiplied by the number of destination positions. You are charged for each route calculated, not each request to the service, so a route matrix with 10 departures and 10 destinations will be billed as 100 routes.

Route terminology

Route calculator resource

An AWS resource that enables you to estimate travel time, distance, and plot routes on a map with traffic and road network data sourced from your chosen data provider.

Using route calculator resources, you calculate routes for different modes of transportation, detours, and traffic conditions.

Route

A route contains details used when traveling along a path from the departure position, waypoint positions, and destination position.

Examples of details in a route include:

- The distance from one position to another position.
- The time it takes to travel from one position to the next position.
- The LineString geometry representing the path of the route.

For more information about routes, see the response syntax for the CalculateRoute operation in the *Amazon Location Service Routes API reference*.

Route matrix

A list of routes, from a set of departure positions to a set of destination positions. Useful as inputs into route planning or optimization software.

For more information about calculating a route matrix, see the syntax for the CalculateRouteMatrix operation in the *Amazon Location Service Routes API reference*.

`LineString` geometry

An Amazon Location route consists of one or more legs (a route from one waypoint to another within the overall route). The geometry of each leg is a polyline represented as a `LineString`. A `LineString` is an ordered array of positions that can be used to plot a route on a map.

The following is an example of a `LineString` with three points:

```
[
    [-122.7565,49.0021],
    [-122.3394,47.6159],
    [-122.1082,45.8371]
]
```

Waypoint

Waypoints are intermediate positions that act as stops along a route between the departure position and destination position. The stopover order on the route follows the order that you provide the waypoint positions in the request.

Leg

A single leg is the journey from one position to another position. If the positions aren't located on a road, they're moved to the nearest road. The number of legs in a route is one less than the total number of positions.

A route with no waypoints consists of a single leg, from the departure position to the destination. A route with 1 waypoint consists of 2 legs, from the departure position to the waypoint, and then from the waypoint to the destination.

Step

A step is a subsection of a leg. Each step provides summary information for that step in the leg.

Geofences and Trackers

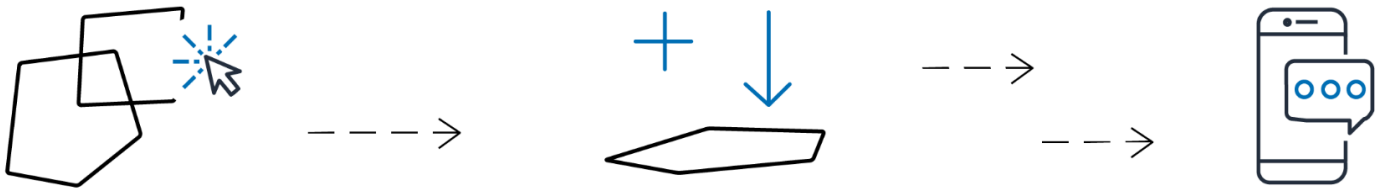
This section provides and overview of the concepts of working with Amazon Location Service geofences and trackers. Geofences are polygon boundaries that you can use to be notified when devices or positions move in and out of the areas. Tracker resources are used to store and update positions for devices as they move.

> **Note**
>
> For information about how to use geofences and trackers in practice, see Geofencing an area of interest using Amazon Location.

Geofences

Geofence collection resources allow you to store and manage geofences—virtual boundaries on a map. You can evaluate locations against a geofence collection resource and get notifications when the location update crosses the boundary of any of the geofences in the geofence collection.

The following shows you how to create and use geofence collection resources:



1. Create a geofence collection resource in your AWS account.
2. Add geofences to that collection. You can do so by either using the geofence upload tool on the Amazon Location console, or by using the Amazon Location Geofences API. For more information about available options, see Accessing Amazon Location.

   Geofences can either be defined by a polygon or by a circle. Use a polygon to find when a device enters a specific area. Use a circle to find when a device comes within a certain distance (radius) of a point.
3. You can start evaluating locations against all your geofences. When a location update crosses the boundaries of one or more geofences, your geofence collection resource emits one of the following geofence event types on Amazon EventBridge:
   - **ENTER** – One event is generated for each geofence where the location update crosses its boundary by entering it.
   - **EXIT** – One event is generated for each geofence where the location update crosses its boundary by exiting it.

   For more information, see the section called "Reacting to events with EventBridge". You can also integrate monitoring using services such as Amazon CloudWatch and AWS CloudTrail. For more information see, the section called "Monitoring with CloudWatch" and the section called "Using CloudTrail with Amazon Location".

For example, if you are tracking a fleet of trucks, and you want to get notified when a truck comes within a certain area of any of your warehouses. You can create a geofence for the area around each warehouse. Then, when the trucks send you updated locations, you can use Amazon Location Service to evaluate those positions and see if a truck has entered (or exited) one of the geofence areas.

> **Note**
>
> Your are billed by the number of geofence collections you evaluate against. Your bill is not affected by the number of geofences in each collection. Since each geofence collection may contain up to 50,000 geofences, you may want to combine your geofences into fewer collections, where possible, to reduce your cost of geofence evaluations. The events generated will include the ID of the individual geofence in the collection, as well as the ID of the collection.

Geofence events

Locations for positions you are monitoring are referenced by an ID called a `DeviceId` (and the positions are referred to as device positions). You can send a list of device positions to evaluate directly to the geofence collection resource, or you can use a tracker. See the next section for more information on trackers.

You receive events (via Amazon EventBridge) only when a device enters or exits a geofence, not for every position change. This means that you will typically receive events and have to respond to them much less frequently than every device position update.

> **Note**
>
> For the first location evaluation for a specific `DeviceID`, it is assumed that the device was previously not in any geofences. So the first update will generate an `ENTER` event, if inside a geofence in the collection, and no event if not.

In order to calculate whether a device has entered or exited a geofence, Amazon Location Service must keep previous position state for the device. This position state is stored for 30 days. After 30 days without an update for a device, a new location update will be treated as the first position update.

Trackers

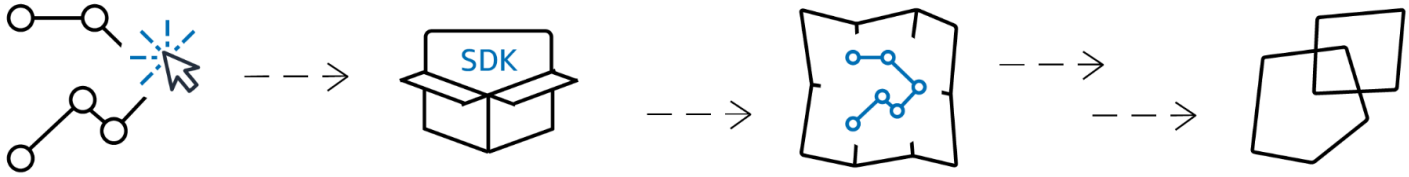
A tracker stores position updates for a collection of devices. The tracker can be used to query the devices' current location or location history. It stores the updates, but reduces storage space and visual noise by filtering the locations before storing them.

Each position update stored in your tracker resources can include a measure of position accuracy and up to 3 fields of metadata about the position or device that you want to store. The metadata is stored as key-value pairs, and can store information such as speed, direction, tire pressure, or engine temperature.

> **Note**
>
> Tracker storage is encrypted with AWS owned keys automatically. You can add another layer of encryption using KMS keys that you manage, to ensure that only you can access your data. For more information, see Data encryption at rest for Amazon Location Service.

Tracker position filtering and storage are useful on their own, but trackers are especially useful when paired with geofences. You can link trackers to one or more of your geofence collection resources, and position updates are evaluated automatically against the geofences in those collections. Proper use of filtering can greatly reduce the costs of your geofence evaluations, as well.

The following diagram shows you how to create and use tracker resources:



1. First, you create a tracker resource in your AWS account.
2. Next, decide how you send location updates to your tracker resources. Use AWS SDKs to integrate tracking capabilities into your mobile applications. Alternately, you can use MQTT by following step-by-step directions in tracking using MQTT.
3. You can now use your tracker resource to record location history and visualize it on a map.
4. You can also link your tracker resource to one or more geofence collections so that every position update sent to your tracker resource is automatically evaluated against all the geofence in all the linked geofence collections. You can link resource on the tracker resource details page of the Amazon Location console or by using the Amazon Location Trackers API.
5. You can then integrate monitoring using services such as Amazon CloudWatch and AWS CloudTrail. For more information see, the section called “Monitoring with CloudWatch” and the section called “Using CloudTrail with Amazon Location”.

Using trackers with geofences

Trackers provide additional functionality when paired with geofences. You associate a tracker with a geofence collection, either through the Amazon Location console or the API, to automatically evaluate tracker locations. Each time the tracker receives an updated location, that location will be evaluated against each geofence in the collection, and the appropriate `ENTER` and `EXIT` events are generated in Amazon EventBridge. You can also apply filtering to the tracker, and, depending on the filtering, you can reduce the costs for geofence evaluations by only evaluating meaningful location updates.

If you associate the tracker with a geofence collection after it has already received some position updates, the first position update after association is treated as an initial update for the geofence evaluations. If it is within a geofence, you will receive an `ENTER` event. If it is not within any geofences you will not receive an `EXIT` event, regardless of the previous state.

Position filtering

Trackers can automatically filter the positions that are sent to them. There are several reasons why you might want to filter out some of your device location updates. If you have a system that only sends reports every minute or so, you might want to filter devices by time, storing and evaluating positions only every 30 seconds. Even if you are monitoring more frequently, you might want to filter position updates to clean up the noisiness of GPS hardware. GPS position locations are inherently noisy. Their accuracy is not 100% perfect, so even a device that is stationary appears to be moving around slightly. At low speeds, this *jitter* causes visual clutter and can cause false entry and exit events if the device is near the edge of a geofence.

The position filtering works as position updates are received by a tracker, reducing visual noise in your device paths (jitter), reducing the number of false geofence entry and exit events, and helping manage costs by reducing the number of position updates stored and geofence evaluations triggered.

Trackers offer three position filtering options to help manage costs and reduce jitter in your location updates.

- **Accuracy-based** – *Use with any device that provides an accuracy measurement. Most GPS and mobile devices provide this information.* The accuracy of each position measurement is affected by many environmental factors, including GPS satellite reception, landscape, and the proximity of wifi and bluetooth devices. Most devices, including most mobile devices, can provide an estimate of the accuracy of the measurement along with the measurement. With `AccuracyBased` filtering, Amazon Location ignores location updates if the device moved less than the measured accuracy. For example, if two consecutive updates from a device have an accuracy range of 5 m and 10 m, Amazon Location ignores the second update if the device has moved less than 15 m. Amazon Location neither evaluates ignored updates against geofences, nor stores them.

  When accuracy is not provided, it is treated as zero, and the measurement is considered perfectly accurate, and no filtering will be applied to the updates.

> **Note**
>
> You can use accuracy-based filtering to remove all filtering. If you select accuracy-based filtering, but override all accuracy data to zero, or omit the accuracy entirely, then Amazon Location will not filter out any updates.

In most scenarios, accuracy-based filtering is a good choice for filtering position updates, providing a balance of tracking location while filtering out unneeded updates, thereby reducing costs.

- **Distance-based** – *Use when your devices do not provide an accuracy measurement, but you still want to take advantage of filtering to reduce jitter and manage costs.* `DistanceBased` filtering ignores location updates in which devices have moved less than 30 m (98.4 ft). When you use `DistanceBased` position filtering, Amazon Location neither evaluates these ignored updates against geofences nor stores the updates.

  The accuracy of most mobile devices, including the average accuracy of iOS and Android devices, is within 15 m. In most applications, `DistanceBased` filtering can reduce the effect of location inaccuracies when displaying device trajectory on a map, and the bouncing effect of multiple consecutive entry and exit events when devices are near the border of a geofence. It can also help reduce the cost of your application, by making fewer calls to evaluate against linked geofences or retrieve device positions.

  Distance-based filtering is useful if you want to filter, but your device doesn't provide accuracy measurements, or you want to filter out a larger number of updates than with accuracy-based.

- **Time-based** – (default) *Use when your devices send position updates very frequently (more than once every 30 seconds), and you want to achieve near real-time geofence evaluations without storing every update.* In `TimeBased` filtering, every location update is evaluated against linked geofence collections, but not every location update is stored. If your update frequency is more often than 30 seconds, only one update per 30 seconds is stored for each unique device ID.

  Time-based filtering is particularly useful when you want to store fewer positions, but want every position update to be evaluated against the associated geofence collections.

> **Note**
>
> Be mindful of the costs of your tracking application when deciding your filtering method and the frequency of position updates. You are billed for every location update and once for evaluating the position update against each linked geofence collection. For example, when using time-based filtering, if your tracker is linked to two geofence collections, every position update will count as one location update request and two geofence collection evaluations. If you are reporting position updates every 5 seconds for your devices and using time-based filtering, you will be billed for 720 location updates and 1,440 geofence evaluations per hour for each device.

Geofence terminology

Geofence Collection

Contains zero or more geofences. It is capable of geofence monitoring by emitting Entry and Exit events, when requested, to evaluate a device position against its geofences.

Geofence

A polygon or circle geometry that defines a virtual boundary on a map.

Polygon geometry

An Amazon Location geofence is a virtual boundary for a geographical area and is represented as a polygon geometry or as a circle.

A circle is a point with a distance around it. Use a circle when you want to be notified if a device is within a certain distance of a location.

A polygon is an array composed of 1 or more linear rings. Use a polygon when you want to define a specific boundary for device notifications. A linear ring is an array of four or more vertices, where the first and last vertex are the same to form a closed boundary. Each vertex is a 2-dimensional point of the form `[longitude, latitude]`, where the units of longitude and latitude are degrees. The vertices must be listed in counter-clockwise order around the polygon.

> **Note**
>
> Amazon Location Service doesn't support polygons with more than one ring. This includes holes, islands or multipolygons. Amazon Location also doesn't support polygons that are wound clockwise, or that cross the antimeridian.

The following is an example of a single linear external ring:

```
[
  [
    [-5.716667, -15.933333],
    [-14.416667, -7.933333],
    [-12.316667, -37.066667],
    [-5.716667, -15.933333]
  ]
]
```

Tracker terminology

Tracker resource

An AWS resource that receives location updates from devices. The tracker resource provides support for location queries, such as current and historic device location. Linking a tracker resource to a geofence collection evaluates location updates against all geofences in the linked geofence collection automatically.

Position data tracked

A tracker resource stores information about your devices over time. The information includes a series of position updates, where each update includes location, time, and optional metadata. The metadata can include a position's accuracy, and up to three key-value pairs to help you track key information about each position, such as speed, direction, tire pressure, remaining fuel, or engine temperature of the vehicle you are tracking. Trackers maintain device location history for 30 days.

Position filtering

Position filtering can help you control costs and improve the quality of your tracking application by filtering out position updates that don't provide valuable information before the updates are stored or evaluated against geofences.

You can choose `AccuracyBased`, `DistanceBased`, or `TimeBased` filtering. By default, position filtering is set to `TimeBased`.

You can configure position filtering when you create or update tracker resources.

RFC 3339 timestamp format

Amazon Location Service Trackers uses the RFC 3339 format, which follows the International Organization for Standardization (ISO) 8601 format for dates and time.

The format is “YYYY-MM-DDThh:mm:ss.sssZ+00:00”:

- `YYYY-MM-DD` — Represents the date format.
- `T` — Indicates that the time values will follow.
- `hh:mm:ss.sss` — Represents the time in 24-hour format.
- `Z` — Indicates that the time zone used is UTC, which can be followed with deviations from the UTC time zone.
- `+00:00` — Optionally indicate deviations from the UTC time zone. For example, +01:00 indicates UTC + 1 hour.

Example

For July 2, 2020, at 12:15:20 in the afternoon, with an adjustment of an additional 1 hour to the UTC time zone.

```
2020-07-02T12:15:20.000Z+01:00
```

Common use cases for using Amazon Location Service

Amazon Location Service lets you build a range of applications, from asset tracking to location-based marketing. The following are common use cases:

User engagement and geomarketing

Use location data to build solutions that improve user engagement with marketing to target customers. For example, Amazon Location can trigger an event that prompts a notification when a customer who ordered a coffee on their mobile app is nearby. Additionally, you can build geotargeting features so that retailers can send discount codes or digital flyers to customers who are near target stores.

Asset tracking

Build asset tracking features to help businesses understand the current and historical locations of their products, personnel, and infrastructure. With asset tracking features, you can build a number of solutions that optimize remote staffing, secure shipment en-route, and maximize dispatch efficacy.

Delivery

Integrate location features into delivery applications to store, track, and coordinate the departure location, delivery vehicles, and their destination. For example, a food delivery application with Amazon Location features built-in has location tracking and geofencing capabilities that can automatically notify a restaurant when a delivery driver is nearby. This reduces the wait time and helps maintain the quality of the food delivered.

This topic provides you an overview of the architecture and steps for applications you can build with Amazon Location.

Topics

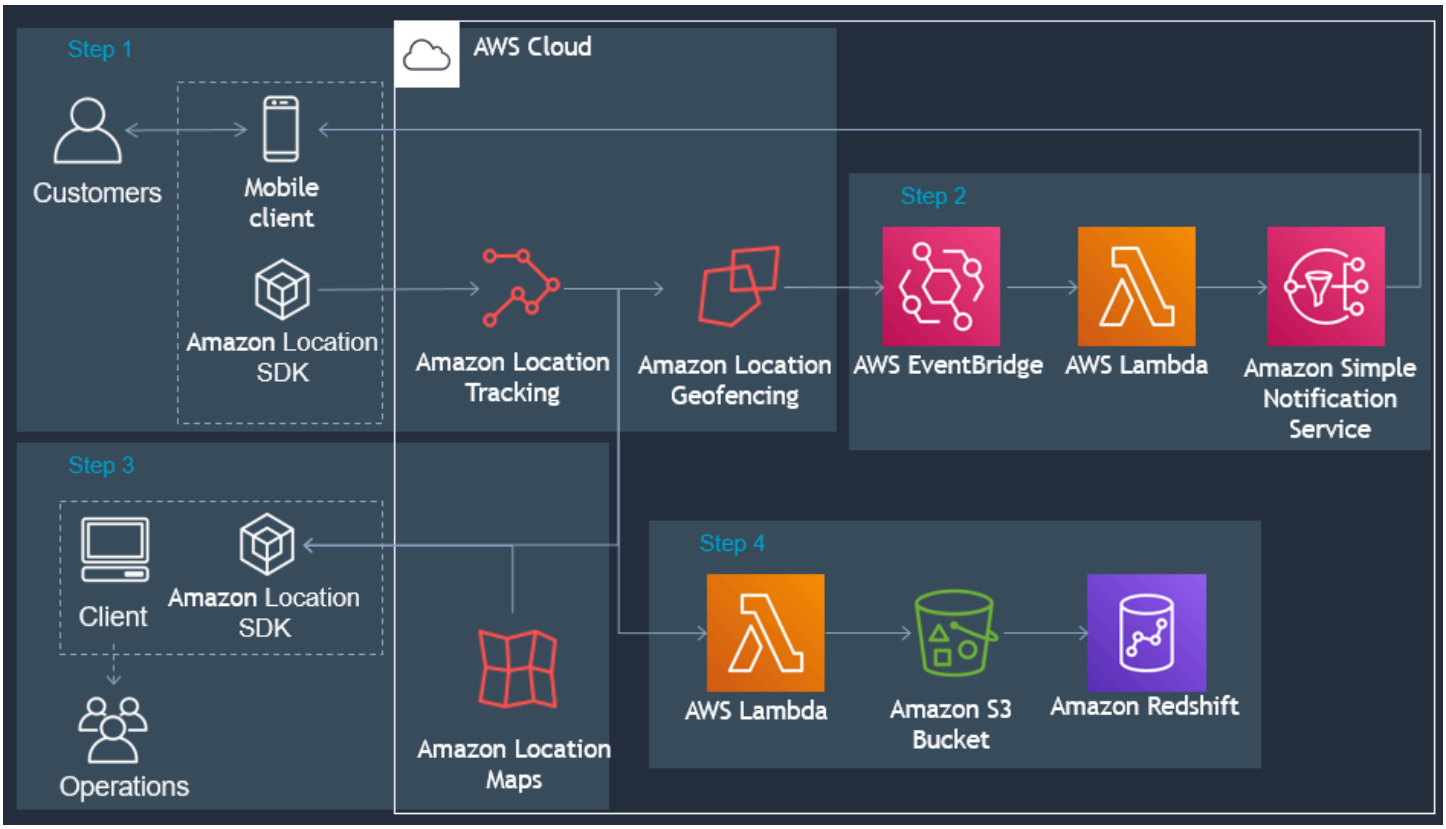
- User engagement and geomarketing applications
- Asset tracking applications
- Delivery applications

User engagement and geomarketing applications

The following is an illustration of a user engagement and geomarketing application architecture using Amazon Location:

With this architecture, you can:

- Initiate events based on the proximity of a target so that you can send offers to nearby customers or engage those who recently left your establishment (called *geotargeting*).
- Visualize customer device locations on a map to monitor trends over time.
- Store customer device locations that you can analyze over time.
- Analyze location history to identify trends and opportunities for optimization.



The following is an overview of the steps required to build a user engagement and geomarketing application:

1. Create your geofences in Geofence Collections and link Trackers to them. For more information, see the section called "Geofencing and tracking".
2. Configure Amazon EventBridge to send a notification to customers who enter or exit a geofenced area of interest. For more information, see the section called "Reacting to events with EventBridge".
3. Display customer locations and geofences on a map. For more information, see Using maps.
4. Save location data to long-term storage for further analysis.

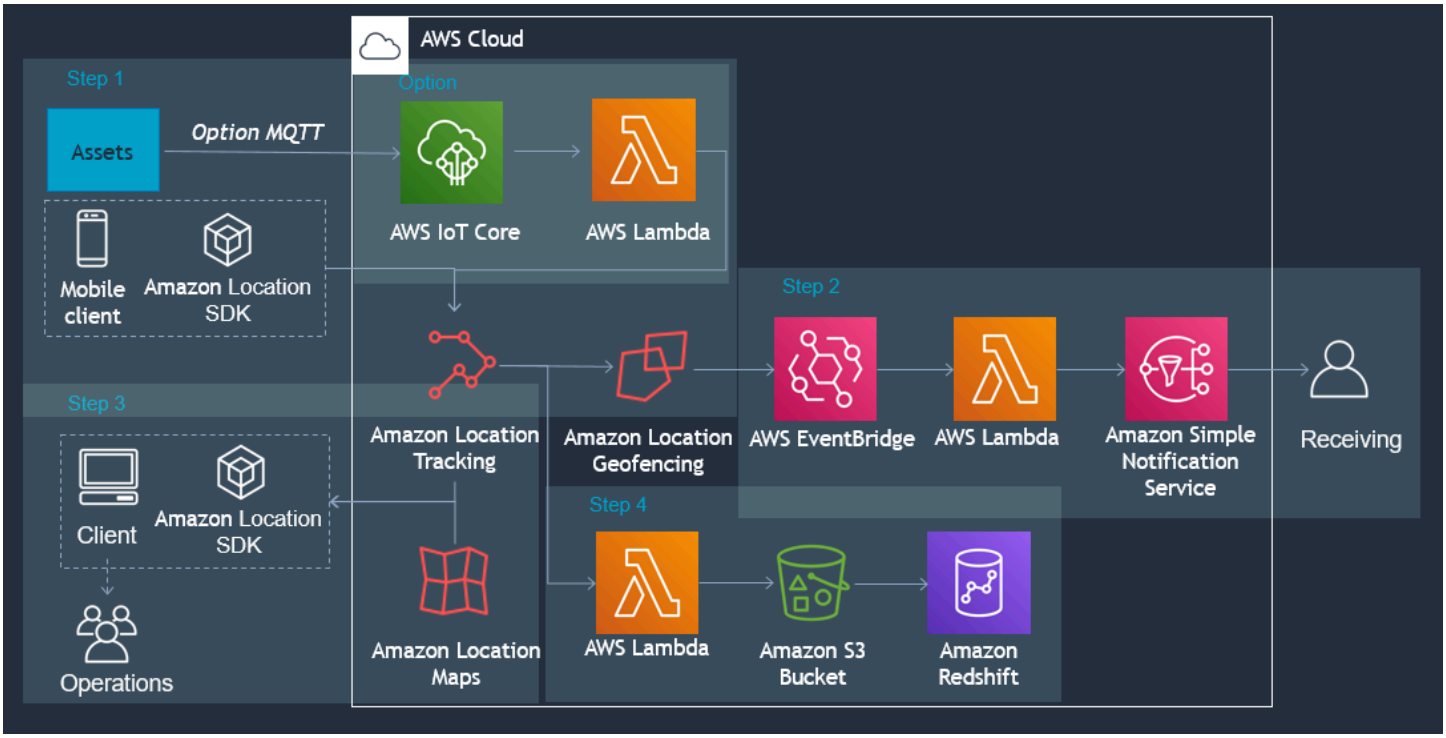
5. Once you have built your application, you can use Amazon CloudWatch and AWS CloudTrail to manage your application. For more information, see the section called “Monitoring with CloudWatch” and the section called “Using CloudTrail with Amazon Location”.

Asset tracking applications

The following is an illustration of an asset tracking application architecture using Amazon Location:

With this architecture, you can:

- Display asset locations on a map to illustrate the big picture. For example, showing a heat map using historical locations or events to help an operations or planning team.
- Initiate events based on asset proximity to provide notice to a receiving department to prepare for a shipment arrival and reduce processing time.
- Store asset locations to initiate actions in your backend applications or to analyze data over time.
- Analyze location history to identify trends and opportunities for optimization.



The following provides an overview of the steps required to build an asset tracking application:

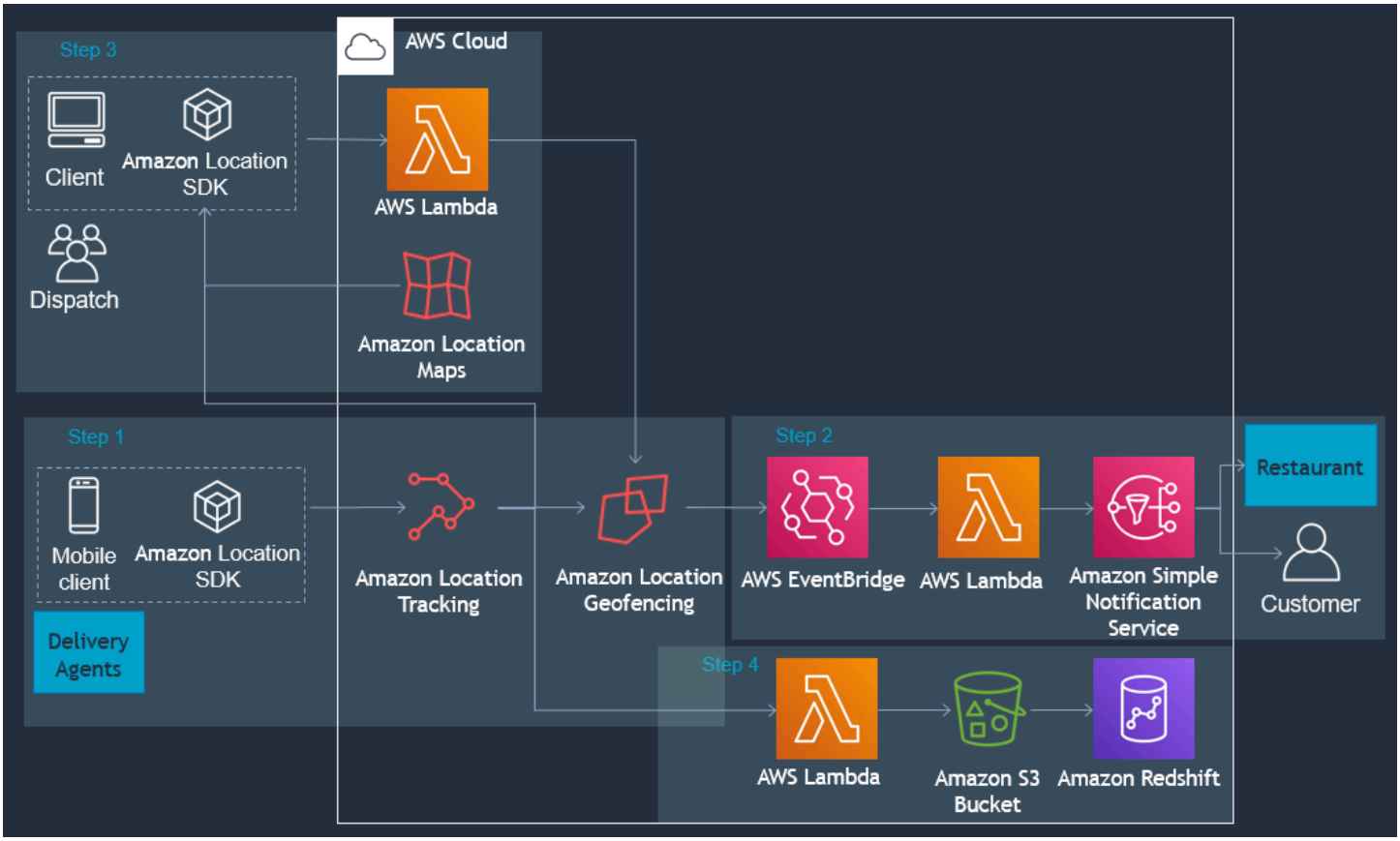
1. Create your geofences in Geofence Collections and link Trackers to them. For more information, see the section called “Geofencing and tracking”.
2. Configure Amazon EventBridge to send a notification or initiate a process. For more information, see the section called “Reacting to events with EventBridge”.
3. Display your tracked assets and your active geofences on a map. For more information, see Using maps.
4. Save location data in long-term storage for further analysis.
5. Once you have built your application, you can use Amazon CloudWatch and AWS CloudTrail to manage your application. For more information, see the section called “Monitoring with CloudWatch” and the section called “Using CloudTrail with Amazon Location”.

Delivery applications

The following is an illustration of a delivery application architecture using Amazon Location.

With this architecture, you can:

- Initiate events based on proximity of delivery agents so that pickups are ready in time and customers can be notified when their delivery is arriving.
- Display driver locations, as well as pick-up and drop-off locations in near-real time on a map to show dispatch teams the big picture.
- Store the locations of delivery agents so that you can act on them in your backend application or analyze them over time.
- Analyze location history to identify trends and opportunities for optimization.



The following is an overview of the steps required to build a delivery application:

1. Create your geofence collections and link tracked devices to the collection. For more information see, the section called “Geofencing and tracking”.
2. Create an AWS Lambda function to automatically add and remove geofences as your orders are booked.
3. Configure Amazon EventBridge to send notifications or initiate a processes. For more information, see the section called “Reacting to events with EventBridge”.
4. Display tracked assets and active geofences on a map. For more information, see Using maps.
5. Save location data to long-term storage for further analysis.
6. Once you have built your application, you can use Amazon CloudWatch and AWS CloudTrail to manage your application. For more information, see the section called “Monitoring with CloudWatch” and the section called “Using CloudTrail with Amazon Location”.

What is a data provider?

Use Amazon Location Service to access geolocation resources from multiple data providers through your AWS account without requiring third-party contracts or integrations. This can help you focus on building your application, without having to manage third-party accounts, credentials, licenses, and billing.

The following Amazon Location services use data providers.

- **Maps** – Choose styles from different map providers when you create a map resource. You can use map resources to build an interactive map to visualize data.
- **Places** – Choose a data provider when you create a place index resource to support queries for geocoding, reverse geocoding, and searches.
- **Routes** – Choose a data provider to support queries for route calculations in different geographies and applications when you create a route calculator resource. With your chosen data provider, Amazon Location Service enables you to calculate routes based on up-to-date road network data, live traffic data, planned closures, and historic traffic patterns.

Each provider gathers and curates their data using different means. They may also have varying expertise in different regions of the world. This section provides details about our data providers. You may select any data provider based on your preference.

Make sure you read the terms of conditions when using Amazon Location Service data providers. For more information, see the AWS Service Terms. Also see the the section called “Data privacy” section for more information about how Amazon Location protects your privacy.

Data provider coverage and features

The following table shows coverage and features at a high level for each data provider.

Data provider	Geographical coverage	Feature coverage	AWS Region
Esri	Global	Maps, Places, Routes	All Regions where Amazon Location is available.

Data provider	Geographical coverage	Feature coverage	AWS Region
Grab	Southeast Asia	Maps, Places, Routes	Asia Pacific (Singapore), `ap-southeast-1` , only.
HERE	Global	Maps, Places, Routes	All Regions where Amazon Location is available.
Open Data	Global	Maps	All Regions where Amazon Location is available.

For more information about each data provider's specific features, see Features by data provider.

Each data provider gathers and produces data in different ways. You can learn more about their coverage areas in the following topics:

- Coverage: Esri
- Coverage: Grab
- Coverage: HERE
- Coverage: Open Data

If you encounter a problem with the data and want to report an error to the data provider, see the following topics:

- Error reporting to Esri
- Error reporting for GrabMaps data
- Error reporting to HERE
- Error reporting and contributing to Open Data

Map styles

Each data provider provides a set of map styles to render the map data that they provide. For example a style might include satellite imagery, or might be optimized to show the roads for navigation. You can find the list and examples of the styles for each provider in the following topics.

- Esri map styles
- Grab map styles
- HERE map styles
- Open Data map styles

More information about each data provider

The following links provide more information about each data provider.

- Esri
- GrabMaps
- HERE Technologies
- Open Data

Esri

Amazon Location Service uses Esri's location services to help AWS customers to use maps, geocode, and calculate routes effectively. Esri's location services are built with high-quality, authoritative, and ready-to-use location data, curated by expert teams of cartographers, geographers, and demographers.

For additional capability information, see Esri on *Amazon Location Service data providers*.

Topics

- Esri map styles
- Coverage: Esri
- Terms of use and data attribution: Esri
- Error reporting to Esri

Esri map styles

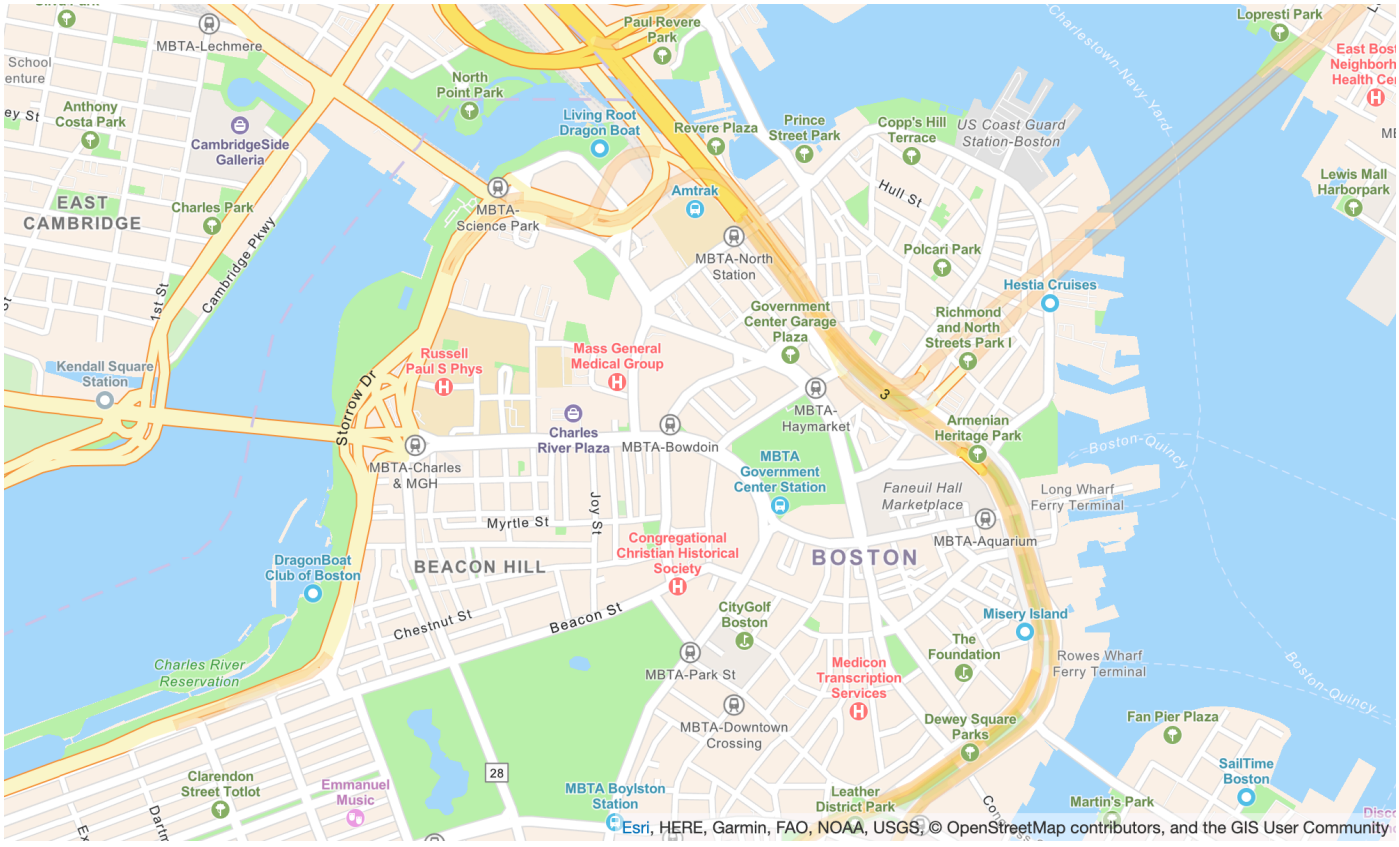
Amazon Location Service supports the following Esri map styles when creating a map resource.

> **Note**
>
> Esri map styles that are not listed in this section are not supported.

The Esri vector styles support alternate Political views.

Esri Navigation

Esri Navigation



Map style name: `VectorEsriNavigation`

This map provides a detailed basemap for the world symbolized with a custom navigation map style that's designed for use during the day in mobile devices.

This comprehensive street map includes highways, major roads, minor roads, railways, water features, cities, parks, landmarks, building footprints, and administrative boundaries. The

vector tile layer in this map is built using the same data sources used for the World Street Map and other Esri basemaps. Enable the `POI` layer by setting it in CustomLayers to leverage the additional places data.

For more information, see Esri World Navigation on the Esri website.

> **Note**
>
> The `VectorEsriNavigation` map pictured above has the `POI` layer enabled.

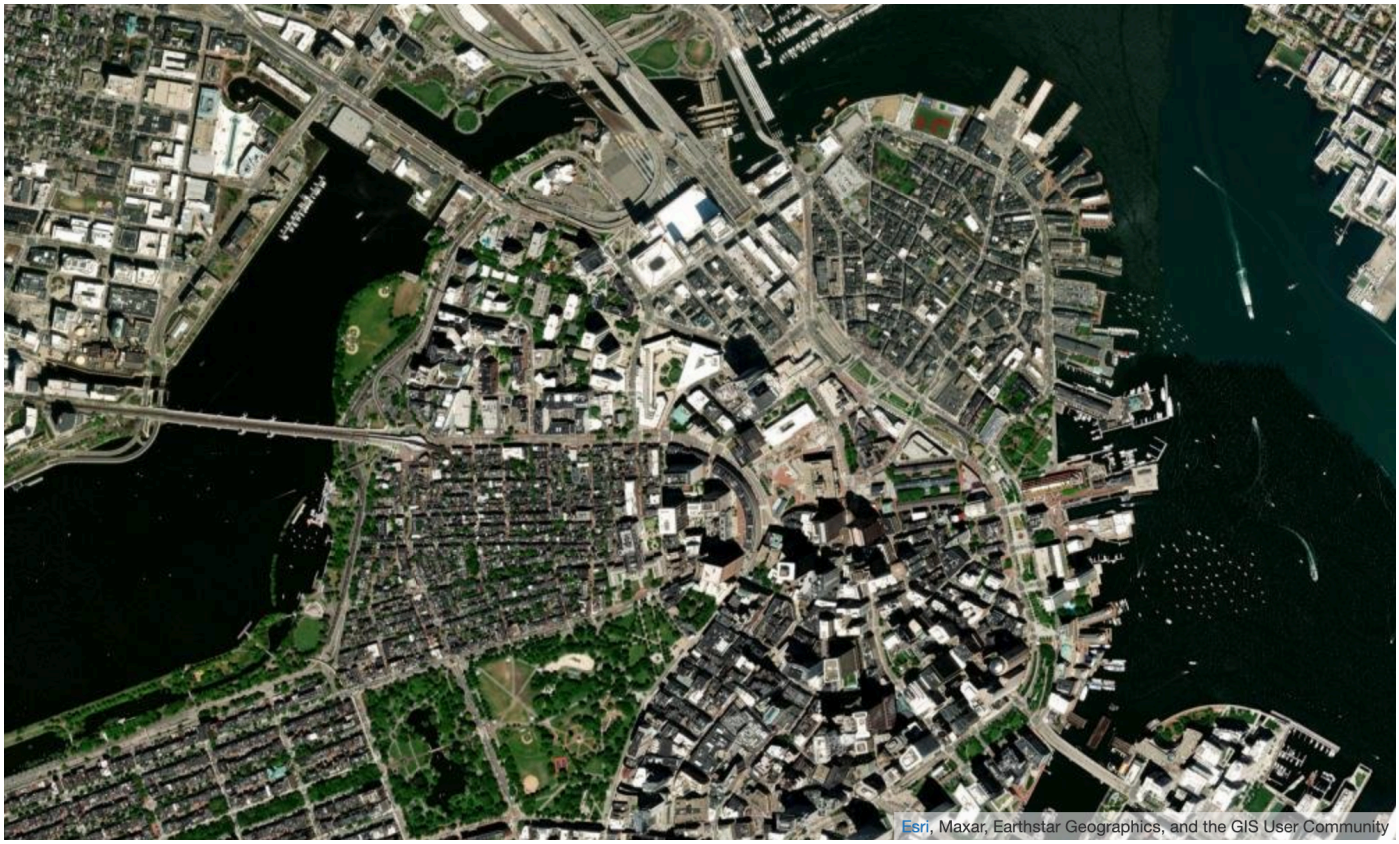
Fonts

Amazon Location serves fonts using `GetMapGlyphs`. The following are available font stacks for this map:

- Arial Italic
- Arial Regular
- Arial Bold
- Arial Unicode MS Bold
- Arial Unicode MS Regular

Esri Imagery

Esri Imagery



Map style name: `RasterEsriImagery`

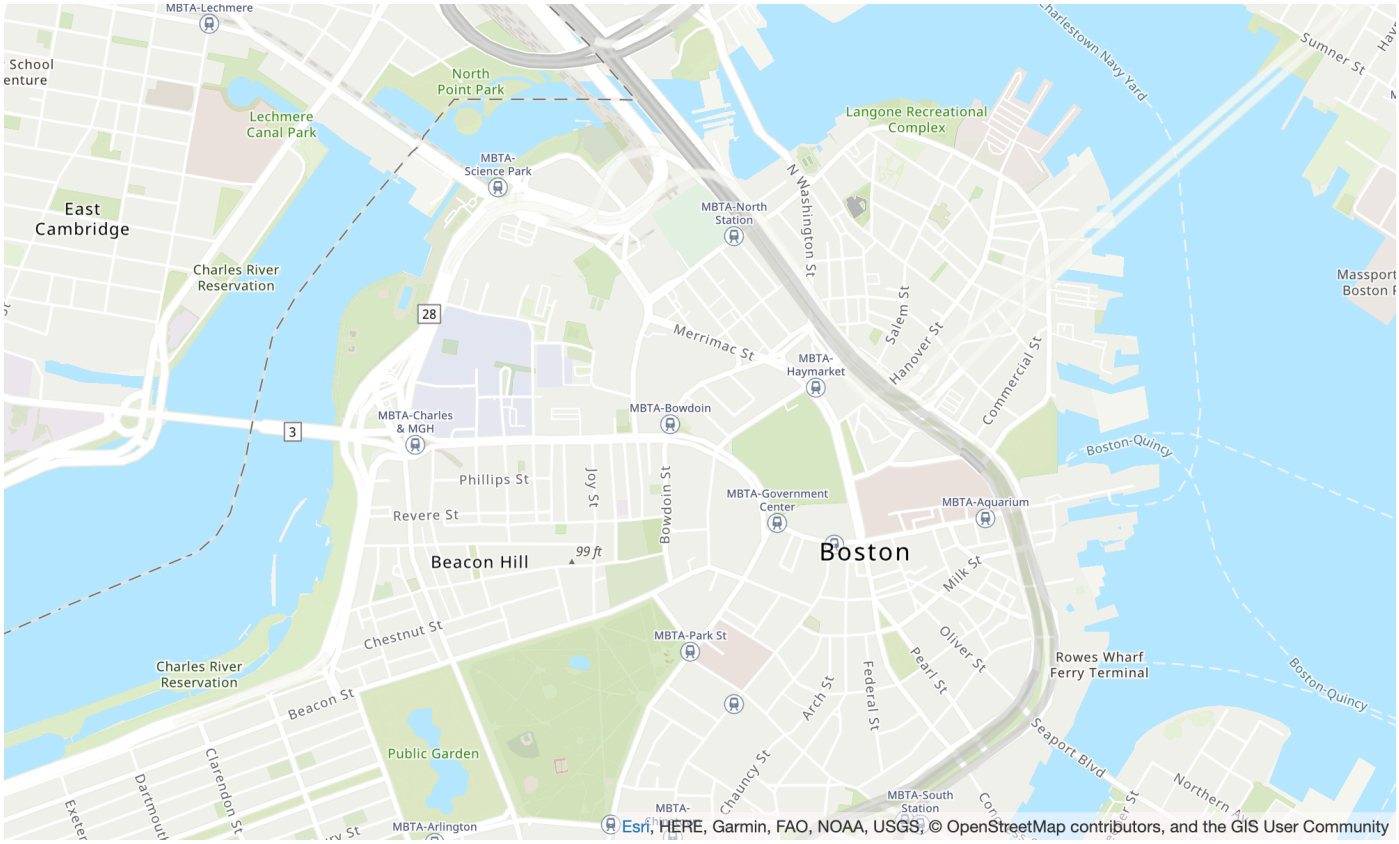
This map provides one meter or better satellite and aerial imagery in many parts of the world and lower resolution satellite imagery worldwide.

The map includes 15m imagery at small and mid-scales (~1:591M down to ~1:72k) and 2.5m SPOT Imagery (~1:288k to ~1:72k) for the world. The map features 0.5m resolution imagery in the continental United States and parts of Western Europe from Maxar. This map features additional Maxar submeter imagery in many parts of the world. In other parts of the world, the GIS User Community has contributed imagery at different resolutions. In select communities, very high-resolution imagery (down to 0.03m) is available down to ~1:280 scale.

For more information, see Esri World Imagery on the Esri website.

Esri Light

Esri Light



Map style name: `VectorEsriTopographic`

This provides a detailed basemap for the world symbolized with a classic Esri map style. This includes highways, major roads, minor roads, railways, water features, cities, parks, landmarks, building footprints, and administrative boundaries.

This basemap is compiled from a variety of authoritative sources from several data providers, including the US Geological Survey (USGS), US Environmental Protection Agency (EPA), US National Park Service (NPS), Food and Agriculture Organization of the United Nations (FAO), Department of Natural Resources Canada (NRCAN), HERE, and Esri. Data for select areas is sourced from OpenStreetMap contributors. Additionally, data is provided by the GIS community.

Fonts

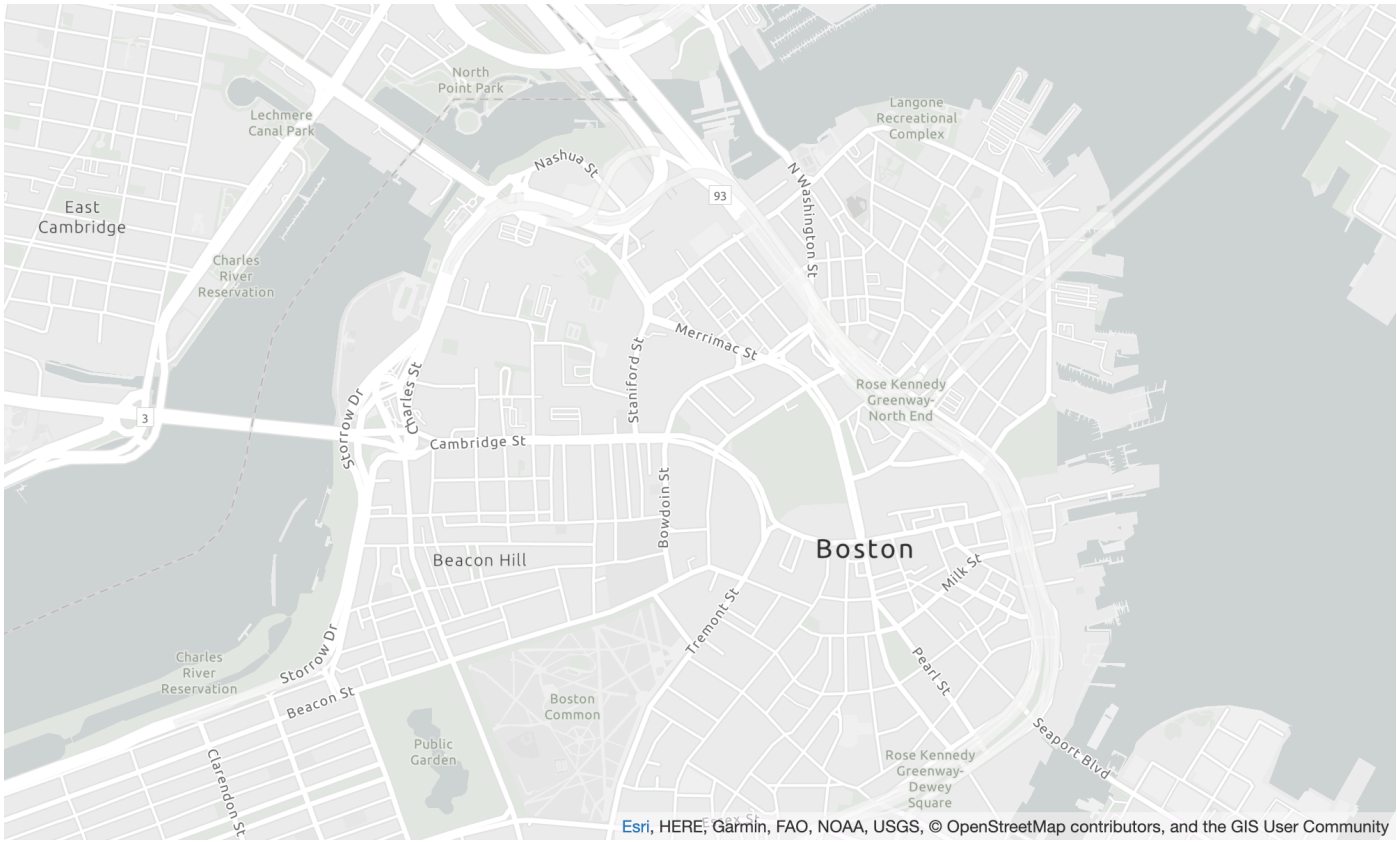
Amazon Location serves fonts using `GetMapGlyphs`. The following are available font stacks for this map:

- Noto Sans Italic
- Noto Sans Regular
- Noto Sans Bold

- Noto Serif Regular
- Roboto Condensed Light Italic

Esri Light Gray Canvas

Esri Light Gray Canvas



Map style name: `VectorEsriLightGrayCanvas`

This map provides a detailed basemap for the world symbolized with a light gray, neutral background style with minimal colors, labels, and features that's designed to draw attention to your thematic content.

This vector tile layer is built using the same data sources used for the Light Gray Canvas and other Esri basemaps. The map includes highways, major roads, minor roads, railways, water features, cities, parks, landmarks, building footprints, and administrative boundaries.

For more information, see Esri Light Gray Canvas on the Esri website.

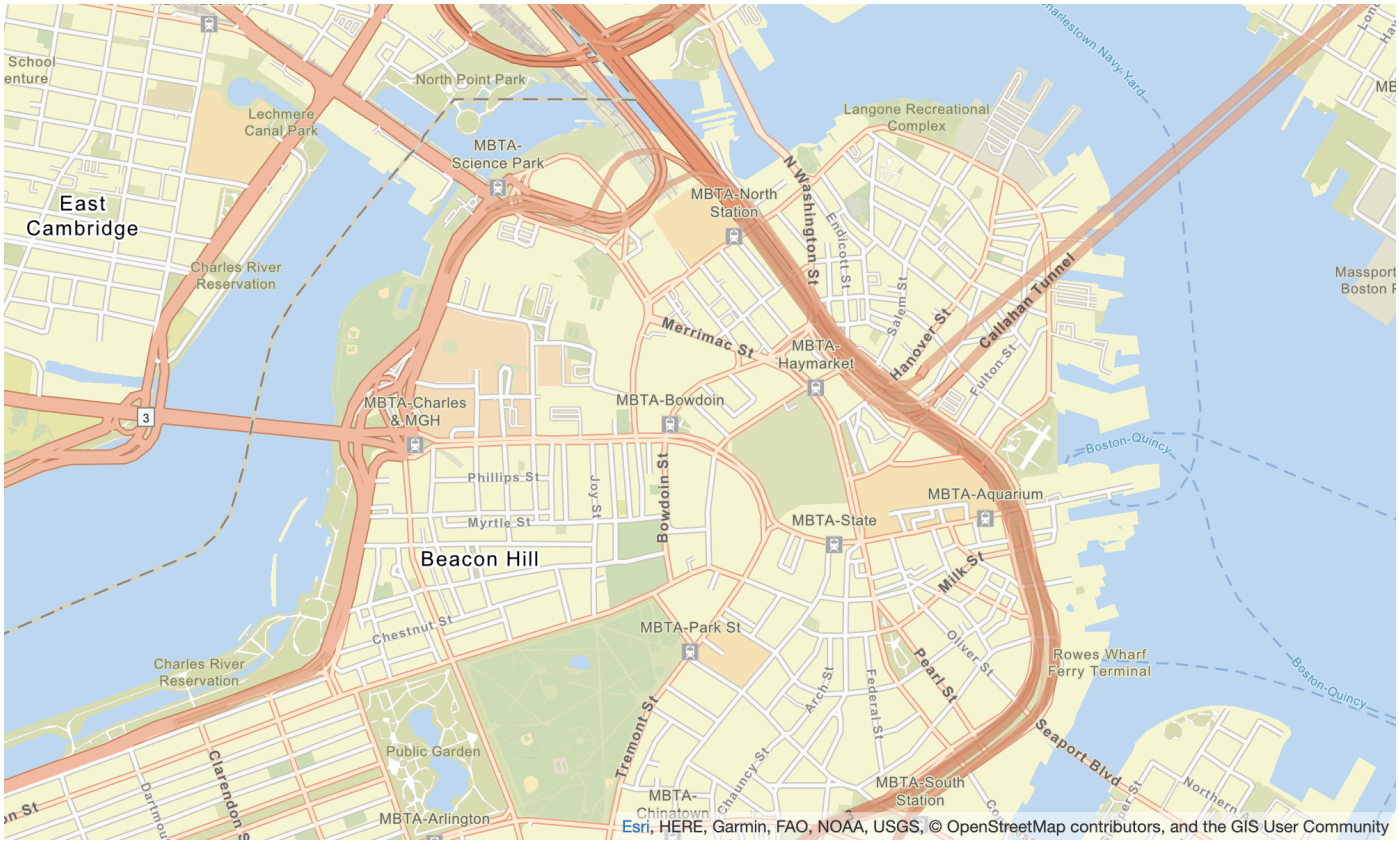
Fonts

Amazon Location serves fonts using `GetMapGlyphs`. The following are available font stacks for this map:

- Ubuntu Italic
- Ubuntu Regular
- Ubuntu Light
- Ubuntu Bold

Esri Street Map

Esri Street Map



Map style name: `VectorEsriStreets`

This map provides a detailed basemap for the world symbolized with a custom navigation map style that's designed for use during the day in mobile devices.

This comprehensive street map includes highways, major roads, minor roads, railways, water features, cities, parks, landmarks, building footprints, and administrative boundaries. It also includes a richer set of places, such as shops, services, restaurants, attractions, and other points

of interest. The vector tile layer in this map is built using the same data sources used for the World Street Map and other Esri basemaps.

For more information, see Esri World Street on the Esri website.

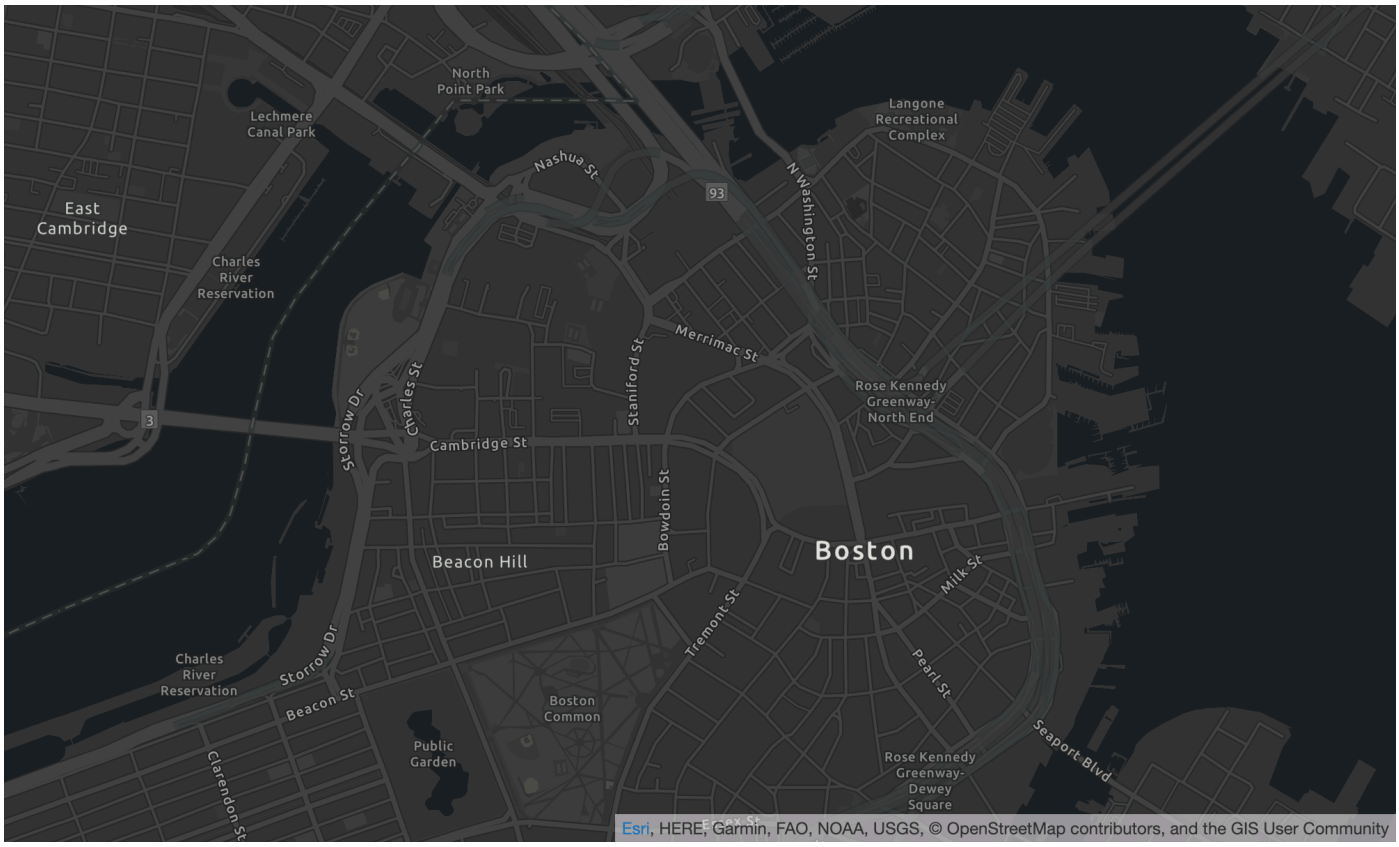
Fonts

Amazon Location serves fonts using `GetMapGlyphs`. The following are available font stacks for this map:

- Arial Italic
- Arial Regular
- Arial Bold
- Arial Unicode MS Bold
- Arial Unicode MS Regular

Esri Dark Gray Canvas

Esri Dark Gray Canvas



Map style name: `VectorEsriDarkGrayCanvas`

This map provides a detailed vector basemap for the world symbolized with a dark gray, neutral background style with minimal colors, labels, and features that's designed to draw attention to your thematic content.

This map includes highways, major roads, minor roads, railways, water features, cities, parks, landmarks, building footprints, and administrative boundaries. The vector tile layers in this map are built using the same data sources used for the Dark Gray Canvas raster map and other Esri basemaps.

For more information, see Esri Dark Gray Canvas on the Esri website.

Fonts

Amazon Location serves fonts using `GetMapGlyphs`. The following are available font stacks for this map:

- Ubuntu Medium Italic
- Ubuntu Medium
- Ubuntu Italic
- Ubuntu Regular
- Ubuntu Bold

Coverage: Esri

You can use Esri as a data provider to support queries for geocoding, reverse geocoding, and searches when you create a place index resource, or to support queries to calculate a route when you create a route calculator resource.

Esri provides different levels of data quality in different regions of the world. For additional information about coverage in your region of interest, see:

- Esri details on geocoding coverage
- Esri details on street networks and traffic coverage

Terms of use and data attribution: Esri

Before you use Esri's data, be sure you can comply with all applicable legal requirements, including license terms applicable to Esri and AWS.

For more information about the AWS requirements, see AWS Service Terms.

For information about Esri's attribution guidelines, see Esri's Data Attributions and Terms of Use.

Error reporting to Esri

If you encounter a problem with the data and want to report errors and discrepancies to Esri, follow Esri's technical support article for How to: Provide feedback on basemaps and geocoding.

GrabMaps

Grab is the largest delivery organization in Southeast Asia, with millions of driver partners and customers. Their subsidiary, GrabMaps, creates up-to-date mapping data in those countries/regions for their own use, and others. Amazon Location Service uses GrabMaps' location services to help AWS customers use maps, geocode, and calculate routes effectively. GrabMaps' location services are built to provide high-quality, authoritative, and ready-to-use location data, specifically for southeast Asian countries.

For information about additional capability, see GrabMaps on *Amazon Location Service data providers*.

> **Important**
>
> Grab provides maps only for southeast Asia, and is available only in the Asia Pacific (Singapore) Region (ap-southeast-1). For more information, see Countries/regions and area covered.

Topics

- Grab map styles
- Coverage: Grab
- Countries/regions and area covered
- Terms of use and data attribution: Grab

- Error reporting for GrabMaps data

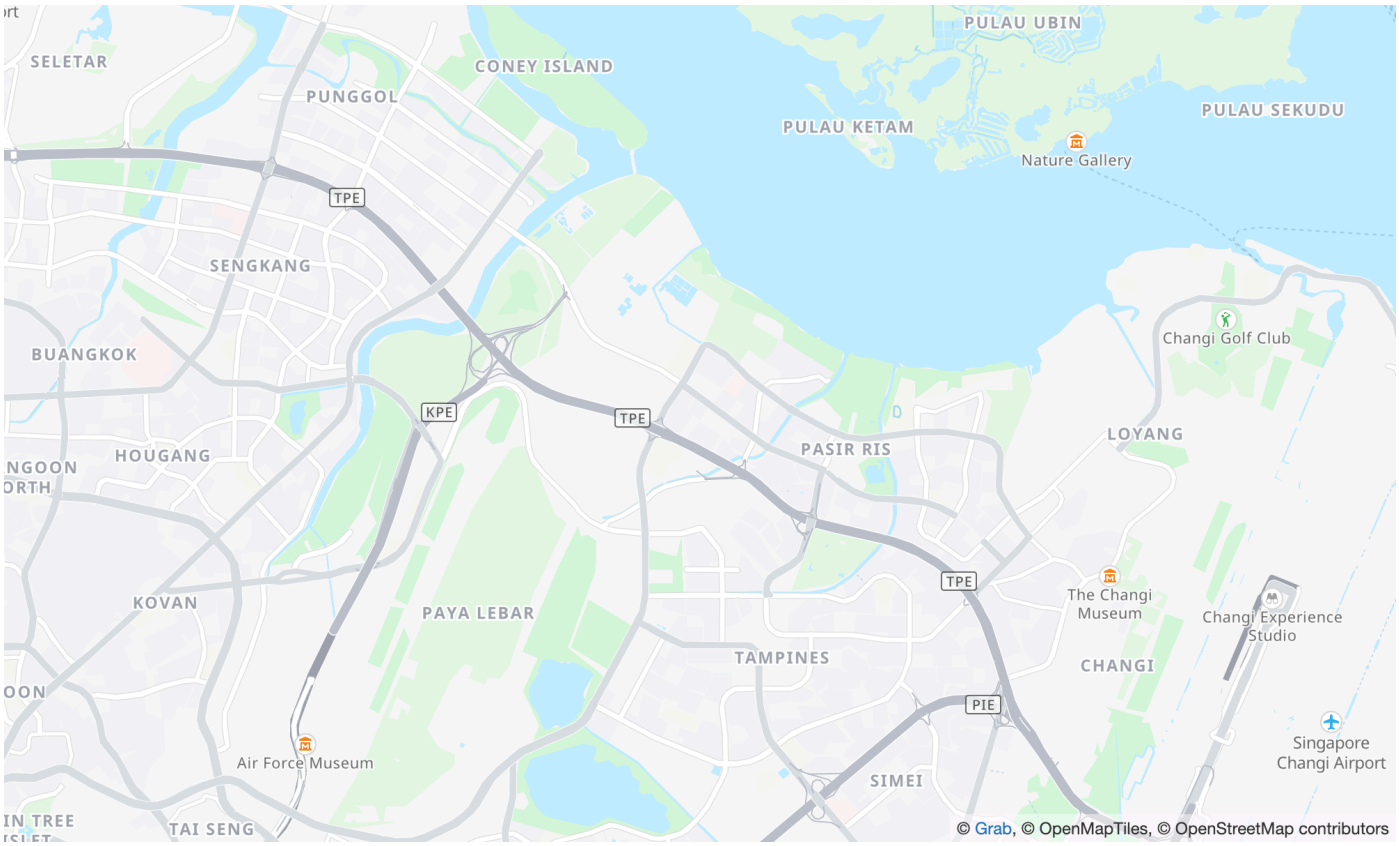
Grab map styles

Amazon Location Service supports the following Grab map styles when creating a map resource:

> **Note**
>
> Grab map styles that are not listed in this section are currently not supported.

Grab Standard Light Map

Grab Standard Light Map



Map style name: `VectorGrabStandardLight`

Grab's standard basemap with detailed land use coloring, area names, roads, landmarks, and points of interest covering Southeast Asia.

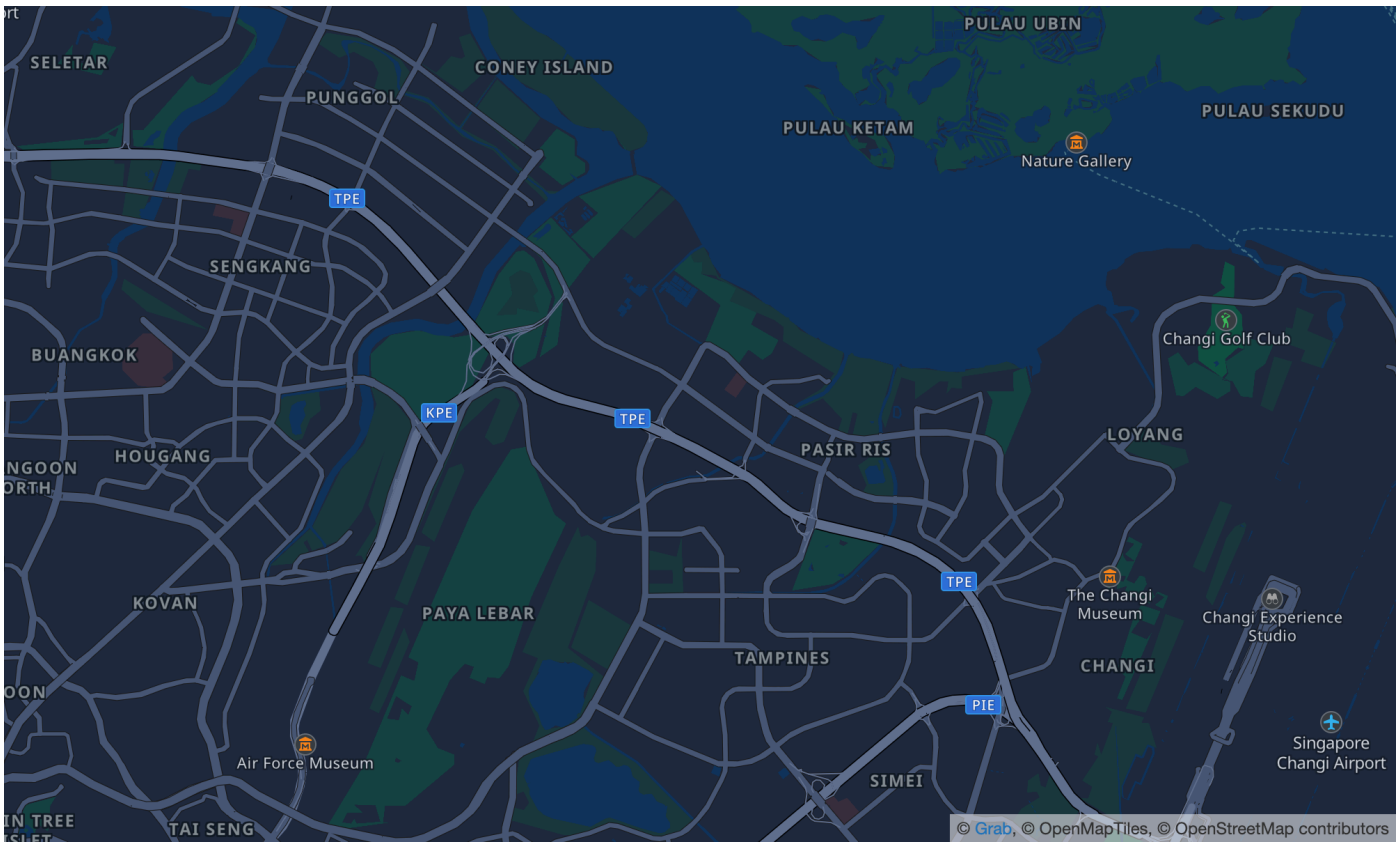
Fonts

Amazon Location serves fonts using `GetMapGlyphs`. The following are available font stacks for this map:

- Noto Sans Regular
- Noto Sans Medium
- Noto Sans Bold

Grab Standard Dark Map

Grab Standard Dark Map



Map style name: `VectorGrabStandardDark`

Grab's dark variation of their standard basemap, with detailed land use coloring, area names, roads, landmarks, and points of interest covering Southeast Asia.

Fonts

Amazon Location serves fonts using `GetMapGlyphs`. The following are available font stacks for this map:

- Noto Sans Regular
- Noto Sans Medium
- Noto Sans Bold

Coverage: Grab

You can use Grab as a data provider to support queries for geocoding, reverse geocoding, and searches when you create a place index resource, or to support queries to calculate a route when you create a route calculator resource.

Countries/regions and area covered

Grab provides maps only for southeast Asia, and is only available in the Asia Pacific (Singapore) Region (`ap-southeast-1`).

Grab provides detailed data for the following countries/regions:

- Malaysia
- Philippines
- Thailand
- Singapore
- Vietnam
- Indonesia
- Myanmar
- Cambodia

> **Note**
>
> Outside of these areas, the Amazon Location Service resources created with Grab as a data provider will not provide any results. This includes search results or routes.

The maps from Grab are within the following boundaries:

- **South** – Latitude -21.943045533438166

- **West** – Longitude 90.0
- **North** – Latitude 31.952162238024968
- **East** – Longitude 146.25

For zoom levels 1–4, Grab includes global coverage. For zoom levels 5 and below, map tiles are provided only within this bounded box.

> **Note**
>
> Outside of this bounded box, the Amazon Location Service map resources created with Grab as a data provider will not return map tiles. To avoid seeing 404 errors in your application, you can limit the map with a bounding box, as described in Setting extents for a map using MapLibre.

Grab routing travel modes

For routing, Grab provides **car** and **motorcycle** routing for all of the previously listed countries/regions.

Grab does not support **truck** routing.

For **bicycle** and **walking** routes, Grab supports the following cities:.

- Singapore
- Jakarta
- Manila
- Klang Valley
- Bangkok
- Ho Chi Minh City
- Hanoi

Terms of use and data attribution: Grab

When using Grab's data, you must comply with all applicable legal requirements, including license terms applicable to Grab and AWS.

For more information about the AWS requirements, see AWS Service Terms.

For information about GrabMaps' attribution guidelines, see Section 9.23 of Grab's Data Attributions and Terms of Use.

Error reporting for GrabMaps data

If you encounter a problem with the data from GrabMaps, and want to report errors or discrepancies, contact AWS technical support.

HERE Technologies

Amazon Location Service uses HERE Technologies' location services to help AWS customers use maps, geocode, and calculate routes effectively. HERE's location data offers a location-centric platform that's open, secure, and private. By selecting HERE location data, you are selecting accurate, fresh, and robust data that's deployed natively on the AWS Cloud.

For additional capability information, see HERE on *Amazon Location Service data providers*.

Topics

- HERE map styles
- Coverage: HERE
- Terms of use and data attribution: HERE
- Error reporting to HERE

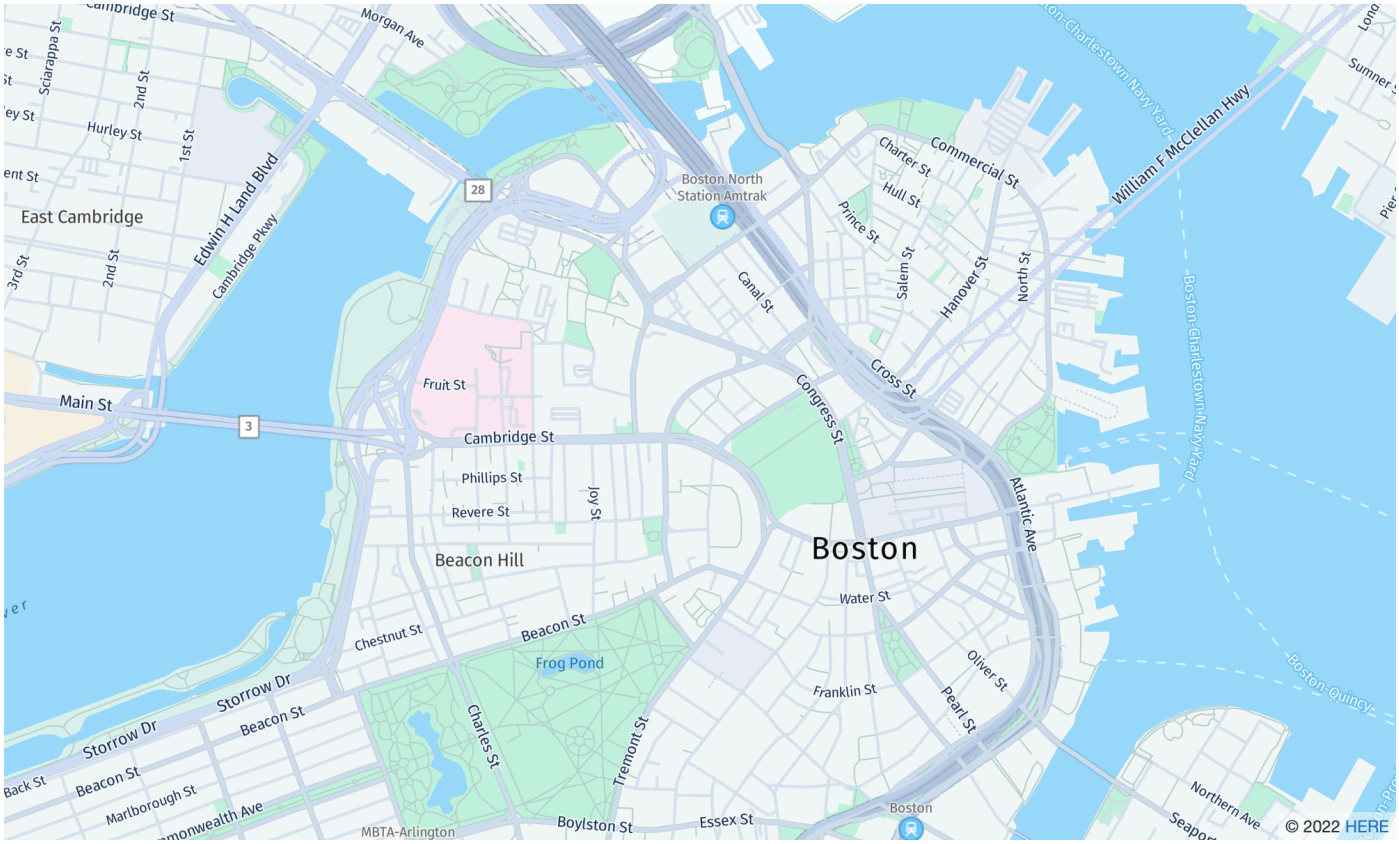
HERE map styles

Amazon Location Service supports the following HERE map styles when creating a map resource:

> **Note**
>
> HERE map styles that are not listed in this section are currently not supported.

HERE Explore

HERE Explore



Map style name: `VectorHereExplore`

HERE Explore

A detailed, neutral base map of the world. The street map includes highways, major roads, minor roads, railways, water features, cities, parks, landmarks, building footprints, and administrative boundaries. Includes a fully designed map of Japan.

Fonts

Amazon Location serves fonts using `GetMapGlyphs`. The following are available font stacks for this map:

- Fira GO Italic
- Fira GO Regular
- Fira GO Bold
- Noto Sans CJK JP Light
- Noto Sans CJK JP Regular
- Noto Sans CJK JP Bold

HERE Imagery

HERE Imagery

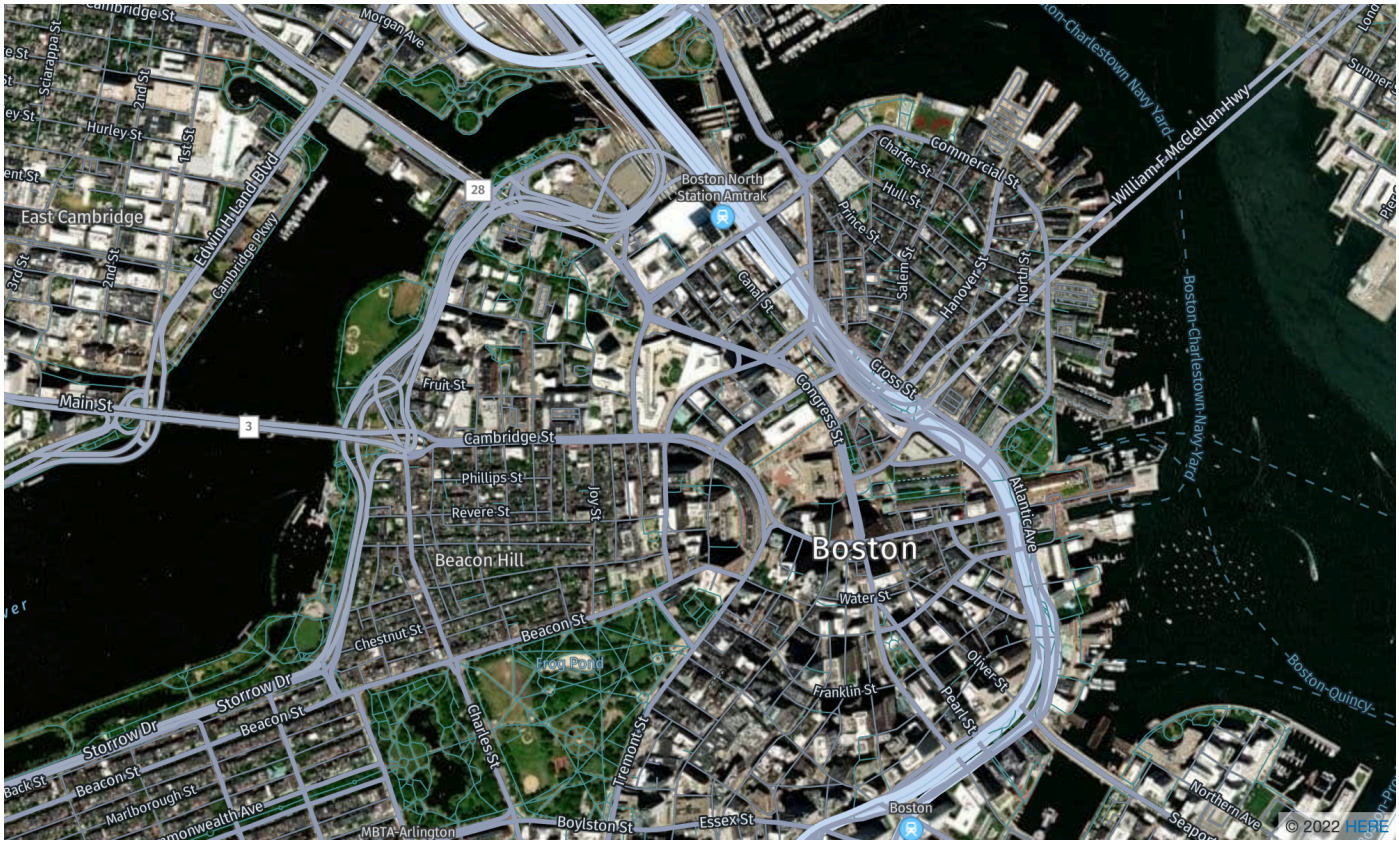
Map style name: `RasterHereExploreSatellite`

HERE Imagery

HERE Imagery provides high resolution satellite imagery with global coverage.

HERE Hybrid

HERE Hybrid



Map style name: `HybridHereExploreSatellite`

HERE Hybrid

HERE Hybrid style displays the road network, street names, and city labels over satellite imagery. This style overlays two map tiles: the satellite image (raster tile) in the background and the road network and labels (vector tile) on top. This style will automatically retrieve both the raster and vector tiles required to render the map.

> **Note**
>
> Hybrid styles use both vector and raster tiles when rendering the map that you see. This means that more tiles are retrieved than when using either vector or raster tiles alone. Your charges will include all tiles retrieved.

Fonts

Amazon Location serves fonts using `GetMapGlyphs`. The following are available font stacks for this map:

- Fira GO Italic
- Fira GO Regular
- Fira GO Bold
- Noto Sans CJK JP Light
- Noto Sans CJK JP Regular
- Noto Sans CJK JP Bold

HERE Contrast (Berlin)

HERE Contrast (Berlin)

Map style name: `VectorHereContrast`

HERE Contrast (Berlin)

A detailed base map of the world that blends 3D and 2D rendering. The high contrast street map includes highways, major roads, minor roads, railways, water features, cities, parks, landmarks, building footprints, and administrative boundaries.

Fonts

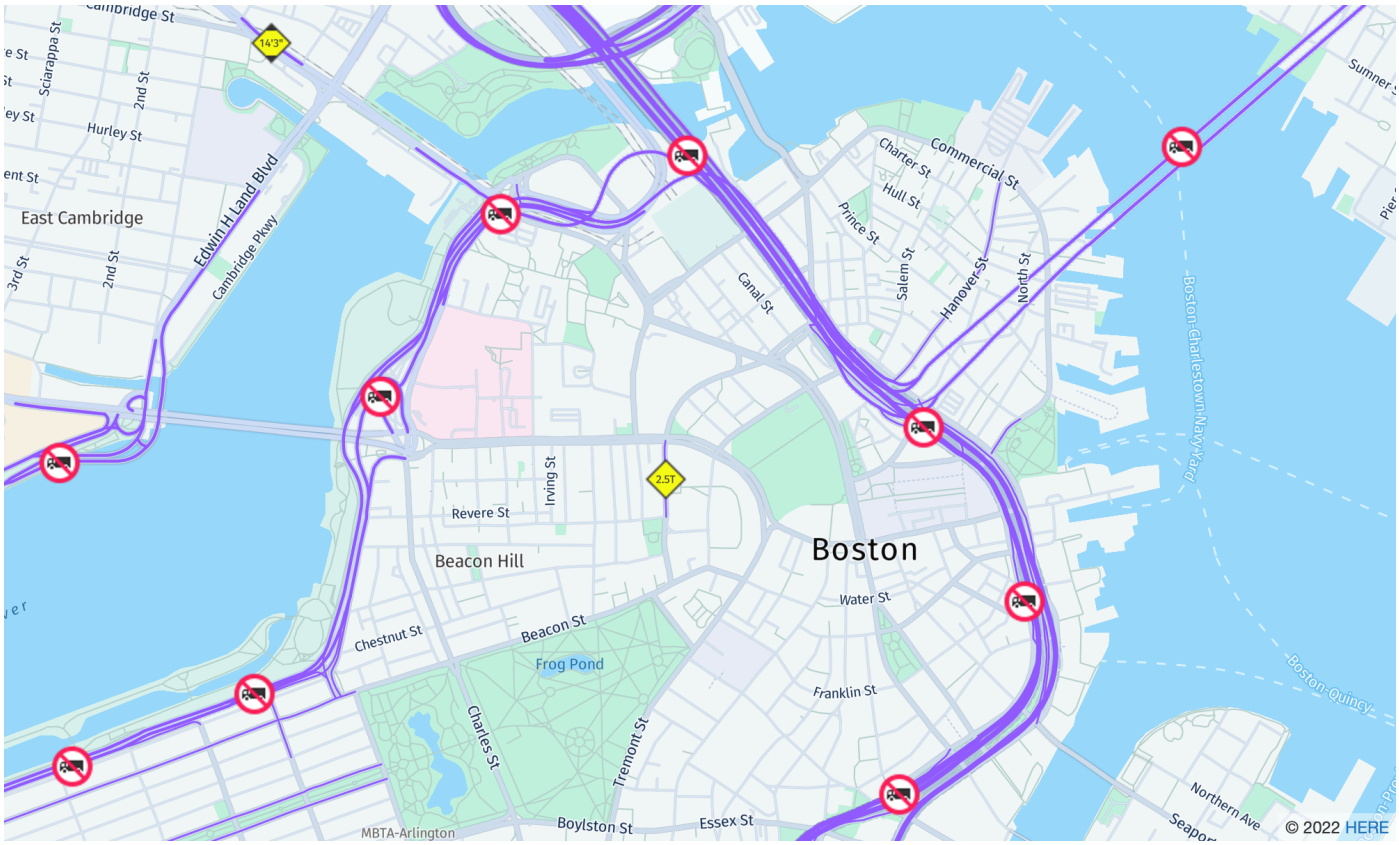
Amazon Location serves fonts using `GetMapGlyphs`. The following are available font stacks for this map:

- Fira GO Regular
- Fira GO Bold

> **Note**
>
> This style was renamed from `VectorHereBerlin` (HERE Berlin maps). `VectorHereBerlin` is deprecated, but will continue to work in applications that use it.

HERE Explore Truck

HERE Explore Truck



Map style name: `VectorHereExploreTruck`

HERE Explore Truck

A detailed, neutral base map of the world. The street map builds on top of the HERE Explore style, and highlights track restrictions and attributes (including width, height, and HAZMAT) with symbols and icons, to support use cases within transport and logistics.

Fonts

Amazon Location serves fonts using `GetMapGlyphs`. The following are available font stacks for this map:

- Fira GO Italic
- Fira GO Regular
- Fira GO Bold
- Noto Sans CJK JP Light
- Noto Sans CJK JP Regular
- Noto Sans CJK JP Bold

For additional information about map data quality in different regions of the world, see HERE map coverage.

Coverage: HERE

You can use HERE as a data provider to support queries for geocoding, reverse geocoding, and searches when you create a place index resource, or to support queries to calculate a route when you create a route calculator resource.

HERE provides different levels of data quality in different regions of the world. For additional information about coverage in your region of interest, see the following:

- HERE geocoding coverage
- HERE car routing coverage
- HERE truck routing coverage

Terms of use and data attribution: HERE

Before you use HERE data, be sure you can comply with all applicable legal requirements, including license terms applicable to HERE and AWS. Because of licensing limitations, you may not use HERE to store geocoding results for locations in Japan.

For information about the AWS requirements, see AWS Service Terms.

For additional information about HERE's attribution guidelines, see Section 2 of HERE Technologies' Supplier Terms Applicable to Location and Other Content.

Error reporting to HERE

To report map errors and discrepancies to HERE, go to https://www.here.com/contact and choose **Report a map error**.

Open Data

Amazon Location Service provides access to open source map data via the Open Data provider. Open Data provides global basemaps built from the Daylight map distribution of OpenStreetMap (OSM), Natural Earth, and other open data sources. The maps provided are designed to support different applications and use cases, including logistics and delivery, and data visualization in web and mobile environments. With over a million map makers, the OSM community updates hundreds of thousands of features per day. Amazon Location Service regularly incorporates these edits.

For additional capability information, see Open Data on *Amazon Location Service data providers*.

Topics

- Open Data map styles
- Coverage: Open Data
- Terms of use and data attribution: Open Data
- Error reporting and contributing to Open Data

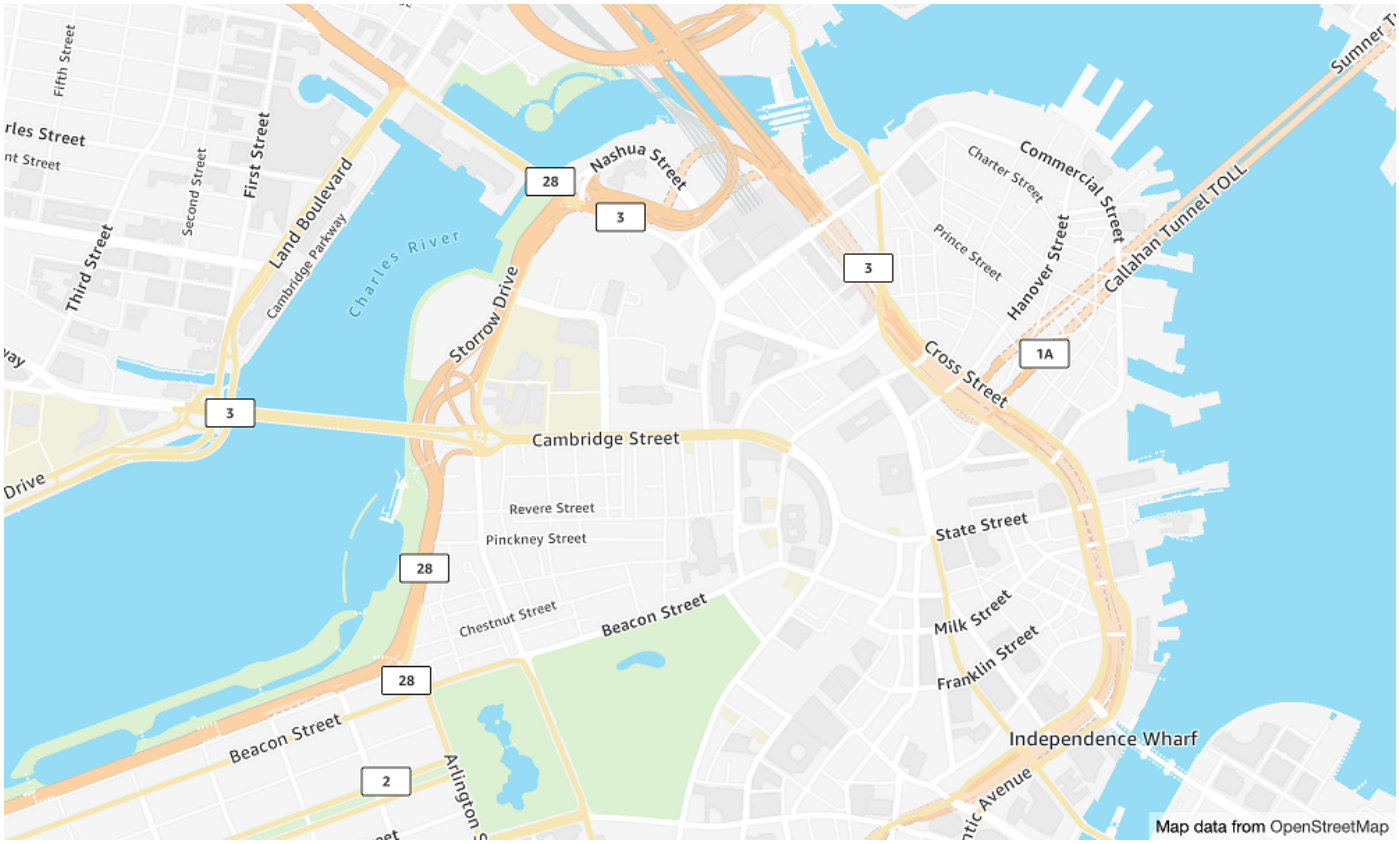
Open Data map styles

Amazon Location Service supports the following map styles when creating a map resource:

Open Data map styles support alternate Political views.

Open Data Standard Light

Open Data Standard Light



Map style name: `VectorOpenDataStandardLight`

This provides a detailed basemap for the world in a light map style, suitable for website and mobile application use. This includes highways, major roads, minor roads, railways, water features, cities, parks, landmarks, building footprints, and administrative boundaries.

This basemap is based on the OSM Daylight map distribution compiled from OpenStreetMap (OSM) contributors. The OSM community includes over 1.8 million contributors who update more than 500,000 features daily. Amazon Location Service incorporates these edits on a regular basis.

Fonts

Amazon Location serves fonts using `GetMapGlyphs`. The following are available font stacks for this map:

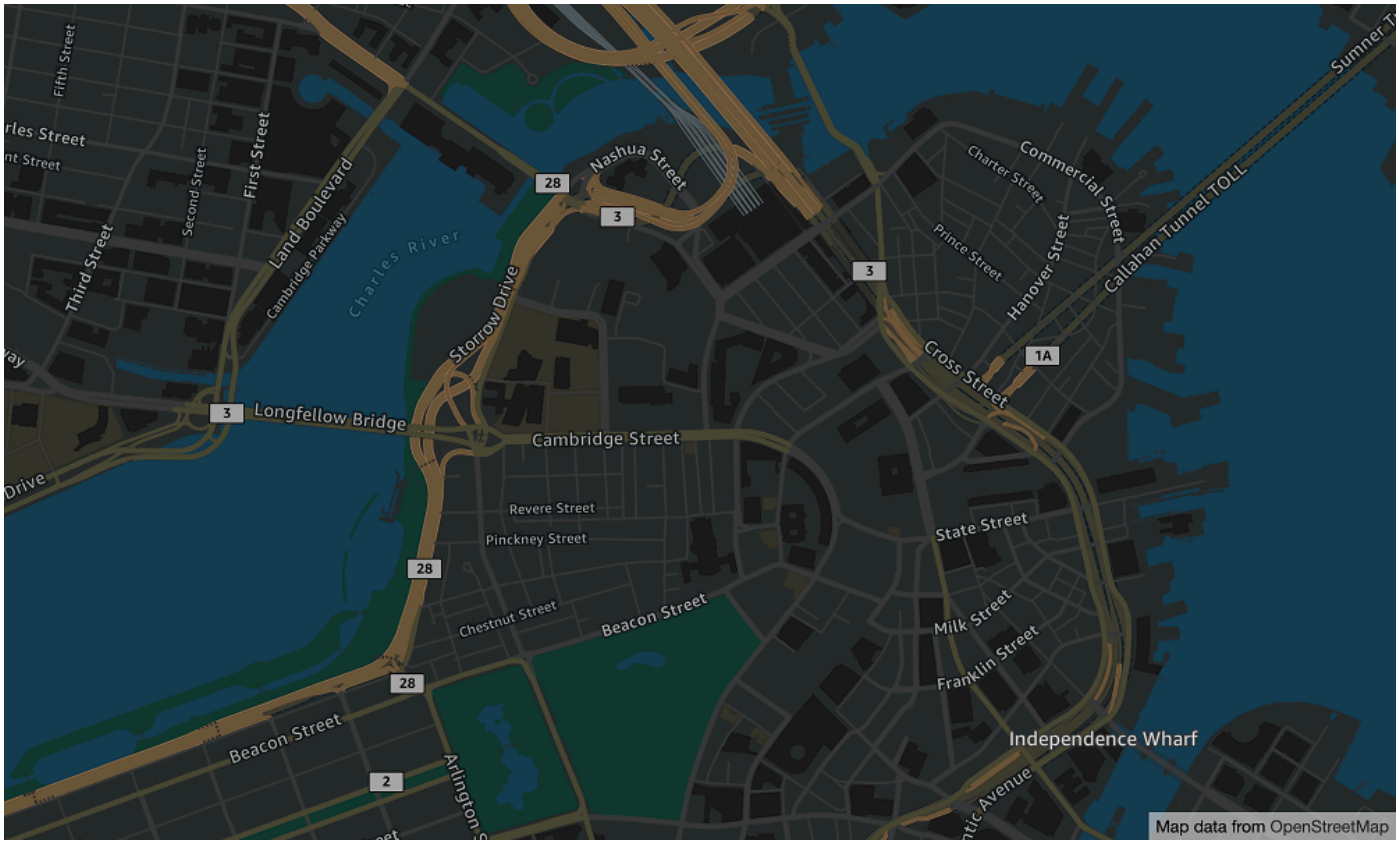
- Amazon Ember Bold,Noto Sans Bold

- Amazon Ember Condensed RC Bold,Noto Sans Bold
- Amazon Ember Condensed RC Regular,Noto Sans Regular
- Amazon Ember Medium,Noto Sans Medium
- Amazon Ember Regular Italic,Noto Sans Italic
- Amazon Ember Regular,Noto Sans Regular
- Amazon Ember Regular,Noto Sans Regular,Noto Sans Arabic Regular
- Amazon Ember Condensed RC Bold,Noto Sans Bold,Noto Sans Arabic Condensed Bold
- Amazon Ember Bold,Noto Sans Bold,Noto Sans Arabic Bold
- Amazon Ember Regular Italic,Noto Sans Italic,Noto Sans Arabic Regular
- Amazon Ember Condensed RC Regular,Noto Sans Regular,Noto Sans Arabic Condensed Regular
- Amazon Ember Medium,Noto Sans Medium,Noto Sans Arabic Medium

> **Note**
>
> The fonts used by `VectorOpenDataStandardLight` are combined fonts that use `Amazon Ember` for most glyphs but `Noto Sans` for glyphs unsupported by `Amazon Ember`.

Open Data Standard Dark

Open Data Standard Dark



Map style name: `VectorOpenDataStandardDark`

This is a dark-themed map style that provides a detailed basemap for the world, suitable for website and mobile application use. This includes highways, major roads, minor roads, railways, water features, cities, parks, landmarks, building footprints, and administrative boundaries.

This basemap is based on the OSM Daylight map distribution compiled from OpenStreetMap (OSM) contributors. The OSM community includes over 1.8 million contributors who update more than 500,000 features daily. Amazon Location Service incorporates these edits on a regular basis.

Fonts

Amazon Location serves fonts using `GetMapGlyphs`. The following are available font stacks for this map:

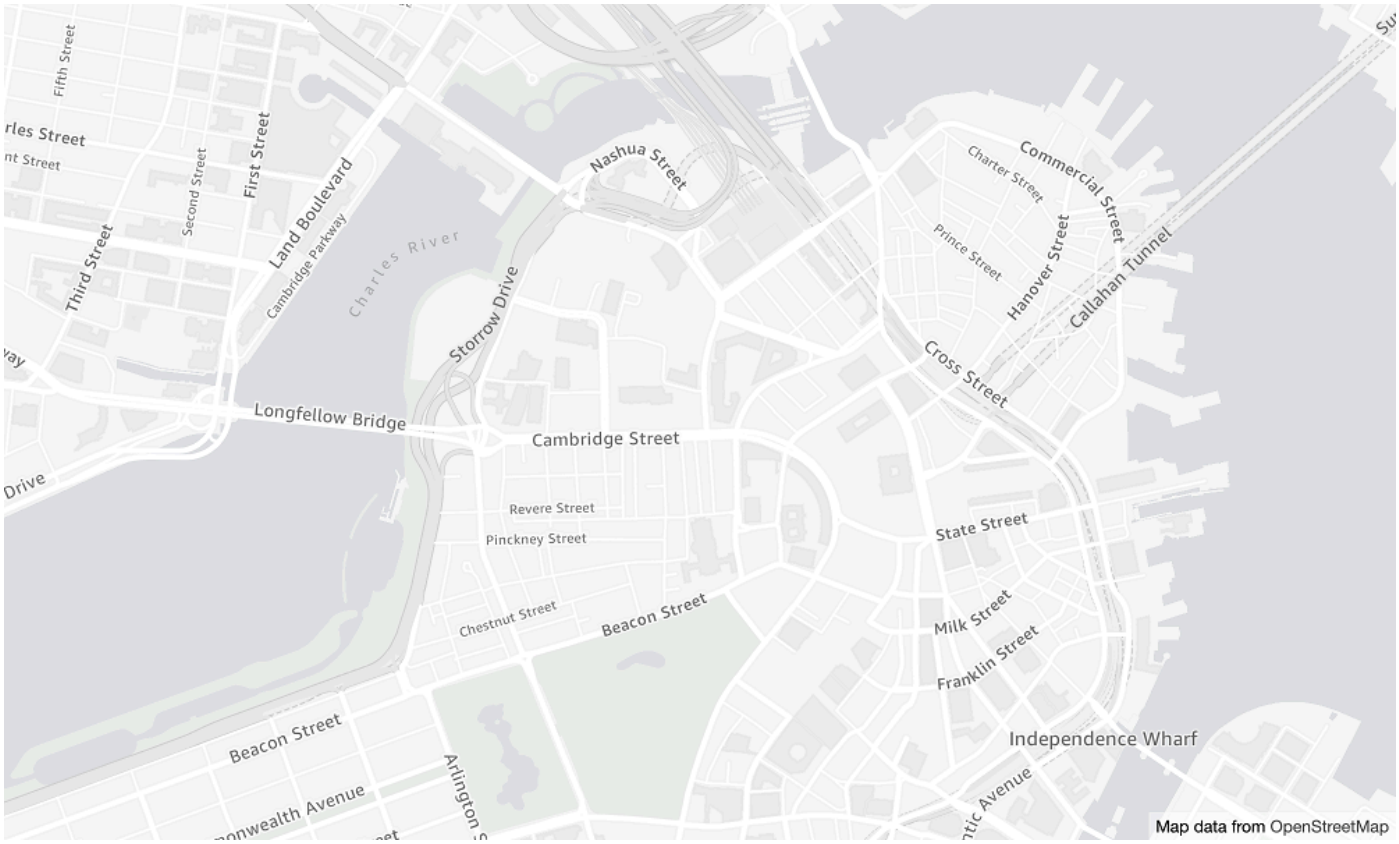
- Amazon Ember Bold,Noto Sans Bold
- Amazon Ember Condensed RC Bold,Noto Sans Bold
- Amazon Ember Condensed RC Regular,Noto Sans Regular

- Amazon Ember Medium,Noto Sans Medium
- Amazon Ember Regular Italic,Noto Sans Italic
- Amazon Ember Regular,Noto Sans Regular
- Amazon Ember Regular,Noto Sans Regular,Noto Sans Arabic Regular
- Amazon Ember Condensed RC Bold,Noto Sans Bold,Noto Sans Arabic Condensed Bold
- Amazon Ember Bold,Noto Sans Bold,Noto Sans Arabic Bold
- Amazon Ember Regular Italic,Noto Sans Italic,Noto Sans Arabic Regular
- Amazon Ember Condensed RC Regular,Noto Sans Regular,Noto Sans Arabic Condensed Regular
- Amazon Ember Medium,Noto Sans Medium,Noto Sans Arabic Medium

> **Note**
>
> The fonts used by `VectorOpenDataStandardDark` are combined fonts that use `Amazon Ember` for most glyphs but `Noto Sans` for glyphs unsupported by `Amazon Ember`.

Open Data Visualization Light

Open Data Visualization Light



Map style name: `VectorOpenDataVisualizationLight`

This is a light-themed style with muted colors and fewer features that aids in understanding overlaid data.

This basemap is based on the OSM Daylight map distribution compiled from OpenStreetMap (OSM) contributors. The OSM community includes over 1.8 million contributors who update more than 500,000 features daily. Amazon Location Service incorporates these edits on a regular basis.

Fonts

Amazon Location serves fonts using `GetMapGlyphs`. The following are available font stacks for this map:

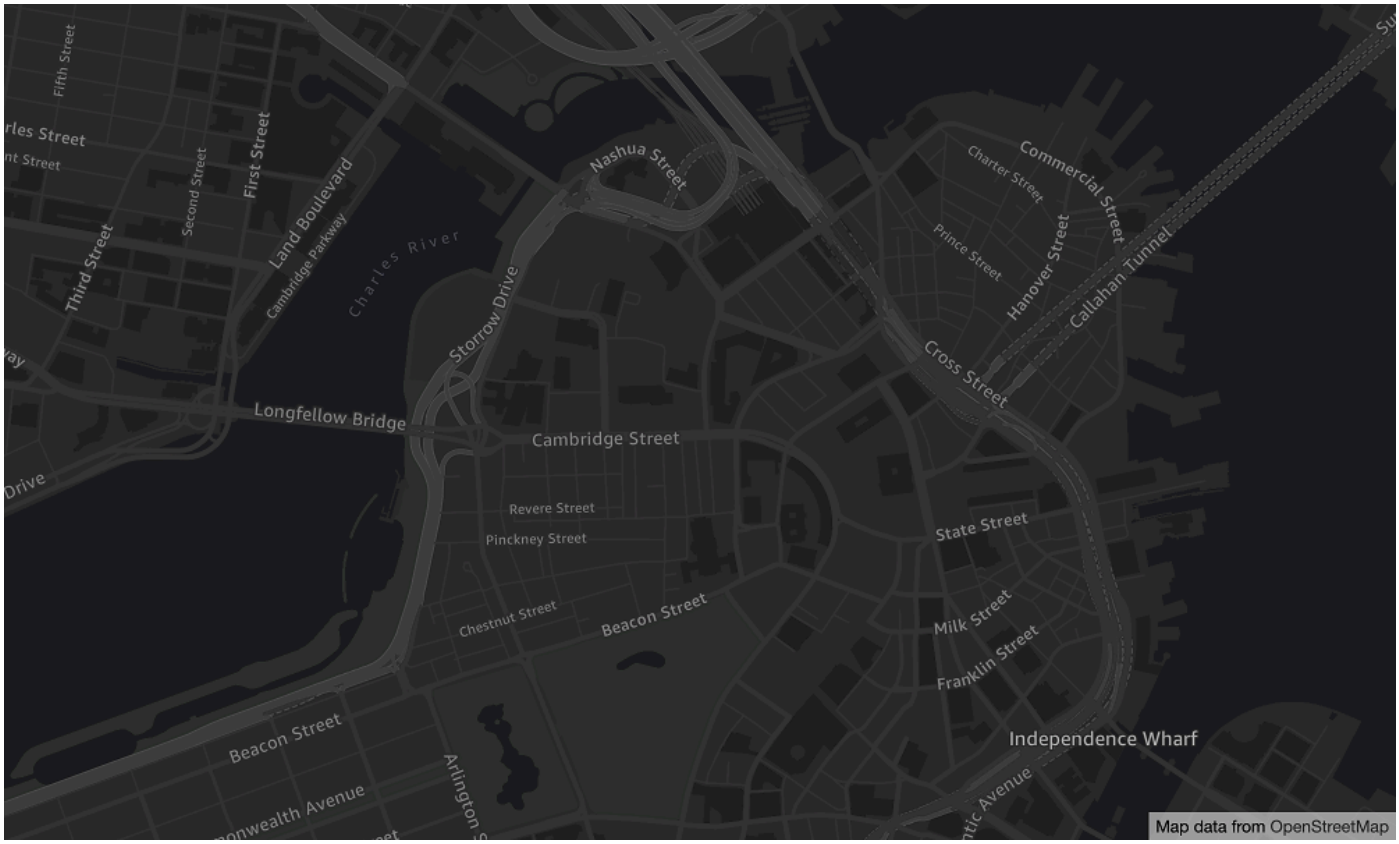
- Amazon Ember Bold,Noto Sans Bold
- Amazon Ember Condensed RC Bold,Noto Sans Bold
- Amazon Ember Condensed RC Regular,Noto Sans Regular
- Amazon Ember Medium,Noto Sans Medium

- Amazon Ember Regular Italic,Noto Sans Italic
- Amazon Ember Regular,Noto Sans Regular
- Amazon Ember Regular,Noto Sans Regular,Noto Sans Arabic Regular
- Amazon Ember Condensed RC Bold,Noto Sans Bold,Noto Sans Arabic Condensed Bold
- Amazon Ember Bold,Noto Sans Bold,Noto Sans Arabic Bold
- Amazon Ember Regular Italic,Noto Sans Italic,Noto Sans Arabic Regular
- Amazon Ember Condensed RC Regular,Noto Sans Regular,Noto Sans Arabic Condensed Regular
- Amazon Ember Medium,Noto Sans Medium,Noto Sans Arabic Medium

> **Note**
>
> The fonts used by `VectorOpenDataVisualizationLight` are combined fonts that use `Amazon Ember` for most glyphs but `Noto Sans` for glyphs unsupported by `Amazon Ember`.

Open Data Visualization Dark

Open Data Visualization Dark



Map style name: `VectorOpenDataVisualizationDark`

This is a dark-themed style with muted colors and fewer features that aids in understanding overlaid data.

This basemap is based on the OSM Daylight map distribution compiled from OpenStreetMap (OSM) contributors. The OSM community includes over 1.8 million contributors who update more than 500,000 features daily. Amazon Location Service incorporates these edits on a regular basis.

Fonts

Amazon Location serves fonts using `GetMapGlyphs`. The following are available font stacks for this map:

- Amazon Ember Bold,Noto Sans Bold
- Amazon Ember Condensed RC Bold,Noto Sans Bold
- Amazon Ember Condensed RC Regular,Noto Sans Regular
- Amazon Ember Medium,Noto Sans Medium

- Amazon Ember Regular Italic,Noto Sans Italic
- Amazon Ember Regular,Noto Sans Regular
- Amazon Ember Regular,Noto Sans Regular,Noto Sans Arabic Regular
- Amazon Ember Condensed RC Bold,Noto Sans Bold,Noto Sans Arabic Condensed Bold
- Amazon Ember Bold,Noto Sans Bold,Noto Sans Arabic Bold
- Amazon Ember Regular Italic,Noto Sans Italic,Noto Sans Arabic Regular
- Amazon Ember Condensed RC Regular,Noto Sans Regular,Noto Sans Arabic Condensed Regular
- Amazon Ember Medium,Noto Sans Medium,Noto Sans Arabic Medium

> **Note**
>
> The fonts used by `VectorOpenDataVisualizationDark` are combined fonts that use `Amazon Ember` for most glyphs but `Noto Sans` for glyphs unsupported by `Amazon Ember`.

Coverage: Open Data

Open Data includes maps with global coverage for rendering with an Amazon Location Service map resource.

> **Note**
>
> Open Data is for use with Amazon Location Service map resources only. You can't use Open Data as a data provider to support queries for geocoding, reverse geocoding, and searches, or to support queries to calculate a route.

Terms of use and data attribution: Open Data

Before you use Open Data, be sure you can comply with all applicable legal requirements, including license terms applicable to Open Data and AWS.

For more information about the AWS requirements, see AWS Service Terms.

For information about Open Data attribution guidelines, see OpenStreetMap's Copyright and License and OpenStreetMap's Licence/Attribution Guidelines.

Error reporting and contributing to Open Data

OpenStreetMap (OSM) and Natural Earth are community-driven open data projects. If you encounter a problem with the data, you can report the errors or directly contribute fixes or suggestions.

- To report an error or offer a suggestion in OSM, you can create a *note* on the map. This is a comment on the map that assists contributors in making fixes to the map. You create notes through the OpenStreetMap website. For more information about notes, see Notes in the *OpenStreetMap wiki*.
- For more information about contributing directly to OpenStreetMap, including adding locations and fixing errors, see Contribute map data in the *OpenStreetMap wiki*.
- To submit a correction request for data in Natural Earth, you can submit an issue through the Natural Earth website.

> **Note**
>
> Correcting errors in OpenStreetMap can happen quickly, however, it can take time for corrections to appear in the Daylight map distribution of the OSM data that is used by the Open Data provider. The Daylight Map Distribution website provides more information about the process. Additionally, Amazon Location Service updates the map data used in Amazon Location Service approximately monthly.

Features by data provider

This section describes the features available in Amazon Location Service, categorized by data provider.

The following table provides a high-level overview of the features.

Data provider	Geographical coverage	Feature coverage	AWS Region
Esri	Global	Maps, Places, Routes	All Regions where Amazon Location is available.
Grab	Southeast Asia	Maps, Places, Routes	Asia Pacific (Singapore), `ap-southeast-1` , only.
HERE	Global	Maps, Places, Routes	All Regions where Amazon Location is available.
Open Data	Global	Maps	All Regions where Amazon Location is available.

The following tabs show details within each feature area.

Map Features

The following table shows the map features by data provider. For more information about map concepts, see Maps.

Data provider	Supported map types	Vector zoom levels	Raster zoom levels
Esri	Vector Raster (imagery) For more information, see Esri map styles.	0-15	0-23
Grab	Vector	0-14	none

Data provider	Supported map types	Vector zoom levels	Raster zoom levels
	(Southeast Asia only) For more information, see Grab map styles.		
HERE	Vector Raster (imagery) Hybrid For more information, see HERE map styles.	1-17	0-19
Open Data	Vector For more information, see Open Data map styles.	0-15	none

> **Note**
>
> Zoom levels represent the maximum and minimum settings, as defined in each provider's APIs. Different areas of the map may have different maximums; for example, ocean tiles may have fewer detailed zoom levels than areas in major cities. MapLibre (and other map rendering engines) allow you to set minimum and maximum zoom levels, and will also honor the data provider zoom levels in an area, so you do not have to write code to handle these discrepancies.

Places and Search

The following table shows the place and search features by data provider. For more information about place concepts, see Places search.

Data provider	Geocoding	Reverse Geocoding	Autocomplete	GetPlace
Esri	All features, except: *PlaceId*	All features, except: *TimeZone* *PlaceId*	All features	All features
Grab	All features, except: *unit type* Categories not supported	All features	All features	All features, except: *unit type* *SubMunicipality*
HERE	All features, except: *unit number* *unit type* *relevance* Additional limitations on filtering	All features	All features	All features, except: *unit number* *unit type* *SubMunicipality*
Open Data	Not supported	Not supported	Not supported	Only supports: *SubMunicipality*

Route features

The following table shows the route features by data provider. For more information about route concepts, see Routes. For more detailed descriptions of route matrix limitations, see Restrictions on departure and destination positions.

Data provider	Travel modes	Calculate route	Route matrix
Esri	Car, Truck, Walking	Departure and destination must be within 400 km of each other. The total travel time can't be more than 400 minutes. ArrivalTime is not supported.	Up to 10 departure and destination positions. Not supported in Korea. All departure and destination pairs must be within 400 km of each other.
Grab	Car, Motorcycle. Walking and Bicycle in selected cities.	No distance limits.	Up to 350 departure and destination positions.
HERE	Car, Truck, Walking	No distance limit. Routes that go more than 10 km outside a circle around the departure and destination positions will not be calculated.	Up to 350 departure and destination positions. All departure and destination positions must fall with a 180 km circle. Longer routes are supported, with additional restrictions.

Data provider	Travel modes	Calculate route	Route matrix
Open Data	Not supported	Not supported	Not supported

Terms of use and data attribution for data providers

Before you use a data provider, be sure you can comply with all applicable legal requirements, including license terms applicable to the use of the provider.

For more information about the AWS requirements, see AWS Service Terms.

When using a data provider with your Amazon Location resources for your application or documentation, be sure to provide attributions for each data provider you use.

For more information on compliance and attribution for each data provider, see the following topics.

- **Esri** – Terms of use and data attribution: Esri
- **Grab** – Terms of use and data attribution: Grab
- **HERE** – Terms of use and data attribution: HERE
- **Open data** – Terms of use and data attribution: Open Data

Amazon Location Regions and endpoints

Amazon Location is available in the following AWS Regions:

Regions

Region Name	Region	Endpoint	Protocol	
US East (Ohio)	us-east-2	geo.us-east-2.amazonaws.com	HTTPS	
US East (N. Virginia)	us-east-1	geo.us-east-1.amazonaws.com	HTTPS	

Region Name	Region	Endpoint	Protocol	
US West (Oregon)	us-west-2	geo.us-west-2.amazonaws.com	HTTPS	
Asia Pacific (Mumbai)	ap-south-1	geo.ap-south-1.amazonaws.com	HTTPS	
Asia Pacific (Singapore)	ap-southeast-1	geo.ap-southeast-1.amazonaws.com	HTTPS	
Asia Pacific (Sydney)	ap-southeast-2	geo.ap-southeast-2.amazonaws.com	HTTPS	
Asia Pacific (Tokyo)	ap-northeast-1	geo.ap-northeast-1.amazonaws.com	HTTPS	
Canada (Central)	ca-central-1	geo.ca-central-1.amazonaws.com	HTTPS	
Europe (Frankfurt)	eu-central-1	geo.eu-central-1.amazonaws.com	HTTPS	
Europe (Ireland)	eu-west-1	geo.eu-west-1.amazonaws.com	HTTPS	
Europe (London)	eu-west-2	geo.eu-west-2.amazonaws.com	HTTPS	
Europe (Stockholm)	eu-north-1	geo.eu-north-1.amazonaws.com	HTTPS	

Region Name	Region	Endpoint	Protocol
South America (São Paulo)	sa-east-1	geo.sa-east-1.amazonaws.com	HTTPS
AWS GovCloud (US-West)	us-gov-west-1	geo.us-gov-west-1.amazonaws.com geo-fips.us-gov-west-1.amazonaws.com	HTTPS HTTPS

Note

For more information about how to use the endpoints in this table, see the following section.

Endpoints

The general syntax for an Amazon Location regional endpoint is as follows:

```
protocol://service-code.geo.region-code.amazonaws.com
```

Within this syntax, Amazon Location uses the following service codes:

Service	Service code
Amazon Location Maps	maps
Amazon Location Places	places
Amazon Location Geofences	geofencing
Amazon Location Trackers	tracking
Amazon Location Routes	routes

For example, the regional endpoint for Amazon Location Maps for US East (N. Virginia) would be: https://*maps*.geo.*us-east-1*.amazonaws.com.

API operation Endpoints

The syntax for an Amazon Location Service control plane endpoint is as follows:

```
protocol://cp.service-code.geo.region-code.amazonaws.com
```

The control plane actions for Amazon Location Service are:

Service	Endpoint	API operation
Amazon Location Maps	https://cp.maps.geo.*region*.amazonaws.com	CreateMap DeleteMap DescribeMap ListMaps UpdateMap
Amazon Location Places	https://cp.places.geo.*region*.amazonaws.com	CreatePlaceIndex DeletePlaceIndex DescribePlaceIndex ListPlaceIndexes UpdatePlaceIndex
Amazon Location Geofences	https://cp.geofencing.geo.*region*.amazonaws.com	CreateGeofenceCollection DeleteGeofenceCollection DescribeGeofenceCollection ListGeofenceCollections UpdateGeofenceCollection

Service	Endpoint	API operation
Amazon Location Trackers	https://cp.tracking.geo.*region*.amazonaws.com	CreateTracker DeleteTracker DescribeTracker UpdateTracker ListTrackers AssociateTrackerConsumer DisassociateTrackerConsumer ListTrackerConsumers
Amazon Location Routes	https://cp.routes.geo.*region*.amazonaws.com	CreateRouteCalculator DeleteRouteCalculator DescribeRouteCalculator ListRouteCalculators UpdateRouteCalculator
Amazon Location Metadata	https://cp.metadata.geo.*region*.amazonaws.com	CreateKey DeleteKey DescribeKey ListKeys UpdateKey

Amazon Location Service quotas

This topic provides a summary of rate limits and quotas for Amazon Location Service.

> **Note**
>
> If you require a higher quota, you can use the Service Quotas console to request quota increases for adjustable quotas. When requesting a quota increase, select the Region you require the quota increase in since most quotas are specific to the AWS Region.

Service Quotas are maximum number of resources you can have per AWS account and AWS Region. Amazon Location Service denies additional requests that exceed the service quota.

Rate limits (quotas that start with *Rate of...*) are the maximum number of requests per second, with a burst rate of 80 percent of the limit within any part of the second, defined for each API operation. Amazon Location Service throttles requests that exceed the operation's rate limit.

Name	Default	Adjuse	Description
API Key resources per account	Each supported Region: 500	No	The maximum number of API key resources (active or expired) that you can have per account.
Geofence Collection resources per account	Each supported Region: 1,500	Yes	The maximum number of Geofence Collection resources that you can create per account.
Geofences per Geofence Collection	Each supported Region: 50,000	No	The maximum number of Geofences that you can create per Geofence Collection.
Map resources per account	Each supported Region: 40	Yes	The maximum number of Map resources that you can create per account.
Place Index resources per account	Each supported Region: 40	Yes	The maximum number of Place Index resources

Name	Default	Adjustable	Description
			that you can create per account.
Rate of AssociateTrackerConsumer API requests	Each supported Region: 10 per second	Yes	The maximum number of AssociateTrackerConsumer requests that you can make per second. Additional requests are throttled.
Rate of BatchDeleteDevicePositionHistory API requests	Each supported Region: 50 per second	Yes	The maximum number of BatchDeleteDevicePositionHistory requests that you can make per second. Additional requests are throttled.
Rate of BatchDeleteGeofence API requests	Each supported Region: 50 per second	Yes	The maximum number of BatchDeleteGeofence requests that you can make per second. Additional requests are throttled.
Rate of BatchEvaluateGeofences API requests	Each supported Region: 50 per second	Yes	The maximum number of BatchEvaluateGeofences requests that you can make per second. Additional requests are throttled.

Name	Default	Adjustable	Description
Rate of BatchGetDevicePosition API requests	Each supported Region: 50 per second	Yes	The maximum number of BatchGetDevicePosition requests that you can make per second. Additional requests are throttled.
Rate of BatchPutGeofence API requests	Each supported Region: 50 per second	Yes	The maximum number of BatchPutGeofence requests that you can make per second. Additional requests are throttled.
Rate of BatchUpdateDevicePosition API requests	Each supported Region: 50 per second	Yes	The maximum number of BatchUpdateDevicePosition requests that you can make per second. Additional requests are throttled.
Rate of CalculateRoute API requests	Each supported Region: 10 per second	Yes	The maximum number of CalculateRoute requests that you can make per second. Additional requests are throttled.
Rate of CalculateRouteMatrix API requests	Each supported Region: 5 per second	Yes	The maximum number of CalculateRouteMatrix requests that you can make per second. Additional requests are throttled.

Name	Default	Adjustable	Description
Rate of CreateGeofenceCollection API requests	Each supported Region: 10 per second	Yes	The maximum number of CreateGeofenceCollection requests that you can make per second. Additional requests are throttled.
Rate of CreateKey API requests	Each supported Region: 10 per second	Yes	The maximum number of CreateKey requests that you can make per second. Additional requests are throttled.
Rate of CreateMap API requests	Each supported Region: 10 per second	Yes	The maximum number of CreateMap requests that you can make per second. Additional requests are throttled.
Rate of CreatePlaceIndex API requests	Each supported Region: 10 per second	Yes	The maximum number of CreatePlaceIndex requests that you can make per second. Additional requests are throttled.
Rate of CreateRouteCalculator API requests	Each supported Region: 10 per second	Yes	The maximum number of CreateRouteCalculator requests that you can make per second. Additional requests are throttled.

Name	Default	Adjustable	Description
Rate of CreateTracker API requests	Each supported Region: 10 per second	Yes	The maximum number of CreateTracker requests that you can make per second. Additional requests are throttled.
Rate of DeleteGeofenceCollection API requests	Each supported Region: 10 per second	Yes	The maximum number of DeleteGeofenceCollection requests that you can make per second. Additional requests are throttled.
Rate of DeleteKey API requests	Each supported Region: 10 per second	Yes	The maximum number of DeleteKey requests that you can make per second. Additional requests are throttled.
Rate of DeleteMap API requests	Each supported Region: 10 per second	Yes	The maximum number of DeleteMap requests that you can make per second. Additional requests are throttled.
Rate of DeletePlaceIndex API requests	Each supported Region: 10 per second	Yes	The maximum number of DeletePlaceIndex requests that you can make per second. Additional requests are throttled.

Name	Default	Adjustable	Description
Rate of DeleteRouteCalculator API requests	Each supported Region: 10 per second	Yes	The maximum number of DeleteRouteCalculator requests that you can make per second. Additional requests are throttled.
Rate of DeleteTracker API requests	Each supported Region: 10 per second	Yes	The maximum number of DeleteTracker requests that you can make per second. Additional requests are throttled.
Rate of DescribeGeofenceCollection API requests	Each supported Region: 10 per second	Yes	The maximum number of DescribeGeofenceCollection requests that you can make per second. Additional requests are throttled.
Rate of DescribeKey API requests	Each supported Region: 10 per second	Yes	The maximum number of DescribeKey requests that you can make per second. Additional requests are throttled.
Rate of DescribeMap API requests	Each supported Region: 10 per second	Yes	The maximum number of DescribeMap requests that you can make per second. Additional requests are throttled.

Name	Default	Adjustable	Description
Rate of DescribePlaceIndex API requests	Each supported Region: 10 per second	Yes	The maximum number of DescribePlaceIndex requests that you can make per second. Additional requests are throttled.
Rate of DescribeRouteCalculator API requests	Each supported Region: 10 per second	Yes	The maximum number of DescribeRouteCalculator requests that you can make per second. Additional requests are throttled.
Rate of DescribeTracker API requests	Each supported Region: 10 per second	Yes	The maximum number of DescribeTracker requests that you can make per second. Additional requests are throttled.
Rate of DisassociateTrackerConsumer API requests	Each supported Region: 10 per second	Yes	The maximum number of DisassociateTrackerConsumer requests that you can make per second. Additional requests are throttled.
Rate of ForecastGeofenceEvents API requests	Each supported Region: 50 per second	Yes	The maximum number of ForecastGeofenceEvents requests that you can make per second. Additional requests are throttled.

Name	Default	Adjustable	Description
Rate of GetDevicePosition API requests	Each supported Region: 50 per second	Yes	The maximum number of GetDevicePosition requests that you can make per second. Additional requests are throttled.
Rate of GetDevicePositionHistory API requests	Each supported Region: 50 per second	Yes	The maximum number of GetDevicePositionHistory requests that you can make per second. Additional requests are throttled.
Rate of GetGeofence API requests	Each supported Region: 50 per second	Yes	The maximum number of GetGeofence requests that you can make per second. Additional requests are throttled.
Rate of GetMapGlyphs API requests	Each supported Region: 50 per second	Yes	The maximum number of GetMapGlyphs requests that you can make per second. Additional requests are throttled.
Rate of GetMapSprites API requests	Each supported Region: 50 per second	Yes	The maximum number of GetMapSprites requests that you can make per second. Additional requests are throttled.

Name	Default	Adjustable	Description
Rate of GetMapStyleDescriptor API requests	Each supported Region: 50 per second	Yes	The maximum number of GetMapStyleDescriptor requests that you can make per second. Additional requests are throttled.
Rate of GetMapTile API requests	Each supported Region: 500 per second	Yes	The maximum number of GetMapTile requests that you can make per second. Additional requests are throttled.
Rate of GetPlace API requests	Each supported Region: 50 per second	Yes	The maximum number of GetPlace requests that you can make per second. Additional requests are throttled.
Rate of ListDevicePositions API requests	Each supported Region: 50 per second	Yes	The maximum number of ListDevicePositions requests that you can make per second. Additional requests are throttled.
Rate of ListGeofenceCollections API requests	Each supported Region: 10 per second	Yes	The maximum number of ListGeofenceCollections requests that you can make per second. Additional requests are throttled.

Name	Default	Adjustable	Description
Rate of ListGeofences API requests	Each supported Region: 50 per second	Yes	The maximum number of ListGeofences requests that you can make per second. Additional requests are throttled.
Rate of ListKeys API requests	Each supported Region: 10 per second	Yes	The maximum number of ListKeys requests that you can make per second. Additional requests are throttled.
Rate of ListMaps API requests	Each supported Region: 10 per second	Yes	The maximum number of ListMaps requests that you can make per second. Additional requests are throttled.
Rate of ListPlaceIndexes API requests	Each supported Region: 10 per second	Yes	The maximum number of ListPlaceIndexes requests that you can make per second. Additional requests are throttled.
Rate of ListRouteCalculators API requests	Each supported Region: 10 per second	Yes	The maximum number of ListRouteCalculators requests that you can make per second. Additional requests are throttled.

Name	Default	Adjustable	Description
Rate of ListTagsForResource API requests	Each supported Region: 10 per second	Yes	The maximum number of ListTagsForResource requests that you can make per second. Additional requests are throttled.
Rate of ListTrackerConsumers API requests	Each supported Region: 10 per second	Yes	The maximum number of ListTrackerConsumers requests that you can make per second. Additional requests are throttled.
Rate of ListTrackers API requests	Each supported Region: 10 per second	Yes	The maximum number of ListTrackers requests that you can make per second. Additional requests are throttled.
Rate of PutGeofence API requests	Each supported Region: 50 per second	Yes	The maximum number of PutGeofence requests that you can make per second. Additional requests are throttled.
Rate of SearchPlaceIndexForPosition API requests	Each supported Region: 50 per second	Yes	The maximum number of SearchPlaceIndexForPosition requests that you can make per second. Additional requests are throttled.

Name	Default	Adjustable	Description
Rate of SearchPlaceIndexForSuggestions API requests	Each supported Region: 50 per second	Yes	The maximum number of SearchPlaceIndexForSuggestions requests that you can make per second. Additional requests are throttled.
Rate of SearchPlaceIndexForText API requests	Each supported Region: 50 per second	Yes	The maximum number of SearchPlaceIndexForText requests that you can make per second. Additional requests are throttled.
Rate of TagResource API requests	Each supported Region: 10 per second	Yes	The maximum number of TagResource requests that you can make per second. Additional requests are throttled.
Rate of UntagResource API requests	Each supported Region: 10 per second	Yes	The maximum number of UntagResource requests that you can make per second. Additional requests are throttled.
Rate of UpdateGeofenceCollection API requests	Each supported Region: 10 per second	Yes	The maximum number of UpdateGeofenceCollection requests that you can make per second. Additional requests are throttled.

Name	Default	Adjustable	Description
Rate of UpdateKey API requests	Each supported Region: 10 per second	Yes	The maximum number of UpdateKey requests that you can make per second. Additional requests are throttled.
Rate of UpdateMap API requests	Each supported Region: 10 per second	Yes	The maximum number of UpdateMap requests that you can make per second. Additional requests are throttled.
Rate of UpdatePlaceIndex API requests	Each supported Region: 10 per second	Yes	The maximum number of UpdatePlaceIndex requests that you can make per second. Additional requests are throttled.
Rate of UpdateRouteCalculator API requests	Each supported Region: 10 per second	Yes	The maximum number of UpdateRouteCalculator requests that you can make per second. Additional requests are throttled.
Rate of UpdateTracker API requests	Each supported Region: 10 per second	Yes	The maximum number of UpdateTracker requests that you can make per second. Additional requests are throttled.

Name	Default	Adjustable	Description
Rate of VerifyDevicePosition API requests	Each supported Region: 50 per second	Yes	The maximum number of VerifyDevicePosition requests that you can make per second. Additional requests are throttled.
Route Calculator resources per account	Each supported Region: 40	Yes	The maximum number of Route Calculator resources that you can create per account.
Tracker consumers per tracker	Each supported Region: 5	No	The maximum number of Geofence Collection that Tracker resource can be associated with.
Tracker resources per account	Each supported Region: 500	Yes	The maximum number of Tracker resources that you can create per account.

Note

You can monitor your usage against your quotas with Cloudwatch. For more information, see Using CloudWatch to monitor usage against quotas.

Managing your Amazon Location service quotas

Amazon Location Service is integrated with Service Quotas, an AWS service that enables you to view and manage your quotas from a central location. For more information, see What Is Service Quotas? in the *Service Quotas User Guide*.

Service Quotas makes it easy to look up the value of your Amazon Location service quotas.

AWS Management Console

To view Amazon Location service quotas using the console

1. Open the Service Quotas console at https://console.aws.amazon.com/servicequotas/.
2. In the navigation pane, choose **AWS services**.
3. From the **AWS services** list, search for and select **Amazon Location**.

   In the **Service quotas** list, you can see the service quota name, applied value (if it is available), AWS default quota, and whether the quota value is adjustable.
4. To view additional information about a service quota, such as the description, choose the quota name.
5. (Optional) To request a quota increase, select the quota that you want to increase, select **Request quota increase**, enter or select the required information, and select **Request**.

To work more with service quotas using the console see the Service Quotas User Guide. To request a quota increase, see Requesting a quota increase in the *Service Quotas User Guide*.

AWS CLI

To view Amazon Location service quotas using the AWS CLI

Run the following command to view the default Amazon Location quotas.

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code geo \
    --output table
```

To work more with service quotas using the AWS CLI, see the Service Quotas AWS CLI Command Reference. To request a quota increase, see the `request-service-quota-increase` command in the AWS CLI Command Reference.

Getting started as a developer using Amazon Location Service

You can use Amazon Location Service to provide geographic-related functionality for apps across many different form factors and systems, including backend web services, web applications, and mobile applications. There are many tools provided to help you build your applications, including SDKs, libraries, and sample code.

This section provides information and links to help you get started with Amazon Location. In particular, the following topics provide information that can be most helpful to you:

- Scenarios and use cases – A list of development scenarios and how Amazon Location Service can help you complete them.
- Amazon Location SDKs and tools – The software development kits (SDKs) and libraries that will help you when programming with Amazon Location.
- Amazon Location Service API Reference – A reference to the core Amazon Location APIs that ship with the AWS SDK.
- Code examples – This section provides samples that will help you get started or to add functionality to your existing application.
- Quick start tutorial – This tutorial shows you how to create your first application. There are versions of the tutorial for creating a web application or an Android-based mobile application.
- Amazon Location Service concepts – This section of this guide describes the basic concepts of Amazon Location, including sections on Maps, Places search, Routes, and Geofences and Trackers.
- Amplify – Amplify is a complete solution that encapsulates much of the functionality needed for creating web and mobile applications using the AWS Cloud. If you are already using Amplify, or choose to use Amplify, it has a geo library using Amazon Location Service built-in that you can use. To get started with Amplify Geo, see the documentation here.

Scenarios and use cases

Amazon Location Service is a service that runs in the AWS Cloud. You may call it from your own Amazon EC2 instances in the cloud, but many mapping applications will run on devices, or a

combination of devices and the cloud. The following lists just a few typical scenarios and how you might approach developing them.

- A backend application that helps you to optimize routes for drivers in your fleet.

  You can write an application that runs on Amazon EC2 in the AWS Cloud that uses the Amazon Location Service to calculate route matrices as an input to a route optimizer for your fleet. Use the AWS SDK to make calls to Amazon Location.
- A web application that allows your customers to find the locations of your business.

  You can create a website that runs on Amazon EC2 instances, including a location-based application. Use the AWS SDK for JavaScript to develop a web application to look up locations using places search, and display results on a map using MapLibre. Use the Amazon Location SDK to make programming with location easier.
- Add location features to an existing iOS or Android application.

  You can use the AWS SDK for Swift (iOS) or Kotlin (Android) to make calls to Amazon Location to add places search and maps functionality to your application. Use MapLibre to render maps. There are additional AWS SDKs available for other languages.
- Track assets (devices or vehicles), and get updates when they are enter or exit areas that you define.

  An application to track devices consists of several parts.
  - Each device that you are tracking must have a tracker resource created to track it. It must send position updates to Amazon Location Service, for example, by using MQTT.
  - Create geofences to define areas that you want to get enter and exit events for your assets.
  - You can use Amazon EC2 or AWS Lambda to respond to your events as assets enter or exit the geofence areas.
  - You can expand upon this to create web or device applications to track and display your asset locations on maps.

The following section gives details on tools and libraries available to use with each aspect of the Amazon Location Service.

SDKs and tools for using Amazon Location Service

There are several tools that will help you to use Amazon Location Service.

- **AWS SDKs** – The AWS software development kits (SDKs) are available in many popular programming languages, providing an API, code examples, and documentation that makes it easier to build applications in your preferred language. The AWS SDKs include the core Amazon Location APIs and functionality, including access to Maps, Places search, Routes, Geofence, and Trackers. To learn more about the SDKs available to use with Amazon Location Service for different applications and languages, see SDKs by language.
- **MapLibre** – Amazon Location Service recommends rendering maps using the MapLibre rendering engine. MapLibre is an engine for displaying maps in web or mobile applications. MapLibre also has a plugin model, and supports user interface for searching and routes in some languages and platforms. To learn more about using MapLibre and the functionality it provides, see MapLibre.
- **Amazon Location SDK** – The Amazon Location SDK is a set of open source libraries that make it easier to develop applications with Amazon Location Service. The libraries provide functionality to support authentication for mobile and web applications, location tracking for mobile applications, conversion between Amazon Location data types and GeoJSON, as well as a hosted package of the Amazon Location client for the AWS SDK v3. To learn more about the Amazon Location SDK, see Amazon Location SDK.

SDKs by language

The following tables provide information about AWS SDKs and MapLibre versions for languages and frameworks, by application type: web, mobile, or backend application.

> **SDK Versions**
>
> We recommend that you use the most recent build of the AWS SDK, and any other SDKs, that you use in your projects, and to keep the SDKs up to date. The AWS SDK provides you with the latest features and functionality, and also security updates. To find the latest build of the AWS SDK for JavaScript, for example, see the browser installation topic in the *AWS SDK for JavaScript* documentation.

Web frontend

The following AWS SDKs and MapLibre versions are available for web frontend application development.

Language / Framework	AWS SDK	Rendering Framework
Fully supported		
JavaScript	https://aws.amazon.com/sdk-for-javascript/	https://maplibre.org/projects/maplibre-gl-js/
ReactJS	https://aws.amazon.com/sdk-for-javascript/	https://github.com/maplibre/maplibre-react-native
TypeScript	https://aws.amazon.com/sdk-for-javascript/	https://maplibre.org/projects/maplibre-gl-js/
Partially supported		
Flutter	https://docs.amplify.aws/start/q/integration/flutter/ Flutter is not yet fully supported by AWS, but limited support is offered via Amplify.	https://github.com/maplibre/flutter-maplibre-gl The MapLibre Flutter library is considered experimental.
Node.js	https://aws.amazon.com/sdk-for-javascript/	There is no MapLibre support for Node.js.
PHP	https://aws.amazon.com/sdk-for-php/	There is no MapLibre support for PHP.

Mobile frontend

The following AWS SDKs and MapLibre versions are available for mobile frontend application development.

Language / Framework	AWS SDK	Rendering Framework
Fully supported		

Language / Framework	AWS SDK	Rendering Framework
Java	https://aws.amazon.com/sdk-for-java/	https://maplibre.org/projects/maplibre-native/
Kotlin	https://aws.amazon.com/sdk-for-kotlin/ Amazon Location Service Mobile Authentication SDK for Android: https://github.com/aws-geospatial/amazon-location-mobile-auth-sdk-android Amazon Location Service Mobile Tracking SDK for Android: https://github.com/aws-geospatial/amazon-location-mobile-tracking-sdk-android	https://maplibre.org/projects/maplibre-native/ Requires custom bindings, as MapLibre is Java-based.
ObjectiveC	https://github.com/aws-amplify/aws-sdk-ios	https://maplibre.org/projects/maplibre-native/
ReactNative	https://aws.amazon.com/sdk-for-javascript/	https://github.com/maplibre/maplibre-react-native

Language / Framework	AWS SDK	Rendering Framework
Swift	https://aws.amazon.com/sdk-for-swift/ Amazon Location Service Mobile Authentication SDK for iOS: https://github.com/aws-geospatial/amazon-location-mobile-auth-sdk-ios Amazon Location Service Mobile Tracking SDK for iOS: https://github.com/aws-geospatial/amazon-location-mobile-tracking-sdk-ios	https://maplibre.org/projects/maplibre-native/
Partially supported		
Flutter	https://docs.amplify.aws/start/q/integration/flutter/ Flutter is not yet fully supported by AWS, but limited support is offered via Amplify.	https://github.com/maplibre/flutter-maplibre-gl The MapLibre Flutter library is considered experimental.

Backend application

The following AWS SDKs are available for backend application development. MapLibre is not listed here, because map rendering is not typically needed for backend applications.

Language	AWS SDK
.NET	https://aws.amazon.com/sdk-for-net/
C++	https://aws.amazon.com/sdk-for-cpp/

Language	AWS SDK
Go	https://aws.amazon.com/sdk-for-go/
Java	https://aws.amazon.com/sdk-for-java/
JavaScript	https://aws.amazon.com/sdk-for-javascript/
Node.js	https://aws.amazon.com/sdk-for-javascript/
TypeScript	https://aws.amazon.com/sdk-for-javascript/
Kotlin	https://aws.amazon.com/sdk-for-kotlin/
PHP	https://aws.amazon.com/sdk-for-php/
Python	https://aws.amazon.com/sdk-for-python/
Ruby	https://aws.amazon.com/sdk-for-ruby/
Rust	https://aws.amazon.com/sdk-for-rust/ The AWS SDK for Rust is in developer preview.

Using MapLibre tools and libraries with Amazon Location

One of the important tools for creating interactive applications with Amazon Location is MapLibre. MapLibre is primarily a rendering engine for displaying maps in a web or mobile application. However, it also includes support for plug-ins, and provides functionality for working with other aspects of Amazon Location. The following describes tools that you can use, based on the area of location that you want to work with.

> **Note**
>
> To use any aspect of Amazon Location, install the AWS SDK for the language that you want to use.

- **Maps**

To display maps in your application, you need a map rendering engine that will use the data provided by Amazon Location, and draw to the screen. Map rendering engines also provide functionality to pan and zoom the map, or to add markers or pushpins and other annotations to the map.

Amazon Location Service recommends rendering maps using the MapLibre rendering engine. MapLibre GL JS is an engine for displaying maps in JavaScript, while MapLibre Native provides maps for either iOS or Android.

MapLibre also has a plug-in ecosystem to extend the core functionality. For more information, visit https://maplibre.org/maplibre-gl-js-docs/plugins/.

- **Places search**

  To make creating a search user interface simpler, you can use the MapLibre geocoder for web (Android applications can use the Android Places plug-in).

  Use the Amazon Location for Maplibre geocoder library to simplify the process of using Amazon Location with `amazon-location-for-maplibre-gl-geocoder` in JavaScript Applications.

- **Routes**

  To display routes on the map, use MapLibre directions.

- **Geofences and Trackers**

  MapLibre doesn't have any specific rendering or tools for geofences and tracking, but you can use the rendering functionality and plug-ins to show the geofences and tracked devices on the map.

  The devices being tracked can use MQTT or manually send updates to Amazon Location Service. Geofence events can be responded to using AWS Lambda.

Many open source libraries are available to provide additional functionality for Amazon Location Service, for example Turf which provide spatial analysis functionality.

Many libraries use the open standard GeoJSON formatted data. Amazon Location Service provides a library to support using GeoJSON in JavaScript applications. For more information, see the next section, Amazon Location SDK and libraries.

Amazon Location MapLibre Geocoder Plugin

The Amazon Location MapLibre geocoder plugin is designed to make it easier for you to incorporate Amazon Location functionality into your JavaScript applications, when working with map rendering and geocoding using the maplibre-gl-geocoder library.

Installation

You can install Amazon Location MapLibre geocoder plugin from NPM for usage with modules, with this command:

```
npm install @aws/amazon-location-for-maplibre-gl-geocoder
```

You can import into an HTML file for usage directly in the browser, with a script:

```
<script src="https://www.unpkg.com/@aws/amazon-location-for-maplibre-gl-
geocoder@1">/script<
```

Usage with Module

This code sets up a Maplibre GL JavaScript map with Amazon Location geocoding capabilities. It uses authentication via Amazon Cognito Identity Pool to access Amazon Location resources. The map is rendered with a specified style and center coordinates, and allows to search for places on the map.

```
// Import MapLibre GL JS
import maplibregl from "maplibre-gl";
// Import from the AWS JavaScript SDK V3
import { LocationClient } from "@aws-sdk/client-location";
// Import the utility functions
import { withIdentityPoolId } from "@aws/amazon-location-utilities-auth-helper";
// Import the AmazonLocationWithMaplibreGeocoder
import { buildAmazonLocationMaplibreGeocoder, AmazonLocationMaplibreGeocoder } from
 "@aws/amazon-location-for-maplibre-gl-geocoder"

const identityPoolId = "Identity Pool ID";
const mapName = "Map Name";
const region = "Region"; // region containing the Amazon Location resource
const placeIndex = "PlaceIndexName" // Name of your places resource in your AWS
 Account.

// Create an authentication helper instance using credentials from Amazon Cognito
```

```
const authHelper = await withIdentityPoolId("Identity Pool ID");

const client = new LocationClient({
  region: "Region", // Region containing Amazon Location resources
  ...authHelper.getLocationClientConfig(), // Configures the client to use
 credentials obtained via Amazon Cognito
});

// Render the map
const map = new maplibregl.Map({
    container: "map",
    center: [-123.115898, 49.295868],
    zoom: 10,
    style: `https://maps.geo.${region}.amazonaws.com/maps/v0/maps/${mapName}/style-
descriptor`,
    ...authHelper.getMapAuthenticationOptions(),
});

// Gets an instance of the AmazonLocationMaplibreGeocoder Object.
const amazonLocationMaplibreGeocoder = buildAmazonLocationMaplibreGeocoder(client,
 placeIndex, {enableAll: true});

// Now we can add the Geocoder to the map.
map.addControl(amazonLocationMaplibreGeocoder.getPlacesGeocoder());
```

Usage with a browser

This example uses the Amazon Location Client to make a request that authenticates using Amazon Cognito.

> **Note**
>
> Some of these example use the Amazon Location Client. The Amazon Location Client is based on the AWS SDK for JavaScript V3 and allows for making calls to Amazon Location through a script referenced in an HTML file.

Include the following in an HTML file:

```
< Import the Amazon Location With Maplibre Geocoder>
<script src="https://www.unpkg.com/@aws/amazon-location-with-maplibre-geocoder@1"></
script>
```

```
<Import the Amazon Location Client>
<script src="https://www.unpkg.com/@aws/amazon-location-client@1"></script>
<!Import the utility library>
<script src="https://www.unpkg.com/@aws/amazon-location-utilities-auth-helper@1"></
script>
```

Include the following in a JavaScript file:

```
const identityPoolId = "Identity Pool ID";
const mapName = "Map Name";
const region = "Region"; // region containing Amazon Location resource

// Create an authentication helper instance using credentials from Amazon Cognito
const authHelper = await
 amazonLocationAuthHelper.withIdentityPoolId(identityPoolId);

// Render the map
const map = new maplibregl.Map({
  container: "map",
  center: [-123.115898, 49.295868],
  zoom: 10,
  style: `https://maps.geo.${region}.amazonaws.com/maps/v0/maps/${mapName}/style-
descriptor`,
  ...authHelper.getMapAuthenticationOptions(),
});

// Initialize the AmazonLocationMaplibreGeocoder object
const amazonLocationMaplibreGeocoderObject =
 amazonLocationMaplibreGeocoder.buildAmazonLocationMaplibreGeocoder(client,
 placesName, {enableAll: true});

// Use the AmazonLocationWithMaplibreGeocoder object to add a geocoder to the map.
map.addControl(amazonLocationMaplibreGeocoderObject.getPlacesGeocoder());
```

Listed below are the functions, and commands used in the Amazon Location MapLibre geocoder plugin:

- **`buildAmazonLocationMaplibreGeocoder`**

  This class creates an instance of the `AmazonLocationMaplibreGeocder` which is the entry point to the other all other calls:

```
const amazonLocationMaplibreGeocoder = buildAmazonLocationMaplibreGeocoder(client,
 placesIndex, {enableAll: true});
```

- **`getPlacesGeocoder`**

  Returns a ready to use IControl object that can be added directly to a map.

```
const geocoder = getPlacesGeocoder();

// Initialize map
let map = await initializeMap();

// Add the geocoder to the map.
map.addControl(geocoder);
```

Amazon Location SDK and libraries

The Amazon Location SDK is a set of open source libraries that provide useful functionality for developing Amazon Location applications. The following functionality is included:

- **Amazon Location client** – The Amazon Location objects in the AWS SDK v3 are bundled and packaged for ease of use in web development.
- **Authentication** – The authentication utility simplifies the authentication (using Amazon Cognito or API keys) when building a web page, JavaScript, iOS, or Android application for Amazon Location Service.
- **Tracking** – The mobile tracking SDKs are available for iOS and Android. This SDK makes it easier for mobile applications to interact with Amazon Location Trackers.
- **Amazon Location GeoJSON functions** – The GeoJSON conversion utilities make it easy to convert between the industry-standard GeoJSON formatted data, and the Amazon Location API formats.

Topics

- How to start using the Amazon Location SDK
- Amazon Location client
- JavaScript Authentication helper
- GeoJSON conversion helpers

- Android Mobile Authentication SDK
- iOS Mobile Authentication SDK
- Android Mobile Tracking SDK
- iOS Mobile Tracking SDK

How to start using the Amazon Location SDK

The Amazon Location SDK is a set of functions that can make using the Amazon Location Service in an application simpler. You can install and import these functions into your JavaScript application. The following sections describe the Amazon Location client, and the authentication and GeoJSON helper libraries.

Amazon Location client

With AWS SDK v3, the SDK is separated out by service. You can install just the parts that you need. For example, to install the Amazon Location client and the credentials provider for Amazon Cognito, use the following commands.

```
npm install @aws-sdk/client-location
npm install @aws-sdk/credential-providers
```

In order to facilitate using Amazon Location Service in JavaScript web frontend applications, AWS provides a hosted bundle of the Amazon Location library and the credentials provider. To use the bundled client, add it to your HTML in a script tag, as follows:

```
<script src="https://unpkg.com/@aws/amazon-location-client@1.x/dist/
amazonLocationClient.js"></script>
```

> **Note**
>
> The package is kept up to date and backward compatible for ease of use. Using this script tag or NPM install will always get the latest version.

JavaScript Authentication helper

The Amazon Location JavaScript authentication helper makes it simpler to authenticate when making Amazon Location API calls from your JavaScript application. This authentication helper

specifically help you when using Amazon Cognito or API keys as your authentication method. This is an open source library that is available on GitHub, here: https://github.com/aws-geospatial/amazon-location-utilities-auth-helper-js.

> **Note**
>
> The Amazon Cognito support in the authentication helper does not support the federated identities feature of Amazon Cognito.

Installation

You can use the libraries with a local install, if you use a build system like webpack, or by including pre-built JavaScript bundles with `<script>` tags in your html.

- Use the following command to install the library, using NPM:

```
npm install @aws/amazon-location-utilities-auth-helper
```

- Use the following command in your HTML file to load the script:

```
<script src="https://unpkg.com/@aws/amazon-location-utilities-auth-helper@1.x/dist/
amazonLocationAuthHelper.js"></script>
```

Import

To use a specific function in your JavaScript application, you must import that function. The following code is used to import the function `withIdentityPoolId` into your application.

```
import { withIdentityPoolId } from '@aws/amazon-location-utilities-auth-helper';
```

Authentication functions

The Amazon Location authentication helpers include the following functions that return an `AuthHelper` object:

- `async withIdentityPoolId( identityPoolId: string): AuthHelper` – This function returns an AuthHelper object, initialized to work with Amazon Cognito

- `async withAPIKey( API_KEY: string): AuthHelper` – This function returns an AuthHelper object, initialized to work with API Keys.

The `AuthHelper` object provides the following functions:

- `AuthHelper.getMapAuthenticationOptions()` – This function of the AuthHelper object returns a JavaScript object with the `transformRequest` that can be used with the map options in MapLibre JS. Only provided when initialized with an identity pool.
- `AuthHelper.getLocationClientConfig()` – This function of the AuthHelper object returns a JavaScript object with the `credentials` that can be used to initialize a LocationClient.
- `AuthHelper.getCredentials()` – This function of the AuthHelper object returns the internal credentials from Amazon Cognito. Only provided when initialized with an identity pool.

Example: Initializing MapLibre map object with Amazon Cognito, using an AuthHelper

```
import { withIdentityPoolId } from '@aws/amazon-location-utilities-auth-helper';

const authHelper = await withIdentityPoolId("identity-pool-id"); // use Cognito pool id
 for credentials

const map = new maplibregl.Map({
    container: "map", // HTML element ID of map element
    center: [-123.1187, 49.2819], // initial map center point
    zoom: 16, // initial map zoom
    style: https://maps.geo.region.amazonaws.com/maps/v0/maps/mapName/style-
descriptor', // Defines the appearance of the map
    ...authHelper.getMapAuthenticationOptions(), // Provides credential options
 required for requests to Amazon Location
});
```

Example: Initializing MapLibre map object with an API key (`AuthHelper` is not needed in this case)

```
const map = new maplibregl.Map({
    container: "map", // HTML element ID of map element
    center: [-123.1187, 49.2819], // initial map center point
    zoom: 16, // initial map zoom
    style: https://maps.geo.region.amazonaws.com/maps/v0/maps/${mapName}/style-
descriptor?key=api-key-id',
```

```
});
```

Example: Initialize the Location client from the AWS SDK for JS, using Amazon Cognito and AuthHelper

This example uses AWS SDK for JavaScript v3.

```
import { withIdentityPoolId } from '@aws/amazon-location-utilities-auth-helper';

const authHelper = await withIdentityPoolId("identity-pool-id"); // use Cognito pool id
 for credentials

//initialize the Location client:
const client = new LocationClient({
    region: "region",
    ...authHelper.getLocationClientConfig() // sets up the Location client to use the
 Cognito pool defined above
});

//call a search function with the location client:
const result = await client.send(new SearchPlaceIndexForPositionCommand({
    IndexName: "place-index", // Place index resource to use
    Position: [-123.1187, 49.2819], // position to search near
    MaxResults: 10 // number of results to return
});
```

Example: Initialize the Location client from the AWS SDK for JS, using an API key and AuthHelper

This example uses AWS SDK for JavaScript v3.

```
import { withAPIKey } from '@aws/amazon-location-utilities-auth-helper';

const authHelper = await withAPIKey("api-key-id"); // use API Key id for credentials

//initialize the Location client:
const client = new LocationClient({
    region: "region",
    ...authHelper.getLocationClientConfig() // sets up the Location client to use the
 API Key defined above
});
```

```
//call a search function with the location client:
const result = await client.send(new SearchPlaceIndexForPositionCommand({
    IndexName: "place-index", // Place index resource to use
    Position: [-123.1187, 49.2819], // position to search near
    MaxResults: 10 // number of results to return
});
```

GeoJSON conversion helpers

The Amazon Location GeoJSON conversion helpers provide tools to convert Amazon Location Service data types to and from the industry-standard GeoJSON format. GeoJSON is used, for example, with MapLibre to render geographic data on the map. This is an open source library that is available on GitHub, here: https://github.com/aws-geospatial/amazon-location-utilities-datatypes-js.

Installation

You can use the libraries with a local install, like webpack, or by including pre-built JavaScript bundles with `<script>` tags in your html.

- Use the following command to install the library, using NPM.

```
npm install @aws/amazon-location-utilities-datatypes
```

- Use the following command in your HTML file to load the script:

```
<script src="https://unpkg.com/@aws/amazon-location-utilities-datatypes@1.x/dist/
amazonLocationDataConverter.js"></script>
```

Import

To use a specific function in your JavaScript application, you must import that function.
The following code is used to import the function `placeToFeatureCollection` into your application.

```
import { placeToFeatureCollection } from '@aws/amazon-location-utilities-datatypes';
```

GeoJSON conversion functions

The Amazon Location GeoJSON conversion helpers include the following functions:

- `placeToFeatureCollection(place: GetPlaceResponse | searchPlaceIndexForPositionResponse | searchPlaceIndexForTextResponse, keepNull: boolean): Feature` – This function converts responses from the place search functions to a GeoJSON FeatureCollection with 1 or more Point features.
- `devicePositionToFeatureCollection(devicePositions: GetDevicePositionResponse | BatchGetDevicePositionResponse | GetDevicePositionHistoryResponse | ListDevicePositionsResponse, keepNull: boolean)` – This function converts responses from the tracker device position functions to a GeoJSON FeatureCollection with 1 or more Point features.
- `routeToFeatureCollection(legs: CalculateRouteResponse): FeatureCollection` – This function converts responses from the calculate route function to a GeoJSON FeatureCollection with a single MultiStringLine feature. Each leg of the route is represented by a LineString entry in the MultiStringLine.
- `geofenceToFeatureCollection(geofences: GetGeofenceResponse | PutGeofenceRequest | BatchPutGeofenceRequest | ListGeofencesResponse): FeatureCollection` – This function converts geofence functions request or response to a GeoJSON FeatureCollection with Polygon features. It can convert geofences both in the response and the request, allowing you to show geofences on a map before uploading them with PutGeofence or BatchPutGeofence.

  This function will convert a circle geofence to a feature with an approximated polygon, but will also have "center" and "radius" properties to recreate the circle geofence, if necessary (see the next function).
- `featureCollectionToGeofences(featureCollection: FeatureCollection): BatchPutGeofenceRequestEntry[]` – This function converts a GeoJSON FeatureCollection with Polygon features to an array of BatchPutGeofenceRequestEntry objects, so the result can be used to create a request to BatchPutGeofence.

  If a Feature in the FeatureCollection has "center" and "radius" properties, it will be converted into a circle geofence request entry, ignoring the geometry of the polygon.

Example: Convert search results to a point layer in MapLibre

This example uses AWS SDK for JavaScript v3.

```
import { placeToFeatureCollection } from '@aws/amazon-location-utility-datatypes';
```

```
...

let map; // map here is an initialized MapLibre instance

const client = new LocationClient(config);
const input = { your_input };
const command = new searchPlaceIndexForTextCommand(input);
const response = await client.send(command);

// calling utility function to convert the response to GeoJSON
const featureCollection = placeToFeatureCollection(response);
map.addSource("search-result", featureCollection);
map.addLayer({
    id: "search-result",
    type: "circle",
    source: "search-result",
    paint: {
        "circle-radius": 6,
        "circle-color": "#B42222",
    },
});
```

Android Mobile Authentication SDK

These utilities help you authenticate when making Amazon Location Service API calls from your Android applications. This specifically helps when using Amazon Cognito or API keys as the authentication method.

The Android mobile authentication SDK is available on github: Amazon Location Service Mobile Authentication SDK for Android. Additionally, both the mobile authentication SDK and the AWS SDK are available on the AWS Maven repository.

Installation

To use the mobile authentication SDK, add the following import statements to your `build.gradle` file in Android Studio.

```
implementation("software.amazon.location:auth:0.0.1")
implementation("com.amazonaws:aws-android-sdk-location:2.72.0")
```

Authentication Functions

The authentication helper SDK has the following functions:

- `authHelper.authenticateWithApiKey("My-Amazon-Location-API-Key"): LocationCredentialsProvider`: This function returns a `LocationCredentialsProvider` initialized to work with an API Key.
- `authHelper.authenticateWithCognitoIdentityPool("My-Cognito-Identity-Pool-Id"): LocationCredentialsProvider`: This function returns a `LocationCredentialsProvider` initialized to work with an Amazon Cognito identity pool.

Usage

To use the SDK in your code, import the following classes:

```
import com.amazonaws.services.geo.AmazonLocationClient
import software.amazon.location.auth.AuthHelper
import software.amazon.location.auth.LocationCredentialsProvider
```

You have two options when creating the authentication helper and location client provider instances. You can create an instance using Amazon Location API keys or Amazon Cognito.

- To create an authentication helper instance using an Amazon Location API Key, declare the helper class as follows:

```
var authHelper = AuthHelper(applicationContext)
var locationCredentialsProvider : LocationCredentialsProvider =
 authHelper.authenticateWithApiKey("My-Amazon-Location-API-Key")
```

- To create an authentication helper instance using Amazon Cognito, declare the helper class as follows:

```
var authHelper = AuthHelper(applicationContext)
var locationCredentialsProvider : LocationCredentialsProvider =
 authHelper.authenticateWithCognitoIdentityPool("My-Cognito-Identity-Pool-Id")
```

You can create an Amazon Location client instance using the location credentials provider and make calls to the Amazon Location service. The following example searches for places near a specified latitude and longitude.

```
var locationClient =
 authHelper.getLocationClient(locationCredentialsProvider.getCredentialsProvider())
```

```
var searchPlaceIndexForPositionRequest =
 SearchPlaceIndexForPositionRequest().withIndexName("My-Place-Index-
Name").withPosition(arrayListOf(30.405423, -97.718833))
var nearbyPlaces =
 locationClient.searchPlaceIndexForPosition(searchPlaceIndexForPositionRequest)
```

iOS Mobile Authentication SDK

These utilities help you authenticate when making Amazon Location Service API calls from your iOS applications. This specifically helps when using Amazon Cognito or API keys as the authentication method.

The iOS mobile authentication SDK is available on github: Amazon Location Service Mobile Authentication SDK for iOS.

Installation

Install the SDK in an Xcode project:

1. Go to **File**, then select **Add Package Dependencies** in your XCode project.
2. Type the package URL: https://github.com/aws-geospatial/amazon-location-mobile-auth-sdk-ios/ into the search bar and press the enter key.
3. Select the `amazon-location-mobile-auth-sdk-ios` package and press **Add Package**.
4. Select the `AmazonLocationiOSAuthSDK` package product and press **Add Package**.

Authentication Functions

The authentication helper SDK has the following functions:

- `authHelper.authenticateWithApiKey("My-Amazon-Location-API-Key"): LocationCredentialsProvider`: This function returns a `LocationCredentialsProvider` initialized to work with an API Key.
- `authHelper.authenticateWithCognitoIdentityPool("My-Cognito-Identity-Pool-Id"): LocationCredentialsProvider`: This function returns a `LocationCredentialsProvider` initialized to work with an Amazon Cognito identity pool.

Usage

To use the mobile authentication SDK, add the following statements to your activity:

```
import AmazonLocationiOSAuthSDK
import AWSLocationXCF
```

You have two options when creating the authentication helper and location client provider instances. You can create an instance using Amazon Location API keys or Amazon Cognito.

- To create an authentication helper instance using an Amazon Location API Key, declare the helper class as follows:

```
let authHelper = AuthHelper()
let locationCredentialsProvider = authHelper.authenticateWithAPIKey(apiKey: "My-Amazon-Location-API-Key", region: "account-region")
```

- To create an authentication helper instance using Amazon Cognito, declare the helper class as follows:

```
let authHelper = AuthHelper()
let locationCredentialsProvider =
 authHelper.authenticateWithCognitoUserPool(identityPoolId: "My-Amazon-Location-API-Key", region: "account-region")
```

You can create an Amazon Location client instance using the location credentials provider and make calls to the Amazon Location service. The following example searches for places near a specified latitude and longitude.

```
let locationClient = AWSLocation.default()
let searchPlaceIndexForPositionRequest =
 AWSLocationSearchPlaceIndexForPositionRequest()!
searchPlaceIndexForPositionRequest.indexName = "My-Place-Index-Name"
searchPlaceIndexForPositionRequest.position = [30.405423, -97.718833]
let nearbyPlaces = locationClient.searchPlaceIndex(forPosition:
 searchPlaceIndexForPositionRequest)
```

Android Mobile Tracking SDK

The Amazon Location mobile tracking SDK provides utilities which help easily authenticate, capture device positions, and send position updates to Amazon Location Trackers. The SDK supports local filtering of location updates with configurable update intervals. This reduces data costs and optimizes intermittent connectivity for your Android applications.

The Android tracking SDK is available on GitHub: Amazon Location Mobile Tracking SDK for Android. Additionally, both the mobile authentication SDK and the AWS SDK are available on the AWS Maven repository. The Android tracking SDK is designed to work with the general AWS SDK.

This section covers the following topics for the Amazon Location mobile tracking Android SDK:

Topics

- Installation
- Usage
- Filters
- Android Mobile SDK tracking functions
- Examples

Installation

To install the SDK, add the following lines to the dependencies section of your build.gradle file in Android Studio:

```
implementation("software.amazon.location:tracking:0.0.1")
implementation("software.amazon.location:auth:0.0.1")
implementation("com.amazonaws:aws-android-sdk-location:2.72.0")
```

Usage

This procedure shows you how to use the SDK to authenticate and create the `LocationTracker` object:

> **Note**
>
> This procedure assumes you have imported the library mentioned in the Installation section.

1. Import the following classes in your code:

```
import software.amazon.location.tracking.LocationTracker
import software.amazon.location.tracking.config.LocationTrackerConfig
import software.amazon.location.tracking.util.TrackingSdkLogLevel
import com.amazonaws.services.geo.AmazonLocationClient
```

```
import software.amazon.location.auth.AuthHelper
import software.amazon.location.auth.LocationCredentialsProvider
```

2. Next create an `AuthHelper`, since the `LocationCredentialsProvider` parameter is required for creating a `LocationTracker` object:

```
// Create an authentication helper using credentials from Cognito
val authHelper = AuthHelper(applicationContext)
val locationCredentialsProvider : LocationCredentialsProvider =
 authHelper.authenticateWithCognitoIdentityPool("My-Cognito-Identity-Pool-Id")
```

3. Now, use the `LocationCredentialsProvider` and `LocationTrackerConfig` to create a `LocationTracker` object:

```
val config = LocationTrackerConfig(
    trackerName = "MY-TRACKER-NAME",
    logLevel = TrackingSdkLogLevel.DEBUG,
    accuracy = Priority.PRIORITY_HIGH_ACCURACY,
    latency = 1000,
    frequency = 5000,
    waitForAccurateLocation = false,
    minUpdateIntervalMillis = 5000,
)
locationTracker = LocationTracker(
    applicationContext,
    locationCredentialsProvider,
    config,
)
```

Filters

The Amazon Location mobile tracking Android SDK has three inbuilt location filters.

- `TimeLocationFilter`: Filters the current location to be uploaded based on a defined time interval.
- `DistanceLocationFilter`: Filters location updates based on a specified distance threshold.
- `AccuracyLocationFilter`: Filters location updates by comparing the distance moved since the last update with the current location's accuracy.

This example adds filters in the `LocationTracker` at the creation time:

```
val config = LocationTrackerConfig(
    trackerName = "MY-TRACKER-NAME",
    logLevel = TrackingSdkLogLevel.DEBUG,
    accuracy = Priority.PRIORITY_HIGH_ACCURACY,
    latency = 1000,
    frequency = 5000,
    waitForAccurateLocation = false,
    minUpdateIntervalMillis = 5000,
    locationFilters = mutableListOf(TimeLocationFilter(), DistanceLocationFilter(),
 AccuracyLocationFilter())
)
locationTracker = LocationTracker(
    applicationContext,
    locationCredentialsProvider,
    config,
)
```

This example enables and disables filter at runtime with `LocationTracker`:

```
// To enable the filter
locationTracker?.enableFilter(TimeLocationFilter())

// To disable the filter
locationTracker?.disableFilter(TimeLocationFilter())
```

Android Mobile SDK tracking functions

The Amazon Location mobile tracking SDK for Android includes the following functions:

- **Class**: `LocationTracker`

  `constructor(context: Context,locationCredentialsProvider: LocationCredentialsProvider,trackerName: String)`, or `constructor(context: Context,locationCredentialsProvider: LocationCredentialsProvider,clientConfig: LocationTrackerConfig)`

  This is an initializer function to create a `LocationTracker` object. It requires instances of `LocationCredentialsProvider` , `trackerName` and optionally an instance of `LocationTrackingConfig`. If the config is not provided it will be initialized with default values.
- **Class**: `LocationTracker`

`start(locationTrackingCallback: LocationTrackingCallback)`

Starts the process of accessing the user's location and sending it to an Amazon Location tracker.

- **Class**: `LocationTracker`

  `isTrackingInForeground()`

  Checks if location tracking is currently in progress.

- **Class**: `LocationTracker`

  `stop()`

  Stops the process of tracking the user's location.

- **Class**: `LocationTracker`

  `startTracking()`

  Starts the process of accessing the user's location and sending it to the AWS tracker.

- **Class**: `LocationTracker`

  `startBackground(mode: BackgroundTrackingMode, serviceCallback: ServiceCallback)`

  Starts the process of accessing the user's location and sending it to the AWS tracker while the application is in the background. BackgroundTrackingMode has the following options:

  - `ACTIVE_TRACKING`: This option actively tracks a user's location updates.
  - `BATTERY_SAVER_TRACKING`: This option tracks user's location updates every 15 minutes.

- **Class**: `LocationTracker`

  `stopBackgroundService()`

  Stops the process of accessing the user's location and sending it to the AWS tracker while the application is in the background.

- **Class**: `LocationTracker`

  `getTrackerDeviceLocation()`

  Retrieves the device location from Amazon Location services.

- **Class**: `LocationTracker`

  ```
  getDeviceLocation(locationTrackingCallback: LocationTrackingCallback?)
  ```

  Retrieves the current device location from the fused location provider client and uploads it to Amazon Location tracker.

- **Class**: `LocationTracker`

  ```
  uploadLocationUpdates(locationTrackingCallback:
  LocationTrackingCallback?)
  ```

  Uploads the device location to Amazon Location services after filtering based on the configured location filters.

- **Class**: `LocationTracker`

  ```
  enableFilter(filter: LocationFilter)
  ```

  Enables a particular location filter.

- **Class**: `LocationTracker`

  ```
  checkFilterIsExistsAndUpdateValue(filter: LocationFilter)
  ```

  Disable particular location filter.

- **Class**: `LocationTrackerConfig`

  ```
  LocationTrackerConfig( // Required var trackerName:
  String, // Optional var locationFilters: MutableList =
  mutableListOf( TimeLocationFilter(), DistanceLocationFilter(), ), var
  logLevel: TrackingSdkLogLevel = TrackingSdkLogLevel.DEBUG, var accuracy:
  Int = Priority.PRIORITY_HIGH_ACCURACY, var latency: Long = 1000, var
  frequency: Long = 1500, var waitForAccurateLocation: Boolean = false, var
  minUpdateIntervalMillis: Long = 1000, var persistentNotificationConfig:
  NotificationConfig = NotificationConfig())
  ```

  This initializes the `LocationTrackerConfig` with user-defined parameter values. If a parameter value is not provided, it will be set to a default value.

- **Class**: `LocationFilter`

`shouldUpload(currentLocation: LocationEntry, previousLocation: LocationEntry?): Boolean`

The `LocationFilter` is a protocol that users can implement for their custom filter implementation. You need to implement the `shouldUpload` function to compare previous and current location and return if the current location should be uploaded.

Examples

The following code sample shows the mobile tracking SDK functionality.

This example uses the `LocationTracker` to start and stop tracking in background:

```
// For starting the location tracking
locationTracker?.startBackground(
BackgroundTrackingMode.ACTIVE_TRACKING,
object : ServiceCallback {
    override fun serviceStopped() {
        if (selectedTrackingMode == BackgroundTrackingMode.ACTIVE_TRACKING) {
            isLocationTrackingBackgroundActive = false
        } else {
            isLocationTrackingBatteryOptimizeActive = false
        }
    }
},
)

// For stopping the location tracking
locationTracker?.stopBackgroundService()
```

iOS Mobile Tracking SDK

The Amazon Location mobile tracking SDK provides utilities which help easily authenticate, capture device positions, and send position updates to Amazon Location Trackers. The SDK supports local filtering of location updates with configurable update intervals. This reduces data costs and optimizes intermittent connectivity for your iOS applications.

The iOS tracking SDK is available on GitHub: Amazon Location Mobile Tracking SDK for iOS.

This section covers the following topics for the Amazon Location mobile tracking iOS SDK:

Topics

- Installation
- Usage
- Filters
- iOS Mobile SDK tracking functions
- Examples

Installation

Use the following procedure to install the mobile tracking SDK for iOS:

1. In your Xcode project, go to **File** and select **Add Package Dependencies**.
2. Type the following URL: https://github.com/aws-geospatial/amazon-location-mobile-tracking-sdk-ios/ into the search bar and press the enter key.
3. Select the `amazon-location-mobile-tracking-sdk-ios` package and click on **Add Package**.
4. Select the `AmazonLocationiOSTrackingSDK` package product and click on **Add Package**.

Usage

The following procedure shows you how to create an authentication helper using credentials from Cognito.

1. After installing the library, you need to add one or both of the descriptions into your `info.plist` file:

```
Privacy - Location When In Use Usage Description
Privacy - Location Always and When In Use Usage Description
```

2. Next, import the AuthHelper in your class:

```
import AmazonLocationiOSAuthSDKimport AmazonLocationiOSTrackingSDK
```

3. Then you will create an `AuthHelper` object and use it with the AWS SDK, by creating an authentication helper using credentials from Amazon Cognito.

```
let authHelper = AuthHelper()
```

```
let locationCredentialsProvider =
 authHelper.authenticateWithCognitoUserPool(identityPoolId: "My-Cognito-Identity-
Pool-Id", region: "My-region") //example: us-east-1
let locationTracker = LocationTracker(provider: locationCredentialsProvider,
 trackerName: "My-tracker-name")

// Optionally you can set ClientConfig with your own values in either initialize or
 in a separate function
// let trackerConfig = LocationTrackerConfig(locationFilters:
 [TimeLocationFilter(), DistanceLocationFilter()],

trackingDistanceInterval: 30,
trackingTimeInterval: 30,
logLevel: .debug)

// locationTracker = LocationTracker(provider: credentialsProvider, trackerName:
 "My-tracker-name",config: trackerConfig)
// locationTracker.setConfig(config: trackerConfig)
```

Filters

The Amazon Location mobile tracking iOS SDK has three inbuilt location filters.

- `TimeLocationFilter`: Filters the current location to be uploaded based on a defined time interval.
- `DistanceLocationFilter`: Filters location updates based on a specified distance threshold.
- `AccuracyLocationFilter`: Filters location updates by comparing the distance moved since the last update with the current location's accuracy.

This example adds filters in the `LocationTracker` at the creation time:

```
val config = LocationTrackerConfig(
    trackerName = "MY-TRACKER-NAME",
    logLevel = TrackingSdkLogLevel.DEBUG,
    accuracy = Priority.PRIORITY_HIGH_ACCURACY,
    latency = 1000,
    frequency = 5000,
    waitForAccurateLocation = false,
    minUpdateIntervalMillis = 5000,
```

```
    locationFilters = mutableListOf(TimeLocationFilter(), DistanceLocationFilter(),
 AccuracyLocationFilter())
)

locationTracker = LocationTracker(
    applicationContext,
    locationCredentialsProvider,
    config,
)
```

This example enables and disables filter at runtime with `LocationTracker`:

```
// To enable the filter
locationTracker?.enableFilter(TimeLocationFilter())

// To disable the filter
locationTracker?.disableFilter(TimeLocationFilter())
```

iOS Mobile SDK tracking functions

The Amazon Location mobile tracking SDK for iOS includes the following functions:

- **Class**: `LocationTracker`

  `init(provider: LocationCredentialsProvider, trackerName: String, config: LocationTrackerConfig? = nil)`

  This is an initializer function to create a `LocationTracker` object. It requires instances of `LocationCredentialsProvider` , `trackerName` and optionally an instance of `LocationTrackingConfig`. If the config is not provided it will be initialized with default values.
- **Class**: `LocationTracker`

  `setTrackerConfig(config: LocationTrackerConfig)`

  This sets Tracker's config to take effect at any point after initialization of location tracker
- **Class**: `LocationTracker`

  `getTrackerConfig()`

  This gets the location tracking config to use or modify in your app.

Returns: `LocationTrackerConfig`

- **Class**: `LocationTracker`

  `getDeviceId()`

  Gets the location tracker's generated device Id.

  Returns: `String?`

- **Class**: `LocationTracker`

  `startTracking()`

  Starts the process of accessing the user's location and sending it to the AWS tracker.

- **Class**: `LocationTracker`

  `resumeTracking()`

  Resumes the process of accessing the user's location and sending it to the AWS tracker.

- **Class**: `LocationTracker`

  `stopTracking()`

  Stops the process of tracking the user's location.

- **Class**: `LocationTracker`

  `startBackgroundTracking(mode: BackgroundTrackingMode)`

  Starts the process of accessing the user's location and sending it to the AWS tracker while the application is in the background. `BackgroundTrackingMode` has the following options:

  - `Active:` This option doesn't automatically pauses location updates.
  - `BatterySaving:` This option automatically pauses location updates
  - `None:` This option overall disables background location updates

- **Class**: `LocationTracker`

  `resumeBackgroundTracking(mode: BackgroundTrackingMode)`

  Resumes the process of accessing the user's location and sending it to the AWS tracker while the application is in the background.

- **Class**: `LocationTracker`

  `stopBackgroundTracking()`

  Stops the process of accessing the user's location and sending it to the AWS tracker while the application is in the background.

- **Class**: `LocationTracker`

  `getTrackerDeviceLocation(nextToken: String?, startTime: Date? = nil, endTime: Date? = nil, completion: @escaping (Result<GetLocationResponse, Error>)`

  Retrieves the uploaded tracking locations for the user's device between start and end date and time.

  Returns: `Void`

- **Class**: `LocationTrackerConfig`

  `init()`

  This initializes the LocationTrackerConfig with default values.

- **Class**: `LocationTrackerConfig`

  `init(locationFilters: [LocationFilter]? = nil, trackingDistanceInterval: Double? = nil, trackingTimeInterval: Double? = nil, trackingAccuracyLevel: Double? = nil, uploadFrequency: Double? = nil, desiredAccuracy: CLLocationAccuracy? = nil, activityType: CLActivityType? = nil, logLevel: LogLevel? = nil)`

  This initializes the `LocationTrackerConfig` with user-defined parameter values. If a parameter value is not provided it will be set to a default value.

- **Class**: `LocationFilter`

  `shouldUpload(currentLocation: LocationEntity, previousLocation: LocationEntity?, trackerConfig: LocationTrackerConfig)`

  The `LocationFilter` is a protocol that users can implement for their custom filter implementation. A user would need to implement `shouldUpload` function to compare previous and current location and return if the current location should be uploaded.

Examples

This sections details examples of using the Amazon Location Mobile Tracking SDK for iOS.

> **Note**
>
> Ensure that the necessary permissions are set in the `info.plist` file. These are the same permissions listed in the Usage section.

The following example demonstrates functionality for tracking device location and retrieving tracked locations:

```
Privacy - Location When In Use Usage Description
Privacy - Location Always and When In Use Usage Description
```

Start tracking the location:

```
do {
    try locationTracker.startTracking()
    }
catch TrackingLocationError.permissionDenied {
        // Handle permissionDenied by showing the alert message or opening the app
 settings
        }
```

Resume tracking the location:

```
do {
    try locationTracker.resumeTracking()
    }
catch TrackingLocationError.permissionDenied {
    // Handle permissionDenied by showing the alert message or opening the app settings
    }
```

Stop tracking the location:

```
locationTracker.stopTracking()
```

Start background tracking:

```
do {

 locationTracker.startBackgroundTracking(mode: .Active) // .Active, .BatterySaving, .None
    }
catch TrackingLocationError.permissionDenied {
   // Handle permissionDenied by showing the alert message or opening the app settings
   }
```

Resume background tracking:

```
do {
    locationTracker.resumeBackgroundTracking(mode: .Active)
    }
catch TrackingLocationError.permissionDenied {
    // Handle permissionDenied by showing the alert message or opening the app settings
    }
```

To stop background tracking:

```
locationTracker.stopBackgroundTracking()
```

Retrieve device's tracked locations from the tracker:

```
func getTrackingPoints(nextToken: String? = nil) {
let startTime: Date = Date().addingTimeInterval(-86400) // Yesterday's day date and
 time
let endTime: Date = Date()
locationTracker.getTrackerDeviceLocation(nextToken: nextToken, startTime: startTime,
 endTime: endTime, completion: { [weak self] result in
    switch result {
    case .success(let response):

        let positions = response.devicePositions
        // You can draw positions on map or use it further as per your requirement

        // If nextToken is available, recursively call to get more data
        if let nextToken = response.nextToken {
            self?.getTrackingPoints(nextToken: nextToken)
        }
    case .failure(let error):
        print(error)
    }
```

```
})
}
```

Amazon Location APIs

Amazon Location Service provides API operations to programmatically access the location functionality. This includes APIs for Maps, Places, Routes, Trackers, Geofences, and tagging your resources. For information about the available API actions, see the Amazon Location Service API reference.

You can find samples in the Code examples chapter of this guide.

Using Amazon Location with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that makes it easier for developers to build AWS applications in their preferred language.

For more information about the SDKs available for use with Amazon Location Service by language, see SDKs by language in this guide.

> **SDK Versions**
>
> We recommend that you use the most recent build of the AWS SDK, and any other SDKs, that you use in your projects, and to keep the SDKs up to date. The AWS SDK provides you with the latest features and functionality, and also security updates. To find the latest build of the AWS SDK for JavaScript, for example, see the browser installation topic in the *AWS SDK for JavaScript* documentation.

Amazon Location API error message updates

Beginning August 1, 2023, the Amazon Location team is changing API error messages as described in the following tables. Error codes will not be changed. If your applications depend on exact error message strings, you must update your applications with the new strings. For help with questions or problems, contact AWS Support.

Topics

- Places

- Maps
- Trackers
- Routes
- Metadata
- Geofences

Places

Places

Error code	Exception	Old error message	New error message
500	InternalServerException	Internal Server Exception	Internal server error. Try again later.
404	ResourceNotFoundException	resource <PlaceIndexName> not found, reason: <Reason> Resource '<PlaceIndexName>' not found placeIdx<PlaceIndexName> not found, reason: <Reason> no place index with name '%s' found	Place index not found: <PlaceIndexName>.
404	ResourceNotFoundException	place not found	Place not found: <PlaceId>.
400	ValidationException	PlaceIndex <PlaceIndexName> cannot be used for SearchPlaceIndexForSuggestions because it	A place index with 'IntendedUse' set to Storage does not support 'SearchPlaceIndexForSuggestion' operation.

Error code	Exception	Old error message	New error message
		has IntendedUse <IntendedUse>	
400	ValidationException	only one of 'BiasPosition' or 'FilterBBox' may be set	Only one of 'BiasPosition' or 'FilterBBox' may be set.
400	ValidationException	BiasPosition must have exactly 2 entries	'BiasPosition' must have exactly 2 entries.
400	ValidationException	BiasPosition[0] must be between -180 and 180	'BiasPosition[0]' must be between -180 and 180.
400	ValidationException	BiasPosition[1] must be between -90 and 90	'BiasPosition[1]' must be between -90 and 90.
400	ValidationException	FilterBBox must have exactly 4 entries	'FilterBBox' must have exactly 4 entries.
400	ValidationException	FilterBBox[0] must be between -180 and 180	'FilterBBox[0]' must be between -180 and 180.
400	ValidationException	FilterBBox[1] must be between -90 and 90	'FilterBBox[1]' must be between -90 and 90.
400	ValidationException	FilterBBox[2] must be between -180 and 180	'FilterBBox[2]' must be between -180 and 180.

Error code	Exception	Old error message	New error message
400	ValidationException	FilterBBox[3] must be between -90 and 90	'FilterBBox[3]' must be between -90 and 90.
400	ValidationException	FilterBBox must have more southwesterly point before more northeasterly point	'FilterBBox' must have more southwesterly position before more northeasterly position.
400	ValidationException	Position must have exactly 2 entries	'Position' must have exactly 2 entries.
400	ValidationException	Position[0] must be between -180 and 180	'Position[0]' must be between -180 and 180.
400	ValidationException	Position[1] must be between -90 and 90	'Position[1]' must be between -90 and 90.
400	ValidationException	Language is not a valid BCP 47 language tag	'Language' must comply with the BCP 47 Language Tag standard, but was set to <GivenValue>. For more information, see https://wikipedia.org/wiki/IETF_language_tag.
400	ValidationException	'placeID' is invalid	'PlaceId' must be a valid ID.
400	ValidationException	no customer account ID parameter found	'RequesterAccountID' is a required field.

Error code	Exception	Old error message	New error message
400	ValidationException	Invalid token	'NextToken' must be a valid token.
400	ValidationException	Expired token	'NextToken' must not be expired.
400	ValidationException	unsupported price plan '<PricingPlan>'	'PricingPlan' must be set to RequestBasedUsage.
400	ValidationException	'DataSource' must be one of: Here, Esri	'DataSource' must be one of Esri, Grab, Here.
400	ValidationException	Grab is only supported in the ap-southeast-1 region	'DataSource' Grab must only be used in following regions: ap-southeast-1.
400	ValidationException	'IntendedUse' and 'PricingPlan' must both be provided to update either property	'IntendedUse' and 'PricingPlan' must both be provided to update either attribute.
402	ServiceQuotaExceededException	Place resources per account exceeded quota limits. For more info, see https://aws.amazon.com/premiumsupport/knowledge-center/manage-service-limits/	Place index resources have exceeded the quota per account per region. For more information, see https://aws.amazon.com/premiumsupport/knowledge-center/manage-service-limits/.

Error code	Exception	Old error message	New error message
409	ConflictException	Resource already exists	Place index already exists: <PlaceIndexName>.

Maps

Maps

Error code	Exception	Old error message	New error message
500	InternalServerException	Internal Server Exception unable to find style template Error fetching style was not able to serialize the map style file	Internal server error. Try again later.
404	ResourceNotFoundException	Map not found	Map not found: <MapName>.
404	ResourceNotFoundException	Sprites are not supported for this resource	Sprite not found: <SpriteName>.
400	ValidationException	Resource name should be set	'MapName' is a required field.
400	ValidationException	Must provide a valid number for start and end of Range	Font Unicode range start and end numbers must both be provided.

Error code	Exception	Old error message	New error message
400	ValidationException	Start of range is an invalid number: <StartValue>	Start of font Unicode range must be a valid number.
400	ValidationException	End of range is an invalid number: <StartValue>	End of font Unicode range must be a valid number.
400	ValidationException	End of range must be exactly 255 higher from start of range, difference found: <Difference>	The difference between the start and end of the font Unicode range must be exactly 255. Difference found: <Difference>.
400	ValidationException	Start of range must be a multiple of 256, found <StartValue>	Start of font Unicode range must be a multiple of 256, but was set to: <StartValue>.
400	ValidationException	Request font is empty	'FontStack' is a required field.

Error code	Exception	Old error message	New error message
400	ValidationException	Request font is not valid for the datasource <DataSource>	<FontStack> is not a supported font stack for data source <DataSource>. For more informati on about the list of supported font stacks, see https://d ocs.aws.amazon.com /location/latest/A PIReference/API_Ge tMapGlyphs.html.
400	ValidationException	Request font is not valid	<FontStack> is not a supported font stack for data source <DataSource>. For more informati on about the list of supported font stacks, see https://d ocs.aws.amazon.com /location/latest/A PIReference/API_Ge tMapGlyphs.html.
400	ValidationException	DataSource is invalid: <DataSource>	'DataSource' must be one of Esri, Grab, Here.
400	ValidationException	Request filename is empty	'FileName' is a required field.

Error code	Exception	Old error message	New error message
400	ValidationException	Request filename is not valid	<SpriteFile> is not a supported sprite file name. For more information about the list of supported sprite file names, see https://docs.aws.amazon.com/location/latest/APIReference/API_GetMapSprites.html.
400	ValidationException	Filename is invalid: <FileName>	<SpriteFile> is not a supported sprite file name. For more information about the list of supported sprite file names, see https://docs.aws.amazon.com/location/latest/APIReference/API_GetMapSprites.html.
400	ValidationException	Filename is an invalid content type: <FileName>	<SpriteFile> is not a supported sprite file name. For more information about the list of supported sprite file names, see https://docs.aws.amazon.com/location/latest/APIReference/API_GetMapSprites.html.

Error code	Exception	Old error message	New error message
400	ValidationException	Filename is invalid: <FileName>	'Filename' must not be empty.
400	ValidationException	y-coordinate part of 'Y' must be a valid integer	y- coordinate part of 'Y' must be an integer.
400	ValidationException	tile resolution part of 'Y' must be a valid integer followed by 'x'	Tile resolution part of 'Y' must be an integer followed by 'X'.
400	ValidationException	file type extension part of 'Y' must not be empty if a '.' is present	File type extension part of 'Y' must not be empty if a '.' is present.
400	ValidationException	'Z' must be a valid integer	'Z' must be an integer.
400	ValidationException	'X' must be a valid integer	'X' must be an integer.
400	ValidationException	'Z' must not be less than minimum zoom of style '<Style>' (<Minimum Value>)	'Z' must not be less than minimum zoom of style <Style> (<MinimumValue>).
400	ValidationException	'Z' must not be greater than maximum zoom of style '<Style' (<Maximum Value>)	'Z' must not be greater than maximum zoom of style Style (<MaximumValue>).

Error code	Exception	Old error message	New error message
400	ValidationException	'Z' value not supported	'Z' must be between 0 and 63.
400	ValidationException	tile resolution part of 'Y' must be omitted because '<Style>' is a vector style	Tile resolution part of 'Y' must be omitted for style <Style>.
400	ValidationException	tile resolution part of 'Y' must be at least 1	Tile resolution part of 'Y' must be at least 1.
400	ValidationException	tile resolution part of 'Y' must not be greater than max resolution of style '<Style>' (<Maximum Resolution>)	Tile resolution part of 'Y' must not be greater than maximum resolution of style <Style> (max <MaxResolution>).
400	ValidationException	file type extension part of 'Y' must be one of <SupportedFileFormats> (or may be omitted) for style '<Style>'	File type extension part of 'Y' must be one of <SupportedFileFormats> (or may be omitted) for style <Style>.
400	ValidationException	file type extension part of 'Y' must be omitted for style '<Style>'	File type extension part of 'Y' must be omitted for style <Style>.
400	ValidationException	y-coordinate part of 'Y' must be an integer in the range 0..2^Zoom-1 (0..<MaxTileCoordinate>)	y-coordinate part of 'Y' must be an integer in the range 0..2^Zoom-1 (0..<MaxTileCoordinate>).

Error code	Exception	Old error message	New error message
400	ValidationException	'DataSource' must be one of: Here, Esri	'DataSource' must be one of Esri, Grab, Here.
400	ValidationException	unsupported price plan '<PricingPlan>'	'PricingPlan' must be set to RequestBasedUsage.
400	ValidationException	Invalid token	'NextToken' must be a valid token.
400	ValidationException	Expired token	'NextToken' must not be expired.
400	ValidationException	Unsupported Map Style: <Style>	<Style> is not a supported map style. For more information about list of supported map styles, see https://docs.aws.amazon.com/location/latest/APIReference/API_MapConfiguration.html.
402	ServiceQuotaExceededException	Map resources per account exceeded quota limits. For more info, see https://aws.amazon.com/premiumsupport/knowledge-center/manage-service-limits/	Map resources have exceeded the quota per account per region. For more information, see https://aws.amazon.com/premiumsupport/knowledge-center/manage-service-limits/.

Error code	Exception	Old error message	New error message
409	ConflictException	Resource already exists	Map already exists: <MapName>.

Trackers

Trackers

Error code	Exception	Old error message	New error message
500	InternalServerException	Internal Server Exception internal server error unable to retrieve point from the storage unable to verify tracker Error processing List request	Internal server error. Try again later.
404	ResourceNotFoundException	tracker not found: <TrackerName> Tracker with name <TrackerName> was not found	Tracker not found: <TrackerName>.
404	ResourceNotFoundException	association not found: TrackerName <TrackerName>; and ConsumerArn < ConsumerArn >	Association between tracker <TrackerName> and consumer <ConsumerArn> is not found.

Error code	Exception	Old error message	New error message
400	ValidationException	'ConsumerArn' must refer to a geofence collection resource	'ConsumerArn' must refer to a geofence collection resource.
400	ValidationException	'ConsumerArn' must refer to a resource in the same region as the tracker it is associated to	'ConsumerArn' must refer to a resource in the same region as the tracker it is associated with.
400	ValidationException	'ConsumerArn' must refer to a resource in the same AWS account as the tracker is it associated to	'ConsumerArn' must refer to a resource in the same AWS account as the tracker it is associated with.
400	ValidationException	'DataSource' must be one of: Here, Esri	'DataSource' must be one of Esri, Grab, Here.
400	ValidationException	Nothing to update.	At least one of the following fields must be set: 'Description', 'PositionFiltering'
400	ValidationException	Invalid token	'NextToken' must be a valid token.
400	ValidationException	Expired token	'NextToken' must not be expired.
400	ValidationException	request.TrackerName not found on request	'TrackerName ' is a required field.
400	ValidationException	no deviceId parameter found	'DeviceId' is a required field.

Error code	Exception	Old error message	New error message
400	ValidationException	unsupported price plan '<PricingPlan>'	'PricingPlan' must be set to RequestBasedUsage
400	ValidationException	Invalid token	'NextToken' must be a valid token.
400	ValidationException	Expired token	'NextToken' must not be expired.
400	ValidationException	provided start time is incorrect, should follow the format YYYY-MM-DDThh:mm:ss.sssZ“	'StartTimeInclusive' must follow the format YYYY-MM-DDThh:mm:ss.sssZ.
400	ValidationException	provided end time is incorrect, should follow the format YYYY-MM-DDThh:mm:ss.sssZ	'EndTimeExclusive' must follow the format YYYY-MM-DDThh:mm:ss.sssZ.
400	ValidationException	end time must be after start time	'EndTimeExclusive' must be after 'StartTimeInclusive'.

Error code	Exception	Old error message	New error message
400	ValidationException	invalid key state	KMS key must be a symmetric Customer Master Key (CMK). Invalid state found. For more informati on about how key state affects the use of a KMS key, see https://docs.aws.a mazon.com/kms/late st/developerguide/ key-state.html.
400	ValidationException	key not found	Invalid KMS key. '<KmsKeyId>' <KmsKeyIdValue> not found.
400	ValidationException	key is disabled	Symmetric Customer Master Key (CMK) must be enabled.
400	ValidationException	access denied	Symmetric Customer Master Key (CMK) must allow Amazon Location to create grants to its KMS key.

Error code	Exception	Old error message	New error message
402	ServiceQuotaExceededException	Tracker <TrackerName> may not have more than <Max> consumer associations	Tracker resource may not have more than <Max> consumer associations. For more information, see https://aws.amazon.com/premiumsupport/knowledge-center/manage-service-limits/.
402	ServiceQuotaExceededException	Trackers per account exceeded quota limits. For more info, see https://aws.amazon.com/premiumsupport/knowledge-center/manage-service-limits/	Tracking resources have exceeded the quota per account per region. For more information, see https://aws.amazon.com/premiumsupport/knowledge-center/manage-service-limits/.
409	ConflictException	association already exists: TrackerName <TrackerName>; and ConsumerArn <ConsumerArn>	An association already exists between tracker <TrackerName> and consumer <ConsumerArn>.
409	ConflictException	Tracker already exists: <TrackerName>	Tracker already exists: <TrackerName>.

Routes

Routes

Error code	Exception	Old error message	New error message
500	InternalServerException	Internal Server Exception	Internal server error. Try again later.
404	ResourceNotFoundException	Resource not found	Route calculator not found: <RouteCalculatorName>.
400	ValidationException	Invalid token	'NextToken' must be a valid token.
400	ValidationException	Expired token	'NextToken' must not be expired.
400	ValidationException	'DataSource' must be one of: Here, Esri, Grab	'DataSource' must be one of Esri, Grab, Here.
400	ValidationException	<PricingPlan> pricing plan is not supported	'PricingPlan' must be set to RequestBasedUsage
400	ValidationException	unsupported price plan '<PricingPlan>'	'PricingPlan' must be set to RequestBasedUsage
400	ValidationException	Grab is only supported in the ap-southeast-1 region	'DataSource' <DataSourceName> must only be used in following regions: ap-southeast-1.
400	ValidationException	PricingPlan must be 'RequestBasedUsage'	'PricingPlan' must be set to RequestBasedUsage.

Error code	Exception	Old error message	New error message
400	ValidationException	'DeparturePositions[0][0]' must be between -180 and 180	'DeparturePositions[0][0]' must be between -180 and 180.
400	ValidationException	'DeparturePositions[0][1]' must be between -90 and 90	'DeparturePositions[0][1]' must be between -90 and 90.
400	ValidationException	'DestinationPositions[0][0]' must be between -180 and 180	'DestinationPositions[0][0]' must be between -180 and 180.
400	ValidationException	'DestinationPositions[0][1]' must be between -90 and 90.	'DestinationPositions[0][1]' must be between -90 and 90
400	ValidationException	'DepartNow' may not be true if 'DepartureTime' is set	Only one of 'DepartNow' or 'DepartureTime' may be set.
400	ValidationException	'<TravelModeOption>'' may not be set when 'TravelMode' has value <TravelModeOption>	'<TravelModeOption>' must not be set when 'TravelMode' has value <TravelModeOption>.
400	ValidationException	'CarModeOptions' may not be set when 'TravelMode' has value Walking	'CarModeOptions' must not be set when 'TravelMode' has value Walking.

Error code	Exception	Old error message	New error message
400	ValidationException	'TruckModeOptions' may not be set when 'TravelMode' has value Walking	'TruckModeOptions' must not be set when 'TravelMode' has value Walking.
400	ValidationException	'TruckModeOptions' may not be set when 'TravelMode' has value Car	'TruckModeOptions' must not be set when 'TravelMode' has value Car.
400	ValidationException	'CarModeOptions' may not be set when 'TravelMode' has value Truck	'CarModeOptions' must not be set when 'TravelMode' has value Truck.
400	ValidationException	At least one of [Height, Length, Width] must be set in 'TruckModeOptions.Dimensions'	At least one of the following attribute must be set in TruckModeOptions.Dimensions: Height, Length, Width.
400	ValidationException	At least one of [Total] must be set in 'TruckModeOptions.Weight'	At least one of the following attribute must be set in TruckModeOptions.Weight: Total.
400	ValidationException	'DeparturePositions' count must be 10 or less with DataSource set to Esri	'DeparturePositions' must have length at most 10 for 'DataSource' Esri.

Error code	Exception	Old error message	New error message
400	ValidationException	'DestinationPositions' count must be 10 or less with DataSource set to Esri	'DestinationPositions' must have length at most 10 for 'DataSource' Esri.
400	ValidationException	'DeparturePositions[0]' is more than 40km away from 'DestinationPositions[0]'	'DeparturePositions[0]' must not be more than 40 km away from 'DestinationPositions[0]'.
400	ValidationException	'DeparturePositions[0]' is more than 400km away from 'DestinationPositions[0]'	'DeparturePositions[0]' must not be more than 400 km away from 'DestinationPositions[0]'.
400	ValidationException	DeparturePositions[0] is contained within an unsupported region. Korea is not supported for CalculateRouteMatrix with the provider Esri.	DeparturePositions[0] is located in Korea, which is not supported when using CalculateRouteMatrix with data provider Esri.
400	ValidationException	'<HereTruckDimension>' must be between <Min> and <Max> <Unit>	'HereTruckDimension' must be between <Min> and <Max> <Unit>.
400	ValidationException	'WaypointPositions[0][0]' must be between -180 and 180	'WaypointPositions[0][0]' must be between -180 and 180.

Error code	Exception	Old error message	New error message
400	ValidationException	'WaypointPositions[0][1]' must be between -90 and 90	'WaypointPositions[0][1]' must be between -90 and 90.
400	ValidationException	'WaypointPositions[1][0]' must be between -180 and 180	'WaypointPositions[1][0]' must be between -180 and 180.
400	ValidationException	'WaypointPositions[1][1]' must be between -90 and 90	'WaypointPositions[1][1]' must be between -90 and 90.
400	ValidationException	No road segment could be matched for one or more coordinates within a radius (1km)	One or more provided positions are more than 1 km from the nearest road segment.
400	ValidationException	Some positions in the request are unreachable	Some positions in the request are unreachable.
400	ValidationException	Total distance between all waypoints must be not be greater than 40km for DataSource Esri when using TravelMode Walking	Total distance between all route positions must not be greater than 40 km for 'DataSource' Esri and 'TravelMode' Walking.
400	ValidationException	Total distance between all waypoints must be not be greater than 400km for DataSource Esri	Total distance between all route positions must not be greater than 400 km for 'DataSource' Esri.

Error code	Exception	Old error message	New error message
400	ValidationException	Following positions in the request are unreachable: <UnreachablePositions>	The following positions are unreachable: <UnreachablePositions>.
400	ValidationException	'DepartureTime' contains a badly-formatted timestamp	'DepartureTime' must follow the format YYYY-MM-DDThh:mm:ss.sssZ.
400	ValidationException	'TravelMode' <TravelMode> is not supported by <DataProvider>	'TravelMode' <TravelMode> not supported by data provider <DataProvider>.
400	ValidationException	'DeparturePositions' must be set	'DeparturePositions' must not be empty.
400	ValidationException	'DestinationPositions' must be set	'DestinationPositions' must not be empty.
400	ValidationException	Some inputs in the request are invalid	Some inputs in the request are invalid.
400	ValidationException	No route found between position <FirstPosition> and position <SecondPosition>	No route found between position <FirstPosition> and position <SecondPosition>.

Error code	Exception	Old error message	New error message
400	ValidationException	No route found	No route found. For more information, see https://developer.here.com/documentation/routing-api/dev_guide/topics/notice.html.
400	ValidationException	No route found	No route found.
402	ServiceQuotaExceededException	Route calculators per account exceeded quota limits. For more info, see https://aws.amazon.com/premiumsupport/knowledge-center/manage-service-limits/	Route calculator resources have exceeded the quota per account per region. For more information, see https://aws.amazon.com/premiumsupport/knowledge-center/manage-service-limits/.
409	ConflictException	Resource already exists	Route calculator already exists: <RouteCalculatorName>.

Metadata

Metadata

Error code	Exception	Old error message	New error message
500	InternalServerException	Internal Server Error	Internal server error. Try again later.

Error code	Exception	Old error message	New error message
		Error processing List request	
404	ResourceNotFoundException	APIKey not found	Api key not found: <APIKeyName>.
404	ResourceNotFoundException	APIKeyID not found	ApiKeyId not found: <APIKeyID>.
400	ValidationException	Either ExpireTime or NoExpiry must be provided	At least one of the following fields must be set: 'ExpireTime', 'NoExpiry'.
400	ValidationException	NoExpiry cannot be set to false if no ExpireTime is provided	'ExpireTime' must be set when 'NoExpiry' has value false.
400	ValidationException	ExpireTime cannot be set if NoExpiry is true	'ExpireTime' must not be set when 'NoExpiry' has value true.
400	ValidationException	Expire time '<ExpireTimeValue>' is not a valid time format	'ExpireTime' must follow the format YYYY-MM-DDThh:mm:ss.sssZ.
400	ValidationException	Expire time '<ExpireTimeValue>' cannot be in the past when creating a key	'ExpireTime' must not be in the past.
400	ValidationException	Invalid token	'NextToken' must be a valid token.

Error code	Exception	Old error message	New error message
400	ValidationException	Expired token	'NextToken' must not be expired.
400	ValidationException	The API Key %s has been recently used and the requested update may impact current usage. Specify ForceUpdate=true to update the API Key configuration.	This update may cause some users to lose API access. Because this API Key has been used in the last 7 days, you must set 'ForceUpdate' to true to confirm this change.
400	ValidationException	Expire time '<ExpireTimeValue>' must not be more than 1 minute in the past	'ExpireTime' must not be more than 1 minute in the past.
400	ValidationException	Description, ExpireTime, NoExpiry and Restrictions can't all be empty	At least one of the following fields must be set: 'Description', 'ExpireTime', 'NoExpiry', 'Restrictions'.
400	ValidationException	API Key expired	'ApiKeyId' must not be expired.
409	ConflictException	API key named <APIKeyName> already exists	Api key already exists: <APIKeyName>.

Geofences

Geofences

Error code	Exception	Old error message	New error message
500	InternalServerException	internal server error Internal server error Unsupported geofence geometry encountered geometry marshal error geometry load error unable to get geofence collection unable to delete geofences unable to retrieve geofence Error processing List request	Internal server error. Try again later.
404	ResourceNotFoundException	collection not found: <GeofenceCollectionName> <GeofenceCollectionName> geofence collection not found Resource not found error	Geofence Collection not found: <GeofenceCollectionName>.

Error code	Exception	Old error message	New error message
		no geofence with given name found	
400	ValidationException	unsupported price plan '<PricingPlan>'	'PricingPlan' must be set to RequestBasedUsage.
400	ValidationException	KMS key must be a symmetric CMK. Invalid usage type: <UsageType>	KMS key must be a symmetric Customer Master Key (CMK). Invalid usage type <UsageType>. For how to create a symmetric CMK, refer to https://docs.aws.amazon.com/kms/latest/developerguide/create-keys.html#create-symmetric-cmk.
400	ValidationException	Invalid token	'NextToken' must be a valid token.
400	ValidationException	Expired token	'NextToken' must not be expired.
400	ValidationException	PricingPlanDataSource cannot be updated without updating PricingPlan	'PricingPlan' must be provided to update 'PricingPlanDataSource'.
400	ValidationException	nothing to update	At least one of the following fields must be set: 'Description'

Error code	Exception	Old error message	New error message
400	ValidationException	invalid key state	KMS key must be a symmetric Customer Master Key (CMK). Invalid state <InvalidState>. For more informati on about how key state affects the use of a KMS key, see https://docs.aws.a mazon.com/kms/late st/developerguide/ key-state.html.
400	ValidationException	key not found	Invalid KMS key. '<KmsKeyId>' <KmsKeyIdValue> not found.
400	ValidationException	key is disabled	Symmetric Customer Master Key (CMK) must be enabled.
400	ValidationException	access denied	Symmetric Customer Master Key (CMK) must allow Amazon Location to create grants to its KMS key.
400	ValidationException	duplicate geofence ID in batch	'GeofenceId' <DuplicatedGeofenc eID> is duplicated in batch.
400	ValidationException	missing GeofenceId	'GeofenceId' must not be empty.

Error code	Exception	Old error message	New error message
400	ValidationException	Invalid token	'NextToken' must be a valid token.
400	ValidationException	Expired token	'NextToken' must not be expired.
400	ValidationException	Position[0] must be between -180 and 180	'Position[0]' must be between -180 and 180.
400	ValidationException	Position[1] must be between -90 and 90	'Position[1]' must be between -90 and 90.
400	ValidationException	radius must be less than or equal to 1000km	'Geometry.Circle.Radius' must be less than or equal to 1000km.
400	ValidationException	no geofence with given name found	Geofence not found: <CollectionName>.
400	ValidationException	Geometry must contain either a Circle or Polygon, not both	Only one of 'Circle' or 'Polygon' may be set within 'Geometry'.
400	ValidationException	Geometry must contain a Polygon or a Circle	One of 'Polygon' or 'Circle' must be set within 'Geometry'.
400	ValidationException	radius must be greater than 0m	'Geometry.Circle.Radius' must be greater than 0m.
400	ValidationException	empty polygon	'Geometry.Polygon' must not be empty.

Error code	Exception	Old error message	New error message
400	ValidationException	empty polygon ring	'Geometry.Polygon' must not be empty.
400	ValidationException	circle can not cross antimeridian	'Geometry.Circle' must not cross antimeridian. Cut it in two such that neither part's representation crosses the antimeridian.
400	ValidationException	polygon can not cross antimeridian	'Geometry.Polygon' must not cross antimeridian. Cut it in two such that neither part's representation crosses the antimeridian.
400	ValidationException	polygon can not have interior rings (holes), remove holes	'Geometry.Polygon' must not have interior rings (holes). For more information about interior rings see https://www.rfc-editor.org/rfc/rfc7946.html#appendix-A.3.
400	ValidationException	polygon ring is not closed	'Geometry.Polygon' contains an open ring. Close the ring by ensuring the first and last positions are equal.

Error code	Exception	Old error message	New error message
400	ValidationException	polygon ring has more than 1000 vertices	'Geometry.Polygon' must not have more than 1000 vertices.
400	ValidationException	polygon ring has fewer than 4 positions	Number of vertices in 'Geometry.Polygon' must be greater or equal to 4.
400	ValidationException	invalid center	'Geometry.Circle.Center' must be a valid position (longitude/latitude pair).
400	ValidationException	radius must be greater than 0m	'Geometry.Circle.Radius' must be greater than 0 m.
400	ValidationException	longitude range should be between -180 and 180 degrees	Longitude must be between -180 and 180 degrees, but was set to <Provided Longitude>.
400	ValidationException	latitude range should be between -90 and 90 degrees	Latitude must be between -90 and 90 degrees, but was set to <Provided Longitude>.
400	ValidationException	polygon exterior ring is expected to be counter clockwise	'Geometry.Polygon' must be oriented counter-clockwise.

Error code	Exception	Old error message	New error message
400	ValidationException	polygon interior ring should be clockwise oriented	'Geometry.Polygon' must be oriented clockwise.
400	ValidationException	radius must be less than or equal to 1000km	'Geometry.Circle.Radius' must be less than or equal to 1000 km.
400	ValidationException	timestamp.Parse() error	'SampleTime' must follow the format YYYY-MM-DDThh:mm:ss.sssZ.
400	ValidationException	invalid input	'SourceArn' must refer to a tracker resource.
400	ValidationException	arn: invalid prefix	'SourceArn' must be a valid ARN. For more information, see https://docs.aws.amazon.com/general/latest/gr/AWS-arns-and-namespaces.html.
400	ValidationException	arn: not enough sections	'SourceArn' must be a valid ARN. For more information, see https://docs.aws.amazon.com/general/latest/gr/AWS-arns-and-namespaces.html.

Error code	Exception	Old error message	New error message
400	ValidationException	invalid resource part	'SourceArn' must refer to a tracker resource.
402	ServiceQuotaExceededException	Geofence collections per account exceeded quota limits. For more info, see https://aws.amazon.com/premiumsupport/knowledge-center/manage-service-limits/	Geofence collection resources have exceeded the quota per account per region. For more information, see https://aws.amazon.com/premiumsupport/knowledge-center/manage-service-limits/.
409	ConflictException	collection already exists: <GeofenceCollectionName>	Geofence Collection already exists: <GeofenceCollectionName>.
409	ConflictException	Resource conflict error	Geofence already exists: <GeofenceName>.

Code examples and tutorials for working with Amazon Location Service

This topic shows a list of code examples, tutorials, and blog posts to help you learn about Amazon Location Service. Each code example includes a description of how it works.

You can find additional samples on the AWS Geospatial GitHub page, the AWS samples GitHub page for Amazon Location, and on the AWS blog site.

Note

It is good to understand the difference between the AWS Geospatial GitHub page and the AWS samples GitHub page.

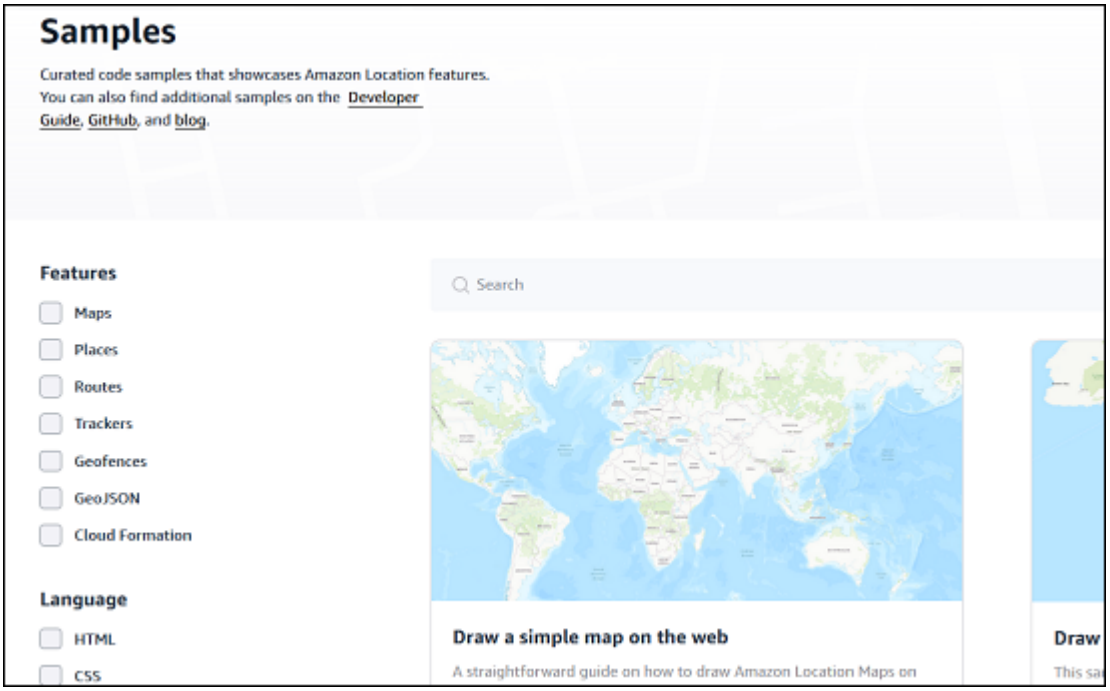
- **Geospatial GitHub** – The AWS Geospatial GitHub page includes samples that are created and maintained by the Amazon Location Service team.
- **Samples GitHub** – The AWS samples GitHub page for Amazon Location includes samples that were created for Amazon Location, but may or may not be actively maintained.

The quick start tutorial is a good place to start before using other samples, as it shows how to complete prerequisites that are useful for most of the samples.

Topics

- Amazon Location Demo site
- Tutorial: Quick start
- Tutorial: Database enrichment
- Example: Explore app
- Example: Style a map
- Example: Draw markers
- Example: Draw clustered points
- Example: Draw a polygon
- Example: Change the map language
- Blog: Estimated delivery time notifications
- Example: Stream Position Updates
- Example: Geofencing and Tracking mobile application

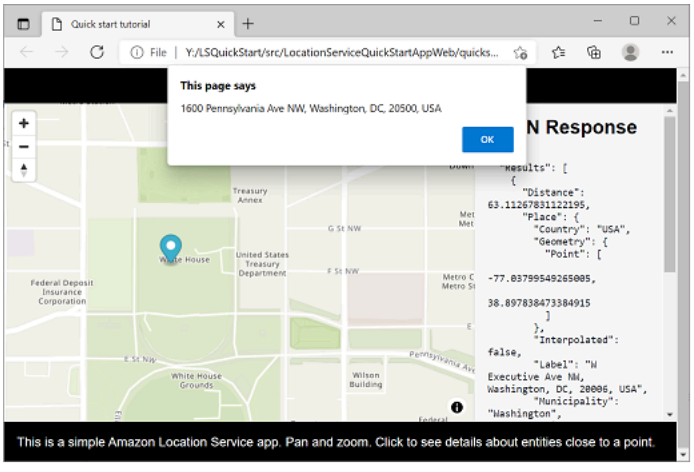
Amazon Location Demo site

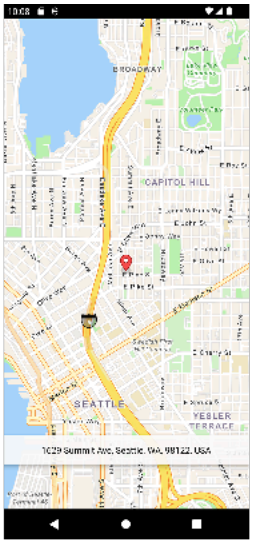


You can see demos with source code of Amazon Location Service in action at the Amazon Location Demo site. This site includes a hosted web demo, and also a demo app for Android.

You can also find a wide array of samples, filterable by features, language, and platform in the site's Samples page.

Tutorial: Quick start

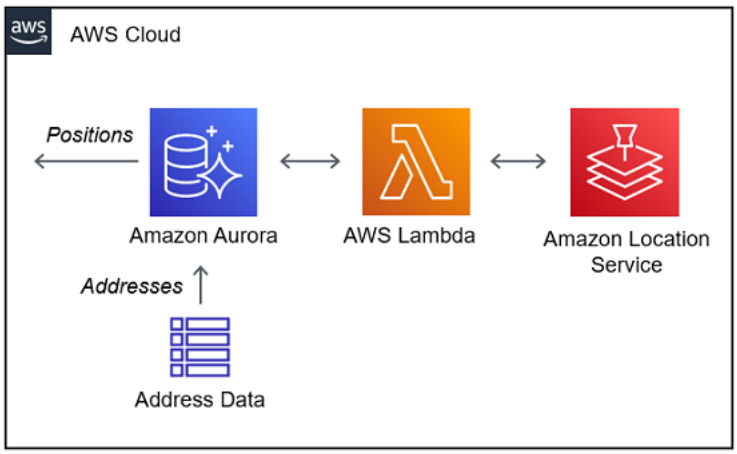




There are quick start tutorials available for web, iOS, and Android devices. For each platform, the tutorial shows you how to add an interactive map to an application, and how to make calls to the Amazon Location Service APIs from your application. The tutorial is available for JavaScript in a static webpage, Kotlin for an Android phone application, or Swift for an iOS application.

- JavaScript for a static webpage documentation link: Creating a web app
- Kotlin for an Android application documentation link: Quick start with Amazon Location Service
- Swift for an iOS app documentation link: Creating an iOS app

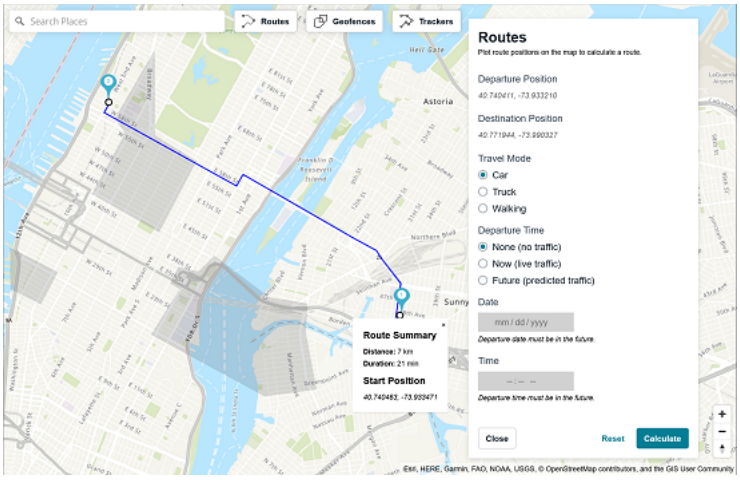
Tutorial: Database enrichment



This tutorial shows you how to use Amazon Location Service, called from AWS Lambda to normalize addresses and add latitude and longitude to records in an Amazon Aurora database. Uses Amazon Aurora and AWS Lambda.

Documentation link: Amazon Aurora PostgreSQL user-defined functions for Amazon Location Service

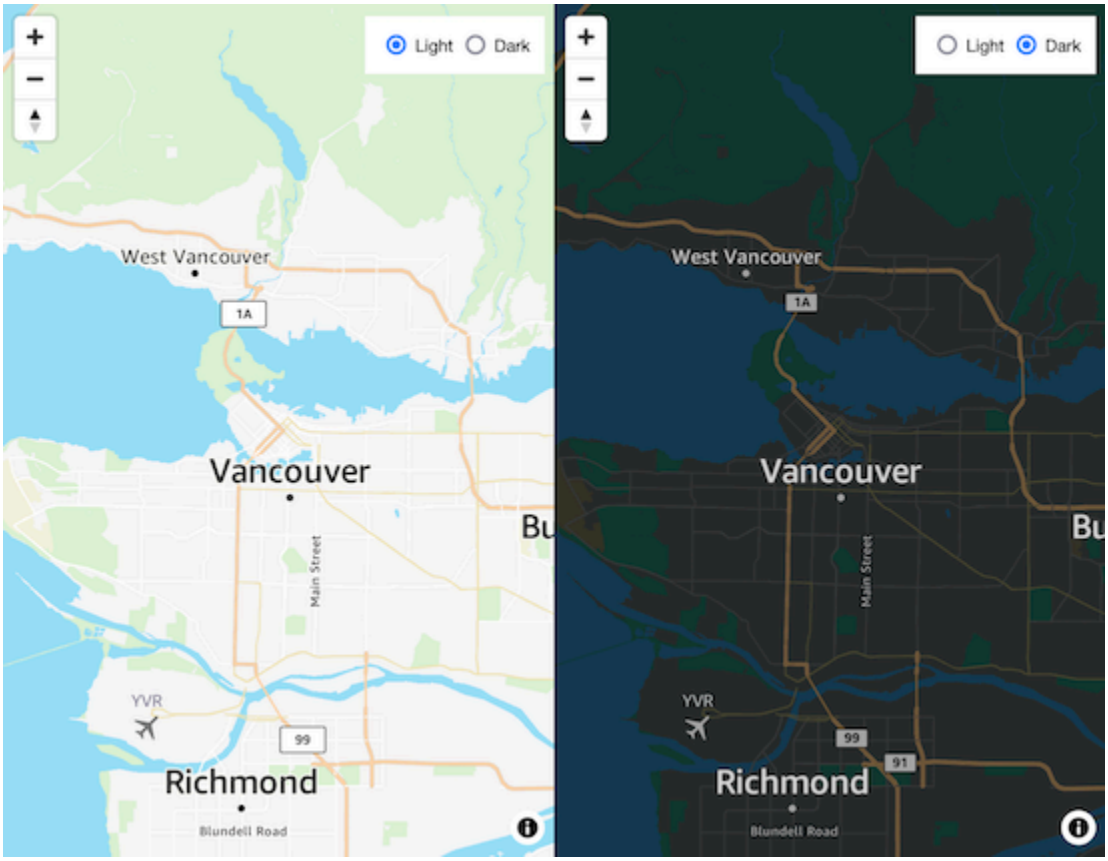
Example: Explore app



One of the best ways to learn about the functionality of Amazon Location Service is to use the Explore functionality within the Amazon Location console. This full web application example mimics the maps, places, routes, geofences, and trackers functionality from the console to show you how to recreate these features in your own app. Uses Amplify, React, and JavaScript.

Samples GitHub link: Explore sample application

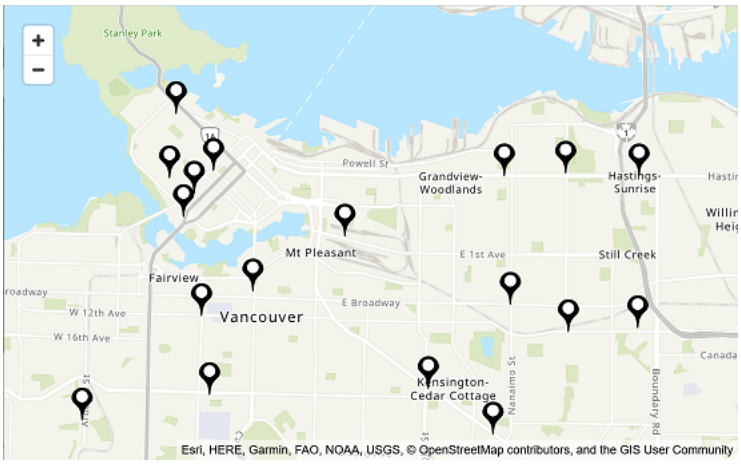
Example: Style a map



This code example shows how to switch between a satellite map and a vector road map, using MapLibre in JavaScript. Uses MapLibre, the Amazon Location authentication helper, and JavaScript.

Geospatial GitHub link: Interactive map with style switching

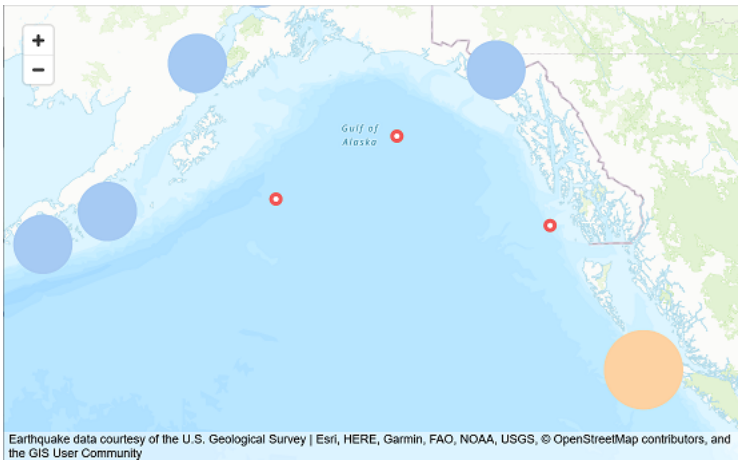
Example: Draw markers



This code example shows Amazon Locker locations in Vancouver, BC, Canada. It shows how to draw markers at point locations. Uses MapLibre, Node.js, React, the Amazon Location authentication helper and JavaScript.

Geospatial GitHub link: Interactive map with markers at points

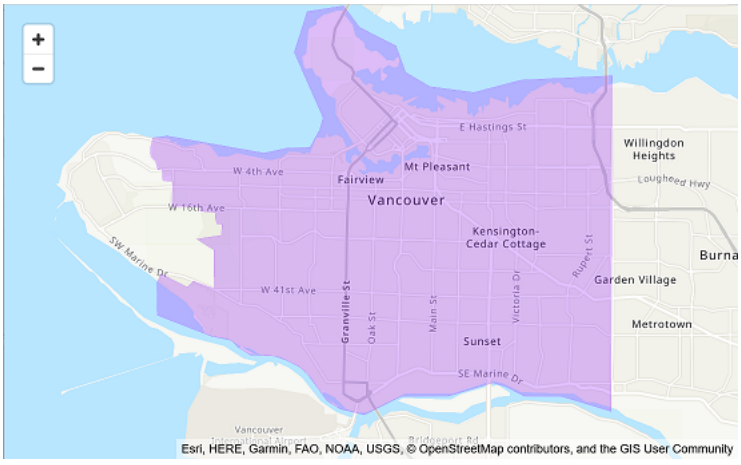
Example: Draw clustered points



Using USGS earthquake data, this code example shows how to draw points that cluster together when they are close together on the map. Uses MapLibre, Node.js, React, Amplify, and JavaScript.

Samples GitHub link: Interactive map with clusters of points

Example: Draw a polygon



This code example shows how to draw a polygon on the map. Uses MapLibre, Node.js, React, the Amazon Location authentication helper, and JavaScript.

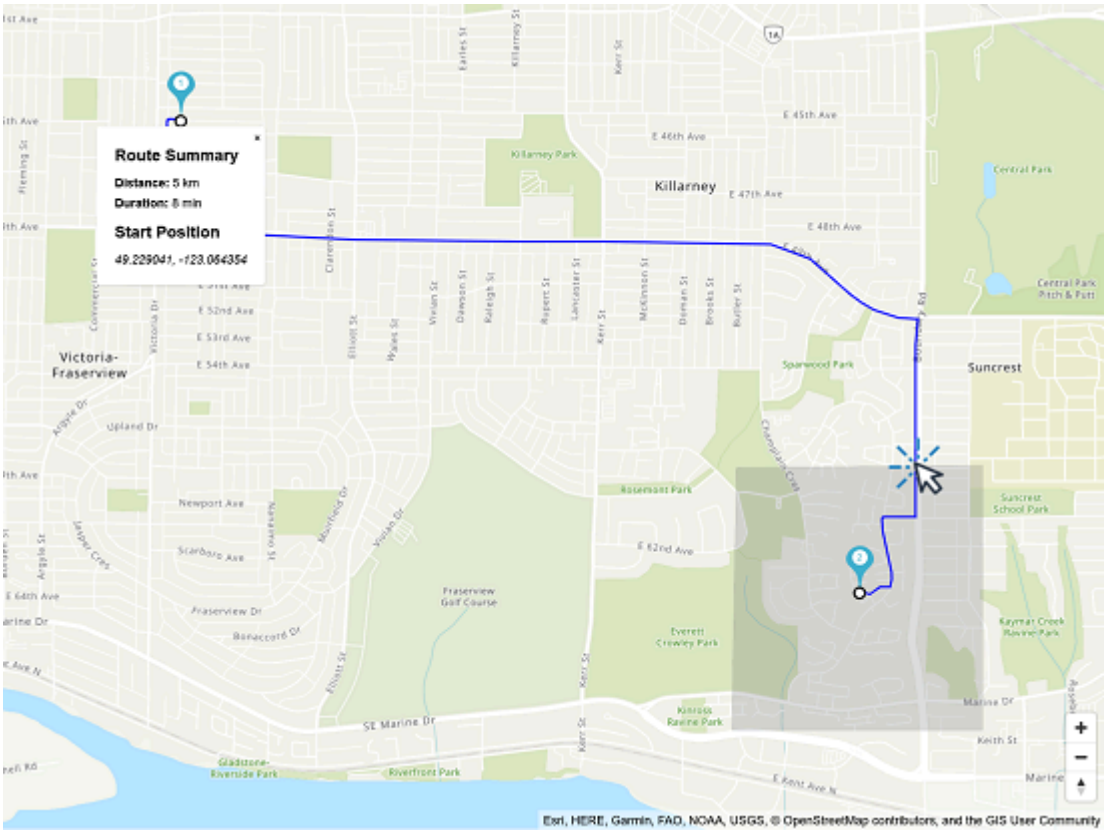
Geospatial GitHub link: Interactive map with polygons

Example: Change the map language

This code example shows how you can change the display language of maps in Amazon Location. Uses Amplify, React, and MapLibre.

Samples GitHub link: [Change Map Language Sample]

Blog: Estimated delivery time notifications



This blog post shows different ways to notify customers with estimated delivery times. It explains using routes to show estimated driving time, and then using trackers and geofences to notify when a driver gets close to the customer. Uses Amplify, React, Amazon EventBridge, and Amazon Simple Notification Service (Amazon SNS).

Blog link: Estimated Time of Arrival and Proximity Notifications

Example: Stream Position Updates

Kinesis Stream To Tracker App: This sample demonstrates how to use Kinesis Data Stream to post tracker updates with Amazon Location Service. The sample is a deployable lambda application written in python that can be integrated with a Kinesis Data Stream to consume the Kinesis events and batch update device positions.

Repository link: Amazon Location Amazon Kinesis Data Streams Stream To Tracker App

For more information on tracking and geofences, see the Geofences and Trackers documentation. Developers can deploy the app by following the AWS's Serverless Application Repository documentation, or dicretly from the Lambda console.

Device Position Streaming Sample App: This code example shows how to stream device position data to a Kinesis Data Stream and how geofence notifications work. This app depends on the Kinesis Stream to Tracker Sample App, listed above, to be running for the streamed tracker positions to be updated in Amazon Location Service.

Repository link: Amazon Location Device Position Streaming Sample App

Example: Geofencing and Tracking mobile application

This sample application shows how a tracker and geofence interact using a combination of Lambda, AWS IoT and Amazon Location features. There are tutorials avaible for iOS and Android.

Tutorial link: Sample Geofence and Tracker mobile application

How to use Amazon Location Service

You can use Amazon Location Service capabilities to complete geographic and location-related tasks. You can then combine these tasks to address more complex uses cases such as geomarketing, delivery, and asset tracking.

When you're ready to build location features into your application, use the following methods to use the Amazon Location Service functionality, depending on your goals and inclinations:

- **Exploration tools** – If you want to experiment with Amazon Location resources, the following tools are the fastest way to access and try out the APIs:
  - The Amazon Location console provides a variety of quick-access tools. You can create and manage your resources and try the APIs using the Explore page. The console is also useful for creating resources (typically a one-time task) in preparation for using any of the other methods described later.
  - The AWS Command Line Interface (CLI) lets you create resources and access the Amazon Location APIs using a terminal. The AWS CLI handles authentication when you configure it with your credentials.
  - You can see code examples and tutorials that show how to perform tasks using the Amazon Location Service APIs. This includes an example that mimics much of the functionality of the Explore page in the console.
- **Platform SDKs** – If you aren't visualizing data on a map, you can use any of the AWS standard tools to build on AWS.
  - The following SDKs are available: C++, Go, Java, JavaScript, .NET, Node.js, PHP, Python, and Ruby.
- **Frontend SDKs and libraries** – If you want to use Amazon Location to build an application on a mobile platform or visualize data on a map on any platform, you have the following options:
  - The AWS Amplify libraries integrate Amazon Location within iOS, Android, and JavaScript web applications.
  - The MapLibre libraries let you render client-side maps into iOS, Android, and JavaScript web applications using Amazon Location.
  - Tangram ES libraries enable you to render 2D and 3D maps from vector data using OpenGL ES within iOS and Android web applications. There is also Tangram for JavaScript web applications.

- **Sending direct HTTPS requests** – If you are working with a programming language for which there is no SDK available, or if you want more control over how you send a request to AWS, you can access Amazon Location by sending direct HTTPS requests authenticated by the Signature Version 4 signing process. For more information on the Signature Version 4 signing process, see the *AWS General Reference*.

This chapter describes many of the tasks that are common to applications using location data. The common use cases section describes how to combine these with other AWS services to achieve more complex use cases.

Topics

- Prerequisites for using Amazon Location Service
- Using Amazon Location Maps in your application
- Searching place and geolocation data using Amazon Location
- Calculating routes using Amazon Location Service
- Geofencing an area of interest using Amazon Location
- Tagging your Amazon Location Service resources
- Granting access to Amazon Location Service
- Monitoring Amazon Location Service
- Creating Amazon Location Service resources with AWS CloudFormation

Prerequisites for using Amazon Location Service

This section describes what you need to do to use Amazon Location Service. You must have an AWS account and have set up access to Amazon Location for users that want to use it.

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Grant access to Amazon Location Service

Your non-admin users have no permissions by default. Before they can access Amazon Location, you must grant permission by attaching an IAM policy with specific permissions. Make sure to follow the principle of least privilege when granting access to resources.

> **Note**
>
> For information about giving unauthenticated users access to Amazon Location Service functionality (for example, in a web-based application), see Granting access to Amazon Location Service.

The following example policy gives a user permission to access all Amazon Location operations. For more examples, see Identity-based policy examples for Amazon Location Service.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "geo:*"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

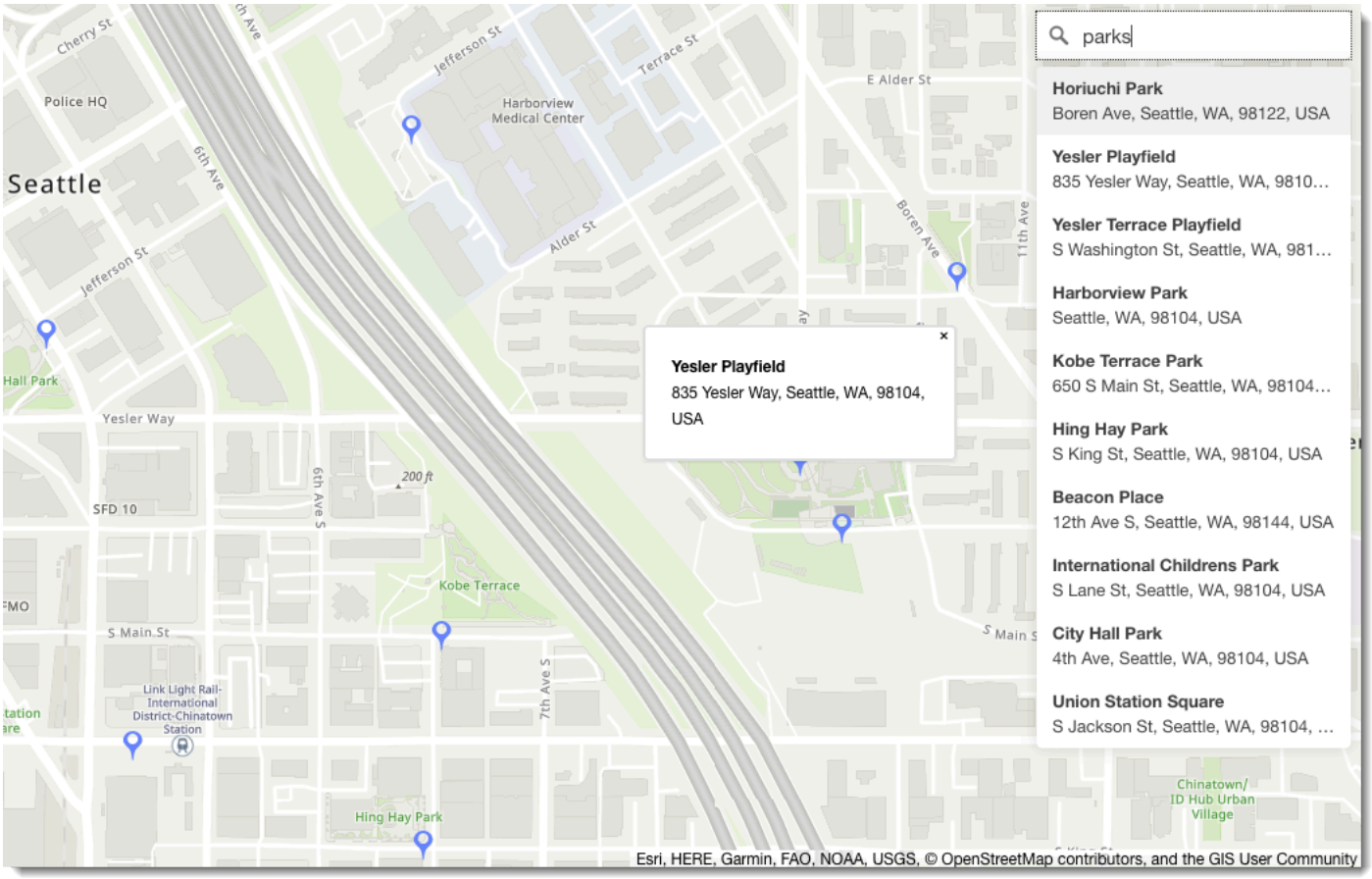
  Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.
- Users managed in IAM through an identity provider:

  Create a role for identity federation. Follow the instructions in Creating a role for a third-party identity provider (federation) in the *IAM User Guide*.
- IAM users:
  - Create a role that your user can assume. Follow the instructions in Creating a role for an IAM user in the *IAM User Guide*.
  - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in Adding permissions to a user (console) in the *IAM User Guide*.

When creating applications that use Amazon Location Service, you may need some users to have unauthenticated access. For these use cases, see Enabling unauthenticated access using Amazon Cognito.

Using Amazon Location Maps in your application

Amazon Location maps are cost-effective and interactive. You can replace an existing map in your application to save money, or add a new one to display location-based data visually, such as your store location.



Amazon Location Service lets you choose a data provider for map operations by creating and configuring a map resource. The map resource configures the data provider and the style that is used to render the map.

After you create your resource, you can send requests by using the AWS SDK directly, or by using a library made specifically for rendering maps in your environment.

> **Note**
>
> For an overview of map concepts, see Maps.

Topics

- Prerequisites
- Display a map in your application
- Drawing data features on a map
- Setting extents for a map using MapLibre

- Managing your map resources

Prerequisites

Before you display a map in your application follow the prerequisite steps:

Topics

- Create a map resource
- Authenticating your requests

Create a map resource

To use a map in your application you must have a map resource, which specifies the map style and data provider to use in your maps.

> **Note**
>
> If your application is tracking or routing assets you use in your business, such as delivery vehicles or employees, you must not use Esri as your geolocation provider. See section 82 of the AWS service terms for more details.

You can create a map resource using the Amazon Location Service console, the AWS CLI, or the Amazon Location APIs.

Console

To create a map resource using the Amazon Location Service console

1. In the Amazon Location console, on the **Maps** page, choose **Create map** to preview map styles.
2. Add a name and description for the new map resource.
3. Choose a map style.

> **Note**
>
> If your application is tracking or routing assets you use in your business, such as delivery vehicles or employees, you must not use Esri as your geolocation provider. See section 82 of the AWS service terms for more details.

4. Choose from the Political views to use.
5. Agree to the **Amazon Location Terms and Conditions**, then choose **Create map**. You can interact with the map that you've chosen: zoom in, zoom out, or pan in any direction.
6. To allow your users to switch styles (for example, to allow them to switch between satellite imagery and vector style), you must create a map resource for each style.

   You can delete resources with map styles that you don't want to use on the Maps home page in the console.

API

To create a map resource using the Amazon Location APIs

Use the `CreateMap` operation from the Amazon Location APIs.

The following example is an API request to create a map resource called *`ExampleMap`* using the *`VectorEsriStreets`* map style.

```
POST /maps/v0/maps HTTP/1.1
Content-type: application/json

{
   "Configuration": {
      "Style": "VectorEsriStreets"
   },
   "MapName": "ExampleMap"
   }
}
```

> **Note**
>
> If your application is tracking or routing assets you use in your business, such as delivery vehicles or employees, you must not use Esri as your geolocation provider. See section 82 of the AWS service terms for more details.

AWS CLI

To create a map resource using AWS CLI commands

Use the `create-map` command.

The following example creates a map resource called *ExampleMap* using *VectorEsriStreets* as the map style.

```
aws location \
  create-map \
  --configuration Style="VectorEsriStreets" \
  --map-name "ExampleMap"
```

> **Note**
>
> If your application is tracking or routing assets you use in your business, such as delivery vehicles or employees, you must not use Esri as your geolocation provider. See section 82 of the AWS service terms for more details.

Authenticating your requests

Once you create a map resource and you're ready to begin building location features into your application, you need to choose how you would authenticate your requests.

> **Note**
>
> Most maps front end applications require unauthenticated access to the maps or other features of Amazon Location Service. Depending on your application, you might want to use AWS Signature v4 to authenticate requests, or you can use Amazon Cognito or Amazon

Location API keys for unauthenticated use. To learn more about all of these options, see Granting access to Amazon Location Service.

Display a map in your application

This section provides tutorials on how to use map rendering tools to display a map in your mobile or web application when using Amazon Location APIs. As mentioned in the How to use Amazon Location Service topic, you have a choice of libraries to use when rendering maps with Amazon Location, including Amplify, MapLibre, and Tangram.

Do one of the following to display a map in your application:

- The most direct way to display a map in your web and mobile front end applications is to use MapLibre. You can follow the MapLibre tutorials or even the Quick start tutorial to learn how to use MapLibre.
- If you are an existing AWS Amplify developer, you may want to use the Amplify Geo SDK. To learn more, follow the Amplify tutorial.
- If you are an existing user of Tangram, and want to continue to use it to render your map, while moving to Amazon Location Service, follow the Tangram tutorial.

Topics

- Using the MapLibre library with Amazon Location Service
- Using the Amplify library with Amazon Location Service
- Using Tangram with Amazon Location Service

Using the MapLibre library with Amazon Location Service

The following tutorials walk you through using the MapLibre Library with Amazon Location.

Topics

- Using MapLibre GL JS with Amazon Location Service
- Using the MapLibre Native SDK for Android with Amazon Location Service
- Using the MapLibre Native SDK for iOS with Amazon Location Service

Using MapLibre GL JS with Amazon Location Service

Use MapLibre GL JS to embed client-side maps into web applications.

MapLibre GL JS is an open-source JavaScript library that's compatible with the styles and tiles provided by the Amazon Location Service Maps API. You can integrate MapLibre GL JS within a basic HTML or JavaScript application to embed customizable and responsive client-side maps.

This tutorial describes how to integrate MapLibre GL JS with Amazon Location within a basic HTML and JavaScript application. The same libraries and techniques presented in this tutorial also apply to frameworks, such as React and Angular.

The sample application for this tutorial is available as part of the Amazon Location Service samples repository on GitHub.

Building the application: Scaffolding

This tutorial creates a web application that uses JavaScript to build a map on an HTML page.

Begin by creating an HTML page (`index.html`) that includes the map's container:

- Enter a `div` element with an `id` of `map` to apply the map's dimensions to the map view. The dimensions are inherited from the viewport.

```
<html>
  <head>
    <style>
      body {
        margin: 0;
      }

      #map {
        height: 100vh; /* 100% of viewport height */
      }
    </style>
  </head>
  <body>
    <!-- map container -->
    <div id="map" />
  </body>
</html>
```

Building the application: Adding dependencies

Add the following dependencies to your application:

- MapLibre GL JS (v3.x), and its associated CSS.
- The Amazon Location JavaScript Authentication helper.

```
<!-- CSS dependencies -->
<link
  href="https://unpkg.com/maplibre-gl@3.x/dist/maplibre-gl.css"
  rel="stylesheet"
/>
<!-- JavaScript dependencies -->
<script src="https://unpkg.com/maplibre-gl@3.x/dist/maplibre-gl.js"></script>
<script src="https://unpkg.com/@aws/amazon-location-authentication-helper.js"></script>
<script>
  // application-specific code
</script>
```

This creates an empty page with the map's container.

Building the application: Configuration

To configure your application using JavaScript:

1. Enter the names and identifiers of your resources.

   ```
   // Cognito Identity Pool ID
   const identityPoolId = "us-east-1:54f2ba88-9390-498d-aaa5-0d97fb7ca3bd";
   // Amazon Location Service Map name
   const mapName = "ExampleMap";
   ```

2. Instantiate a credential provider using the unauthenticated identity pool you created in Using maps - Step 2, Set up authentication. We will put this in a function called `initializeMap`, that will also contain other map initialization code, added in the next step

   ```
   // extract the Region from the Identity Pool ID; this will be used for both Amazon
    Cognito and Amazon Location
   AWS.config.region = identityPoolId.split(":")[0];

   async function initializeMap() {
   ```

```
  // Create an authentication helper instance using credentials from Cognito
  const authHelper = await
 amazonLocationAuthHelper.withIdentityPoolId(identityPoolId);

  // ... more here, later
}
```

Building the application: Map initialization

For the map to display after the page is loaded, you must initialize the map. You can adjust the initial map location, add additional controls, and overlay data.

```
async function initializeMap() {
  // Create an authentication helper instance using credentials from Cognito
  const authHelper = await amazonLocationAuthHelper.withIdentityPoolId(identityPoolId);

  // Initialize the map
  const map = new maplibregl.Map({
    container: "map",
    center: [-123.1187, 49.2819], // initial map centerpoint
    zoom: 10, // initial map zoom
    style: 'https://maps.geo.${region}.amazonaws.com/maps/v0/maps/${mapName}/style-
descriptor',
    ...authHelper.getMapAuthenticationOptions(), // authentication, using cognito
  });

  map.addControl(new maplibregl.NavigationControl(), "top-left");
}

initializeMap();
```

> **Note**
>
> You must provide word mark or text attribution for each data provider that you use, either on your application or your documentation. Attribution strings are included in the style descriptor response under the `sources.esri.attribution`, `sources.here.attribution`, and `sources.grabmaptiles.attribution` keys. MapLibre GL JS will automatically provide attribution. When using Amazon Location resources with data providers, make sure to read the service terms and conditions.

Running the application

You can run this sample application by using it in a local web server, or opening it in a browser.

To use a local web server, you can use npx, because it's installed as part of Node.js. You can use `npx serve` from within the same directory as `index.html`. This serves the application on `localhost:5000`.

> **Note**
>
> If the policy you created for your unauthenticated Amazon Cognito role includes a `referer` condition, you might be blocked from testing with `localhost:` URLs. In this case. you can test with a web server that provides a URL that is in your policy.

After completing the tutorial, the final application looks like the following example.

```
<!-- index.html -->
<html>
  <head>
    <link href="https://unpkg.com/maplibre-gl@3.x/dist/maplibre-gl.css"
 rel="stylesheet" />
    <style>
      body {
        margin: 0;
      }
      #map {
        height: 100vh;
      }
    </style>
  </head>

  <body>
    <!-- map container -->
    <div id="map" />
    <!-- JavaScript dependencies -->
    <script src="https://unpkg.com/maplibre-gl@3.x/dist/maplibre-gl.js"></script>
    <script src="https://unpkg.com/@aws/amazon-location-authentication-helper.js"></
script>
    <script>
      // configuration
```

```
      const identityPoolId = "us-east-1:54f2ba88-9390-498d-aaa5-0d97fb7ca3bd"; //
 Cognito Identity Pool ID
      const mapName = "ExampleMap"; // Amazon Location Service Map Name

      // extract the region from the Identity Pool ID
      const region = identityPoolId.split(":")[0];

      async function initializeMap() {
        // Create an authentication helper instance using credentials from Cognito
        const authHelper = await
 amazonLocationAuthHelper.withIdentityPoolId(identityPoolId);

        // Initialize the map
        const map = new maplibregl.Map({
          container: "map",
          center: [-123.115898, 49.295868],
          zoom: 10,
          style: `https://maps.geo.${region}.amazonaws.com/maps/v0/maps/${mapName}/
style-descriptor`,
          ...authHelper.getMapAuthenticationOptions(),
        });
        map.addControl(new maplibregl.NavigationControl(), "top-left");
      }

      initializeMap();
    </script>
  </body>
</html>
```

Running this application displays a full-screen map using your chosen map style. This sample is available in the Amazon Location Service samples repository on GitHub.

Using the MapLibre Native SDK for Android with Amazon Location Service

Use MapLibre Native SDK to embed interactive maps into your Android applications.

The MapLibre Native SDK for Android is a library based on Mapbox Native, and is compatible with the styles and tiles provided by the Amazon Location Service Maps API. You can integrate MapLibre Native SDK for Android to embed interactive map views with scalable, customizable vector maps in your Android applications.

This tutorial describes how to integrate the MapLibre Native SDK for Android with Amazon Location. The sample application for this tutorial is available as part of the Amazon Location Service samples repository on GitHub.

Building the application: Initialization

To initialize your application:

1. Create a new Android Studio project from the **Empty Activity** template.
2. Ensure that **Kotlin** is selected for the project language.
3. Select a **Minimum SDK of API 14: Android 4.0 (Ice Cream Sandwich)** or newer.
4. Open **Project Structure**, then go to **File** > **Project Structure...** to choose the **Dependencies** section.
5. With **<All Modules>** selected,then choose the **+** button to add a new **Library Dependency**.
6. Add **AWS Android SDK** version 2.20.0 or later. For example: `com.amazonaws:aws-android-sdk-core:2.20.0`
7. Add the **MapLibre Native SDK for Android** version 9.4.0 or later. For example: `org.maplibre.gl:android-sdk:9.4.0`
8. At the project level of your **build.gradle** file, add the following maven repository to access the MapLibre packages for Android:

```
allprojects {
    repositories {
        // Retain your existing repositories
        google()
        jcenter()

        //  Declare the repositories for MapLibre
        mavenCentral()
    }
}
```

Building the application: Configuration

To configure your application with your resources and AWS Region:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
```

```
    <string name="identityPoolId">us-east-1:54f2ba88-9390-498d-aaa5-0d97fb7ca3bd</
string>
    <string name="mapName">ExampleMap</string>
    <string name="awsRegion">us-east-1</string>
</resources>
```

Building the application: Activity layout

Edit `app/src/main/res/layout/activity_main.xml`:

- Add a `MapView`, which renders the map. This will also set the map's initial center point.
- Add a `TextView`, which displays attribution.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <com.mapbox.mapboxsdk.maps.MapView
        android:id="@+id/mapView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:mapbox_cameraTargetLat="49.2819"
        app:mapbox_cameraTargetLng="-123.1187"
        app:mapbox_cameraZoom="12"
        app:mapbox_uiAttribution="false"
        app:mapbox_uiLogo="false" />

    <TextView
        android:id="@+id/attributionView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#80808080"
        android:padding="5sp"
        android:textColor="@android:color/black"
        android:textSize="10sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
```

```
        tools:ignore="SmallSp" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

> **Note**
>
> You must provide word mark or text attribution for each data provider that you use, either on your application or your documentation. Attribution strings are included in the style descriptor response under the `sources.esri.attribution`, `sources.here.attribution`, and `source.grabmaptiles.attribution` keys. When using Amazon Location resources with data providers, make sure to read the service terms and conditions.

Building the application: Request transformation

Create a class named `SigV4Interceptor` to intercept AWS requests and sign them using Signature Version 4. This will be registered with the HTTP client used to fetch map resources when the Main Activity is created.

```
package aws.location.demo.okhttp

import com.amazonaws.DefaultRequest
import com.amazonaws.auth.AWS4Signer
import com.amazonaws.auth.AWSCredentialsProvider
import com.amazonaws.http.HttpMethodName
import com.amazonaws.util.IOUtils
import okhttp3.HttpUrl
import okhttp3.Interceptor
import okhttp3.Request
import okhttp3.Response
import okio.Buffer
import java.io.ByteArrayInputStream
import java.net.URI

class SigV4Interceptor(
    private val credentialsProvider: AWSCredentialsProvider,
    private val serviceName: String
) : Interceptor {
    override fun intercept(chain: Interceptor.Chain): Response {
        val originalRequest = chain.request()
```

```
        if (originalRequest.url().host().contains("amazonaws.com")) {
            val signer = if (originalRequest.url().encodedPath().contains("@")) {
                // the presence of "@" indicates that it doesn't need to be double URL-
encoded
                AWS4Signer(false)
            } else {
                AWS4Signer()
            }

            val awsRequest = toAWSRequest(originalRequest, serviceName)
            signer.setServiceName(serviceName)
            signer.sign(awsRequest, credentialsProvider.credentials)

            return chain.proceed(toSignedOkHttpRequest(awsRequest, originalRequest))
        }

        return chain.proceed(originalRequest)
    }

    companion object {
        fun toAWSRequest(request: Request, serviceName: String): DefaultRequest<Any> {
            // clone the request (AWS-style) so that it can be populated with
 credentials
            val dr = DefaultRequest<Any>(serviceName)

            // copy request info
            dr.httpMethod = HttpMethodName.valueOf(request.method())
            with(request.url()) {
                dr.resourcePath = uri().path
                dr.endpoint = URI.create("${scheme()}://${host()}")

                // copy parameters
                for (p in queryParameterNames()) {
                    if (p != "") {
                        dr.addParameter(p, queryParameter(p))
                    }
                }
            }

            // copy headers
            for (h in request.headers().names()) {
                dr.addHeader(h, request.header(h))
            }
```

```
            // copy the request body
            val bodyBytes = request.body()?.let { body ->
                val buffer = Buffer()
                body.writeTo(buffer)
                IOUtils.toByteArray(buffer.inputStream())
            }

            dr.content = ByteArrayInputStream(bodyBytes ?: ByteArray(0))

            return dr
        }

        fun toSignedOkHttpRequest(
            awsRequest: DefaultRequest<Any>,
            originalRequest: Request
        ): Request {
            // copy signed request back into an OkHttp Request
            val builder = Request.Builder()

            // copy headers from the signed request
            for ((k, v) in awsRequest.headers) {
                builder.addHeader(k, v)
            }

            // start building an HttpUrl
            val urlBuilder = HttpUrl.Builder()
                .host(awsRequest.endpoint.host)
                .scheme(awsRequest.endpoint.scheme)
                .encodedPath(awsRequest.resourcePath)

            // copy parameters from the signed request
            for ((k, v) in awsRequest.parameters) {
                urlBuilder.addQueryParameter(k, v)
            }

            return builder.url(urlBuilder.build())
                .method(originalRequest.method(), originalRequest.body())
                .build()
        }
    }
}
```

Building the application: Main activity

The Main Activity is responsible for initializing the views that will be displayed to users. This involves:

- Instantiating an Amazon Cognito `CredentialsProvider`.
- Registering the Signature Version 4 interceptor.
- Configuring the map by pointing it at a map style descriptor, and displaying appropriate attribution.

`MainActivity` is also responsible for forwarding life cycle events to the map view, allowing it to preserve the active viewport between invocations.

```
package aws.location.demo.maplibre

import android.os.Bundle
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import aws.location.demo.okhttp.SigV4Interceptor
import com.amazonaws.auth.CognitoCachingCredentialsProvider
import com.amazonaws.regions.Regions
import com.mapbox.mapboxsdk.Mapbox
import com.mapbox.mapboxsdk.maps.MapView
import com.mapbox.mapboxsdk.maps.Style
import com.mapbox.mapboxsdk.module.http.HttpRequestUtil
import okhttp3.OkHttpClient

private const val SERVICE_NAME = "geo"

class MainActivity : AppCompatActivity() {
    private var mapView: MapView? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // configuration
        val identityPoolId = getString(R.string.identityPoolId)
        val region = getString(R.string.awsRegion)
        val mapName = getString(R.string.mapName)

        // Credential initialization
```

```
        val credentialProvider = CognitoCachingCredentialsProvider(
            applicationContext,
            identityPoolId,
            Regions.fromName(identityPoolId.split(":").first())
        )

        // initialize MapLibre
        Mapbox.getInstance(this, null)
        HttpRequestUtil.setOkHttpClient(
            OkHttpClient.Builder()
                .addInterceptor(SigV4Interceptor(credentialProvider, SERVICE_NAME))
                .build()
        )

        // initialize the view
        setContentView(R.layout.activity_main)

        // initialize the map view
        mapView = findViewById(R.id.mapView)
        mapView?.onCreate(savedInstanceState)
        mapView?.getMapAsync { map ->
            map.setStyle(
                Style.Builder()
                    .fromUri("https://maps.geo.${region}.amazonaws.com/maps/v0/maps/
${mapName}/style-descriptor")
            ) { style ->
                findViewById<TextView>(R.id.attributionView).text =
 style.sources.first()?.attribution
            }
        }
    }

    override fun onStart() {
        super.onStart()
        mapView?.onStart()
    }

    override fun onResume() {
        super.onResume()
        mapView?.onResume()
    }

    override fun onPause() {
        super.onPause()
```

```
        mapView?.onPause()
    }

    override fun onStop() {
        super.onStop()
        mapView?.onStop()
    }

    override fun onSaveInstanceState(outState: Bundle) {
        super.onSaveInstanceState(outState)
        mapView?.onSaveInstanceState(outState)
    }

    override fun onLowMemory() {
        super.onLowMemory()
        mapView?.onLowMemory()
    }

    override fun onDestroy() {
        super.onDestroy()
        mapView?.onDestroy()
    }
}
```

Running this application displays a full-screen map in the style of your choosing. This sample is available as part of the Amazon Location Service samples repository on GitHub.

Using the MapLibre Native SDK for iOS with Amazon Location Service

Use MapLibre Native SDK for iOS to embed client-side maps into iOS applications.

The MapLibre Native SDK for iOS is a library based on Mapbox GL Native, and is compatible with the styles and tiles provided by the Amazon Location Service Maps API. You can integrate MapLibre Native SDK for iOS to embed interactive map views with scalable, customizable vector maps into your iOS applications.

This tutorial describes how to integrate the MapLibre Native SDK for iOS with Amazon Location. The sample application for this tutorial is available as part of the Amazon Location Service samples repository on GitHub.

Building the application: Initialization

To initialize your application:

1. Create a new Xcode project from the **App** template.
2. Select **SwiftUI** for its interface.
3. Select **SwiftUI** application for its Life Cycle.
4. Select **Swift** for its language.

Adding MapLibre dependencies using Swift Packages

To add a package dependency to your Xcode project:

1. Navigate to **File** > **Swift Packages** > **Add Package Dependency**.
2. Enter the repository URL: **`https://github.com/maplibre/maplibre-gl-native-distribution`**

> **Note**
>
> For more information about Swift Packages see Adding Package Dependencies to Your App at Apple.com

3. In your terminal, install CocoaPods:

```
sudo gem install cocoapods
```

4. Navigate to your application's project directory and initialize the **Podfile** with the CocoaPods package manager:

```
pod init
```

5. Open the **Podfile** to add `AWSCore` as a dependency:

```
platform :ios, '12.0'

target 'Amazon Location Service Demo' do
  use_frameworks!

  pod 'AWSCore'
end
```

6. Download and install dependencies:

```
pod install --repo-update
```

7. Open the Xcode workspace that CocoaPods created:

```
xed .
```

Building the application: Configuration

Add the following keys and values to **Info.plist** to configure the application:

Key	Value
AWSRegion	us-east-1
IdentityPoolId	us-east-1:54f2ba88-9390-498d-aaa5-0d97fb7ca3bd
MapName	ExampleMap

Building the application: ContentView layout

To render the map, edit `ContentView.swift`:

- Add a `MapView` which renders the map.
- Add a `TextField` which displays attribution.

This also sets the map's initial center point.

```
import SwiftUI

struct ContentView: View {
    @State private var attribution = ""

    var body: some View {
        MapView(attribution: $attribution)
            .centerCoordinate(.init(latitude: 49.2819, longitude: -123.1187))
            .zoomLevel(12)
            .edgesIgnoringSafeArea(.all)
```

```
            .overlay(
                TextField("", text: $attribution)
                    .disabled(true)
                    .font(.system(size: 12, weight: .light, design: .default))
                    .foregroundColor(.black)
                    .background(Color.init(Color.RGBColorSpace.sRGB, white: 0.5,
 opacity: 0.5))
                    .cornerRadius(1),
                alignment: .bottomTrailing)
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

Note

You must provide word mark or text attribution for each data provider that you use, either on your application or your documentation. Attribution strings are included in the style descriptor response under the `sources.esri.attribution`, `sources.here.attribution`, and `source.grabmaptiles.attribution` keys. When using Amazon Location resources with data providers, make sure to read the service terms and conditions.

Building the application: Request transformation

Create a new Swift file named `AWSSignatureV4Delegate.swift` containing the following class definition to intercept AWS requests and sign them using Signature Version 4. An instance of this class will be assigned as the offline storage delegate, which is also responsible for rewriting URLs, in the map view.

```
import AWSCore
import Mapbox

class AWSSignatureV4Delegate : NSObject, MGLOfflineStorageDelegate {
    private let region: AWSRegionType
    private let identityPoolId: String
```

```
    private let credentialsProvider: AWSCredentialsProvider

    init(region: AWSRegionType, identityPoolId: String) {
        self.region = region
        self.identityPoolId = identityPoolId
        self.credentialsProvider = AWSCognitoCredentialsProvider(regionType: region,
 identityPoolId: identityPoolId)
        super.init()
    }

    class func doubleEncode(path: String) -> String? {
        return path.addingPercentEncoding(withAllowedCharacters: .urlPathAllowed)?
            .addingPercentEncoding(withAllowedCharacters: .urlPathAllowed)
    }

    func offlineStorage(_ storage: MGLOfflineStorage, urlForResourceOf kind:
 MGLResourceKind, with url: URL) -> URL {
        if url.host?.contains("amazonaws.com") != true {
            // not an AWS URL
            return url
        }

        // URL-encode spaces, etc.
        let keyPath = String(url.path.dropFirst())
        guard let percentEncodedKeyPath =
 keyPath.addingPercentEncoding(withAllowedCharacters: .urlPathAllowed) else {
            print("Invalid characters in path '\(keyPath)'; unsafe to sign")
            return url
        }

        let endpoint = AWSEndpoint(region: region, serviceName: "geo", url: url)
        let requestHeaders: [String: String] = ["host": endpoint!.hostName]

        // sign the URL
        let task = AWSSignatureV4Signer
            .generateQueryStringForSignatureV4(
                withCredentialProvider: credentialsProvider,
                httpMethod: .GET,
                expireDuration: 60,
                endpoint: endpoint!,
                // workaround for https://github.com/aws-amplify/aws-sdk-ios/
issues/3215
                keyPath: AWSSignatureV4Delegate.doubleEncode(path:
 percentEncodedKeyPath),
```

```
                requestHeaders: requestHeaders,
                requestParameters: .none,
                signBody: true)
        task.waitUntilFinished()

        if let error = task.error as NSError? {
            print("Error occurred: \(error)")
        }

        if let result = task.result {
            var urlComponents = URLComponents(url: (result as URL),
 resolvingAgainstBaseURL: false)!
            // re-use the original path; workaround for https://github.com/aws-amplify/
aws-sdk-ios/issues/3215
            urlComponents.path = url.path

            // have Mapbox GL fetch the signed URL
            return (urlComponents.url)!
        }

        // fall back to an unsigned URL
        return url
    }
}
```

Building the application: Map view

The Map View is responsible for initializing an instance of `AWSSignatureV4Delegate` and configuring the underlying `MGLMapView`, which fetches resources and renders the map. It also handles propagating attribution strings from the style descriptor's source back to the `ContentView`.

Create a new Swift file named `MapView.swift` containing the following `struct` definition:

```
import SwiftUI
import AWSCore
import Mapbox

struct MapView: UIViewRepresentable {
    @Binding var attribution: String

    private var mapView: MGLMapView
    private var signingDelegate: MGLOfflineStorageDelegate
```

```
    init(attribution: Binding<String>) {
        let regionName = Bundle.main.object(forInfoDictionaryKey: "AWSRegion") as!
 String
        let identityPoolId = Bundle.main.object(forInfoDictionaryKey: "IdentityPoolId")
 as! String
        let mapName = Bundle.main.object(forInfoDictionaryKey: "MapName") as! String

        let region = (regionName as NSString).aws_regionTypeValue()

        // MGLOfflineStorage doesn't take ownership, so this needs to be a member here
        signingDelegate = AWSSignatureV4Delegate(region: region, identityPoolId:
 identityPoolId)

        // register a delegate that will handle SigV4 signing
        MGLOfflineStorage.shared.delegate = signingDelegate

        mapView = MGLMapView(
            frame: .zero,
            styleURL: URL(string: "https://maps.geo.\(regionName).amazonaws.com/maps/
v0/maps/\(mapName)/style-descriptor"))

        _attribution = attribution
    }

    func makeCoordinator() -> Coordinator {
        Coordinator($attribution)
    }

    class Coordinator: NSObject, MGLMapViewDelegate {
        var attribution: Binding<String>

        init(_ attribution: Binding<String>) {
            self.attribution = attribution
        }

        func mapView(_ mapView: MGLMapView, didFinishLoading style: MGLStyle) {
            let source = style.sources.first as? MGLVectorTileSource
            let attribution = source?.attributionInfos.first
            self.attribution.wrappedValue = attribution?.title.string ?? ""
        }
    }

    // MARK: - UIViewRepresentable protocol
```

```
    func makeUIView(context: UIViewRepresentableContext<MapView>) -> MGLMapView {
        mapView.delegate = context.coordinator

        mapView.logoView.isHidden = true
        mapView.attributionButton.isHidden = true
        return mapView
    }

    func updateUIView(_ uiView: MGLMapView, context:
 UIViewRepresentableContext<MapView>) {
    }

    // MARK: - MGLMapView proxy

    func centerCoordinate(_ centerCoordinate: CLLocationCoordinate2D) -> MapView {
        mapView.centerCoordinate = centerCoordinate
        return self
    }

    func zoomLevel(_ zoomLevel: Double) -> MapView {
        mapView.zoomLevel = zoomLevel
        return self
    }
}
```

Running this application displays a full-screen map in the style of your choosing. This sample is available as part of the Amazon Location Service samples repository on GitHub.

Using the Amplify library with Amazon Location Service

The following tutorial walks you through using AWS Amplify with Amazon Location. Amplify uses MapLibre GL JS to render maps in your JavaScript-based application.

Amplify is a set of open-source client libraries that provide interfaces to different categories of services, including Amplify Geo, which is powered by Amazon Location Service. Learn more about the AWS Amplify Geo JavaScript library.

> **Note**
>
> This tutorial assumes that you have already followed the steps in Using maps - To add a map to your application.

Building the application: Scaffolding

This tutorial creates a web application that uses JavaScript to build a map on an HTML page.

Begin by creating an HTML page (`index.html`) that includes the map's container:

- Enter a `div` element with an `id` of `map` to apply the map's dimensions to the map view. The dimensions are inherited from the viewport.

```
<html>
  <head>
    <style>
      body { margin: 0; }
      #map { height: 100vh; } /* 100% of viewport height */
    </style>
  </head>

  <body>
    <!-- map container -->
    <div id="map" />
  </body>
</html>
```

Building the application: Adding dependencies

Add the following dependencies to your application:

- AWS Amplify map and geo libraries.
- AWS Amplify core library.
- AWS Amplify auth library.
- AWS Amplify stylesheet.

```
<!-- CSS dependencies -->
    <link href="https://cdn.amplify.aws/packages/maplibre-
gl/1.15.2/maplibre-gl.css" rel="stylesheet" integrity="sha384-
DrPVD9GufrxGb7kWwRv0CywpXTmfvbKOZ5i5pN7urmIThew0zXKTME+gutUgtpeD"
 crossorigin="anonymous" referrerpolicy="no-referrer"></link>

<!-- JavaScript dependencies -->
```

```
    <script src="https://cdn.amplify.aws/packages/maplibre-gl/1.15.2/maplibre-gl.js"
 integrity="sha384-rwYfkmAOpciZS2bDuwZ/Xa/Gog6jXem8D/whm3wnsZSVFemDDlprcUXHnDDUcrNU"
 crossorigin="anonymous" referrerpolicy="no-referrer"></script>
    <script src="https://cdn.amplify.aws/packages/core/4.3.0/aws-amplify-core.min.js"
 integrity="sha384-7Oh+5w0l7XGyYvSqbKi2Q7SA5K640V5nyW2/LEbevDQEV1HMJqJLA1A00z2hu8fJ"
 crossorigin="anonymous" referrerpolicy="no-referrer"></script>
    <script src="https://cdn.amplify.aws/packages/auth/4.3.8/aws-amplify-auth.min.js"
 integrity="sha384-jfkXCEfYyVmDXYKlgWNwv54xRaZgk14m7sjeb2jLVBtUXCD2p+WU8YZ2mPZ9Xbdw"
 crossorigin="anonymous" referrerpolicy="no-referrer"></script>
    <script src="https://cdn.amplify.aws/packages/geo/1.1.0/aws-amplify-geo.min.js"
 integrity="sha384-TFMTyWuCbiptXTzvOgzJbV8TPUupG1rA1AVrznAhCSpXTIdGw82bGd8RTk5rr3nP"
 crossorigin="anonymous" referrerpolicy="no-referrer"></script>
    <script src="https://cdn.amplify.aws/packages/maplibre-gl-js-
amplify/1.1.0/maplibre-gl-js-amplify.umd.min.js" integrity="sha384-7/
RxWonKW1nM9zCKiwU9x6bkQTjldosg0D1vZYm0Zj+K/vUSnA3sOMhlRRWAtHPi" crossorigin="anonymous"
 referrerpolicy="no-referrer"></script>
<script>
  // application-specific code
</script>
```

This creates an empty page with the map's container.

Building the application: Configuration

To configure your application using JavaScript:

1. Enter the identifiers of the unauthenticated identity pool that you created in Using maps - Step 2, Set up authentication.

   ```
   // Cognito Identity Pool ID
   const identityPoolId = "region:identityPoolID"; // for example: us-
   east-1:123example-1234-5678
   // extract the Region from the Identity Pool ID
   const region = identityPoolId.split(":")[0];
   ```

2. Configure AWS Amplify to use the resources you've created, including the identity pool and the Map resource (shown here with the default name of `explore.map`).

   ```
   // Configure Amplify
   const { Amplify } = aws_amplify_core;
   const { createMap } = AmplifyMapLibre;

   Amplify.configure({
   ```

```
  Auth: {
    identityPoolId,
    region,
  },
  geo: {
    AmazonLocationService: {
      maps: {
        items: {
          "explore.map": {
            style: "Default style"
          },
        },
        default: "explore.map",
      },
      region,
    },
  }
});
```

Building the application: Map initialization

For the map to display after the page is loaded, you must initialize the map. You can adjust the initial map location, add additional controls, and overlay data.

```
async function initializeMap() {
  const map = await createMap(
    {
      container: "map",
      center: [-123.1187, 49.2819],
      zoom: 10,
      hash: true,
    }
  );

  map.addControl(new maplibregl.NavigationControl(), "top-left");
}

initializeMap();
```

> **Note**
>
> You must provide word mark or text attribution for each data provider that you use, either on your application or your documentation. Attribution strings are included in the style descriptor response under the `sources.esri.attribution`, `sources.here.attribution`, and `sources.grabmaptiles.attribution` keys. Amplify will automatically provide attribution. When using Amazon Location resources with data providers, make sure to read the service terms and conditions.

Running the application

You can run this sample application by using it in a local web server, or opening it in a browser.

To use a local web server, you can use npx, installed as part of Node.js, or any other web server of your choice. To use npx, type `npx serve` from within the same directory as `index.html`. This serves the application on `localhost:5000`.

> **Note**
>
> If the policy that you created for your unauthenticated Amazon Cognito role includes a `referer` condition, you might be blocked from testing with `localhost:` URLs. In this case. you can test with a web server that provides a URL that is in your policy.

After completing the tutorial, the final application looks like the following example.

```
<html>
  <head>
    <!-- CSS dependencies -->
    <link href="https://cdn.amplify.aws/packages/maplibre-
gl/1.15.2/maplibre-gl.css" rel="stylesheet" integrity="sha384-
DrPVD9GufrxGb7kWwRv0CywpXTmfvbKOZ5i5pN7urmIThew0zXKTME+gutUgtpeD"
 crossorigin="anonymous" referrerpolicy="no-referrer"></link>

    <!-- JavaScript dependencies -->
    <script src="https://cdn.amplify.aws/packages/maplibre-gl/1.15.2/maplibre-gl.js"
 integrity="sha384-rwYfkmAOpciZS2bDuwZ/Xa/Gog6jXem8D/whm3wnsZSVFemDDlprcUXHnDDUcrNU"
 crossorigin="anonymous" referrerpolicy="no-referrer"></script>
```

```
    <script src="https://cdn.amplify.aws/packages/core/4.3.0/aws-amplify-core.min.js"
 integrity="sha384-7Oh+5w0l7XGyYvSqbKi2Q7SA5K640V5nyW2/LEbevDQEV1HMJqJLA1A00z2hu8fJ"
 crossorigin="anonymous" referrerpolicy="no-referrer"></script>
    <script src="https://cdn.amplify.aws/packages/auth/4.3.8/aws-amplify-auth.min.js"
 integrity="sha384-jfkXCEfYyVmDXYKlgWNwv54xRaZgk14m7sjeb2jLVBtUXCD2p+WU8YZ2mPZ9Xbdw"
 crossorigin="anonymous" referrerpolicy="no-referrer"></script>
    <script src="https://cdn.amplify.aws/packages/geo/1.1.0/aws-amplify-geo.min.js"
 integrity="sha384-TFMTyWuCbiptXTzvOgzJbV8TPUupG1rA1AVrznAhCSpXTIdGw82bGd8RTk5rr3nP"
 crossorigin="anonymous" referrerpolicy="no-referrer"></script>
    <script src="https://cdn.amplify.aws/packages/maplibre-gl-js-
amplify/1.1.0/maplibre-gl-js-amplify.umd.min.js" integrity="sha384-7/
RxWonKW1nM9zCKiwU9x6bkQTjldosg0D1vZYm0Zj+K/vUSnA3sOMhlRRWAtHPi" crossorigin="anonymous"
 referrerpolicy="no-referrer"></script>

    <style>
      body { margin: 0; }
      #map { height: 100vh; }
    </style>
  </head>

  <body>
    <div id="map" />
    <script type="module">
      // Cognito Identity Pool ID
      const identityPoolId = "region:identityPoolId"; // for example: us-
east-1:123example-1234-5678
      // extract the Region from the Identity Pool ID
      const region = identityPoolId.split(":")[0];

      // Configure Amplify
      const { Amplify } = aws_amplify_core;
      const { createMap } = AmplifyMapLibre;

      Amplify.configure({
        Auth: {
          identityPoolId,
          region,
        },
        geo: {
          AmazonLocationService: {
            maps: {
              items: {
                "explore.map": {
                  style: "Default style"
```

```
                },
              },
              default: "explore.map",
            },
            region,
          },
        }
      });

      async function initializeMap() {
        const map = await createMap(
          {
            container: "map",
            center: [-123.1187, 49.2819],
            zoom: 10,
            hash: true,
          }
        );

        map.addControl(new maplibregl.NavigationControl(), "top-left");
      }

      initializeMap();
    </script>
  </body>
</html>
```

Running this application displays a full-screen map using your chosen map style. This sample is also described on the **Embed map** tab of any Map resource page in the Amazon Location Service console.

After you complete this tutorial, go to the Display a map topic in the AWS Amplify documentation to learn more, including how to display markers on the map.

Using Tangram with Amazon Location Service

This section provides the following tutorials on how to integrate Tangram with Amazon Location.

> **Important**
>
> The Tangram styles in the following tutorials are only compatible with Amazon Location map resources configured with the `VectorHereContrast` style.

The following is an example of an AWS CLI command to create a new map resource called *TangramExampleMap* using the *VectorHereContrast* style:

```
aws --region us-east-1 \
  location \
  create-map \
  --map-name "TangramExampleMap" \
  --configuration "Style=VectorHereContrast"
```

> **Note**
>
> Billing is determined by your usage. You may incur fees for the use of other AWS services. For more information, see Amazon Location Service pricing.

Topics

- Using Tangram with Amazon Location Service
- Using Tangram ES for Android with Amazon Location Service
- Using Tangram ES for iOS with Amazon Location Service

Using Tangram with Amazon Location Service

Tangram is a flexible mapping engine, designed for real-time rendering of 2D and 3D maps from vector tiles. It can be used with Mapzen-designed styles and the HERE tiles provided by the Amazon Location Service Maps API. This guide describes how to integrate Tangram with Amazon Location within a basic HTML/JavaScript application, although the same libraries and techniques also apply when using frameworks like React and Angular.

Tangram is built atop Leaflet, an open-source JavaScript library for mobile-friendly interactive maps. This means that many Leaflet-compatible plugins and controls also work with Tangram.

Tangram styles built to work with the Tilezen schema are largely compatible with Amazon Location when using maps from HERE. These include:

- Bubble Wrap – A full-featured wayfinding style with helpful icons for points of interest
- Cinnabar – A classic look and go-to for general mapping applications
- Refill – A minimalist map style designed for data visualization overlays, inspired by the seminal Toner style by Stamen Design

- Tron – An exploration of scale transformations in the visual language of TRON
- Walkabout – An outdoor-focused style that's perfect for hiking or getting out and about

This guide describes how to integrate Tangram with Amazon Location within a basic HTML/JavaScript application using the Tangram Style called Bubble Wrap. This sample is available as part of the Amazon Location Service samples repository on GitHub.

While other Tangram styles are best accompanied by raster tiles, which encode terrain information, this feature is not yet supported by Amazon Location.

> **Important**
>
> The Tangram styles in the following tutorial are only compatible with Amazon Location map resources configured with the `VectorHereContrast` style.

Building the application: Scaffolding

The application is an HTML page with JavaScript to build the map on your web application. Create an HTML page (`index.html`) and create the map's container:

- Enter a `div` element with an `id` of map to apply the map's dimensions to the map view.
- The dimensions are inherited from the viewport.

```
<html>
  <head>
    <style>
      body {
        margin: 0;
      }

      #map {
        height: 100vh; /* 100% of viewport height */
      }
    </style>
  </head>
  <body>
    <!-- map container -->
    <div id="map" />
```

```
  </body>
</html>
```

Building the application: Adding dependencies

Add the following dependencies:

- Leaflet and its associated CSS.
- Tangram.
- AWS SDK for JavaScript.

```
<!-- CSS dependencies -->
<link
  rel="stylesheet"
  href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css"
  integrity="sha512-xodZBNTC5n17Xt2atTPuE1HxjVMSvLVW9ocqUKLsCC5CXdbqCmblAshOMAS6/keqq/
sMZMZ19scR4PsZChSR7A=="
  crossorigin=""
/>
<!-- JavaScript dependencies -->
<script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"></script>
<script src="https://unpkg.com/tangram"></script>
<script src="https://sdk.amazonaws.com/js/aws-sdk-2.784.0.min.js"></script>
<script>
  // application-specific code
</script>
```

This creates an empty page with the necessary prerequisites. The next step guides you through writing the JavaScript code for your application.

Building the application: Configuration

To configure your application with your resources and credentials:

1. Enter the names and identifiers of your resources.

   ```
   // Cognito Identity Pool ID
   const identityPoolId = "us-east-1:54f2ba88-9390-498d-aaa5-0d97fb7ca3bd";
   // Amazon Location Service map name; must be HERE-backed
   const mapName = "TangramExampleMap";
   ```

2. Instantiate a credential provider using the unauthenticated identity pool you created in Using maps - Step 2, Set up authentication. Since this uses credentials outside the normal AWS SDK work flow, sessions expire after **one** hour.

```
// extract the region from the Identity Pool ID; this will be used for both Amazon
 Cognito and Amazon Location
AWS.config.region = identityPoolId.split(":", 1)[0];

// instantiate a Cognito-backed credential provider
const credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: identityPoolId,
});
```

3. While Tangram allows you to override the URL(s) used to fetch tiles, it doesn't include the ability to intercept requests so that they can be signed.

   To work around this, override `sources.mapzen.url` to point to Amazon Location using a synthetic host name `amazon.location`, which will be handled by a service worker. The following is an example of scene configuration using Bubble Wrap:

```
const scene = {
  import: [
    // Bubble Wrap style
    "https://www.nextzen.org/carto/bubble-wrap-style/10/bubble-wrap-style.zip",
    "https://www.nextzen.org/carto/bubble-wrap-style/10/themes/label-7.zip",
    "https://www.nextzen.org/carto/bubble-wrap-style/10/themes/bubble-wrap-road-
shields-usa.zip",
    "https://www.nextzen.org/carto/bubble-wrap-style/10/themes/bubble-wrap-road-
shields-international.zip",
  ],
  // override values beneath the `sources` key in the style above
  sources: {
    mapzen: {
      // point at Amazon Location using a synthetic URL, which will be handled by
 the service
      // worker
      url: `https://amazon.location/${mapName}/{z}/{x}/{y}`,
    },
    // effectively disable raster tiles containing encoded normals
    normals: {
      max_zoom: 0,
    },
```

```
    "normals-elevation": {
      max_zoom: 0,
    },
  },
};
```

Building the application: Request transformation

To register and initialize the service worker, create a `registerServiceWorker` function to be called before the map is initialized. This registers the JavaScript code provided in a separate file called `sw.js` as the service worker controlling `index.html`.

Credentials are loaded from Amazon Cognito and are passed into the service worker alongside the Region to provide information to sign tile requests with Signature Version 4.

```
/**
 * Register a service worker that will rewrite and sign requests using Signature
 Version 4.
 */
async function registerServiceWorker() {
  if ("serviceWorker" in navigator) {
    try {
      const reg = await navigator.serviceWorker.register("./sw.js");

      // refresh credentials from Amazon Cognito
      await credentials.refreshPromise();

      await reg.active.ready;

      if (navigator.serviceWorker.controller == null) {
        // trigger a navigate event to active the controller for this page
        window.location.reload();
      }

      // pass credentials to the service worker
      reg.active.postMessage({
        credentials: {
          accessKeyId: credentials.accessKeyId,
          secretAccessKey: credentials.secretAccessKey,
          sessionToken: credentials.sessionToken,
        },
        region: AWS.config.region,
```

```
      });
    } catch (error) {
      console.error("Service worker registration failed:", error);
    }
  } else {
    console.warn("Service worker support is required for this example");
  }
}
```

The Service Worker implementation in `sw.js` listens for `message` events to pick up credential and Region configuration changes. It also acts as a proxy server by listening for `fetch` events. `fetch` events targeting the `amazon.location` synthetic host name will be rewritten to target the appropriate Amazon Location API and signed using Amplify Core's `Signer`.

```
// sw.js
self.importScripts(
  "https://unpkg.com/@aws-amplify/core@3.7.0/dist/aws-amplify-core.min.js"
);

const { Signer } = aws_amplify_core;

let credentials;
let region;

self.addEventListener("install", (event) => {
  // install immediately
  event.waitUntil(self.skipWaiting());
});

self.addEventListener("activate", (event) => {
  // control clients ASAP
  event.waitUntil(self.clients.claim());
});

self.addEventListener("message", (event) => {
  const {
    data: { credentials: newCredentials, region: newRegion },
  } = event;

  if (newCredentials != null) {
    credentials = newCredentials;
  }
```

```
  if (newRegion != null) {
    region = newRegion;
  }
});

async function signedFetch(request) {
  const url = new URL(request.url);
  const path = url.pathname.slice(1).split("/");

  // update URL to point to Amazon Location
  url.pathname = `/maps/v0/maps/${path[0]}/tiles/${path.slice(1).join("/")}`;
  url.host = `maps.geo.${region}.amazonaws.com`;
  // strip params (Tangram generates an empty api_key param)
  url.search = "";

  const signed = Signer.signUrl(url.toString(), {
    access_key: credentials.accessKeyId,
    secret_key: credentials.secretAccessKey,
    session_token: credentials.sessionToken,
  });

  return fetch(signed);
}

self.addEventListener("fetch", (event) => {
  const { request } = event;

  // match the synthetic hostname we're telling Tangram to use
  if (request.url.includes("amazon.location")) {
    return event.respondWith(signedFetch(request));
  }

  // fetch normally
  return event.respondWith(fetch(request));
});
```

To automatically renew credentials and send them to the service worker before they expire, use the following function within `index.html`:

```
async function refreshCredentials() {
  await credentials.refreshPromise();

  if ("serviceWorker" in navigator) {
```

```
    const controller = navigator.serviceWorker.controller;

    controller.postMessage({
      credentials: {
        accessKeyId: credentials.accessKeyId,
        secretAccessKey: credentials.secretAccessKey,
        sessionToken: credentials.sessionToken,
      },
    });
  } else {
    console.warn("Service worker support is required for this example.");
  }

  // schedule the next credential refresh when they're about to expire
  setTimeout(refreshCredentials, credentials.expireTime - new Date());
}
```

Building the application: Map initialization

For the map to display after the page is loaded, you must initialize the map. You have the option to adjust the initial map location, add additional controls, and overlay data.

> **Note**
>
> You must provide word mark or text attribution for each data provider that you use, either on your application or your documentation. Attribution strings are included in the style descriptor response under the `sources.esri.attribution`, `sources.here.attribution`, and `source.grabmaptiles.attribution` keys. Because Tangram doesn't request these resources, and is only compatible with maps from HERE, use "© 2020 HERE". When using Amazon Location resources with data providers, make sure to read the service terms and conditions.

```
/**
 * Initialize a map.
 */
async function initializeMap() {
  // register the service worker to handle requests to https://amazon.location
  await registerServiceWorker();

  // Initialize the map
```

```
  const map = L.map("map").setView([49.2819, -123.1187], 10);
  Tangram.leafletLayer({
    scene,
  }).addTo(map);
  map.attributionControl.setPrefix("");
  map.attributionControl.addAttribution("© 2020 HERE");
}

initializeMap();
```

Running the application

To run this sample, you can:

- Use a host that supports HTTPS,
- Use a local web server to comply with service worker security restrictions.

To use a local web server, you can use npx, because it's installed as a part of Node.js. You can use `npx serve` from within the same directory as `index.html` and `sw.js`. This serves the application on localhost:5000.

The following is the `index.html` file:

```
<!-- index.html -->
<html>
  <head>
    <link
      rel="stylesheet"
      href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css"
      integrity="sha512-xodZBNTC5n17Xt2atTPuE1HxjVMSvLVW9ocqUKLsCC5CXdbqCmblAshOMAS6/
keqq/sMZMZ19scR4PsZChSR7A=="
      crossorigin=""
    />
    <style>
      body {
        margin: 0;
      }

      #map {
        height: 100vh;
      }
    </style>
```

```
  </head>

  <body>
    <div id="map" />
    <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"></script>
    <script src="https://unpkg.com/tangram"></script>
    <script src="https://sdk.amazonaws.com/js/aws-sdk-2.784.0.min.js"></script>
    <script>
      // configuration
      // Cognito Identity Pool ID
      const identityPoolId = "<Identity Pool ID>";
      // Amazon Location Service Map name; must be HERE-backed
      const mapName = "<Map name>";

      AWS.config.region = identityPoolId.split(":")[0];

      // instantiate a credential provider
      credentials = new AWS.CognitoIdentityCredentials({
        IdentityPoolId: identityPoolId,
      });

      const scene = {
        import: [
          // Bubble Wrap style
          "https://www.nextzen.org/carto/bubble-wrap-style/10/bubble-wrap-style.zip",
          "https://www.nextzen.org/carto/bubble-wrap-style/10/themes/label-7.zip",
          "https://www.nextzen.org/carto/bubble-wrap-style/10/themes/bubble-wrap-road-
shields-usa.zip",
          "https://www.nextzen.org/carto/bubble-wrap-style/10/themes/bubble-wrap-road-
shields-international.zip",
        ],
        // override values beneath the `sources` key in the style above
        sources: {
          mapzen: {
            // point at Amazon Location using a synthetic URL, which will be handled by
 the service
            // worker
            url: `https://amazon.location/${mapName}/{z}/{x}/{y}`,
          },
          // effectively disable raster tiles containing encoded normals
          normals: {
            max_zoom: 0,
          },
          "normals-elevation": {
```

```
            max_zoom: 0,
          },
        },
      };

      /**
       * Register a service worker that will rewrite and sign requests using Signature
Version 4.
       */
      async function registerServiceWorker() {
        if ("serviceWorker" in navigator) {
          try {
            const reg = await navigator.serviceWorker.register("./sw.js");

            // refresh credentials from Amazon Cognito
            await credentials.refreshPromise();

            await reg.active.ready;

            if (navigator.serviceWorker.controller == null) {
              // trigger a navigate event to active the controller for this page
              window.location.reload();
            }

            // pass credentials to the service worker
            reg.active.postMessage({
              credentials: {
                accessKeyId: credentials.accessKeyId,
                secretAccessKey: credentials.secretAccessKey,
                sessionToken: credentials.sessionToken,
              },
              region: AWS.config.region,
            });
          } catch (error) {
            console.error("Service worker registration failed:", error);
          }
        } else {
          console.warn("Service Worker support is required for this example");
        }
      }

      /**
       * Initialize a map.
       */
```

```
      async function initializeMap() {
        // register the service worker to handle requests to https://amazon.location
        await registerServiceWorker();

        // Initialize the map
        const map = L.map("map").setView([49.2819, -123.1187], 10);
        Tangram.leafletLayer({
          scene,
        }).addTo(map);
        map.attributionControl.setPrefix("");
        map.attributionControl.addAttribution("© 2020 HERE");
      }

      initializeMap();
    </script>
  </body>
</html>
```

The following is the `sw.js` file:

```
// sw.js
self.importScripts(
  "https://unpkg.com/@aws-amplify/core@3.7.0/dist/aws-amplify-core.min.js"
);

const { Signer } = aws_amplify_core;

let credentials;
let region;

self.addEventListener("install", (event) => {
  // install immediately
  event.waitUntil(self.skipWaiting());
});

self.addEventListener("activate", (event) => {
  // control clients ASAP
  event.waitUntil(self.clients.claim());
});

self.addEventListener("message", (event) => {
  const {
    data: { credentials: newCredentials, region: newRegion },
```

```
  } = event;

  if (newCredentials != null) {
    credentials = newCredentials;
  }

  if (newRegion != null) {
    region = newRegion;
  }
});

async function signedFetch(request) {
  const url = new URL(request.url);
  const path = url.pathname.slice(1).split("/");

  // update URL to point to Amazon Location
  url.pathname = `/maps/v0/maps/${path[0]}/tiles/${path.slice(1).join("/")}`;
  url.host = `maps.geo.${region}.amazonaws.com`;
  // strip params (Tangram generates an empty api_key param)
  url.search = "";

  const signed = Signer.signUrl(url.toString(), {
    access_key: credentials.accessKeyId,
    secret_key: credentials.secretAccessKey,
    session_token: credentials.sessionToken,
  });

  return fetch(signed);
}

self.addEventListener("fetch", (event) => {
  const { request } = event;

  // match the synthetic hostname we're telling Tangram to use
  if (request.url.includes("amazon.location")) {
    return event.respondWith(signedFetch(request));
  }

  // fetch normally
  return event.respondWith(fetch(request));
});
```

This sample is available as part of the Amazon Location Service samples repository on GitHub.

Using Tangram ES for Android with Amazon Location Service

Tangram ES is a C++ library for rendering 2D and 3D maps from vector data using OpenGL ES. It's the native counterpart to Tangram.

Tangram styles built to work with the Tilezen schema are largely compatible with Amazon Location when using maps from HERE. These include:

- Bubble Wrap – A full-featured wayfinding style with helpful icons for points of interest.
- Cinnabar – A classic look and go-to for general mapping applications.
- Refill – A minimalist map style designed for data visualization overlays, inspired by the seminal Toner style by Stamen Design.
- Tron – An exploration of scale transformations in the visual language of TRON.
- Walkabout – An outdoor-focused style that's perfect for hiking or getting out and about.

This guide describes how to integrate Tangram ES for Android with Amazon Location using the Tangram style called Cinnabar. This sample is available as part of the Amazon Location Service samples repository on GitHub.

While other Tangram styles are best accompanied by raster tiles, which encode terrain information, this feature isn't yet supported by Amazon Location.

> **Important**
>
> The Tangram styles in the following tutorial are only compatible with Amazon Location map resources configured with the `VectorHereContrast` style.

Building the application: Initialization

To initialize your application:

1. Create a new Android Studio project from the **Empty Activity** template.
2. Ensure that **Kotlin** is selected for the project language.
3. Select a **Minimum SDK of API 16: Android 4.1 (Jelly Bean)** or newer.
4. Open **Project Structure** to select **File**, **Project Structure...**, and choose the **Dependencies** section.
5. With **<All Modules>** selected, choose the **+** button to add a new **Library Dependency**.

6. Add **AWS Android SDK** version 2.19.1 or later. For example: `com.amazonaws:aws-android-sdk-core:2.19.1`
7. Add **Tangram** version 0.13.0 or later. For example: `com.mapzen.tangram:tangram:0.13.0`.

> **Note**
>
> Searching for **Tangram**: `com.mapzen.tangram:tangram:0.13.0` will generate a message that it's "not found", but choosing **OK** will allow it to be added.

Building the application: Configuration

To configure your application with your resources and AWS Region:

1. Create `app/src/main/res/values/configuration.xml`.
2. Enter the names and identifiers of your resources, and also the AWS Region they were created in:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="identityPoolId">us-east-1:54f2ba88-9390-498d-aaa5-0d97fb7ca3bd</
string>
    <string name="mapName">TangramExampleMap</string>
    <string name="awsRegion">us-east-1</string>
    <string name="sceneUrl">https://www.nextzen.org/carto/cinnabar-style/9/cinnabar-
style.zip</string>
    <string name="attribution">© 2020 HERE</string>
</resources>
```

Building the application: Activity layout

Edit `app/src/main/res/layout/activity_main.xml`:

- Add a `MapView`, which renders the map. This will also set the map's initial center point.
- Add a `TextView`, which displays attribution.

This will also set the map's initial center point.

Note

You must provide word mark or text attribution for each data provider that you use, either on your application or your documentation. Attribution strings are included in the style descriptor response under the `sources.esri.attribution`, `sources.here.attribution`, and `source.grabmaptiles.attribution` keys. Because Tangram doesn't request these resources, and is only compatible with maps from HERE, use "© 2020 HERE". When using Amazon Location resources with data providers, make sure to read the service terms and conditions.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <com.mapzen.tangram.MapView
        android:id="@+id/map"
        android:layout_height="match_parent"
        android:layout_width="match_parent" />

    <TextView
        android:id="@+id/attributionView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#80808080"
        android:padding="5sp"
        android:textColor="@android:color/black"
        android:textSize="10sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        tools:ignore="SmallSp" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Building the application: Request transformation

Create a class named `SigV4Interceptor` to intercept AWS requests and sign them using Signature Version 4. This will be registered with the HTTP client used to fetch map resources when the Main Activity is created.

```
package aws.location.demo.okhttp

import com.amazonaws.DefaultRequest
import com.amazonaws.auth.AWS4Signer
import com.amazonaws.auth.AWSCredentialsProvider
import com.amazonaws.http.HttpMethodName
import com.amazonaws.util.IOUtils
import okhttp3.HttpUrl
import okhttp3.Interceptor
import okhttp3.Request
import okhttp3.Response
import okio.Buffer
import java.io.ByteArrayInputStream
import java.net.URI

class SigV4Interceptor(
    private val credentialsProvider: AWSCredentialsProvider,
    private val serviceName: String
) : Interceptor {
    override fun intercept(chain: Interceptor.Chain): Response {
        val originalRequest = chain.request()

        if (originalRequest.url().host().contains("amazonaws.com")) {
            val signer = if (originalRequest.url().encodedPath().contains("@")) {
                // the presence of "@" indicates that it doesn't need to be double URL-
encoded
                AWS4Signer(false)
            } else {
                AWS4Signer()
            }

            val awsRequest = toAWSRequest(originalRequest, serviceName)
            signer.setServiceName(serviceName)
            signer.sign(awsRequest, credentialsProvider.credentials)

            return chain.proceed(toSignedOkHttpRequest(awsRequest, originalRequest))
        }
```

```
        return chain.proceed(originalRequest)
    }

    companion object {
        fun toAWSRequest(request: Request, serviceName: String): DefaultRequest<Any> {
            // clone the request (AWS-style) so that it can be populated with
credentials
            val dr = DefaultRequest<Any>(serviceName)

            // copy request info
            dr.httpMethod = HttpMethodName.valueOf(request.method())
            with(request.url()) {
                dr.resourcePath = uri().path
                dr.endpoint = URI.create("${scheme()}://${host()}")

                // copy parameters
                for (p in queryParameterNames()) {
                    if (p != "") {
                        dr.addParameter(p, queryParameter(p))
                    }
                }
            }

            // copy headers
            for (h in request.headers().names()) {
                dr.addHeader(h, request.header(h))
            }

            // copy the request body
            val bodyBytes = request.body()?.let { body ->
                val buffer = Buffer()
                body.writeTo(buffer)
                IOUtils.toByteArray(buffer.inputStream())
            }

            dr.content = ByteArrayInputStream(bodyBytes ?: ByteArray(0))

            return dr
        }

        fun toSignedOkHttpRequest(
            awsRequest: DefaultRequest<Any>,
            originalRequest: Request
```

```
        ): Request {
            // copy signed request back into an OkHttp Request
            val builder = Request.Builder()

            // copy headers from the signed request
            for ((k, v) in awsRequest.headers) {
                builder.addHeader(k, v)
            }

            // start building an HttpUrl
            val urlBuilder = HttpUrl.Builder()
                .host(awsRequest.endpoint.host)
                .scheme(awsRequest.endpoint.scheme)
                .encodedPath(awsRequest.resourcePath)

            // copy parameters from the signed request
            for ((k, v) in awsRequest.parameters) {
                urlBuilder.addQueryParameter(k, v)
            }

            return builder.url(urlBuilder.build())
                .method(originalRequest.method(), originalRequest.body())
                .build()
        }
    }
}
```

Building the application: Main activity

The Main Activity is responsible for initializing the views that will be displayed to users. This involves:

- Instantiating an Amazon Cognito `CredentialsProvider`.
- Registering the Signature Version 4 interceptor.
- Configuring the map by pointing it at a map style, overriding tile URLs, and displaying appropriate attribution.

`MainActivity` is also responsible for forwarding life cycle events to the map view.

```
package aws.location.demo.tangram
```

```
import android.os.Bundle
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import aws.location.demo.okhttp.SigV4Interceptor
import com.amazonaws.auth.CognitoCachingCredentialsProvider
import com.amazonaws.regions.Regions
import com.mapzen.tangram.*
import com.mapzen.tangram.networking.DefaultHttpHandler
import com.mapzen.tangram.networking.HttpHandler

private const val SERVICE_NAME = "geo"

class MainActivity : AppCompatActivity(), MapView.MapReadyCallback {
    private var mapView: MapView? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContentView(R.layout.activity_main)

        mapView = findViewById(R.id.map)

        mapView?.getMapAsync(this, getHttpHandler())
        findViewById<TextView>(R.id.attributionView).text =
 getString(R.string.attribution)
    }

    override fun onMapReady(mapController: MapController?) {
        val sceneUpdates = arrayListOf(
            SceneUpdate(
                "sources.mapzen.url",
                "https://maps.geo.${getString(R.string.awsRegion)}.amazonaws.com/maps/
v0/maps/${
                    getString(
                        R.string.mapName
                    )
                }/tiles/{z}/{x}/{y}"
            )
        )

        mapController?.let { map ->
            map.updateCameraPosition(
                CameraUpdateFactory.newLngLatZoom(
                    LngLat(-123.1187, 49.2819),
```

```
                    12F
                )
            )
            map.loadSceneFileAsync(
                getString(R.string.sceneUrl),
                sceneUpdates
            )
        }
    }

    private fun getHttpHandler(): HttpHandler {
        val builder = DefaultHttpHandler.getClientBuilder()

        val credentialsProvider = CognitoCachingCredentialsProvider(
            applicationContext,
            getString(R.string.identityPoolId),
            Regions.US_EAST_1
        )

        return DefaultHttpHandler(
            builder.addInterceptor(
                SigV4Interceptor(
                    credentialsProvider,
                    SERVICE_NAME
                )
            )
        )
    }

    override fun onResume() {
        super.onResume()
        mapView?.onResume()
    }

    override fun onPause() {
        super.onPause()
        mapView?.onPause()
    }

    override fun onLowMemory() {
        super.onLowMemory()
        mapView?.onLowMemory()
    }
```

```
    override fun onDestroy() {
        super.onDestroy()
        mapView?.onDestroy()
    }
}
```

Running this application displays a full-screen map in the style of your choosing. This sample is available as part of the Amazon Location Service samples repository on GitHub.

Using Tangram ES for iOS with Amazon Location Service

Tangram ES is a C++ library for rendering 2D and 3D maps from vector data using OpenGL ES. It's the native counterpart to Tangram.

Tangram styles built to work with the Tilezen schema are largely compatible with Amazon Location when using maps from HERE. These include:

- Bubble Wrap – A full-featured wayfinding style with helpful icons for points of interest
- Cinnabar – A classic look and go-to for general mapping applications
- Refill – A minimalist map style designed for data visualization overlays, inspired by the seminal Toner style by Stamen Design
- Tron – An exploration of scale transformations in the visual language of TRON
- Walkabout – An outdoor-focused style that's perfect for hiking or getting out and about

This guide describes how to integrate Tangram ES for iOS with Amazon Location using the Tangram style called Cinnabar. This sample is available as part of the Amazon Location Service samples repository on GitHub.

While other Tangram styles are best accompanied by raster tiles, which encode terrain information, this feature isn't yet supported by Amazon Location.

> **Important**
>
> The Tangram styles in the following tutorial are only compatible with Amazon Location map resources configured with the `VectorHereContrast` style.

Building the application: Initialization

To initialize the application:

1. Create a new Xcode project from the **App** template.
2. Select **SwiftUI** for its interface.
3. Select **SwiftUI** application for its Life Cycle.
4. Select **Swift** for its language.

Building the application: Add dependencies

To add dependencies, you can use a dependency manager, such as CocoaPods:

1. In your terminal, install CocoaPods:

```
sudo gem install cocoapods
```

2. Navigate to your application's project directory and initialize the **Podfile** with the CocoaPods package manager:

```
pod init
```

3. Open the **Podfile** to add `AWSCore` and `Tangram-es` as dependencies:

```
platform :ios, '12.0'

target 'Amazon Location Service Demo' do
  use_frameworks!

  pod 'AWSCore'
  pod 'Tangram-es'
end
```

4. Download and install dependencies:

```
pod install --repo-update
```

5. Open the Xcode workspace that CocoaPods created:

```
xed .
```

Building the application: Configuration

Add the following keys and values to **Info.plist** to configure the application and disable telemetry:

Key	Value
AWSRegion	us-east-1
IdentityPoolId	us-east-1:54f2ba88-9390-498d-aaa5-0d97fb7ca3bd
MapName	ExampleMap
SceneURL	https://www.nextzen.org/carto/cinnabar-style/9/cinnabar-style.zip

Building the application: ContentView layout

To render the map, edit `ContentView.swift`:

- Add a `MapView` which renders the map.
- Add a `TextField` which displays attribution.

This also sets the map's initial center point.

> **Note**
>
> You must provide word mark or text attribution for each data provider that you use, either on your application or your documentation. Attribution strings are included in the style descriptor response under the `sources.esri.attribution`, `sources.here.attribution`, and `source.grabmaptiles.attribution` keys. When using Amazon Location resources with data providers, make sure to read the service terms and conditions.

```
import SwiftUI
import TangramMap
```

```
struct ContentView: View {
    var body: some View {
        MapView()
            .cameraPosition(TGCameraPosition(
                                center: CLLocationCoordinate2DMake(49.2819, -123.1187),
                                zoom: 10,
                                bearing: 0,
                                pitch: 0))
            .edgesIgnoringSafeArea(.all)
            .overlay(
                Text("© 2020 HERE")
                    .disabled(true)
                    .font(.system(size: 12, weight: .light, design: .default))
                    .foregroundColor(.black)
                    .background(Color.init(Color.RGBColorSpace.sRGB, white: 0.5,
 opacity: 0.5))
                    .cornerRadius(1),
                alignment: .bottomTrailing)
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

Building the application: Request transformation

Create a new Swift file named `AWSSignatureV4URLHandler.swift` containing the following class definition to intercept AWS requests and sign them using Signature Version 4. This will be registered as a URL handler within the Tangram `MapView`.

```
import AWSCore
import TangramMap

class AWSSignatureV4URLHandler: TGDefaultURLHandler {
    private let region: AWSRegionType
    private let identityPoolId: String
    private let credentialsProvider: AWSCredentialsProvider

    init(region: AWSRegionType, identityPoolId: String) {
        self.region = region
```

```
        self.identityPoolId = identityPoolId
        self.credentialsProvider = AWSCognitoCredentialsProvider(regionType: region,
identityPoolId: identityPoolId)
        super.init()
    }

    override func downloadRequestAsync(_ url: URL, completionHandler: @escaping
TGDownloadCompletionHandler) -> UInt {
        if url.host?.contains("amazonaws.com") != true {
            // not an AWS URL
            return super.downloadRequestAsync(url, completionHandler:
completionHandler)
        }

        // URL-encode spaces, etc.
        let keyPath = String(url.path.dropFirst())
        guard let keyPathSafe =
keyPath.addingPercentEncoding(withAllowedCharacters: .urlPathAllowed) else {
            print("Invalid characters in path '\(keyPath)'; unsafe to sign")
            return super.downloadRequestAsync(url, completionHandler:
completionHandler)
        }

        // sign the URL
        let endpoint = AWSEndpoint(region: region, serviceName: "geo", url: url)
        let requestHeaders: [String: String] = ["host": endpoint!.hostName]
        let task = AWSSignatureV4Signer
            .generateQueryStringForSignatureV4(
                withCredentialProvider: credentialsProvider,
                httpMethod: .GET,
                expireDuration: 60,
                endpoint: endpoint!,
                keyPath: keyPathSafe,
                requestHeaders: requestHeaders,
                requestParameters: .none,
                signBody: true)
        task.waitUntilFinished()

        if let error = task.error as NSError? {
            print("Error occurred: \(error)")
        }

        if let result = task.result {
            // have Tangram fetch the signed URL
```

```
            return super.downloadRequestAsync(result as URL, completionHandler:
 completionHandler)
        }

        // fall back to an unsigned URL
        return super.downloadRequestAsync(url, completionHandler: completionHandler)
    }
}
```

Building the application: Map view

The map view is responsible for initializing an instance of `AWSSignatureV4Delegate` and configuring the underlying `MGLMapView`, which fetches resources and renders the map. It also handles propagating attribution strings from the style descriptor's source back to the `ContentView`.

Create a new Swift file named `MapView.swift` containing the following `struct` definition:

```
import AWSCore
import TangramMap
import SwiftUI

struct MapView: UIViewRepresentable {
    private let mapView: TGMapView

    init() {
        let regionName = Bundle.main.object(forInfoDictionaryKey: "AWSRegion") as!
 String
        let identityPoolId = Bundle.main.object(forInfoDictionaryKey: "IdentityPoolId")
 as! String
        let mapName = Bundle.main.object(forInfoDictionaryKey: "MapName") as! String
        let sceneURL = URL(string: Bundle.main.object(forInfoDictionaryKey: "SceneURL")
 as! String)!

        let region = (regionName as NSString).aws_regionTypeValue()

        // rewrite tile URLs to point at AWS resources
        let sceneUpdates = [
            TGSceneUpdate(path: "sources.mapzen.url",
                          value: "https://maps.geo.\(regionName).amazonaws.com/maps/v0/
maps/\(mapName)/tiles/{z}/{x}/{y}")]

        // instantiate a TGURLHandler that will sign AWS requests
```

```
        let urlHandler = AWSSignatureV4URLHandler(region: region, identityPoolId:
 identityPoolId)

        // instantiate the map view and attach the URL handler
        mapView = TGMapView(frame: .zero, urlHandler: urlHandler)

        // load the map style and apply scene updates (properties modified at runtime)
        mapView.loadScene(from: sceneURL, with: sceneUpdates)
    }

    func cameraPosition(_ cameraPosition: TGCameraPosition) -> MapView {
        mapView.cameraPosition = cameraPosition

        return self
    }

    // MARK: - UIViewRepresentable protocol

    func makeUIView(context: Context) -> TGMapView {
        return mapView
    }

    func updateUIView(_ uiView: TGMapView, context: Context) {
    }
}
```

Running this application displays a full-screen map in the style of your choosing. This sample is available as part of the Amazon Location Service samples repository on GitHub.

Drawing data features on a map

After you have an application that renders a map, using Amplify, MapLibre, or Tangram to render the map, a natural next step is to draw features on top of the map. For example, you might want to render your customer locations as markers on the map.

In general, you can use the Places search functions to find locations from your data, and then use the functionality of Amplify, MapLibre, or Tangram to render the locations.

To see samples of rendering different types of objects on map, see the following MapLibre samples:

- Example: Draw markers

- Example: Draw clustered points
- Example: Draw a polygon

For more samples and tutorials, see Code examples and tutorials for working with Amazon Location Service.

Setting extents for a map using MapLibre

There are times that you do not want your users to be able to pan or zoom around the entire world. If you are using MapLibre's map control, you can limit the *extents*, or *bounds*, of the map control with the `maxBounds` option, and constrain the zoom with `minZoom` and `maxZoom` options.

The following code example shows how to initialize the map control to constrain panning to a specific boundary (in this case, the extents of the Grab data source).

> **Note**
>
> These samples are in JavaScript, and work within the context of the Creating a web app tutorial.

```
// Set bounds to Grab data provider region
var bounds = [
        [90.0, -21.943045533438166], // Southwest coordinates
        [146.25, 31.952162238024968] // Northeast coordinates
    ];

var mlglMap = new maplibregl.Map(
    {
    container: 'map',
    style: mapName,
    maxBounds: bounds // Sets bounds as max
    transformRequest,
    }
);
```

Similarly, you can set a minimum and maximum zoom level for the map. The values for both can be between 0 and 24, although the defaults are 0 for minimum zoom and 22 for maximum (data

providers may not provide data at all zoom levels. Most map libraries handle this automatically). The following example initializes the `minZoom` and `maxZoom` options on the MapLibre Map control.

```
// Set the minimum and maximum zoom levels
var mlglMap = new maplibregl.Map(
    {
    container: 'map',
    style: mapName,
    maxZoom: 12,
    minZoom: 5,
    transformRequest,
    }
);
```

Tip

The MapLibre Map control also allows setting these options at runtime, rather than during initialization, with `get...` and `set...` functions. For example, use `getMaxBounds` and `setMaxBounds` to change the map bounds at runtime.

Managing your map resources

You can manage your map resources using the Amazon Location console, the AWS CLI, or the Amazon Location APIs.

List map resources

You can view a list of your map resources using the Amazon Location console, the AWS CLI, or the Amazon Location APIs.

Console

To view a list of existing map resources using the Amazon Location console

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. Choose **Maps** from the left navigation pane.
3. View a list of your map resources under **My maps**.

API

Use the `ListMaps` operation from the Amazon Location Maps APIs.

The following example is an API request to get a list of map resources in the AWS account.

```
POST /maps/v0/list-maps
```

The following is an example response for `ListMaps`:

```
{
   "Entries": [
      {
         "CreateTime": 2020-10-30T01:38:36Z,
         "DataSource": "Esri",
         "Description": "string",
         "MapName": "ExampleMap",
         "UpdateTime": 2020-10-30T01:38:36Z
      }
   ],
   "NextToken": "1234-5678-9012"
}
```

CLI

Use the `list-map` command.

The following example is an AWS CLI to get a list of map resources in the AWS account.

```
aws location list-maps
```

Get map resource details

You can get details about any map resource in your AWS account using the Amazon Location console, the AWS CLI, or the Amazon Location APIs.

Console

To view the details of a map resource using the Amazon Location console

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.

2. Choose **Maps** from the left navigation pane.
3. Under **My maps**, select the name link of the target map resource.

API

Use the DescribeMap operation from the Amazon Location Maps APIs.

The following example is an API request to get the map resource details for *ExampleMap*.

```
GET /maps/v0/maps/ExampleMap
```

The following is an example response for DescribeMap:

```
{
   "Configuration": {
      "Style": "VectorEsriNavigation"
   },
   "CreateTime": 2020-10-30T01:38:36Z,
   "DataSource": "Esri",
   "Description": "string",
   "MapArn": "arn:aws:geo:us-west-2:123456789012:maps/ExampleMap",
   "MapName": "ExampleMap",
   "Tags": {
      "Tag1" : "Value1"
   },
   "UpdateTime": 2020-10-30T01:40:36Z
}
```

CLI

Use the describe-map command.

The following example is an AWS CLI to get the map resource details for *ExampleMap*.

```
aws location describe-map \
    --map-name "ExampleMap"
```

Delete a map resource

You can delete a map resource from your AWS account using the Amazon Location console, the AWS CLI, or the Amazon Location APIs.

> **Warning**
>
> This operation deletes the resource permanently.

Console

To delete an existing map resource using the Amazon Location console

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. Choose **Maps** from the left navigation pane.
3. Under **My maps** list, select the target map from the list.
4. Choose **Delete map**.

API

Use the `DeleteMap` operation from the Amazon Location Maps APIs.

The following example is an API request to delete the map resource *ExampleMap*.

```
DELETE /maps/v0/maps/ExampleMap
```

The following is an example success response for `DeleteMap`:

```
HTTP/1.1 200
```

CLI

Use the `delete-map` command.

The following example is an AWS CLI command to delete the map resource *ExampleMap*.

```
aws location delete-map \
```

```
    --map-name "ExampleMap"
```

Searching place and geolocation data using Amazon Location

Amazon Location includes the ability to search the geolocation, or *place*, data of your chosen provider. There are several kinds of searching available.

- **Geocoding** – Geocoding is the process of searching for addresses, regions, business names, or other points of interest, based on text input. It returns details and the location (in latitude and longitude) of the results found.
- **Reverse geocoding** – Reverse geocoding allows you to find places near a given location.
- **Autocomplete** – Autocomplete is the process of making automatic suggestions as the user types in a query. For example, if they type **`Par`** one suggestion might be `Paris, France`.

Amazon Location lets you choose a data provider for place search operations by creating and configuring a place index resource.

Once you create your resource, you can send requests using the AWS SDK for your preferred language, Amplify, or the REST API endpoints. You can use data from the response to mark locations on a map, enrich position data, and to convert positions into human-readable text.

> **Note**
>
> For an overview of searching place concepts, see Places search.

Topics

- Prerequisites
- Geocoding
- Reverse geocoding
- Autocomplete
- Using place IDs
- Place categories and filtering results
- Amazon Aurora PostgreSQL user-defined functions for Amazon Location Service

- Managing your place index resources

Prerequisites

Before you begin geocoding, reverse geocoding or searching for places, follow the prerequisite steps:

Topics

- Creating a place index resource
- Authenticating your requests

Creating a place index resource

Begin by creating a place index resource in your AWS account.

When you create a place index resource, you can choose from the data providers available to support queries for geocoding, reverse geocoding, and searches:

1. **Esri** – For more information about Esri's coverage in your region of interest, see Esri geocoding coverage in the Esri documentation.
2. **HERE Technologies** – For more information about HERE's coverage in your region of interest, see HERE geocoding coverage in the HERE documentation.
3. **Grab** – Grab provides data only for Southeast Asia. For more information about Grab's coverage, see Countries/regions and area covered in this guide.

You can do this using the Amazon Location Service console, the AWS CLI, or the Amazon Location APIs.

Console

To create a place index resource using the Amazon Location Service console

1. Open the Amazon Location Service console at https://console.aws.amazon.com/location/.
2. In the left navigation pane, choose **Place indexes**.
3. Choose **Create place index**.
4. Fill out the following boxes:

- **Name** – Enter a name for the place index resource. For example, *ExamplePlaceIndex*. Maximum 100 characters. Valid entries include alphanumeric characters, hyphens, periods, and underscores.
- **Description** – Enter an optional description.

5. Under **Data providers**, choose an available data provider to use with your place index resource.

> **Note**
>
> If your application is tracking or routing assets you use in your business, such as delivery vehicles or employees, you must not use Esri as your geolocation provider. See section 82 of the AWS service terms for more details.

6. Under **Data storage options**, specify if you intend to store search results from your place index resource.
7. (Optional) Under **Tags**, enter a tag **Key** and **Value**. This adds a tag your new place index resource. For more information, see Tagging your resources.
8. Choose **Create place index**.

API

To create a place index resource using the Amazon Location APIs

Use the `CreatePlaceIndex` operation from the Amazon Location Places APIs.

The following example is an API request to create a place index resource called *ExamplePlaceIndex* using the data provider *Esri*.

```
POST /places/v0/indexes
Content-type: application/json

{
   "DataSource": "Esri",
   "DataSourceConfiguration": {
      "IntendedUse": "SingleUse"
   },
   "Description": "string",
   "IndexName": "ExamplePlaceIndex",
```

```
   "Tags": {
      "Tag1" : "Value1"
   }
}
```

AWS CLI

To create a place index resource using AWS CLI commands

Use the `create-place-index` command.

The following example creates a place index resource called *ExamplePlaceIndex* using *Esri* as the data provider.

```
aws location \
  create-place-index \
  --data-source "Esri" \
  --description "Example place index" \
  --index-name "ExamplePlaceIndex" \
  --tags Tag1=Value1
```

Note

Billing depends on your usage. You may incur fees for the use of other AWS services. For more information, see Amazon Location Service pricing.

Authenticating your requests

Once you create a place index resource and you're ready to begin building location features into your application, choose how you would authenticate your requests:

- To explore ways you can access the services, see Accessing Amazon Location Service.
- If you have a website with anonymous users, you may want to use API Keys or Amazon Cognito.

Example

The following example shows using an API key for authorization, using AWS JavaScript SDK v3, and the Amazon Location JavaScript Authentication helper.

```
import { LocationClient, SearchPlaceIndexForTextCommand } from "@aws-sdk/client-
location";
import { withAPIKey } from "@aws/amazon-location-utilities-auth-helper";

const apiKey = "v1.public.your-api-key-value"; // API key

// Create an authentication helper instance using an API key
const authHelper = await withAPIKey(apiKey);

const client = new LocationClient({
  region: "<region>", // region containing Cognito pool
  ...authHelper.getLocationClientConfig(), // Provides configuration required to make
 requests to Amazon Location
});

const input = {
  IndexName: "ExamplePlaceIndex",
  Text: "Anyplace",
  BiasPosition: [-123.4567, 45.6789]
};

const command = new SearchPlaceIndexForTextCommand(input);

const response = await client.send(command);
```

Geocoding

Geocoding is a process that converts text, such as an address, a region, a business name, or point of interest, into a set of geographic coordinates. You can use place index resources to submit geocoding requests and incorporate data retrieved from geocoding to display data on a map for your web or mobile application.

This section guides you through how to send a simple geocoding request, and how to send geocoding requests with optional specifications.

Geocoding

You can submit a simple request to geocode using the `SearchPlaceIndexForText` operation to convert an address to a set of coordinates. A simple request contains the following required parameter:

- `Text` – An address, name, city, or region to convert to a set of coordinates. For example, the string `Any Town`.

To specify a maximum number of results per pages, use the following optional parameter:

- `MaxResults` – Limits the maximum number of results returned in the query response.

You can use the AWS CLI or the Amazon Location APIs.

API

The following example is a `SearchPlaceIndexForText` request to search the place index resource, *`ExamplePlaceIndex`*, for an address, name, city or region called *`Any Town`*.

```
POST /places/v0/indexes/ExamplePlaceIndex/search/text
Content-type: application/json

{
    "Text": "Any Town",
    "MaxResults": 10
}
```

AWS CLI

The following example is a `search-place-index-for-text` command to search the place index resource, *`ExamplePlaceIndex`*, for an address, name, city or region called *`Any Town`*.

```
aws location \
    search-place-index-for-text \
        --index-name ExamplePlaceIndex \
        --text "Any Town" \
        --max-results 10
```

Geocode near a position

When geocoding, you can geocode near a given position with the following optional parameter:

- `BiasPosition` – The position you want to search nearby. This narrows your search by searching for results closest to the given position. Defined as `[longitude, latitude]`

The following example is a SearchPlaceIndexForText request to search the place index resource for an address, name, city or region called *Any Town* near the position [*-123.4567*,*45.6789*].

```
POST /places/v0/indexes/ExamplePlaceIndex/search/text
Content-type: application/json

{
   "Text": "Any Town",
   "BiasPosition": [-123.4567,45.6789]
}
```

Geocode within a bounding box

You can geocode within a bounding box to narrow your results to coordinates within a given boundary using the following optional parameter:

- `FilterBBox` – A bounding box that you specify to filter your results to coordinates within the box's boundaries. Defined as `[LongitudeSW, LatitudeSW, LongitudeNE, LatitudeNE]`

> **Note**
>
> A request can't contain both the `FilterBBox` and `BiasPosition` parameters. Specifying both parameters in the request returns a `ValidationException` error.

The following example is a SearchPlaceIndexForText request to search within a bounding box for an address, name, city or region called *Any Town*. The bounding box follows that:

- The longitude of the southwest corner is *-124.1450*.
- The latitude of the southwest corner is *41.7045*.
- The longitude of the northeast corner is *-124.1387*.
- The latitude of the northeast corner is *41.7096*.

```
POST /places/v0/indexes/ExamplePlaceIndex/search/text
Content-type: application/json

{
```

```
   "Text": "Any Town",
   "FilterBBox":  [
        -124.1450,41.7045,
        -124.1387,41.7096
    ]
}
```

Geocode within a country

You can geocode within one or more countries you specify by using the following optional parameter:

- `FilterCountries` – The country or region you want to geocode within. You can define up to 100 countries in one request using a ISO 3166 three letter country code. For example, use AUS for Australia.

The following example is a `SearchPlaceIndexForText` request to search for an address, name, city or region called *`Any Town`* in Germany and France.

```
POST /places/v0/indexes/ExamplePlaceIndex/search/text
Content-type: application/json

{
   "Text": "Any Town",
   "FilterCountries": ["DEU","FRA"]
}
```

Filtering by category

You can filter the categories that are returned in your geocode request by using the following optional parameter:

- `FilterCategories` – The categories of results you want returned in your query. You can specify up to 5 categories in one request. You can find the list of Amazon Location Service categories in the Categories section. For example, you can specify `Hotel` to specify only returning hotels in your query.

The following example is a `SearchPlaceIndexForText` request to search for an coffee shop called *`Hometown Coffee`* in the United States.

```
POST /places/v0/indexes/ExamplePlaceIndex/search/text
Content-type: application/json

{
   "Text": "Hometown Coffee",
   "FilterCategories": ["Coffee Shop"],
   "FilterCountries": ["USA"]
}
```

For more details about filtering on categories, see Place categories and filtering results

Geocode in a preferred language

You can set a language preference for results of your search by using the optional `Language` parameter. For example, a search for **`100 Main St, Anytown, USA`** may return `100 Main St, Any Town, USA` by default. But if you select `fr` as the `Language`, then the results may return `100 Rue Principale, Any Town, États-Unis` instead.

- `Language` – A language code to use for rendering the results of your query. The value must be a valid BCP 47 language code. For example, `en` for English.

> **Note**
>
> If `Language` is not specified, or the specified language is not supported for a result, the partner's default language for that result will be used.

The following example is a `SearchPlaceIndexforText` request to search for a place called **Any Town** with the preferred language specified as `de`.

```
POST /places/v0/indexes/ExamplePlaceIndex/search/text
Content-type: application/json
{
   "Text": "Any Town",
   "Language": "de"
}
```

Example response

Example

The following is an example response when you call the SearchPlaceIndexForText operation from the Amazon Location Places APIs. The results include relevant places and the request summary. Two responses are shown, based on selecting Esri or HERE as the partner.

Example request

```
POST /places/v0/indexes/ExamplePlaceIndex/search/text
Content-type: application/json

{
   "Text": "Amazon",
   "MaxResults": 1,
   "FilterCountries": ["USA"],
   "BiasPosition": [-112.10, 46.32]
}
```

Example response (Esri)

```
{
    "Results": [
        {
            "Place": {
                "Country": "USA",
                "Geometry": {
                    "Point": [
                        -112.10667999999998,
                        46.319090000000074
                    ]
                },
                "Interpolated": false,
                "Label": "Amazon, MT, USA",
                "Municipality": "Amazon",
                "Region": "Montana",
                "SubRegion": "Jefferson County"
            },
            "Distance": 523.4619749879726,
            "Relevance": 1
        }
    ],
```

```
    "Summary": {
        "BiasPosition": [
            -112.1,
            46.32
        ],
        "DataSource": "Esri",
        "FilterCountries": [
            "USA"
        ],
        "MaxResults": 1,
        "ResultBBox": [
            -112.10667999999998,
            46.319090000000074,
            -112.10667999999998,
            46.319090000000074
        ],
        "Text": "Amazon"
    }
}
```

Example response (HERE)

```
{
    "Summary": {
        "Text": "Amazon",
        "BiasPosition": [
            -112.1,
            46.32
        ],
        "FilterCountries": [
            "USA"
        ],
        "MaxResults": 1,
        "ResultBBox": [
            -112.10668,
            46.31909,
            -112.10668,
            46.31909
        ],
        "DataSource": "Here"
    },
    "Results": [
        {
```

```
            "Place": {
                "Label": "Amazon, Jefferson City, MT, United States",
                "Geometry": {
                    "Point": [
                        -112.10668,
                        46.31909
                    ]
                },
                "Neighborhood": "Amazon",
                "Municipality": "Jefferson City",
                "SubRegion": "Jefferson",
                "Region": "Montana",
                "Country": "USA",
                "Interpolated": false,
                "TimeZone": {
                    "Name": "America/Denver",
                    "Offset": -25200
                }
            },
            "PlaceId": "AQAAAIAADsn2T3KdrRWeaXLeVEyjNx_JfeTsMB0NVCEAnAZoJ-
o3nqdlJZAdgcT2oWi1w9pS4wXXOk3O1vsKlGsPyHjV4EJxsu289i3hVO_BUPgP7SFoWAi8BW2v7LvAjQ5NfUPy7a1v9a
et39ZQDWSPLZUzgcjN-6VD2gyKkH0Po7gSm8YSJNSQ",            "Distance":
 523.4619749905755
        }
    ]
}
```


Reverse geocoding

Reverse geocoding is a process that converts a set of coordinates into meaningful text, such as an address, a region, a business name, or point of interest. You can use place index resources to submit reverse geocoding requests and incorporate data retrieved from reverse geocoding to display data on a map for your web or mobile application.

This section guides you through how to send a simple reverse geocoding request.

Reverse geocoding

You can submit a simple request to reverse geocode a set of coordinates and convert them to a meaningful address, a point of interest or a general location without an address using the

SearchPlaceIndexForPosition operation. A simple request contains the following required parameter:

- `Position` – A set of coordinates that you want to convert to an address, point of interest, or general location. Defined using the format `[longitude,latitude]`.

To specify a maximum number of results per pages, add the following optional parameter:

- `MaxResults` – Limits the maximum number of results returned in the query response.

If you want to specify a preferred language for the results of your query, use the following optional parameter:

- `Language` – A language code to be used for rendering results. The value must be a valid BCP 47 language code. For example, `en` for English.

> **Note**
>
> If `Language` is not specified, or the specified language is not supported for a result, the partner's default language for that result will be used.

You can use the AWS CLI or the Amazon Location APIs.

API

The following example is a SearchPlaceIndexForPosition request to search the place index resource, *ExamplePlaceIndex*, for a meaningful address, point of interest or general location near the position [*122.3394*,*47.6159*].

```
POST /places/v0/indexes/ExamplePlaceIndex/search/position
Content-type: application/json

{
   "Position": [-122.3394,47.6159],
   "MaxResults": 5,
   "Language": "de"
}
```

AWS CLI

The following example is a search-place-index-for-position command to search the place index resource, *ExamplePlaceIndex*, for a meaningful address, point of interest or general location near the position [*122.3394*,*47.6159*].

```
aws location \
    search-place-index-for-position \
        --index-name ExamplePlaceIndex \
        --position -122.3394 47.6159 \
        --max-results 5 \
        --language de
```

Example response

Example

The following is an example response when calling the SearchPlaceIndexForPosition operation from the Amazon Location Places APIs. The results return relevant places and the request summary. Two responses are shown, based on selecting Esri or Here as the partner.

Example request

```
POST /places/v0/indexes/ExamplePlaceIndex/search/position
Content-type: application/json

{
   "Position": [-122.3394,47.6159],
   "MaxResults": 1
}
```

Example response (Esri)

```
{
    "Results": [
        {
            "Place": {
                "AddressNumber": "2111",
                "Country": "USA",
                "Geometry": {
                    "Point": [
```

```
                        -122.33937999999995,
                        47.61591000000004
                    ]
                },
                "Interpolated": false,
                "Label": "The Spheres, 2111 7th Ave, Seattle, WA, 98121, USA",
                "Municipality": "Seattle",
                "Neighborhood": "Belltown",
                "PostalCode": "98121",
                "Region": "Washington",
                "SubRegion": "King County"
            },
            "Distance": 1.8685861313438727
        }
    ],
    "Summary": {
        "DataSource": "Esri",
        "MaxResults": 1,
        "Position": [
            -122.3394,
            47.6159
        ]
    }
}
```

Example response (HERE)

```
{
    "Summary": {
        "Position": [
            -122.3394,
            47.6159
        ],
        "MaxResults": 1,
        "DataSource": "Here"
    },
    "Results": [
        {
            "Place": {
                "Label": "2111 7th Ave, Seattle, WA 98121-5114, United States",
                "Geometry": {
                    "Point": [
                        -122.33938,
```

```
                        47.61591
                    ]
                },
                "AddressNumber": "2111",
                "Street": "7th Ave",
                "Neighborhood": "Belltown",
                "Municipality": "Seattle",
                "SubRegion": "King",
                "Region": "Washington",
                "Country": "USA",
                "PostalCode": "98121-5114",
                "Interpolated": false,
                "TimeZone": {
                    "Name": "America/Los_Angeles",
                    "Offset": -28800
                }
            },
            "PlaceId": "AQAAAIAADsn2T3KdrRWeaXLeVEyjNx_JfeTsMB0NVCEAnAZoJ-
o3nqdlJZAdgcT2oWi1w9pS4wXXOk3O1vsKlGsPyHjV4EJxsu289i3hVO_BUPgP7SFoWAi8BW2v7LvAjQ5NfUPy7a1v9a
et39ZQDWSPLZUzgcjN-6VD2gyKkH0Po7gSm8YSJNSQ",
            "Distance": 1.868586125090601
        }
    ]
}
```


Autocomplete

Autocomplete provides responsive feedback to end users as they are typing their search query. It provides suggestions for addresses and points of interest based on partial or misspelled free-form text. You can use place index resources to request autocomplete suggestions, and display the resulting suggestions in your application.

Amazon Location does not support storage of autocomplete suggestions. An error is returned if the place index used for an autocomplete call is configured for use with stored geocodes. To use stored geocodes and query for suggestions, create and configure multiple place indexes.

This section describes how to send an autocomplete request. It starts with the most basic form of the request, and then shows optional parameters that you can use to increase the relevance of autocomplete search results.

Using autocomplete

You can submit a simple request for autocomplete suggestions by using the `SearchPlaceIndexForSuggestions` operation. The simplest form of the request has a single required parameter, the query `Text`:

- `Text` – The free-form partial text to use to generate place suggestions. For example, the string `eiffel tow`.

To limit the number of results returned, add the optional `MaxResults` parameter:

- `MaxResults` – Limits the number of results returned in the query response.

You can use the Amazon Location APIs or the AWS CLI.

API

The following example is a `SearchPlaceIndexForSuggestions` request to search the place index resource, *`ExamplePlaceIndex`*, for up to *`5`* suggestions based on the partial place name *`kamp`*.

```
POST /places/v0/indexes/ExamplePlaceIndex/search/suggestions
Content-type: application/json

{
    "Text": "kamp",
    "MaxResults": 5
}
```

AWS CLI

The following example is a `search-place-index-for-suggestions` command to search the place index resource, *`ExamplePlaceIndex`*, for up to *`5`* suggestions based on the partial place name *`kamp`*.

```
aws location \
              search-place-index-for-suggestions \
              --index-name ExamplePlaceIndex \
              --text kamp \
              --max-results 5
```

The call to `SearchPlaceIndexForSuggestions` results in a list of places with a name and an ID for each. You can use those results to present suggestions of what the user might be searching for, as they are typing, such as providing a dropdown list of choices underneath a text box. For example, here are the results for suggestions, based on a user typing *kamp*.


```
{
    "Summary": {
        "Text": "kamp",
        "MaxResults": 5,
        "DataSource": "Esri"
    },
    "Results": [
        {
            "Text": "Kampuchea",
            "PlaceId": "AQAAAIAADsn2T3KdrRWeaXLeVEyjNx_JfeTsMB0NVCEAnAZoJ-
o3nqdlJZAdgcT2oWi1w9pS4wXXOk3O1vsKlGsPyHjV4EJxsu289i3hVO_BUPgP7SFoWAi8BW2v7LvAjQ5NfUPy7a1v9ajT3
et39ZQDWSPLZUzgcjN-6VD2gyKkH0Po7gSm8YSJNSQ"
        },
        {
            "Text": "Kampoul, Kabul, AFG",
            "PlaceId":
 "AQAAAIAAA1mxl_-9ffzXD07rBgo9fh6E01Pd1YKvuT5rz2qBDxqBkhTlgkeiOPR2s5sa3YBLxUqQI8bhymsYcu9R-
DkX3L9QSi3CB5LhNPu160iSFJo6H8S1CrxO3QsJALhrr9mdbg0R4R4YDywkhkeBlnbn7g5C5LI_wYx873WeQZuilwtsGm8j
UeXcb_bg"
        },
        {
            "Text": "Kampala, UGA",
            "PlaceId":
 "AQAAAIAAzZfZt3qMruKGObyhP6MM0pqy2L8SULlVWT7a3ertLBRS6Q5n7I4s9D7E0nRHADAj7mL7kvX1Q8HD-
mpuiATXNJ1Ix4_V_1B15zHe8jlYKMWvXbgbO8cMpgR2fqYqZMR1x-
dfBOO8OoqujKZldvPIDK1kNe3GwcaqvvMWWPMeaGd203brFynubAe-MmFF-Gjz-WBMfUy9og6MV7bkk6NGCA"
        },
        {
            "Text": "Kampar, Riau, IDN",
            "PlaceId": "AQAAAIAAvbXXx-
srOi111tHOkPdao0GF7WQ_KaZ444SEnevycp6Gtf_2JWgPfCE5bIQCYwya1uZQpX2a8YJoFm2K7Col4fLu7IK0yYOLhZx4k
        },
        {
            "Text": "Kampung Pasir Gudang Baru, Johor, MYS",
            "PlaceId":
 "AQAAAIAA4HLQHdjUDcaaXLE9wtNIT1cjQYLgkBnMoG2eNN0AaQ8PJoWabLRXmmPUaAj8MAD6vT0i6zqaun5Mixyj7vnYX
        }
    ]
```

```
}
```

The next section explains how to use the `PlaceID` from these results.

Using the autocomplete results

The call to `SearchPlaceIndexForSuggestions` results in a list of places with a name and an ID for each. You can use those results to present suggestions of what the user might be searching for, as they are typing, such as providing a dropdown list of choices underneath a text box. When the user chooses one of the results, you can then call the GetPlace operation with the ID of their selection to return the details of that place, including location, address, or other details.

> **Note**
>
> A `PlaceId` is valid only if all of the following are the same in the original search request, and the call to `GetPlace`.
>
> - Customer AWS account
> - AWS Region
> - Data provider specified in the place index resource

Typically, you use `GetPlace` with the Amazon Location APIs. The following example is a `GetPlace` request to find one of the suggestions from the previous section. This example is based on the partial place name *kamp*.

```
POST /places/v0/indexes/ExamplePlaceIndex/
places/AQAAAIAADsn2T3KdrRWeaXLeVEyjNx_JfeTsMB0NVCEAnAZoJ-
o3nqdlJZAdgcT2oWi1w9pS4wXXOk3O1vsKlGsPyHjV4EJxsu289i3hVO_BUPgP7SFoWAi8BW2v7LvAjQ5NfUPy7a1v9ajT3
et39ZQDWSPLZUzgcjN-6VD2gyKkH0Po7gSm8YSJNSQ
```

Autocomplete near a position

When you search for autocomplete place suggestions by using `SearchPlaceIndexForSuggestions`, you can get more locally-relevant suggestions by adding the following optional parameter:

- `BiasPosition` – The position you want to search nearby. Defined as `[longitude, latitude]`.

The following example uses a SearchPlaceIndexForSuggestions request to search the place index resource *ExamplePlaceIndex* for place suggestions matching the partial query *kamp* near the position [*32.5827*,*0.3169*].

```
POST /places/v0/indexes/ExamplePlaceIndex/search/suggestions
Content-type: application/json

{
    "Text": "kamp",
    "BiasPosition": [32.5827,0.3169]
}
```

The suggestions returned for the same `Text` can be different if a different `BiasPosition` is chosen, such as [*-96.7977*, *32.7776*].

Autocomplete within a bounding box

You can narrow your autocomplete search to receive only suggestions for places which are located within a given boundary by adding the following optional parameter:

- `FilterBBox` – A bounding box that you specify to filter your results to coordinates within the box's boundaries. Defined as `[LongitudeSW, LatitudeSW, LongitudeNE, LatitudeNE]`

> **Note**
>
> A request can't contain both the `FilterBBox` and `BiasPosition` parameters. Specifying both parameters in the request returns a `ValidationException` error.

The following example uses a SearchPlaceIndexForSuggestions request to search the place index resource *ExamplePlaceIndex* for place suggestions matching the partial query *kamp*, and which are contained within the bounding box where:

- The longitude of the southwest corner of the bounding box is *32.5020*.
- The latitude of the southwest corner of the bounding box is *0.2678*.
- The longitude of the northeast corner of the bounding box is *32.6129*.
- The latitude of the northeast corner of the bounding box is *0.3502*.

```
POST /places/v0/indexes/ExamplePlaceIndex/search/suggestions
Content-type: application/json

{
    "Text": "kamp",
    "FilterBBox": [
        32.5020, 0.2678,
        32.6129, 0.3502
    ]
}
```

The suggestions returned for the same `Text` are different if a different `FilterBBox` is chosen, such as [*-97.9651*, *32.0640*, *-95.1196*, *34.0436*].

Autocomplete within a country

You can narrow your autocomplete search to receive only suggestions for places which are located within a given country, or set of countries, by adding the following optional parameter:

- `FilterCountries` – The countries you want to search for place suggestions within. You can specify up to 100 countries in one request using a ISO 3166 three-letter country code. For example, use AUS for Australia.

The following example uses a `SearchPlaceIndexForSuggestions` request to search the place index resource *ExamplePlaceIndex* for place suggestions matching the partial query *kamp* and which are contained within Uganda, Kenya, or Tanzania:

```
POST /places/v0/indexes/ExamplePlaceIndex/search/suggestions
Content-type: application/json

{
    "Text": "kamp",
    "FilterCountries": ["UGA", "KEN", "TZA"]
}
```

The suggestions returned for the same `Text` are different if a different `FilterCountries` list is chosen, such as ["*USA*"].

Example response

The following is an example response of suggested autocompletions for the SearchPlaceIndexForSuggestions operation, using the text *kamp*.


```
{
    "Summary": {
        "Text": "kamp",
        "MaxResults": 5,
        "DataSource": "Esri"
    },
    "Results": [
        {
            "Text": "Kampuchea",
            "PlaceId": "AQAAAIAADsn2T3KdrRWeaXLeVEyjNx_JfeTsMB0NVCEAnAZoJ-
o3nqdlJZAdgcT2oWi1w9pS4wXXOk3O1vsKlGsPyHjV4EJxsu289i3hVO_BUPgP7SFoWAi8BW2v7LvAjQ5NfUPy7a1v9ajT3
et39ZQDWSPLZUzgcjN-6VD2gyKkH0Po7gSm8YSJNSQ"
        },
        {
            "Text": "Kampoul, Kabul, AFG",
            "PlaceId":
 "AQAAAIAAA1mxl_-9ffzXD07rBgo9fh6E01Pd1YKvuT5rz2qBDxqBkhTlgkeiOPR2s5sa3YBLxUqQI8bhymsYcu9R-
DkX3L9QSi3CB5LhNPu160iSFJo6H8S1CrxO3QsJALhrr9mdbg0R4R4YDywkhkeBlnbn7g5C5LI_wYx873WeQZuilwtsGm8j
UeXcb_bg"
        },
        {
            "Text": "Kampala, UGA",
            "PlaceId":
 "AQAAAIAAzZfZt3qMruKGObyhP6MM0pqy2L8SULlVWT7a3ertLBRS6Q5n7I4s9D7E0nRHADAj7mL7kvX1Q8HD-
mpuiATXNJ1Ix4_V_1B15zHe8jlYKMWvXbgbO8cMpgR2fqYqZMR1x-
dfBOO8OoqujKZldvPIDK1kNe3GwcaqvvMWWPMeaGd203brFynubAe-MmFF-Gjz-WBMfUy9og6MV7bkk6NGCA"
        },
        {
            "Text": "Kampar, Riau, IDN",
            "PlaceId": "AQAAAIAAvbXXx-
srOi111tHOkPdao0GF7WQ_KaZ444SEnevycp6Gtf_2JWgPfCE5bIQCYwya1uZQpX2a8YJoFm2K7Col4fLu7IK0yYOLhZx4k
        },
        {
            "Text": "Kampung Pasir Gudang Baru, Johor, MYS",
            "PlaceId":
 "AQAAAIAA4HLQHdjUDcaaXLE9wtNIT1cjQYLgkBnMoG2eNN0AaQ8PJoWabLRXmmPUaAj8MAD6vT0i6zqaun5Mixyj7vnYX
        }
    ]
```

```
}
```

Using place IDs

Searching for places returns a list of results. Most results include a `PlaceId` for that result. You can use a `PlaceId` in a `GetPlace` operation to return the information about that place (including name, address, location, or other details).

> **Note**
>
> Using SearchPlaceIndexForSuggestions will return `PlaceId` results for any place indexes created with any data source. Using SearchPlaceIndexForText or SearchPlaceIndexForPosition will return a `PlaceId` only if the data source used is HERE.

Each `PlaceId` uniquely defines the place it refers to, but a single place can have more than one `PlaceId` over time, and based on the context. The following rules describe the uniqueness and longevity of a `PlaceId`.

- The `PlaceId` returned in calls that you make is specific to your AWS account, to the AWS Region, and to the data provider in your `PlaceIndex` resource. `GetPlace` will find results only when these three attributes match the original call that created the `PlaceId`.
- The `PlaceId` for a place will change when the data about that place changes. For example, when the business that it refers to moves location or changes names.
- The `PlaceId` returned from a repeated search call may change when the backend service makes an update. The older `PlaceId` will continue to be found, but new calls to search may return a different ID.

The `PlaceId` is a string. There is no specific limit to the length of a `PlaceId`. The following is an example of a valid `PlaceId`.

```
AQAAAIAADsn2T3KdrRWeaXLeVEyjNx_JfeTsMB0NVCEAnAZoJ-
o3nqdlJZAdgcT2oWi1w9pS4wXXOk3O1vsKlGsPyHjV4EJxsu289i3hVO_BUPgP7SFoWAi8BW2v7LvAjQ5NfUPy7a1v9ajT3
et39ZQDWSPLZUzgcjN-6VD2gyKkH0Po7gSm8YSJNSQ
```

Calling `GetPlace` with a `PlaceId` for a place whose data has changed (for example, a business location that has gone out of business), will result in a `404`, `ResourceNotFound` error. Calling

`GetPlace` with a `PlaceId` that is not valid, or one out of context, such as from another AWS account, will return a `400`, `ValidationException` error.

While you can use PlaceID in subsequent requests, PlaceID is not intended to be a permanent identifier and the ID can change between consecutive API calls. Please see the following PlaceID behaviour for each data provider:

- **Esri**: Place IDs will change every quarter at a minimum. The typical time period for these changes would be March, June, September, and December. Place IDs might also change between the typical quarterly change but that will be much less frequent.
- **HERE**: We recommend that you cache data for no longer than a week to keep your data data fresh. You can assume that less than 1% ID shifts will release over release which is approximately 1 - 2 times per week.
- **Grab**: Place IDs can expire or become invalid in the following situations.
  - **Data operations**: The POI may be removed from Grab POI database by Grab Map Ops based on the ground-truth, such as being closed in the real world, being detected as a duplicate POI, or having incorrect information. Grab will synchronize data to the Waypoint environment on weekly basis.
  - **Interpolated POI**: Interpolated POI is a temporary POI generated in real time when serving a request, and it will be marked as derived in the `place.result_type` field in the response. The information of interpolated POIs will be retained for at least 30 days, which means that within 30 days, you are able to obtain POI details by Place ID from Place Details API. After 30 days, the interpolated POIs(both Place ID and details) may expire and inaccessible from the Places Details API.

Place categories and filtering results

Places are categorized. If you search for a business, the business might be a `Restaurant`, for example. Even the results of a search for an address can be categorized by whether it was matched to an *address*, *street*, or *intersection*.

Broadly, Amazon Location Service categorizes places into *Place types*. Points of interest are further categorized into *Point of interest types*.

> **Note**
>
> Not all results will have categories.

You can use the categories to filter your geocoding searches.

Filtering results

When you are using `SearchPlaceIndexForText`, you can filter the results that are returned by the categories that you want to use. For example:

- If you want to search for a place called "Hometown Coffee", and only return results that are categorized as coffee shops, you can do that by calling `SearchPlaceIndexForText` and include the *Point of interest category*, `Coffee Shop` in the `FilterCategories` parameter.
- When searching for "123 Main St, Anytown, WA, 98123, USA", you can filter result to just addresses, so you don't get matches on, for example, the postal code. Filter to just addresses by including *Place type*, `AddressType` in the `FilterCategories` parameter.

> **Note**
>
> Not all data providers support filtering, or support it in the same way. For more information, see Filtering limitations by data provider.

The next section lists the categories that you can filter on.

Categories

The following lists show the categories that Amazon Location Service uses to categorize and filter. These categories are used in all languages, independent of the language parameter is set to a different language.

> **Note**
>
> Amazon Location Service maps data provider categories to this set of categories. If a data provider puts a place into a category that is not part of the Amazon Location Service

category list, the provider category will be included in the results as a *supplemental category*.

Place types – These types are used to indicate the type of match that was used to find the result.

- `AddressType` – Returned when the result was matched to an address.
- `StreetType` – Returned when the result was matched to a street.
- `IntersectionType` – Returned when the result was matched to the intersection of two streets.
- `PointOfInterestType` – Returned when the result was match to a point of interest, such as a business, or civic location.
- `CountryType` – Returned when the result was matched to a country or major region.
- `RegionType` – Returned when the result was matched to a region within a country, such as a state or province.
- `SubRegionType` – Returned when the result was matched to a subregion within a country, such as a county or metropolitan area.
- `MunicipalityType` – Returned when the result was matched to a city or town.
- `NeighborhoodType` – Returned when the result was matched to a neighborhood or area within a city.
- `PostalCodeType` – Returned when the result was matched to a postal code.

Point of interest categories – These categories are used to indicate the type of business or location for point of interest results.

- `Airport`
- `Amusement Park`
- `Aquarium`
- `Art Gallery`
- `ATM`
- `Bakery`
- `Bank`
- `Bar`

- Beauty Salon
- Bus Station
- Car Dealer
- Car Rental
- Car Repair
- Car Wash
- Cemetery
- Cinema
- City Hall
- Clothing Store
- Coffee Shop
- Consumer Electronics Store
- Convenience Store
- Court House
- Dentist
- Embassy
- Fire Station
- Fitness Center
- Gas Station
- Government Office
- Grocery
- Higher Education
- Hospital
- Hotel
- Laundry
- Library
- Liquor Store
- Lodging

- Market
- Medical Clinic
- Motel
- Museum
- Nightlife
- Nursing Home
- Park
- Parking
- Pet Store
- Pharmacy
- Plumbing
- Police Station
- Post Office
- Religious Place
- Restaurant
- School
- Shopping Mall
- Sports Center
- Storage
- Taxi Stand
- Tourist Attraction
- Train Station
- Veterinary Care
- Zoo

Filtering limitations by data provider

Not all providers have the same filtering functionality. The following table describes the differences.

Provider	APIs with filter support	Categories supported for filtering	Return values
Esri	`SearchPlaceIndexForText`, `SearchPlaceIndexForSuggestions`	Filter by *Place types* and *Point of interest categories*.	Categories are returned by `SearchPlaceIndexForText`, `SearchPlaceIndexForPosition`, and `GetPlace`
Here	`SearchPlaceIndexForText`, `SearchPlaceIndexForSuggestions`	Filter by *Place types* only.	Categories are returned by `SearchPlaceIndexForText` and `SearchPlaceIndexForSuggestions`, `SearchPlaceIndexForPosition`, and `GetPlace`
Grab	not supported	not supported	not supported
Open Data	n/a (searching places not supported)	n/a	n/a

Amazon Aurora PostgreSQL user-defined functions for Amazon Location Service

You can use Amazon Location Service to work with coordinates and addresses stored in database tables to clean and enrich your geospatial data.

For example:

- You can use geocoding to convert addresses to coordinates to normalize and fill gaps in data for addresses stored in a database table.
- You can geocode addresses to obtain their positions and use the coordinates with database spatial functions, such as a function that shows rows in a specified area.
- You can use enriched data to generate automated reporting, such as generating an automated report that illustrates all devices in a given area, or an automated report for machine learning that illustrates areas with higher failure rates when sending location updates.

This tutorial shows how to format and enrich addresses stored in an Amazon Aurora PostgreSQL database table using Amazon Location Service.

- **Amazon Aurora PostgreSQL** – A fully managed relational database engine, compatible with MySQL and PostgreSQL, that outputs up to five times the throughput of MySQL and up to three times the throughput of PostgreSQL without changing most of your existing application. For more information, see What is Amazon Aurora? in the *Amazon Aurora User Guide*.

> **Important**
>
> The resulting application in this tutorial uses a place index that stores geocoding results. For information about applicable charges for storing geocoding results, see Amazon Location Service pricing.

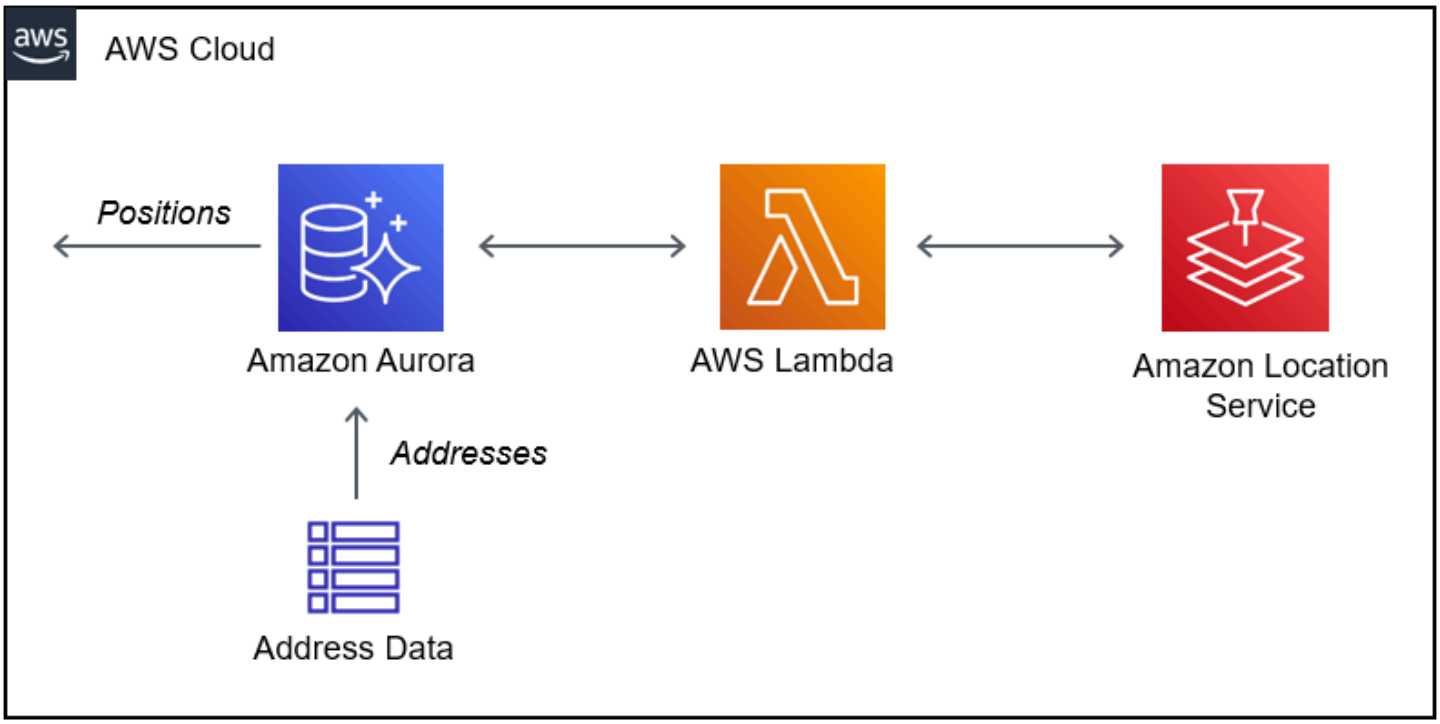
Sample code is available in the Amazon Location Service samples repository on GitHub, which includes an AWS CloudFormation template.

Topics

- Overview
- Prerequisites
- Quick start
- Create a place index resource
- Create an AWS Lambda function for geocoding
- Grant Amazon Aurora PostgreSQL access to AWS Lambda

- Invoke the AWS Lambda function
- Enriching a database containing address data
- Next steps

Overview



The architecture involves the following integrations:

- This solution uses an Amazon Location place index resource to support geocoding queries using the operation `SearchPlaceIndexForText`.
- AWS Lambda uses a Python Lambda that geocodes addresses when an IAM policy gives permission to allow AWS Lambda to call the Amazon Location geocoding operation, `SearchPlaceIndexForText`.
- Grant permission to Amazon Aurora PostgreSQL to invoke the geocoding Lambda function using an SQL user-defined function.

Prerequisites

Before you begin, you need the following prerequisites:

- An Amazon Aurora PostgreSQL cluster. For more information about Creating an Amazon Aurora DB cluster, see the *Amazon Aurora User Guide*.

> **Note**
>
> If your Amazon Aurora cluster isn't publicly available, you must also configure Amazon Aurora to connect to AWS Lambda in a virtual private cloud (VPC) in your AWS account. For more information, see Grant Amazon Aurora PostgreSQL access to AWS Lambda.

- An SQL developer tool to connect to the Amazon Aurora PostgreSQL cluster.

Quick start

As an alternative to going through the steps in this tutorial, you can launch a quick stack to deploy an AWS Lambda function supporting the Amazon Location operation `SearchPlaceIndexForText`. This automatically configures your AWS account to allow Amazon Aurora to call AWS Lambda.

Once you configure your AWS account, you will need to:

- Add the Lambda feature to Amazon Aurora. See Add the IAM role to a Amazon Aurora DB cluster in Grant Amazon Aurora PostgreSQL access to AWS Lambda.
- Load the user-defined function into your database. See Invoke the AWS Lambda function.

Create a place index resource

Start by creating a place index resource to support geocoding queries.

1. Open the Amazon Location Service console at https://console.aws.amazon.com/location/.
2. In the left navigation pane, choose **Place indexes**.
3. Fill out the following boxes:

   - **Name** – Enter a name for the place index resource. For example, *`AuroraPlaceIndex`*. Maximum 100 characters. Valid entries include alphanumeric characters, hyphens, periods, and underscores.

- **Description** – Enter an optional description. For example, *Place index for Amazon Aurora*.

4. Under **Data providers**, choose an available data provider to use with your place index resource. If you have no preference, we recommend starting with *Esri*.
5. Under **Data storage options**, specify **Yes, results will be stored**. This indicates that you intend to save the geocoding results in a database.
6. (Optional) Under **Tags**, enter a tag **Key** and **Value**. This adds a tag your new place index resource. For more information, see Tagging your resources.
7. Choose **Create place index**.

Create an AWS Lambda function for geocoding

To create a connection between Amazon Aurora PostgreSQL and Amazon Location Service, you need an AWS Lambda function to handle requests from the database engine. This function translates the Lambda user-defined function event and calls the Amazon Location operation `SearchPlaceIndexForText`.

You can create the function using the AWS Lambda console, the AWS Command Line Interface, or the AWS Lambda APIs.

To create a Lambda user-defined function using the console

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. From the left navigation, choose **Functions**.
3. Choose **Create Function**, and make sure that **Author from scratch** is selected.
4. Fill out the following boxes:

   - **Function name** – Enter a unique name for your function. Valid entries include alphanumeric characters, hyphens, and underscores with no spaces. For example, *AuroraGeocoder*.
   - **Runtime** – Choose *Python 3.8*.

5. Choose **Create function**.
6. Choose the **Code** tab to open the editor.
7. Overwrite the placeholder code in `lambda_function.py` with the following:

```
from os import environ
```

```
import boto3
from botocore.config import Config

# load the place index name from the environment, falling back to a default
PLACE_INDEX_NAME = environ.get("PLACE_INDEX_NAME", "AuroraPlaceIndex")

location = boto3.client("location", config=Config(user_agent="Amazon Aurora
 PostgreSQL"))

"""
This Lambda function receives a payload from Amazon Aurora and translates it to
an Amazon Location `SearchPlaceIndex` call and returns the results as-is, to be
post-processed by a PL/pgSQL function.
"""
def lambda_handler(event, context):
    kwargs = {}

    if event.get("biasPosition") is not None:
        kwargs["BiasPosition"] = event["biasPosition"]

    if event.get("filterBBox") is not None:
        kwargs["FilterBBox"] = event["filterBBox"]

    if event.get("filterCountries") is not None:
        kwargs["FilterCountries"] = event["filterCountries"]

    if event.get("maxResults") is not None:
        kwargs["MaxResults"] = event["maxResults"]

    return location.search_place_index_for_text(
        IndexName=PLACE_INDEX_NAME,
        Text=event["text"],
        **kwargs)["Results"]
```

8. If you've named your place index something other than *AuroraPlaceIndex*, create an environment variable named PLACE_INDEX_NAME to assign the resource name to:

   - From the **Configuration** tab, choose **Environment Variables**.
   - Choose **Edit**, then choose **Add environment variable**.
   - For **Key**: Enter PLACE_INDEX_NAME.
   - For **Value**: Enter the name of your place index resource.

9. Choose **Deploy** to save the updated function.

10. From the **Test** drop-down menu, choose **Configure test Event**.
11. Choose **Create new test event**.
12. Enter the following test event:

```
{
  "text": "Baker Beach",
  "biasPosition": [-122.483, 37.790],
  "filterCountries": ["USA"]
}
```

13. Choose **Test** to test the Lambda function.
14. Choose the **Configuration** tab.
15. Under **General configuration**: Choose **Permissions**.
16. Under **Execution role**: Choose the hyper linked **Role name** to grant Amazon Location Service permissions to your Lambda function.
17. Under the **Permissions** tab: Select the **Add permissions** drop down, then choose **Create inline policy**.
18. Choose the **JSON** tab.
19. Add the following IAM policy:

    - The following policy gives permission to send `SearchPlaceIndexForText` to the place index resource *`AuroraPlaceIndex`*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "geo:SearchPlaceIndexForText",
      "Resource": "arn:aws:geo:<Region>:<AccountId>:place-index/AuroraPlaceIndex"
    }
  ]
}
```

20. Choose **Review policy**.
21. Enter a policy name. For example, *`AuroraPlaceIndexReadOnly`*.
22. Choose **Create policy**.

Grant Amazon Aurora PostgreSQL access to AWS Lambda

Before Amazon Aurora PostgreSQL can invoke an AWS Lambda function, you must grant access permission.

If your Amazon Aurora PostgreSQL cluster isn't publicly accessible, you will need to first create a VPC endpoint for AWS Lambda in order for Amazon Aurora to call your Lambda function.

Create a VPC Endpoint for AWS Lambda

> **Note**
>
> This step is only required if your Amazon Aurora PostgreSQL cluster isn't publicly accessible.

1. Open the Amazon Virtual Private Cloud Console.
2. In the left navigation, choose **Endpoints**.
3. Choose **Create endpoint**.
4. In the **Service Name** filter, enter "lambda", then choose `com.amazonaws.<region>.lambda`.
5. Choose the VPC containing your Aurora cluster.
6. Choose a subnet for each availability zone.
7. In the **Security group** filter, enter "default" or the name of the security group your Aurora cluster is a member of, then choose the security group.
8. Choose **Create endpoint**.

Create an IAM policy to grant permission to invoke your AWS Lambda function

1. Open the IAM console.
2. In the left navigation, expand **Access Management** to choose **Policies**.
3. Choose **Create policy**.
4. On the **JSON** tab, input the following policy:

   - The following is an example of an IAM policy that grants Amazon Aurora PostgreSQL permission to invoke the `AuroraGeocoder` AWS Lambda function.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": [
                "arn:aws:lambda:<Region>:<AccountId>:function:AuroraGeocoder"
            ]
        }
    ]
}
```

5. Choose **Next: Tags** to add optional tags.
6. Choose **Next: Review**.
7. Review your policy and enter the following details for the policy:
   - **Name** – Use alphanumeric and '+=,.@-_' characters. Maximum 128 characters. For example, *AuroraGeocoderInvoke*.
   - **Description** – Enter an optional description. Use alphanumeric and '+=,.@-_' characters. Maximum 1000 characters.
8. Choose **Create policy**. Note the ARN for this policy, which you use to attach the policy to an IAM role.

Create an IAM role to give permission to Amazon Relational Database Service (Amazon RDS)

By creating an IAM role, Amazon Aurora PostgreSQL can assume the role on your behalf to access your Lambda function. For more information, see Creating a role to delegate permissions to an IAM user in the *IAM User Guide*.

The following example is an AWS CLI command that creates a role named *AuroraGeocoderInvokeRole*:

```
aws iam create-role  --role-name rds-lambda-role --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
```

```
        "Principal": {
            "Service": "rds.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}'
```

Attach your IAM policy to the IAM role

When you have an IAM role, attach the IAM policy that you've created.

The following example is an AWS CLI command that attaches the policy *AuroraGeocoderInvoke* to the role *AuroraGeocoderInvokeRole*.

```
aws iam attach-role-policy  --policy-arn AuroraGeocoderInvoke  --role-
name AuroraGeocoderInvokeRole
```

Add the IAM role to a Amazon Aurora DB cluster

The following example is an AWS CLI command to add an IAM role to a Amazon Aurora PostgreSQL DB cluster named *MyAuroraCluster*.

```
aws rds add-role-to-db-cluster \
--db-cluster-identifier MyAuroraCluster \
--feature-name Lambda \
--role-arn AuroraGeocoderInvokeRole   \
--region your-region
```

Invoke the AWS Lambda function

After you grant permission to Amazon Aurora PostgreSQL to invoke your geocoding Lambda function, you can create an Amazon Aurora PostgreSQL user-defined function to invoke the geocoding AWS Lambda function. For more information, see Invoking an AWS Lambda function from an Amazon Aurora PostgreSQL DB cluster in the *Amazon Aurora User Guide*.

Install the required PostgreSQL extensions

To install the required PostgreSQL extensions `aws_lambda` and `aws _commons` extensions, see Overview of using a Lambda function in the *Amazon Aurora User Guide*.

```
CREATE EXTENSION IF NOT EXISTS aws_lambda CASCADE;
```

Install the required PostGIS extensions

PostGIS is an extension to PostgreSQL for storing and managing spatial information. For more information, see Working with the PostGIS extension on the *Amazon Relational Database Service User Guide*.

```
CREATE EXTENSION IF NOT EXISTS postgis;
```

Create an SQL user-defined function that invokes the Lambda function

In an SQL editor, create a new user-defined function `f_SearchPlaceIndexForText` to invoke the function *`AuroraGeocoder`*:

```
CREATE OR REPLACE FUNCTION f_SearchPlaceIndexForText(
  text text,
  bias_position geometry(Point, 4326) DEFAULT NULL,
  filter_bbox box2d DEFAULT NULL,
  filter_countries text[] DEFAULT NULL,
  max_results int DEFAULT 1
)
 RETURNS TABLE (
   label text,
   address_number text,
   street text,
   municipality text,
   postal_code text,
   sub_region text,
   region text,
   country text,
   geom geometry(Point, 4326)
 )
 LANGUAGE plpgsql
 IMMUTABLE
AS $function$
begin
    RETURN QUERY
    WITH results AS (
      SELECT json_array_elements(payload) rsp
      FROM aws_lambda.invoke(
```

```
        aws_commons.create_lambda_function_arn('AuroraGeocoder'),
        json_build_object(
          'text', text,
          'biasPosition',
          CASE WHEN bias_position IS NOT NULL THEN
            array_to_json(ARRAY[ST_X(bias_position), ST_Y(bias_position)])
          END,
          'filterBBox',
          CASE WHEN filter_bbox IS NOT NULL THEN
            array_to_json(ARRAY[ST_XMin(filter_bbox), ST_YMin(filter_bbox),
 ST_XMax(filter_bbox), ST_YMax(filter_bbox)])
          END,
          'filterCountries', filter_countries,
          'maxResults', max_results
        )
      )
    )
    SELECT
      rsp->'Place'->>'Label' AS label,
      rsp->'Place'->>'AddressNumber' AS address_number,
      rsp->'Place'->>'Street' AS street,
      rsp->'Place'->>'Municipality' AS municipality,
      rsp->'Place'->>'PostalCode' AS postal_code,
      rsp->'Place'->>'SubRegion' AS sub_region,
      rsp->'Place'->>'Region' AS region,
      rsp->'Place'->>'Country' AS country,
      ST_GeomFromGeoJSON(
        json_build_object(
          'type', 'Point',
            'coordinates', rsp->'Place'->'Geometry'->'Point'
        )
      ) geom
    FROM results;
end;
$function$;
```

Call the SQL function to geocode from Aurora

Running the SQL statement invokes the Lambda function *AuroraGeocoder*, which takes address records from the database table in the Amazon Aurora PostgreSQL database and geocodes them using a place index resource.

> **Note**
>
> Amazon Aurora PostgreSQL invokes the Lambda function for each call to the SQL user-defined function.
> If you are geocoding 50 rows, Amazon Aurora PostgreSQL invokes the Lambda function 50 times. One invocation for each row.

The following `f_SearchPlaceIndexForText` SQL function makes requests to Amazon Location's `SearchPlaceIndexForText` API through the *`AuroraGeocoder`* Lambda function. The function returns a `geom` column that's a PostGIS geometry, which `ST_AsText(geom)` converts to text.

```
SELECT *, ST_AsText(geom)
FROM f_SearchPlaceIndexForText('Vancouver, BC');
```

By default, the return will contain one row. To request additional rows, up to the `MaxResults` limit, run the following SQL statement while providing a `BiasPosition` and limiting to results in Canada.

```
SELECT *
FROM f_SearchPlaceIndexForText('Mount Pleasant', ST_MakePoint(-123.113, 49.260), null,
 '{"CAN"}', 5);
```

To filter results using a bounding box, then pass a `Box2D` as `filter_bbox`:

- `FilterBBox` – Filters the results by returning places within a bounding box. This is an optional parameter.

```
SELECT *
FROM f_SearchPlaceIndexForText('Mount Pleasant', null, 'BOX(-139.06 48.30, -114.03
 60.00)'::box2d, '{"CAN"}', 5);
```

For more information on PostGIS types and functions, see the PostGIS Reference.

Enriching a database containing address data

You can construct a formatted address and simultaneously normalize and geocode using the Amazon Location operation `SearchPlaceIndexForText` given a database table with the following data broken out into the following columns:

- `id`
- `address`
- `city`
- `state`
- `zip`

```
WITH source_data AS (
  SELECT
    id,
    address || ', ' || city || ', ' || state || ', ' || zip AS formatted_address
  FROM addresses
),
geocoded_data AS (
  SELECT
    *,
    (f_SearchPlaceIndexForText(formatted_address)).*
  FROM source_data
)
SELECT
  id,
  formatted_address,
  label normalized_address,
  ST_Y(geom) latitude,
  ST_X(geom) longitude
FROM geocoded_data
-- limit the number of rows that will be geocoded; remove this to geocode the entire
 table
LIMIT 1;
```

The following example illustrates one resulting datatable row:

```
 id |      formatted_address         |            normalized_address              |
 latitude     |     longitude
```

```
----+--------------------------------+-------------------------------------------
+-----------------+------------------
 42 | 123 Anytown Ave N, Seattle, WA | 123 Anytown Ave N, Seattle, WA, 12345, USA |
 47.6223000127926 | -122.336745971039
(1 row)
```

Update the database table and populate columns

The following example updates the table and populates columns with results of `SearchPlaceIndexForText` queries:

```
WITH source_data AS (
  -- select rows that have not been geocoded and created a formatted address for each
  SELECT
    id,
    address || ', ' || city || ', ' || state || ', ' || zip AS formatted_address
  FROM addresses
  WHERE label IS NULL
  -- limit the number of rows that will be geocoded; remove this to geocode the entire
 table
  LIMIT 1
),
geocoded_data AS (
  -- geocode each row and keep it linked to the source's ID
  SELECT
    id,
    (f_SearchPlaceIndexForText(formatted_address)).*
  FROM source_data
)
UPDATE addresses
-- populate columns
SET
  normalized_address = geocoded_data.label,
  latitude = ST_Y(geocoded_data.geom),
  longitude = ST_X(geocoded_data.geom)
FROM geocoded_data
-- ensure that rows match
WHERE addresses.id = geocoded_data.id;
```

Next steps

Sample code is available in the Amazon Location Service samples repository on GitHub, which includes an AWS CloudFormation template.

Managing your place index resources

You can manage your place index resources using the Amazon Location console, the AWS CLI, or the Amazon Location APIs.

List your place index resources

You can view your place index resources list using the Amazon Location console, the AWS CLI, or the Amazon Location APIs:

Console

To view a list of place index resources using the Amazon Location console

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. Choose **Place indexes** from the left navigation pane.
3. View a list of your place index resources under the **My place indexes**.

API

Use the `ListPlaceIndexes` operation from the Amazon Location Places APIs.

The following example is an API request to get a list of place index resources in the AWS account.

```
POST /places/v0/list-indexes
```

The following is an example response for `ListPlaceIndexes`:

```
{
   "Entries": [
      {
         "CreateTime": 2020-10-30T01:38:36Z,
         "DataSource": "Esri",
         "Description": "string",
         "IndexName": "ExamplePlaceIndex",
         "UpdateTime": 2020-10-30T01:40:36Z
      }
   ],
   "NextToken": "1234-5678-9012"
```

```
}
```

CLI

Use the `list-place-indexes` command.

The following example is an AWS CLI to get a list of place index resources in the AWS account.

```
aws location list-place-indexes
```

Get place index resource details

You can get details about any place index resource in your AWS account using the Amazon Location console, the AWS CLI, or the Amazon Location APIs:

Console

To view the details of a place index resource using the Amazon Location console

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. Choose **Place indexes** from the left navigation pane.
3. Under **My place indexes**, select the name link of the target place index resource.

API

Use the `DescribePlaceIndex` operation from the Amazon Location Place APIs.

The following example is an API request to get the place index resource details for *`ExamplePlaceIndex`*.

```
GET /places/v0/indexes/ExamplePlaceIndex
```

The following is an example response for `DescribePlaceIndex`:

```
{
   "CreateTime": 2020-10-30T01:38:36Z,
   "DataSource": "Esri",
   "DataSourceConfiguration": {
```

```
      "IntendedUse": "SingleUse"
   },
   "Description": "string",
   "IndexArn": "arn:aws:geo:us-west-2:123456789012:place-indexes/ExamplePlaceIndex",
   "IndexName": "ExamplePlaceIndex",
   "Tags": {
      "string" : "string"
   },
   "UpdateTime": 2020-10-30T01:40:36Z
}
```

CLI

Use the `describe-place-index` command.

The following example is an AWS CLI to get the place index resource details for *ExamplePlaceIndex*.

```
aws location describe-place-index \
    --index-name "ExamplePlaceIndex"
```

Delete a place index resource

You can delete a place index resource from your AWS account using the Amazon Location console, the AWS CLI, or the Amazon Location APIs:

Console

To delete a place index resource using the Amazon Location console

> **Warning**
>
> This operation deletes the resource permanently.

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. Choose **Place indexes** from the left navigation pane.
3. Under **My place index**, select the target place index resource.

4. Choose **Delete place index**.

API

Use the DeletePlaceIndex operation from the Amazon Location Places APIs.

The following example is an API request to delete the place index resource *ExamplePlaceIndex*.

```
DELETE /places/v0/indexes/ExamplePlaceIndex
```

The following is an example success response for DeletePlaceIndex:

```
HTTP/1.1 200
```

CLI

Use the delete-place-index command.

The following example is an AWS CLI command to delete the place index resource *ExamplePlaceIndex*.

```
aws location delete-place-index \
    --index-name "ExamplePlaceIndex"
```

Calculating routes using Amazon Location Service

Amazon Location lets you select a data provider for calculating a route by creating and configuring a route calculator resource.

You can use the route calculator resource to calculate a route given specific parameters using the AWS SDK, or the REST API endpoints. Use this route calculator resource to calculate routes between an origin, a destination and up to 23 waypoints for different modes of transportation, avoidances, and traffic conditions.

You can also use the route calculator resource to create inputs for your route planning algorithms or products by calculating a route matrix. Calculate the travel time and travel distance between a set of departure positions and a set of destination positions. Route planning software can use that time and distance data to optimize a route or a set of routes; for example, if you are planning

multiple delivery routes, and want to find the best route and time for each stop. You can calculate a matrix of routes for different modes of transportation, avoidances, and traffic conditions.

> **Note**
>
> For an overview of routing concepts, see Routes.

Topics

- Prerequisites
- Calculate a route
- Route planning with a route matrix
- Positions not located on a road
- Departure time
- Travel mode
- Managing your route calculator resources

Prerequisites

Before you begin calculating routes, follow the prerequisite steps:

Topics

- Create a route calculator resource
- Authenticating your requests

Create a route calculator resource

Before you can calculate a route, create a route calculator resource in your AWS account.

When you create a route calculator resource, you can choose from the data providers available:

1. **Esri** – For more information about Esri's coverage in your region of interest, see Esri details on street networks and traffic coverage.
2. **HERE Technologies** – For more information about HERE's coverage in your region of interest, see HERE car routing coverage and HERE truck routing coverage.
3. **Grab** – For more information about Grab's coverage, see Countries/regions and area covered.

Note

If your application is tracking or routing assets you use in your business, such as delivery vehicles or employees, you must not use Esri as your geolocation provider. See section 82 of the AWS service terms for more details.

You can do this using the Amazon Location Service console, the AWS CLI, or the Amazon Location APIs.

Console

To create a route calculator resource using the Amazon Location console

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. In the left navigation pane, choose **Route calculators**.
3. Choose **Create route calculator**.
4. Fill out the following boxes:
   - **Name** – Enter a name for the route calculator resource. For example, *ExampleCalculator*. Maximum 100 characters. Valid entries include alphanumeric characters, hyphens, periods, and underscores.
   - **Description** – Enter an optional description.
5. For **Data providers**, choose a data provider to use as a route calculator.
6. (Optional) Under **Tags**, enter a tag **Key** and **Value**. This adds a tag your new route calculator resource. For more information, see Tagging your resources.
7. Choose **Create route calculator**.

API

To create a route calculator resource using the Amazon Location APIs

Use the `CreateRouteCalculator` operation from the Amazon Location Places APIs.

The following example is an API request to create a route calculator resource called *ExampleCalculator* using the data provider *Esri*.

```
POST /routes/v0/calculators
```

```
Content-type: application/json

{
   "CalculatorName": "ExampleCalculator",
   "DataSource": "Esri",
   "Description": "string",
   "Tags": {
      "Tag1" : "Value1"
   }
}
```

AWS CLI

To create a route calculator resource using AWS CLI commands

Use the `create-route-calculator` command.

The following example creates a route calculator resource called *ExampleCalculator* using *Esri* as the data provider.

```
aws location \
  create-route-calculator \
  --calculator-name "ExampleCalculator" \
  --data-source "Esri" \
  --tags Tag1=Value1
```

Note

Billing depends on your usage. You may incur fees for the use of other AWS services. For more information, see Amazon Location Service pricing.

Authenticating your requests

Once you create a route calculator resource and you're ready to begin building location features into your application, choose how you would authenticate your requests:

- To explore ways you can access the services, see Accessing Amazon Location Service.
- If you have a website with anonymous users, you may want to use API Keys or Amazon Cognito.

Example

The following example shows using an API key for authorization, using AWS JavaScript SDK v3, and the Amazon Location JavaScript Authentication helper.

```
import { LocationClient, CalculateRouteCommand } from "@aws-sdk/client-location";
import { withAPIKey } from "@aws/amazon-location-utilities-auth-helper";

const apiKey = "v1.public.your-api-key-value"; // API key

// Create an authentication helper instance using an API key
const authHelper = await withAPIKey(apiKey);

const client = new LocationClient({
  region: "<region>", // region containing Cognito pool
  ...authHelper.getLocationClientConfig(), // Provides configuration required to make
 requests to Amazon Location
});

const input = {
  CalculatorName: "ExampleCalculator",
  DeparturePosition: [-123.4567, 45.6789],
  DestinationPosition: [-123.123, 45.234],
};

const command = new CalculateRouteCommand(input);

const response = await client.send(command);
```

Calculate a route

You can use Amazon Location Service to calculate routes between an origin and a destination, with up to 23 waypoints along the route, for different modes of transportation, avoidances, and traffic conditions.

> **Note**
>
> You must first create a route calculator resource and set up authentication for your requests to Amazon Location. For more information, see Prerequisites.

Start calculating routes

Submit a simple request by using the `CalculateRoute` operation. A simple request contains the following required fields:

- `DeparturePosition` – The starting position for which to calculate the route from. Defined as `[longitude, latitude]`
- `DestinationPosition` – The end position to which to calculate the route. Defined as `[longitude, latitude]`.

> **Note**
>
> If you specify a departure or destination position that's not located on a road, Amazon Location moves the position to the nearest road.

You can optionally specify waypoints, a departure time, and a travel mode in your request.

You can use the AWS CLI or the Amazon Location APIs.

API

The following example is a `CalculateRoute` request using the route calculator resource *`ExampleCalculator`*. The request specifies calculating a route from a departure position [*`-122.7565`*, *`49.0021`*] to a destination position [*`-122.3394`*, *`47.6159`*].

```
POST /routes/v0/calculators/ExampleCalculator/calculate/route
Content-type: application/json
{
   "DeparturePosition": [-122.7565,49.0021],
   "DestinationPosition": [-122.3394, 47.6159]
}
```

AWS CLI

The following example is a `calculate-route` command using the route calculator resource *`ExampleCalculator`*. The request specifies calculating a route from a departure position [*`-122.7565`*, *`49.0021`*] to a destination position [*`-122.3394`*, *`47.6159`*].

```
aws location \
```

```
    calculate-route \
        --calculator-name ExampleCalculator \
        --departure-position -122.7565 49.0021 \
        --destination-position -122.3394 47.6159
```

By default, the response returns `Distance` in kilometers. You can change the unit of measurement to miles using the following optional parameter:

- `DistanceUnit` – Specifies the unit system to use for the distance results.

Example

```
POST /routes/v0/calculators/ExampleCalculator/calculate/route
Content-type: application/json
{
   "DeparturePosition": [-122.7565,49.0021],
   "DestinationPosition": [-122.3394, 47.6159],
   "DistanceUnit": "Miles"
}
```

Setting waypoints

When calculating a route, you can specify up to 23 intermediate stopover points between the departure position and the destination position using waypoint positions.

- `WaypointPositions` – Specifies an ordered list of intermediate positions to include along a route between the departure position and destination position.

> **Note**
>
> If you specify a waypoint position that's not located on a road, Amazon Location moves the position to the nearest road.

Example

The following `CalculateRoute` request calculates a route with 2 waypoints:

- The departure position is [-122.7565, 49.0021], and the destination position is [-122.3394, 47.6159].
- For the request parameter `WaypointPositions`:
  - The first stop over position is [*-122.1884, 48.0936*].
  - The second stop over position is [*-122.3493, 47.6205*].
- To include the leg linestring geometry between these two waypoints, set the following optional parameter to *true*:
  - `IncludeLegGeometry` – Includes the geometry of each path between a pair of positions in the response.

```
POST /routes/v0/calculators/ExampleCalculator/calculate/route
Content-type: application/json
{
   "DeparturePosition": [-122.7565,49.0021],
   "DestinationPosition": [-122.3394, 47.6159],
   "WaypointPositions":[
        [-122.1884,48.0936],
        [-122.3493,47.6205]
    ],
   "IncludeLegGeometry": true
}
```

Example response

The following is an example request with the corresponding response when calling the `CalculateRoute` operation from the Amazon Location Routes API with the `IncludeLegGeometry` set to *true*, which includes the linestring geometry of each path between a pair of positions in the response.

Example request

```
POST /routes/v0/calculators/ExampleCalculator/calculate/route
Content-type: application/json
{
   "DeparturePosition": [-122.7565,49.0021],
   "DestinationPosition": [-122.3394, 47.6159],
   "IncludeLegGeometry": true
}
```

Example response


```
{
   "Legs": [
      {
         "Distance": 178.5,
         "DurationSeconds": 6480,
         "EndPosition": [-122.3394,47.6159],
         "Geometry": {
            "LineString": [
               [-122.7565,49.0021],
               [-122.3394,47.6159]
            ]
         },
         "StartPosition": [-122.7565,49.0021],
         "Steps": [
            {
               "Distance": 178.5,
               "DurationSeconds": 6480,
               "EndPosition": [-122.3394,47.6159],
               "GeometryOffset": 0,
               "StartPosition": [-122.7565,49.0021]
            }
         ]
      }
   ],
   "Summary": {
      "DataSource": "Esri",
      "Distance": 178.5,
      "DistanceUnit": "Kilometers",
      "DurationSeconds": 6480,
      "RouteBBox": [
        -122.7565,49.0021,
        -122.3394,47.6159
    ]
   }
}
```

Route planning with a route matrix

You can use Amazon Location Service to create inputs to your route planning and optimization software. You can create route results, including travel time and travel distance, for routes between a set of departure positions and a set of destination positions.

For example, given departure positions A and B, and destination positions X and Y, Amazon Location Service will return travel time and travel distance for routes from A to X, A to Y, B to X, and B to Y.

You can calculate the routes with different modes of transportation, avoidances, and traffic conditions. For example, you can specify that the vehicle is a truck that is 35 feet long, and the route calculated will use those restrictions to determine the travel time and travel distance.

The number of results returned (and routes calculated) is the number of departure positions multiplied by the number of destination positions. You are charged for each route calculated, not each request to the service, so a route matrix with 10 departures and 10 destinations will be billed as 100 routes.

Calculating a route matrix

You can calculate a matrix of routes between a set of departure positions and a set of destination positions. The route results will include travel time and travel distance.

Prerequisite

- You must first create a route calculator resource and set up authentication for your requests to Amazon Location. For more information, see Prerequisites.

Submit a request by using the `CalculateRouteMatrix` operation. A minimal request contains the following required fields:

- `DeparturePositions` – The set of starting positions for which to calculate the routes. Defined as an array of `[longitude, latitude]`
- `DestinationPositions` – The set of end positions for which to calculate the routes. Defined as an array of `[longitude, latitude]`.

> **Note**
>
> If you specify a departure or destination position that's not located on a road, Amazon Location moves the position to the nearest road.

You can optionally specify a departure time, and a travel mode in your request.

You can use the AWS CLI or the Amazon Location APIs.

API

The following example is a `CalculateRouteMatrix` request using the route calculator resource *`ExampleCalculator`*. The request specifies calculating the matrix of routes from departure positions [*`-122.7565`*, *`49.0021`*] and [*`-122.2014`*, *`47.6101`*] to destination positions [*`-122.3394`*, *`47.6159`*] and [*`-122.4813`*, *`48.7511`*].

```
POST /routes/v0/calculators/ExampleCalculator/calculate/route-matrix
Content-type: application/json
{
   "DeparturePositions": [
       [-122.7565,49.0021],
       [-122.2014,47.6101]
   ],
   "DestinationPositions": [
       [-122.3394, 47.6159],
       [-122.4813,48.7511]
   ]
}
```

AWS CLI

The following example is a `calculate-route-matrix` command using the route calculator resource *`ExampleCalculator`*. The request specifies calculating the matrix of routes from departure positions [*`-122.7565`*, *`49.0021`*] and [*`-122.2014`*, *`47.6101`*] to destination positions [*`-122.3394`*, *`47.6159`*] and [*`-122.4813`*, *`48.7511`*].

```
aws location \
    calculate-route-matrix \
        --calculator-name ExampleCalculator \
        --departure-positions "[[-122.7565,49.0021],[-122.2014,47.6101]]" \
```

```
    --destination-positions "[[-122.3394,47.6159],[-122.4813,48.7511]]"
```

By default, the response returns `Distance` in kilometers. You can change the unit of measurement to miles using the following optional parameter:

- `DistanceUnit` – Specifies the unit system to use for the distance results.

Example

```
POST /routes/v0/calculators/ExampleCalculator/calculate/route-matrix
Content-type: application/json
{
   "DeparturePositions": [
       [-122.7565,49.0021],
       [-122.2014,47.6101]
   ],
   "DestinationPositions": [
       [-122.3394, 47.6159],
       [-122.4813,48.7511]
   ],
   "DistanceUnit": "Miles"
}
```

Restrictions on departure and destination positions

When calculating a route matrix, there are restrictions on the departure and destination positions. These restrictions vary depending on the provider used by the `RouteCalculator` resource.

Limitation	Esri	Grab	HERE
Number of positions	Up to 10 departure positions and 10 destination positions.	Up to 350 departure positions and 350 destination positions.	Up to 350 departure positions and 350 destination positions. For longer routes, additional restricti ons apply. See the section.

Limitation	Esri	Grab	HERE
Distance between positions	Any pair of departure and destination positions must be within 400 km of each other (40km for walking routes).		All departure and destination positions must fall within a 180 km diameter circle. For longer routes, additional restricti ons apply. See the section.
Route length	Routes will not be completed if the total travel time for the route is over 400 minutes.		Routes that deviate more than 10 km outside a circle around the departure and destination points will not be calculated. For longer routes, additional restricti ons apply. See the section.
Regions	Calculating a route matrix is not supported in Korea.	Available in Southeast Asia. For a list of supported countries/regions and more information, see Countries/regions and area covered.	No additional restricti ons.

Longer route planning

Calculating a matrix of route results is useful for efficient route planning, but the calculation can take some time. All of the Amazon Location Service data providers put limitations on the number

of routes or the distance of the routes that can be calculated. For example, HERE allows creating routes between 350 departure and destination positions, but those positions must fall within a 180km circle. What if you want to plan with longer routes?

You can calculate a matrix of routes with unrestricted lengths for a smaller numbers of routes, using, a `RouteCalculator` with HERE as the data provider. This does not change the way that you call the `CalculateRouteMatrix` API, Amazon Location simply allows longer routes when you meet the requirements.

The requirements for longer length route calculations are:

- The `RouteCalculator` must use the HERE data provider.
- The number of departure positions must not be greater than 15.
- The total number of routes to calculate must not be greater than 100.
- Long distance routing is not allowed for truck routing with toll avoidances when the routes are greater than 1,000 km. This combination is slower to calculate, and can cause the call to time out. You can calculate these routes individually with the CalculateRoute operation.

If your call does not meet these requirements (for example, you are requesting 150 route calculations in a single call), then `CalculateRouteMatrix` will revert to only allowing the shorter route rules. You can then calculate the routes, as long as the positions are within a 180km circle.

When calculating longer routes, keep these points in mind:

- Longer routes can take longer to calculate, even longer than the maximum time for Amazon Location APIs. If you get frequent timeouts with specific routes, you can try a smaller number of routes in each call to `CalculateRouteMatrix`.
- If you add more destination or departure positions to your `CalculateRouteMatrix` request, the operation can switch into the more restricted mode, and you can get an error for a route that can be calculated without issue when there are fewer routes to create. In this case, reduce the number of destination or departure positions, and make multiple requests to get the full set of route calculations that you need.

Example response

The following is an example request with the corresponding response when calling the `CalculateRouteMatrix` operation from the Amazon Location Routes API.

Example request

```
POST /routes/v0/calculators/ExampleCalculator/calculate/route-matrix
Content-type: application/json
{
   "DeparturePositions": [
       [-122.7565,49.0021],
       [-122.2014,47.6101]
   ],
   "DestinationPositions": [
       [-122.3394, 47.6159],
       [-122.4813,48.7511]
   ]
}
```

Example response

```
{
    "RouteMatrix": [
        [
            {
                "Distance": 178.764,
                "DurationSeconds": 7565
            },
            {
                "Distance": 39.795,
                "DurationSeconds": 1955
            }
        ],
        [
            {
                "Distance": 15.31,
                "DurationSeconds": 1217
            },
            {
                "Distance": 142.506,
                "DurationSeconds": 6279
            }
        ]
    ],
    "Summary": {
        "DataSource": "Here",
        "RouteCount": 4,
```

```
        "ErrorCount": 0,
        "DistanceUnit": "Kilometers"
    }
}
```

Positions not located on a road

When using `CalculateRoute` or `CalculateRouteMatrix`, if you specify a departure, destination, or waypoint position that's not located on a road Amazon Location moves the position to a nearby road.

The following `CalculateRoute` request specifies a departure position and destination position that's not located on a road:

```
POST /routes/v0/calculators/ExampleCalculator/calculate/route
Content-type: application/json
{
   "DeparturePosition": [-123.128014, 49.298472],
   "DestinationPosition": [-123.134701, 49. 294315]
}
```

The resulting response returns a position that's snapped to a nearby road:

```
{
   "Legs": [
      {
         "StartPosition": [-123.12815, 49.29717],
         "EndPosition":   [-123.13375, 49.2926],
         "Distance": 4.223,
         "DurationSeconds": 697,
         "Steps": [
            {
               "StartPosition": [ -123.12815, 49.29717 ],
               "EndPosition":   [ -123.12806, 49.29707 ],
               "Distance": 0.013,
               "DurationSeconds": 8
            },
            {
               "StartPosition": [ -123.12806, 49.29707 ],
               "EndPosition":   [ -123.1288, 49.29659 ],
               "Distance": 0.082,
```

```
            "DurationSeconds": 36
        },
        {
            "StartPosition": [ -123.1288, 49.29659 ],
            "EndPosition":   [ -123.12021, 49.29853 ],
            "Distance": 0.742,
            "DurationSeconds": 128
        },
        {
            "StartPosition": [ -123.12021, 49.29853 ],
            "EndPosition":   [ -123.1201, 49.29959 ],
            "Distance": 0.131,
            "DurationSeconds": 26
        },
        {
            "StartPosition": [ -123.1201, 49.29959 ],
            "EndPosition":   [ -123.13562, 49.30681 ],
            "Distance": 1.47,
            "DurationSeconds": 238
        },
        {
            "StartPosition": [ -123.13562, 49.30681 ],
            "EndPosition":   [ -123.13693, 49.30615 ],
            "Distance": 0.121,
            "DurationSeconds": 28
        },
        {
            "StartPosition": [ -123.13693, 49.30615 ],
            "EndPosition":   [ -123.13598, 49.29755 ],
            "Distance": 0.97,
            "DurationSeconds": 156
        },
        {
            "StartPosition": [ -123.13598, 49.29755 ],
            "EndPosition":   [ -123.13688, 49.29717 ],
            "Distance": 0.085,
            "DurationSeconds": 15
        },
        {
            "StartPosition": [ -123.13688, 49.29717 ],
            "EndPosition":   [ -123.13375, 49.2926 ],
            "Distance": 0.609,
            "DurationSeconds": 62
        }
```

```
         ]
      }
   ],
   "Summary": {
      "RouteBBox": [ -123.13693, 49.2926, -123.1201, 49.30681 ],
      "DataSource": "Here",
      "Distance": 4.223,
      "DurationSeconds": 697,
      "DistanceUnit": "Kilometers"
   }
}
```

Departure time

By default, when you call `CalculateRoute` or `CalculateRouteMatrix`, if you don't provide a departure time in the request, the calculated routes reflects optimal traffic conditions.

You can set a specific a departure time to use live and predictive traffic conditions from your chosen data provider, by using one of the following options:

- `DepartNow` – When set to *`true`*, it uses live traffic conditions to calculate the fastest travel path.
- `DepartureTime` – When provided, it uses predictive and known traffic conditions for the requested time. Defined in the following format: `YYYY-MM-DDThh:mm:ss.sssZ`.

Example

The following `CalculateRoute` request sets the departure time to July 2, 2024, at 12:15:20 UTC.

```
POST /routes/v0/calculators/ExampleCalculator/calculate/route
Content-type: application/json
{
   "DeparturePosition": [-122.7565,49.0021],
   "DestinationPosition": [-122.3394, 47.6159],
   "WaypointPositions":[
        [-122.1884,48.0936],
        [-122.3493,47.6205]
    ]
    "IncludeLegGeometry": true,
    "DepartureTime": 2024-07-02T12:15:20.000Z,
}
```

Travel mode

You can set a travel mode when using `CalculateRoute` or `CalculateRouteMatrix`. The mode of travel affects speed of travel and road compatibility. While the default mode of travel is by car, you can specify which mode of travel you're using while traveling along a route with the following optional parameter:

- `TravelMode` – Specifies the mode of transport when calculating a route, such as: *`Bicycle`*, *`Car`*, *`Motorcycle`*, *`Truck`*, or *`Walking`*.

Limitations

- If you specify `Walking` for the travel mode and your data provider is Esri, the start and destination must be within 40km.
- `Bicycle` or `Motorcycle` are available only when using Grab as the data provider.
- Grab provides only `Bicycle` and `Walking` routes in certain cities. For more information, see Countries/regions and area covered.
- `Truck` is not available when using Grab as the data provider.

Additional preferences

If you specify a `TravelMode` of *`Car`*, you can specify additional route preferences with the following optional parameter:

- `CarModeOptions` – Specifies route preferences when traveling in a car, such as *`AvoidFerries`* or *`AvoidTolls`*.

If you specify a `TravelMode` of *`Truck`*, you can specify additional route preferences with the following optional parameter:

- `TruckModeOptions` – Specifies route preferences when traveling in a truck, such as *`AvoidFerries`* or *`AvoidTolls`*, in addition to specifying routes that can accommodate the *`TruckDimensions`* and *`TruckWeight`*.

Example

The following `CalculateRoute` request specifies *`Truck`* as the mode of travel. Additional route restrictions include: avoiding routes that use ferries and avoiding roads that can't accommodate the truck dimensions and weight.

```
{
   "DeparturePosition": [-122.7565,49.0021],
   "DestinationPosition": [-122.3394, 47.6159],
   "DepartNow": true,
   "TravelMode": "Truck",
   "TruckModeOptions": {
      "AvoidFerries": true,
      "AvoidTolls": false,
      "Dimensions": {
         "Height": 4.5,
         "Length": 15.5,
         "Unit": "Meters",
         "Width": 4.5
      },
      "Weight": {
         "Total": 7500,
         "Unit": "Pounds"
      }
   }
}
```

Managing your route calculator resources

You can manage your route calculator resources using the Amazon Location console, the AWS CLI, or the Amazon Location APIs.

List your route calculator resources

You can view your route calculator list using the Amazon Location console, the AWS CLI, or the Amazon Location APIs:

Console

To view a list of route calculators using the Amazon Location console

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.

2. Choose **Route calculators** from the left navigation pane.
3. View the route calculator details under **My route calculators**.

API

Use the `ListRouteCalculators` operation from the Amazon Location Routes APIs.

The following example is an API request to get a list of route calculators in the AWS account.

```
POST /routes/v0/list-calculators
```

The following is an example response for `ListRouteCalculators`:

```
{
       "Entries": [
          {
             "CalculatorName": "ExampleCalculator",
             "CreateTime": 2020-09-30T22:59:34.142Z,
             "DataSource": "Esri",
             "Description": "string",
             "UpdateTime": 2020-09-30T23:59:34.142Z
          }
       ],
       "NextToken": "1234-5678-9012"
    }
```

CLI

Use the `list-route-calculators` command.

The following example is an AWS CLI to get a list of route calculators in the AWS account.

```
aws location list-route-calculators
```

Get route calculator details

You can get details about any route calculator resource in your AWS account using the Amazon Location console, the AWS CLI, or the Amazon Location APIs:

Console

To view the details of a route calculator using the Amazon Location console

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. Choose **Route calculators** from the left navigation pane.
3. Under **My route calculators**, select the name link of the target route calculator.

API

Use the `DescribeRouteCalculator` operation from the Amazon Location Routes APIs.

The following example is an API request to get the route calculator details for *ExampleCalculator*.

```
GET /routes/v0/calculators/ExampleCalculator
```

The following is an example response for `DescribeRouteCalculator`:

```
{
       "CalculatorArn": "arn:aws:geo:us-west-2:123456789012:route-
calculator/ExampleCalculator",
       "CalculatorName": "ExampleCalculator",
       "CreateTime": 2020-09-30T22:59:34.142Z,
       "DataSource": "Esri",
       "Description": "string",
       "Tags": {
          "Tag1" : "Value1"
       },
       "UpdateTime": 2020-09-30T23:59:34.142Z
    }
```

CLI

Use the `describe-route-calculator` command.

The following example is an AWS CLI to get the route calculator details for *ExampleCalculator*.

```
aws location describe-route-calculator \
```

```
        --calculator-name "ExampleCalculator"
```

Delete a route calculator

You can delete a route calculator from your AWS account using the Amazon Location console, the AWS CLI, or the Amazon Location APIs:

Console

To delete a route calculator using the Amazon Location console

> **Warning**
>
> This operation deletes the resource permanently.

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. Choose **Route calculators** from the left navigation pane.
3. Under **My route calculators**, select the target route calculator.
4. Choose **Delete route calculator**.

API

Use the `DeleteRouteCalculator` operation from the Amazon Location Routes APIs.

The following example is an API request to delete the geofence collection *ExampleCalculator*.

```
DELETE /routes/v0/calculators/ExampleCalculator
```

The following is an example response for `DeleteRouteCalculator`:

```
HTTP/1.1 200
```

CLI

Use the `delete-route-calculator` command.

The following example is an AWS CLI command to delete the geofence collection *ExampleCalculator*.

```
aws location delete-route-calculator \
        --calculator-name "ExampleCalculator"
```

Geofencing an area of interest using Amazon Location

A geofencing application evaluates a tracked device's position relative to previously registered areas of interest. This enables actions to be taken based on position updates. For example, you can initiate an event that prompts a notification when a customer who ordered coffee on their mobile app is near a store.

> **Note**
>
> For an overview of geofencing and tracker concepts, see Geofences and Trackers.

This section of the guide provides step-by-step instructions for creating a geofencing application using Amazon Location Service.

Overview of steps

1. Add geofences around areas of interest and store them in a geofence collection resource.
2. Start tracking your target devices and store the device location history in a tracker resource.
3. Link your tracker resource to your geofence collection resource so that device location updates are automatically evaluated against all your geofences.
4. You can evaluate device positions directly against your geofence collection resources if you don't want to use Amazon Location Trackers to keep your devices' location history.

After you implement your geofencing solution, your geofence collection resource emits the following events:

- ENTER — A tracked device enters a geofence within a geofence collection.
- EXIT — A tracked device exits a geofence within a geofence collection.

You can use Amazon EventBridge to react to events by routing them elsewhere.

As an alternative to sending updates via the Amazon Location Service APIs from each device, you can use MQTT to send device updates.

The following topics describe these steps and alternatives in detail.

Topics

- Add geofences
- Start tracking
- Link a tracker to a geofence collection
- Evaluate device positions against geofences
- Verify device positions
- Reacting to Amazon Location Service events with Amazon EventBridge
- Tracking using AWS IoT and MQTT with Amazon Location Service
- Managing your geofence collection resources
- Managing your tracker resources
- Sample Geofencing and Tracking mobile application

Add geofences

Geofences contain points and vertices that form a closed boundary, which defines an area of interest. Geofence collections store and manage one or multiple geofences.

Amazon Location geofence collections stores geofences defined by using a standard geospatial data format called GeoJSON (RFC 7946). You can use tools, such as geojson.io, at no charge to draw your geofences graphically and save the output GeoJSON file.

> **Note**
>
> Amazon Location doesn't support polygons with holes, multipolygons, clockwise polygons, and geofences that cross the antimeridian.

Create a geofence collection

Create a geofence collection to store and manage geofences by using the Amazon Location console, the AWS CLI, or the Amazon Location APIs.

Console

To create a geofence collection using the Amazon Location console

1. Open the Amazon Location Service console at https://console.aws.amazon.com/location/.
2. In the left navigation pane, choose **Geofence collections**.
3. Choose **Create geofence collection**.
4. Fill out the following boxes:
   - **Name** – Enter a unique name. For example, *ExampleGeofenceCollection*. Maximum 100 characters. Valid entries include alphanumeric characters, hyphens, periods, and underscores.
   - **Description** – Enter an optional description to differentiate your resources.
5. Under **EventBridge rule with CloudWatch as a target**, you can create an optional EventBridge rule to get started reacting to geofence events. This enables Amazon Location to publish events to Amazon CloudWatch Logs.
6. (Optional) Under **Tags**, enter a tag **Key** and **Value**. This adds a tag your new geofence collection. For more information, see Tagging your Amazon Location Service resources.
7. (Optional) Under **Customer managed key encryption**, you can choose to **Add a customer managed key**. This adds a symmetric customer managed key that you create, own, and manage over the default AWS owned encryption. For more information, see Encrypting data at rest.
8. Choose **Create geofence collection**.

API

To create a geofence collection using the Amazon Location APIs

Use the `CreateGeofenceCollection` operation from the Amazon Location Geofences APIs.

The following example uses an API request to create a geofence collection called *ExampleGeofenceCollection*. The geofence collection is associated with a customer managed AWS KMS key to encrypt customer data.

```
POST /geofencing/v0/collections
Content-type: application/json

{
   "CollectionName": "ExampleGeofenceCollection",
   "Description": "Geofence collection 1 for shopping center",
   "KmsKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
   "Tags": {
      "Tag1" : "Value1"
   }
}
```

AWS CLI

To create a geofence collection using AWS CLI commands

Use the `create-geofence-collection` command.

The following example uses an AWS CLI to create a geofence collection called *ExampleGeofenceCollection*. The geofence collection is associated with a customer managed AWS KMS key to encrypt customer data.

```
aws location \
  create-geofence-collection \
  --collection-name "ExampleGeofenceCollection" \
  --description "Shopping center geofence collection" \
  --kms-key-id "1234abcd-12ab-34cd-56ef-1234567890ab" \
  --tags Tag1=Value1
```

Note

Billing depends on your usage. You may incur fees for the use of other AWS services. For more information, see Amazon Location Service pricing.

Draw geofences

Now that you've created your geofence collection, you can define your geofences. Geofences are defined either as a polygon or as a circle. To draw a polygon geofence you can use a GeoJSON editing tool, such as geojson.io.

To create a geofence as a circle, you must define the center point of the circle, and the radius. For example, if you want to create a geofence to be notified whenever a device comes within 50 meters of a specific location, you would use the latitude and longitude of that location and specify the radius as 50 meters.

Using the Amazon Location Service APIs, you can also add metadata to your geofence, in the form of key-value pairs. These can be useful for storing information about the geofence, such as its type, or other information that is specific to your application. You can use this metadata when Reacting to Amazon Location Service events with Amazon EventBridge.

Adding polygon geofences

This section describes creating polygon geofences

Draw geofences using a GeoJSON tool

Now that you've created your geofence collection, you can define your geofences by using a GeoJSON editing tool, such as geojson.io.

To create a GeoJSON file

1. Open a GeoJSON editing tool. For example, geojson.io.
2. Choose the **Draw a polygon** icon and draw your area of interest.
3. Choose **Save**, then choose **GeoJSON** from the dropdown menu.

Put GeoJSON geofences in a geofence collection

You can use the resulting GeoJSON file to upload your geofences using the Amazon Location Service console, the AWS CLI, or the Amazon Location APIs:

Console

To add a geofence to a geofence collection using the Amazon Location Service console

1. Open the Amazon Location Service console at https://console.aws.amazon.com/location/.
2. In the left navigation pane, choose **Geofence collections**.
3. From the **Geofence collections** list, select the name link for the target geofence collection.
4. Under **Geofences**, choose **Create geofences**.
5. In the **Add geofences** window, drag, and drop your GeoJSON into the window.

6. Choose **Add geofences**.

API

To add geofences using the Amazon Location APIs

Use the PutGeofence operation from the Amazon Location Geofences APIs.

The following example uses an API request to add a geofence given the ID *GEOFENCE-EXAMPLE1* to a geofence collection called *ExampleGeofenceCollection*. It also specifies a single geofence metadata property with the key `Type` and value `loadingArea`.

```
PUT /geofencing/v0/collections/ExampleGeofenceCollection/geofence/GEOFENCE-EXAMPLE1
    Content-type: application/json

    {
       "GeofenceProperties": {
          "Type" : "loadingArea"
       },
       "Geometry": {
          "Polygon": [
            [
                [-5.716667, -15.933333],
                [-14.416667, -7.933333],
                [-12.316667, -37.066667],
                [-5.716667, -15.933333]
            ]
          ]
       }
    }
```

Alternatively, you can add more than one geofence using the BatchPutGeofence operation.

```
POST /geofencing/v0/collections/ExampleGeofenceCollection/put-geofences
    Content-type: application/json

    {
       "Entries": [
          {
             "GeofenceProperties": {
                "Type" : "loadingArea"
             },
```

```
            "GeofenceId": "GEOFENCE-EXAMPLE1",
            "Geometry": {
               "Polygon": [
                  [
                     [-5.716667, -15.933333],
                     [-14.416667, -7.933333],
                     [-12.316667, -37.066667],
                     [-5.716667, -15.933333]
                  ]
               ]
            }
         }
      ]
   }
```

AWS CLI

To add a geofence to a geofence collection using AWS CLI commands

Use the put-geofence command.

The following example uses an AWS CLI to add a geofence to a geofence collection called *ExampleGeofenceCollection*.

```
$ aws location \
        put-geofence \
            --collection-name ExampleGeofenceCollection \
            --geofence-id ExampleGeofenceTriangle \
            --geofence-properties '{"Type": "loadingArea"}' \
            --geometry 'Polygon=[[[-5.716667, -15.933333],[-14.416667, -7.933333],
[-12.316667, -37.066667],[-5.716667, -15.933333]]]'
    {
        "CreateTime": "2020-11-11T00:16:14.487000+00:00",
        "GeofenceId": "ExampleGeofenceTriangle",
        "UpdateTime": "2020-11-11T00:19:59.894000+00:00"
    }
```

Adding circular geofences

This section describes creating circular geofences. You must know the latitude and longitude of the point that you want to be the center of the circle, and the radius in meters of the circle. You can create circular geofences with the Amazon Location APIs or the AWS CLI.

API

To add circular geofences using the Amazon Location APIs

Use the PutGeofence operation from the Amazon Location Geofences APIs.

The following example uses an API request to add a geofence given the ID *GEOFENCE-EXAMPLE2* to a geofence collection called *ExampleGeofenceCollection*:

```
PUT /geofencing/v0/collections/ExampleGeofenceCollection/geofence/GEOFENCE-EXAMPLE2
    Content-type: application/json

    {
       "Geometry": {
          "Circle": {
            "Center": [-5.716667, -15.933333],
            "Radius": 50
          }
       }
    }
```

AWS CLI

To add a circular geofence to a geofence collection using AWS CLI commands

Use the put-geofence command.

The following example uses an AWS CLI to add a geofence to a geofence collection called *ExampleGeofenceCollection*.

```
$ aws location \
        put-geofence \
            --collection-name ExampleGeofenceCollection \
            --geofence-id ExampleGeofenceCircle \
            --geometry 'Circle={Center=[-5.716667, -15.933333], Radius=50}'
```

Note

You can also put JSON for complex geometry into its own file as in the following example.

```
$ aws location \
```

```
        put-geofence \
            --collection-name ExampleGeofenceCollection \
            --geofence-id ExampleGeofenceCircle \
            --geometry file:circle.json
```

In the example, the circle.json file includes JSON for the circle geometry.

```
{
    "Circle": {
        "Center": [-74.006975, 40.717127],
        "Radius": 287.7897969218057
    }
}
```

Start tracking

This section guides you through building a tracking application that captures device locations.

Create a tracker

Create a tracker resource to store and process position updates from your devices. You can use the Amazon Location Service console, the AWS CLI, or the Amazon Location APIs.

Each position update stored in your tracker resources can include a measure of position accuracy, and up to three fields of metadata about the position or device that you want to store. The metadata is stored as key-value pairs, and can store information such as speed, direction, tire pressure, or engine temperature.

Trackers filter position updates as they are received. This reduces visual noise in your device paths (called *jitter*), and reduces the number of false geofence entry and exit events. This also helps manage costs by reducing the number of geofence evaluations initiated.

Trackers offer three position filtering options to help manage costs and reduce jitter in your location updates.

- **Accuracy-based** – *Use with any device that provides an accuracy measurement. Most mobile devices provide this information.* The accuracy of each position measurement is affected by many environmental factors, including GPS satellite reception, landscape, and the proximity of Wi-Fi and Bluetooth devices. Most devices, including most mobile devices, can provide an estimate of

the accuracy of the measurement along with the measurement. With `AccuracyBased` filtering, Amazon Location ignores location updates if the device moved less than the measured accuracy. For example, if two consecutive updates from a device have an accuracy range of 5 m and 10 m, Amazon Location ignores the second update if the device has moved less than 15 m. Amazon Location neither evaluates ignored updates against geofences, nor stores them.

When accuracy is not provided, it is treated as zero, and the measurement is considered perfectly accurate.

> **Note**
>
> You can also use accuracy-based filtering to remove all filtering. If you select accuracy-based filtering, but override all accuracy data to zero, or omit the accuracy entirely, then Amazon Location will not filter out any updates.

- **Distance-based** – *Use when your devices do not provide an accuracy measurement, but you still want to take advantage of filtering to reduce jitter and manage costs.* `DistanceBased` filtering ignores location updates in which devices have moved less than 30 m (98.4 ft). When you use `DistanceBased` position filtering, Amazon Location neither evaluates these ignored updates against geofences nor stores the updates.

  The accuracy of most mobile devices, including the average accuracy of iOS and Android devices, is within 15 m. In most applications, `DistanceBased` filtering can reduce the effect of location inaccuracies when displaying device trajectory on a map, and the bouncing effect of multiple consecutive entry and exit events when devices are near the border of a geofence. It can also help reduce the cost of your application, by making fewer calls to evaluate against linked geofences or retrieve device positions.

- **Time-based** – (default) *Use when your devices send position updates very frequently (more than once every 30 seconds), and you want to achieve near real-time geofence evaluations without storing every update.* In `TimeBased` filtering, every location update is evaluated against linked geofence collections, but not every location update is stored. If your update frequency is more often than 30 seconds, only one update per 30 seconds is stored for each unique device ID.

> **Note**
>
> Be mindful of the costs of your tracking application when deciding your filtering method and the frequency of position updates. You are billed for every location update and once

for evaluating the position update against each linked geofence collection. For example, when using time-based filtering, if your tracker is linked to two geofence collections, every position update will count as one location update request and two geofence collection evaluations. If you are reporting position updates every 5 seconds for your devices and using time-based filtering, you will be billed for 720 location updates and 1,440 geofence evaluations per hour for each device.
Your bill is not affected by the number of geofences in each collection. Since each geofence collection may contain up to 50,000 geofences, you may want to combine your geofences into fewer collections, where possible, to reduce your cost of geofence evaluations.

By default, you will get EventBridge events each time a tracked device enters or exits a linked geofence. For more information, see Link a tracker to a geofence collection.

You can enable events for all filtered position updates for a tracker resource. For more information, see Enable update events for a tracker.

Note

If you wish to encrypt your data using your own AWS KMS customer managed key, then the Bounding Polygon Queries feature will be disabled by default. This is because by using this Bounding Polygon Queries feature, a representation of your device positions will not be encrypted using the your AWS KMS managed key. However, the exact device position is still encrypted using your managed key.
You can choose to opt-in to the Bounding Polygon Queries feature by setting the `KmsKeyEnableGeospatialQueries` parameter to true when creating or updating a Tracker.

Console

To create a tracker using the Amazon Location console

1. Open the Amazon Location Service console at https://console.aws.amazon.com/location/.
2. In the left navigation pane, choose **Trackers**.
3. Choose **Create tracker**.
4. Fill the following fields:

- **Name** – Enter a unique name. For example, *ExampleTracker*. Maximum 100 characters. Valid entries include alphanumeric characters, hyphens, periods, and underscores.
- **Description** – Enter an optional description.

5. Under **Position filtering**, choose the option that best fits how you intend to use your tracker resource. If you do not set **Position filtering**, the default setting is `TimeBased`. For more information, see Trackers in this guide, and `PositionFiltering` in the Amazon Location Service Trackers API Reference.
6. (Optional) Under **Tags**, enter a tag **Key** and **Value**. This adds a tag your new geofence collection. For more information, see Tagging your resources.
7. (Optional) Under **Customer managed key encryption**, you can choose to **Add a customer managed key**. This adds a symmetric customer managed key that you create, own, and manage over the default AWS owned encryption. For more information, see Encrypting data at rest.
8. (Optional) Under **KmsKeyEnableGeospatialQueries**, you can choose to enable **Geospatial Queries**. This allows you use the Bounding Polygon Queries feature, while encrypting your data using a customer AWS KMS managed key.

   > **Note**
   >
   > When you use the Bounding Polygon Queries feature a representation of your device positions is not be encrypted using the your AWS KMS managed key. However, the exact device position is still encrypted using your managed key.

9. (Optional) Under **EventBridge configuration**, you can choose to enable EventBridge events for filtered position updates. This will send an event each time a position update for a device in this tracker meets the position filtering evaluation.
10. Choose **Create tracker**.

API

To create a tracker by using the Amazon Location APIs

Use the `CreateTracker` operation from the Amazon Location Trackers APIs.

The following example uses an API request to create a tracker called *ExampleTracker*. The tracker resource is associated with a customer managed AWS KMS key to encrypt customer data, and does not enable position updates in EventBridge.

```
POST /tracking/v0/trackers
Content-type: application/json

{

   "TrackerName": "ExampleTracker",
   "Description": "string",
   "KmsKeyEnableGeospatialQueries": false,
   "EventBridgeEnabled": false,
   "KmsKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
   "PositionFiltering": "AccuracyBased",
   "Tags": {
      "string" : "string"
   }
}
```

Create a tracker with `KmsKeyEnableGeospatialQueries` enabled

The following example has the parameter `KmsKeyEnableGeospatialQueries` set to true. This allows you use the Bounding Polygon Queries feature, while encrypting your data using a customer AWS KMS managed key.

For information on using the Bounding Polygon Queries feature, see ???

> **Note**
>
> When you use the Bounding Polygon Queries feature a representation of your device positions is not be encrypted using the your AWS KMS managed key. However, the exact device position is still encrypted using your managed key.

```
POST /tracking/v0/trackers
Content-type: application/json

{

   "TrackerName": "ExampleTracker",
```

```
   "Description": "string",
   "KmsKeyEnableGeospatialQueries": true,
   "EventBridgeEnabled": false,
   "KmsKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
   "PositionFiltering": "AccuracyBased",
   "Tags": {
      "string" : "string"
   }
}
```

AWS CLI

To create a tracker using AWS CLI commands

Use the `create-tracker` command.

The following example uses the AWS CLI to create a tracker called *ExampleTracker*. The tracker resource is associated with a customer managed AWS KMS key to encrypt customer data, and does not enable position updates in EventBridge.

```
aws location \
  create-tracker \
  --tracker-name "ExampleTracker" \
  --position-filtering "AccuracyBased" \
  --event-bridge-enabled false \
  --kms-key-enable-geospatial-queries false \
  --kms-key-id "1234abcd-12ab-34cd-56ef-1234567890ab"
```

Create a tracker with `KmsKeyEnableGeospatialQueries` enabled

The following example has the parameter `KmsKeyEnableGeospatialQueries` set to true. This allows you use the Bounding Polygon Queries feature, while encrypting your data using a customer AWS KMS managed key.

For information on using the Bounding Polygon Queries feature, see ???

> **Note**
>
> When you use the Bounding Polygon Queries feature a representation of your device positions is not be encrypted using the your AWS KMS managed key. However, the exact device position is still encrypted using your managed key.

```
aws location \
  create-tracker \
  --tracker-name "ExampleTracker" \
  --position-filtering "AccuracyBased" \
  --event-bridge-enabled false \
  --kms-key-enable-geospatial-queries true \
  --kms-key-id "1234abcd-12ab-34cd-56ef-1234567890ab"
```

Note

Billing depends on your usage. You may incur fees for the use of other AWS services. For more information, see Amazon Location Service pricing.

You can edit the **Description**, **Position filtering**, and **EventBridge configuration** after the tracker is created by choosing **Edit tracker**.

Authenticating your requests

Once you create a tracker resource and you're ready to begin evaluating device positions against geofences, choose how you would authenticate your requests:

- To explore ways you can access the services, see Accessing Amazon Location Service.
- If you want to publish device positions with unauthenticated requests,you may want to use Amazon Cognito.

  Example

  The following example shows using an Amazon Cognito identity pool for authorization, using AWS JavaScript SDK v3, and the Amazon Location JavaScript Authentication helper.

```
import { LocationClient, BatchUpdateDevicePositionCommand } from "@aws-sdk/client-
location";
import { withIdentityPoolId } from "@aws/amazon-location-utilities-auth-helper";

// Unauthenticated identity pool you created
const identityPoolId = "us-east-1:1234abcd-5678-9012-abcd-sample-id";

// Create an authentication helper instance using credentials from Cognito
const authHelper = await withIdentityPoolId(identityPoolId);
```

```
const client = new LocationClient({
  region: "us-east-1", // The region containing both the identity pool and tracker
 resource
  ...authHelper.getLocationClientConfig(), // Provides configuration required to make
 requests to Amazon Location
});

const input = {
  TrackerName: "ExampleTracker",
  Updates: [
    {
      DeviceId: "ExampleDevice-1",
      Position: [-123.4567, 45.6789],
      SampleTime: new Date("2020-10-02T19:09:07.327Z"),
    },
    {
      DeviceId: "ExampleDevice-2",
      Position: [-123.123, 45.123],
      SampleTime: new Date("2020-10-02T19:10:32Z"),
    },
  ],
};

const command = new BatchUpdateDevicePositionCommand(input);

// Send device position updates
const response = await client.send(command);
```

Update your tracker with a device position

To track your devices, you can post device position updates to your tracker. You can later retrieve these device positions or the device position history from your tracker resource.

Each position update must include the device ID, a timestamp , and a position. You may optionally include other metadata, including accuracy and up to 3 key-value pairs for your own use.

If your tracker is linked to one or more geofence collections, updates will be evaluated against those geofences (following the filtering rules that you specified for the tracker). If a device breaches a geofenced area (by moving from inside the area to outside, or vice versa), you will receive events in EventBridge. These `ENTER` or `EXIT` events include the position update details, including the device ID, the timestamp, and any associated metadata.

Note

For more information about position filtering, see Create a tracker.
For more information about geofence events, see Reacting to Amazon Location Service events with Amazon EventBridge.

Use either of these methods to send device updates:

- Send MQTT updates to an AWS IoT Core resource and link it to your tracker resource.
- Send location updates using the Amazon Location Trackers API, by using the AWS CLI, or the Amazon Location APIs. You can use the AWS SDKs to call the APIs from your iOS or Android application.

API

To send a position update using the Amazon Location APIs

Use the `BatchUpdateDevicePosition` operation from the Amazon Location Trackers APIs.

The following example uses an API request to post a device position update for *`ExampleDevice`* to a tracker *`ExampleTracker`*.

```
POST /tracking/v0/trackers/ExampleTracker/positions
Content-type: application/json
{
 "Updates": [
    {
    "DeviceId": "1",
    "Position": [
   -123.12245146162303, 49.27521118043802
    ],
    "SampleTime": "2022-10-24T19:09:07.327Z",
     "PositionProperties": {
            "name" : "device1"
        },
        "Accuracy": {
           "Horizontal": 10
        }
    },
```

```
    {
    "DeviceId": "2",
    "Position": [
   -123.1230104928471, 49.27752402723152
    ],
    "SampleTime": "2022-10-02T19:09:07.327Z"
    },
    {
    "DeviceId": "3",
    "Position": [
    -123.12325592118916, 49.27340530543111
    ],
    "SampleTime": "2022-10-02T19:09:07.327Z"
    },
    {
    "DeviceId": "4",
    "Position": [
    -123.11958813096311, 49.27774641063121
    ],
    "SampleTime": "2022-10-02T19:09:07.327Z"
    },
    {
    "DeviceId": "5",
    "Position": [
    -123.1277418058896, 49.2765989015285
    ],
    "SampleTime": "2022-10-02T19:09:07.327Z"
    },
    {
    "DeviceId": "6",
    "Position": [
   -123.11964267059481, 49.274188155916534
    ],
    "SampleTime": "2022-10-02T19:09:07.327Z"
    }
    ]
}
```

AWS CLI

To send a position update using AWS CLI commands

Use the batch-update-device-position command.

The following example uses an AWS CLI to post a device position update for *ExampleDevice-1* and *ExampleDevice-2* to a tracker *ExampleTracker*.

```
aws location batch-update-device-position \
--tracker-name ExampleTracker \
--updates '[{"DeviceId":"ExampleDevice-1","Position":
[-123.123,47.123],"SampleTime":"2021-11-30T21:47:25.149Z"},
{"DeviceId":"ExampleDevice-2","Position":
[-123.123,47.123],"SampleTime":"2021-11-30T21:47:25.149Z","Accuracy":
{"Horizontal":10.30},"PositionProperties":{"field1":"value1","field2":"value2"}}]'
```

Get a device's location history from a tracker

Your Amazon Location tracker resource maintains the location history of all your tracked devices for a period of 30 days. You can retrieve device location history, including all associated metadata, from your tracker resource. The following examples use the AWS CLI, or the Amazon Location APIs.

API

To get the device location history from a tracker using the Amazon Location APIs

Use the GetDevicePositionHistory operation from the Amazon Location Trackers APIs.

The following example uses an API URI request to get the device location history of *ExampleDevice* from a tracker called *ExampleTracker* starting from `19:05:07` (inclusive) and ends at `19:20:07` (exclusive) on `2020–10–02`.

```
POST /tracking/v0/trackers/ExampleTracker/devices/ExampleDevice/list-positions
Content-type: application/json
{
  "StartTimeInclusive": "2020-10-02T19:05:07.327Z",
  "EndTimeExclusive": "2020-10-02T19:20:07.327Z"
}
```

AWS CLI

To get the device location history from a tracker using AWS CLI commands

Use the get-device-position-history command.

The following example uses an AWS CLI to get the device location history of *ExampleDevice* from a tracker called *ExampleTracker* starting from 19:05:07 (inclusive) and ends at 19:20:07 (exclusive) on 2020–10–02.

```
aws location \
    get-device-position-history \
        --device-id "ExampleDevice" \
        --start-time-inclusive "2020-10-02T19:05:07.327Z" \
        --end-time-exclusive "2020-10-02T19:20:07.327Z" \
        --tracker-name "ExampleTracker"
```

List your device positions

You can view a list device positions for a tracker using the AWS CLI, or the Amazon Location APIs, with the ListDevicePositions API. When you call the ListDevicePositions API, a list of the latest positions for all devices associated with a given tracker is returned. By default this API returns 100 of the latest device positions per page of results for a given tracker. To only return devices within a specific region use the `FilterGeometry` parameter to create a Bounding Polygon Query. This way when you call ListDevicePositions, only devices inside the polygon will be returned.

> **Note**
>
> If you wish to encrypt your data using your own AWS KMS customer managed key, then the Bounding Polygon Queries feature will be disabled by default. This is because by using this feature, a representation of your device positions will not be encrypted using the your AWS KMS managed key. The exact device position, however; is still encrypted using your managed key.
> You can choose to opt-in to the Bounding Polygon Queries feature. This is done by setting the `KmsKeyEnableGeospatialQueries` parameter to true when creating or updating a Tracker.

API

Use the `ListDevicePositions` operation from the Amazon Location Trackers APIs.

The following example is an API request to get a list of device positions in polygonal area, using the optional parameter `FilterGeometry`. The example returns 3 device locations present in the area defined by the `Polygon` array.

```
POST /tracking/v0/trackers/TrackerName/list-positions HTTP/1.1
Content-type: application/json

{
   "FilterGeometry": {
        "Polygon": [
          [
            [
              -123.12003339442259,
              49.27425121147397
            ],
            [
              -123.1176984148229,
              49.277063620879744
            ],
            [
              -123.12389509145294,
              49.277954183760926
            ],
            [
              -123.12755921328647,
              49.27554025235713
            ],
            [
              -123.12330236586217,
              49.27211836076236
            ],
            [
              -123.12003339442259,
              49.27425121147397
            ]
          ]
        ]
    },
   "MaxResults": 3,
   "NextToken": "1234-5678-9012"
}
```

The following is an example response for [ListDevicePositions](#):

```
{
    "Entries": [
        {
```

```
            "DeviceId": "1",
            "SampleTime": "2022-10-24T19:09:07.327Z",
            "Position": [
                -123.12245146162303,
                49.27521118043802
            ],
            "Accuracy": {
                "Horizontal": 10
            },
            "PositionProperties": {
                "name": "device1"
            }
        },
        {
            "DeviceId": "3",
            "SampleTime": "2022-10-02T19:09:07.327Z",
            "Position": [
                -123.12325592118916,
                49.27340530543111
            ]
        },
        {
            "DeviceId": "2",
            "SampleTime": "2022-10-02T19:09:07.327Z",
            "Position": [
                -123.1230104928471,
                49.27752402723152
            ]
        }
    ],
   "NextToken": "1234-5678-9012"
}
```

CLI

Use the `list-trackers` command.

The following example is an AWS CLI to get a list of devices in a polygonal area.

```
aws location list-device-positions TODO: add arguments add props for filter geo
```

Link a tracker to a geofence collection

Now that you have a geofence collection and a tracker, you can link them together so that location updates are automatically evaluated against all of your geofences. If you don't want to evaluate all location updates, or alternatively, if you aren't storing some of your locations in a tracker resource, you can evaluate device positions against geofences on demand.

When device positions are evaluated against geofences, events are generated. You can set an action to these events. For more information about actions that you can set for geofence events, see Reacting to Amazon Location Service events with Amazon EventBridge.

An Amazon Location event includes the attributes of the device position update that generates it and some attributes of the geofence that is entered or exited. For more information about the data included in a geofence event, see Amazon EventBridge event examples for Amazon Location Service.

The following examples link a tracker resource to a geofence collection using the console, the AWS CLI, or the Amazon Location APIs.

Console

To link a tracker resource to a geofence collection using the Amazon Location Service console

1. Open the Amazon Location Service console at https://console.aws.amazon.com/location/.
2. In the left navigation pane, choose **Trackers**.
3. Under **Device trackers**, select the name link of the target tracker.
4. Under **Linked Geofence Collections**, choose **Link Geofence Collection**.
5. In the **Linked Geofence Collection** window, select a geofence collection from the dropdown menu.
6. Choose **Link**.

After you link the tracker resource, it will be assigned an **Active** status.

API

To link a tracker resource to a geofence collection using the Amazon Location APIs

Use the `AsssociateTrackerConsumer` operation from the Amazon Location Trackers APIs.

The following example uses an API request that associates *ExampleTracker* with a geofence collection using its Amazon Resource Name (ARN).

```
POST /tracking/v0/trackers/ExampleTracker/consumers
Content-type: application/json

{
   "ConsumerArn": "arn:aws:geo:us-west-2:123456789012:geofence-
collection/ExampleGeofenceCollection"
}
```

AWS CLI

To link a tracker resource to a geofence collection using AWS CLI commands

Use the `associate-tracker-consumer` command.

The following example uses an AWS CLI to create a geofence collection called *ExampleGeofenceCollection*.

```
aws location \
    associate-tracker-consumer \
        --consumer-arn "arn:aws:geo:us-west-2:123456789012:geofence-
collection/ExampleGeofenceCollection" \
        --tracker-name "ExampleTracker"
```

Evaluate device positions against geofences

There are two ways to evaluate positions against geofences to generate geofence events:

- You can link Trackers and Geofence Collections. For more information, see the section: Link a tracker to a geofence collection.
- You can make a direct request to the geofence collection resource to evaluate one or more positions, using the BatchEvaluateGeofences API.

Additionally, you can forecast incoming geofence events for a device entering, exiting, or remaining idle within a geofence. Use the ForecastGeofenceEvents API to forecast events.

If you also want to track your device location history or display locations on a map, link the tracker with a geofence collection. Alternatively, you may not want to evaluate all location updates, or you

don't intend to store location data in a tracker resource. If either of these is the case, you can make a direct request to the geofence collection and evaluate one or more device positions against its geofences.

Evaluating device positions against geofences generates events. You can react to these events and route them to other AWS services. For more information about actions that you can take when receiving geofence events, see Reacting to Amazon Location Service events with Amazon EventBridge.

An Amazon Location event includes the attributes of the device position update that generates it, including the time, position, accuracy, and key-value metadata, and some attributes of the geofence that is entered or exited. For more information about the data included in a geofence event, see Amazon EventBridge event examples for Amazon Location Service.

The following examples use the AWS CLI, or the Amazon Location APIs.

API

To evaluate device positions against the position of geofences using the Amazon Location APIs

Use the `BatchEvaluateGeofences` operation from the Amazon Location Geofences APIs.

The following example uses an API request to evaluate the position of device *`ExampleDevice`* to an associated geofence collection *`ExampleGeofenceCollection`*. Replace these values with your own geofence and device IDs.

```
POST /geofencing/v0/collections/ExampleGeofenceCollection/positions HTTP/1.1
Content-type: application/json

{
   "DevicePositionUpdates": [
      {
         "DeviceId": "ExampleDevice",
         "Position": [-123.123, 47.123],
         "SampleTime": "2021-11-30T21:47:25.149Z",
         "Accuracy": {
            "Horizontal": 10.30
         },
         "PositionProperties": {
            "field1": "value1",
            "field2": "value2"
```

```
            }
        }
    ]
}
```

AWS CLI

To evaluate device positions against the position of geofences using AWS CLI commands

Use the `batch-evaluate-geofences` command.

The following example uses an AWS CLI to evaluate the position of *ExampleDevice* against an associated geofence collection *ExampleGeofenceCollection*. Replace these values with your own geofence and device IDs.

```
aws location \
    batch-evaluate-geofences \
        --collection-name ExampleGeofenceCollection \
        --device-position-updates '[{"DeviceId":"ExampleDevice","Position":
[-123.123,47.123],"SampleTime":"2021-11-30T21:47:25.149Z","Accuracy":
{"Horizontal":10.30},"PositionProperties":{"field1":"value1","field2":"value2"}}]'
```

Evaluating device positions against geofences generates events. Traditionally you can react to the events by using Amazon EventBridge, but this process only lets you react to events after then have happened. If you need to anticipate when a device enters or exits a geofence, for example if a device is crossing a border and will be subject to a different regulations as a consequence, then you can use the ForecastGeofenceEvents API to predict future geofence events.

The ForecastGeofenceEvents API uses criteria such as the device's time-to-breach, proximity, speed, and position to predict events. There API will return a `ForecastedBreachTime`, which signals the estimated time the geofence event will occur.

The following example uses the Amazon Location APIs.

API

To forecast geofence events using the Amazon Location APIs

Use the `ForecastGeofenceEvents` operation from the Amazon Location Geofences APIs.

The following example uses an API request to forecast geofence events for an *ExampleDevice* relative to an *ExampleGeofence*. Replace these values with your own geofence and device IDs.

```
POST /geofencing/v0/collections/CollectionName/forecast-geofence-events HTTP/1.1
Content-type: application/json

{
   "DeviceState": {
      "Position": [ number ],
      "Speed": number
   },
   "DistanceUnit": "string",
   "MaxResults": number,
   "NextToken": "string",
   "SpeedUnit": "string",
   "TimeHorizonMinutes": number
}
```

Verify device positions

To check the integrity of a device position use the VerifyDevicePosition API. This API returns information about the integrity of the device's position, by evaluating properties such as the device's cell signal, Wi-Fi access point, Ipv4 address, and if a proxy is in use.

Prerequisites

Before being able to use the listed APIs for device verification, make sure you have the following prerequisite:

- You have created a tracker for the device or devices you want to check. For more information, see Start tracking.

The following example shows a request for the Amazon Location VerifyDevicePosition API.

API

To verify device positions using the Amazon Location APIs

Use the `VerifyDevicePosition` operation from the Amazon Location Tracking APIs.

The following example shows an API request to evaluate the integrity of the position of a device. Replace these values with your own device IDs.

```
POST /tracking/v0/trackers/TrackerName/positions/verify HTTP/1.1
Content-type: application/json

{
   "DeviceState": {
      "Accuracy": {
         "Horizontal": number
      },
      "CellSignals": {
         "LteCellDetails": [
            {
               "CellId": number,
               "LocalId": {
                  "Earfcn": number,
                  "Pci": number
               },
               "Mcc": number,
               "Mnc": number,
               "NetworkMeasurements": [
                  {
                     "CellId": number,
                     "Earfcn": number,
                     "Pci": number,
                     "Rsrp": number,
                     "Rsrq": number
                  }
               ],
               "NrCapable": boolean,
               "Rsrp": number,
               "Rsrq": number,
               "Tac": number,
               "TimingAdvance": number
            }
         ]
      },
      "DeviceId": "ExampleDevice",
      "Ipv4Address": "string",
      "Position": [ number ],
      "SampleTime": "string",
      "WiFiAccessPoints": [
         {
            "MacAddress": "string",
            "Rss": number
```

```
            }
         ]
      },
      "DistanceUnit": "string"
}
```

> **Note**
>
> The **Location Integrity SDK** provides enhanced features related to device verification, and it is available for use by request. To get access to the SDK, contact Sales Support.

Reacting to Amazon Location Service events with Amazon EventBridge

Amazon EventBridge is a serverless event bus that efficiently connects applications together using data from AWS services like Amazon Location. EventBridge receives events from Amazon Location and routes that data to targets like AWS Lambda. You can set up routing rules to determine where to send your data to build application architectures that react in real time.

Only geofence events (`ENTER` and `EXIT` events, as devices enter or leave the geofenced areas) are sent to EventBridge by default. You can also enable all filtered position update events for a tracker resource. For more information, see Enable update events for a tracker.

For more information, see the Events and Event Patterns in *the Amazon EventBridge User Guide*.

Topics

- Enable update events for a tracker
- Create event rules for Amazon Location
- Amazon EventBridge event examples for Amazon Location Service

Enable update events for a tracker

By default, Amazon Location sends only `ENTER` and `EXIT` geofence events to EventBridge. You can enable all filtered position `UPDATE` events for a tracker to be sent to EventBridge. You can do this when you create or update a tracker.

For example, to update an existing tracker using the AWS CLI, you can use the following command (use the name of your tracker resource in place of *`MyTracker`*).

```
aws location update-tracker --tracker-name MyTracker --event-bridge-enabled
```

To turn off position events for a tracker, you must use the API or the Amazon Location Service console.

Create event rules for Amazon Location

You can create up to 300 rules per event bus in EventBridge to configure actions taken in response to an Amazon Location event.

For example, you can create a rule for geofence events where a push notification will be sent when a phone is detected within a geofenced boundary.

To create a rule for Amazon Location events

Using the following values, create an EventBridge rule based on Amazon Location events:

- For **Rule type**, choose **Rule with an event pattern**.
- In the **Event pattern** box, add the following pattern:

```
{
  "source": ["aws.geo"],
  "detail-type": ["Location Geofence Event"]
}
```

To create a rule for tracker position updates, you can instead use the following pattern:

```
{
  "source": ["aws.geo"],
  "detail-type": ["Location Device Position Event"]
}
```

You can optionally specify only `ENTER` or `EXIT` events by adding a `detail` tag (if your rule is for tracker position updates, there is only a single `EventType`, so there is no need to filter on it):

```
{
  "source": ["aws.geo"],
  "detail-type": ["Location Geofence Event"],
  "detail": {
    "EventType": ["ENTER"]
  }
```

```
}
```

You can also optionally filter on properties of the position or geofence:

```
{
  "source": ["aws.geo"],
  "detail-type": ["Location Geofence Event"],
  "detail": {
    "EventType": ["ENTER"],
    "GeofenceProperties": {
      "Type": "LoadingDock"
    },
    "PositionProperties": {
      "VehicleType": "Truck"
    }
  }
}
```

- For **Select targets**, choose the target action to take when an event is received from Amazon Location Service.

  For example, use an Amazon Simple Notification Service (SNS) topic to send an email or text message when an event occurs. You first need to create an Amazon SNS topic using the Amazon SNS console. For more information, see Using Amazon SNS for user notifications.

> **Warning**
>
> It's best practice to confirm that the event rule was successfully applied or your automated action may not initiate as expected. To verify your event rule, initiate conditions for the event rule. For example, simulate a device entering a geofenced area.

You can also capture all events from Amazon Location, by just excluding the `detail-type` section. For example:

```
{
  "source": [
    "aws.geo"
  ]
}
```

Note

The same event may be delivered more than one time. You can use the event id to de-duplicate the events that you receive.

Amazon EventBridge event examples for Amazon Location Service

The following is an example of an event for entering a geofence initiated by calling `BatchUpdateDevicePosition`.


```
{
  "version": "0",
  "id": "aa11aa22-33a-4a4a-aaa5-example",
  "detail-type": "Location Geofence Event",
  "source": "aws.geo",
  "account": "636103698109",
  "time": "2020-11-10T23:43:37Z",
  "region": "eu-west-1",
  "resources": [
    "arn:aws:geo:eu-west-1:0123456789101:geofence-collection/GeofenceEvents-
GeofenceCollection_EXAMPLE",
    "arn:aws:geo:eu-west-1:0123456789101:tracker/Tracker_EXAMPLE"
  ],
  "detail": {
    "EventType": "ENTER",
    "GeofenceId": "polygon_14",
    "DeviceId": "Device1-EXAMPLE",
    "SampleTime": "2020-11-10T23:43:37.531Z",
    "Position": [
      -123.12390073297821,
      49.23433613216247
    ],
    "Accuracy": {
      "Horizontal": 15.3
    },
    "GeofenceProperties": {
      "ExampleKey1": "ExampleField1",
      "ExampleKey2": "ExampleField2"
    },
    "PositionProperties": {
      "ExampleKey1": "ExampleField1",
```

```
      "ExampleKey2": "ExampleField2"
    }
  }
}
```

The following is an example of an event for exiting a geofence initiated by calling `BatchUpdateDevicePosition`.

```
{
  "version": "0",
  "id": "aa11aa22-33a-4a4a-aaa5-example",
  "detail-type": "Location Geofence Event",
  "source": "aws.geo",
  "account": "123456789012",
  "time": "2020-11-10T23:41:44Z",
  "region": "eu-west-1",
  "resources": [
    "arn:aws:geo:eu-west-1:0123456789101:geofence-collection/GeofenceEvents-
GeofenceCollection_EXAMPLE",
    "arn:aws:geo:eu-west-1:0123456789101:tracker/Tracker_EXAMPLE"
  ],
  "detail": {
    "EventType": "EXIT",
    "GeofenceId": "polygon_10",
    "DeviceId": "Device1-EXAMPLE",
    "SampleTime": "2020-11-10T23:41:43.826Z",
    "Position": [
      -123.08569321875426,
      49.23766166742559
    ],
    "Accuracy": {
      "Horizontal": 15.3
    },
    "GeofenceProperties": {
      "ExampleKey1": "ExampleField1",
      "ExampleKey2": "ExampleField2"
    },
    "PositionProperties": {
      "ExampleKey1": "ExampleField1",
      "ExampleKey2": "ExampleField2"
    }
  }
}
```

The following is an example of an event for a position update, initiated by calling `BatchUpdateDevicePosition`.

```
{
  "version": "0",
  "id": "aa11aa22-33a-4a4a-aaa5-example",
  "detail-type": "Location Device Position Event",
  "source": "aws.geo",
  "account": "123456789012",
  "time": "2020-11-10T23:41:44Z",
  "region": "eu-west-1",
  "resources": [
    "arn:aws:geo:eu-west-1:0123456789101:tracker/Tracker_EXAMPLE"
  ],
  "detail": {
    "EventType": "UPDATE",
    "TrackerName": "tracker_2",
    "DeviceId": "Device1-EXAMPLE",
    "SampleTime": "2020-11-10T23:41:43.826Z",
    "ReceivedTime": "2020-11-10T23:41:39.235Z",
    "Position": [
      -123.08569321875426,
      49.23766166742559
    ],
    "Accuracy": {
      "Horizontal": 15.3
    },
    "PositionProperties": {
      "ExampleKey1": "ExampleField1",
      "ExampleKey2": "ExampleField2"
    }
  }
}
```

Tracking using AWS IoT and MQTT with Amazon Location Service

MQTT is a lightweight and widely adopted messaging protocol designed for constrained devices. AWS IoT Core supports device connections that use the MQTT protocol and MQTT over WebSocket Secure (WSS) protocol.

AWS IoT Core connects devices to AWS and enables you to send and receive messages between them. The AWS IoT Core rules engine stores queries about your devices' message topics and

enables you to define actions for sending messages to other AWS services, such as Amazon Location Service. Devices that are aware of their location as coordinates can have their locations forwarded to Amazon Location through the rules engine.

> **Note**
>
> Devices may know their own position, for example via built-in GPS. AWS IoT also has support for third party device location tracking. For more information, see AWS IoT Core Device Location in the *AWS IoT Core Developer Guide*.

The following walkthrough describes tracking using AWS IoT Core rules. You can also send the device information to your own AWS Lambda function, if you need to process it before sending to Amazon Location. For more details about using Lambda to process your device locations, see Using AWS Lambda with MQTT.

Topics

- Prerequisite
- Create an AWS IoT Core rule
- Test your AWS IoT Core rule in the console
- Using AWS Lambda with MQTT

Prerequisite

Before you can begin tracking, you must complete the following prerequisites:

- Create a tracker resource that you will send the device location data to.
- Create an IAM role for granting AWS IoT Core access to your tracker.

  When following those steps, use the following policy to give access to your tracker:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "WriteDevicePosition",
      "Effect": "Allow",
      "Action": "geo:BatchUpdateDevicePosition",
```

```
      "Resource": "arn:aws:geo:*:*:tracker/*"
    }
  ]
}
```

Create an AWS IoT Core rule

Next, create an AWS IoT Core rule to forward your devices' positional telemetry to Amazon Location Service. For more information about creating rules, see the following topics in the *AWS IoT Core Developer Guide*:

- Creating an AWS IoT rule for information about creating a new rule.
- Location action for information specific to creating a rule for publishing to Amazon Location

Test your AWS IoT Core rule in the console

If no devices are currently publishing telemetry that includes location, you can test your rule using the AWS IoT Core console. The console has a test client where you can publish a sample message to verify the results of the solution.

1. Sign in to the AWS IoT Core console at https://console.aws.amazon.com/iot/.
2. In the left navigation, expand **Test**, and choose **MQTT test client**.
3. Under **Publish to a topic**, set the **Topic name** to *iot/topic* (or the name of the topic that you set up in your AWS IoT Core rule, if different), and provide the following for the **Message payload**.

   ```
   {
     "payload": {
       "deviceid": "thing123",
       "timestamp": 1604940328,
       "location": { "lat": 49.2819, "long": -123.1187 },
       "accuracy": { "Horizontal": 20.5 },
       "positionProperties": { "field1": "value1", "field2": "value2" }
     }
   }
   ```

4. Choose **Publish** to topic to send the test message.

5. To validate that the message was received by Amazon Location Service, use the following AWS CLI command. If you modified it during setup, replace the tracker name with the one that you used.

```
aws location batch-get-device-position --tracker-name MyTracker --device-ids
 thing123
```

Using AWS Lambda with MQTT

While using AWS Lambda is no longer required when sending device location data to Amazon Location for tracking, you may still want to use Lambda in some cases. For example, if you wish to process your device location data yourself, before sending it on to Amazon Location. The following topics describe how to use Lambda to process messages before sending them to your tracker. For more information about this pattern, see the reference architecture.

Topics

- Prerequisite
- Create a Lambda function
- Create an AWS IoT Core rule
- Test your AWS IoT Core rule in the console

Prerequisite

Before you can begin tracking, you must create a tracker resource. To create a tracker resource, you can use the Amazon Location console, the AWS CLI, or the Amazon Location APIs.

The following example uses the Amazon Location Service console to create the tracker resource:

1. Open the Amazon Location Service console at https://console.aws.amazon.com/location/.
2. In the left navigation pane, choose **Trackers**.
3. Choose **Create tracker**.
4. Fill out the following boxes:
   - **Name** – Enter a unique name that has a maximum of 100 characters. Valid entries include alphanumeric characters, hyphens, and underscores. For example, *MyTracker*.

- **Description** – Enter an optional description. For example, *Tracker for storing AWS IoT Core device positions*.
- **Position filtering** – Select the filtering that you want to use for position updates. For example, **Accuracy-based filtering**.

5. Choose **Create tracker**.

Create a Lambda function

To create a connection between AWS IoT Core and Amazon Location Service, you need an AWS Lambda function to process messages forwarded by AWS IoT Core. This function will extract any positional data, format it for Amazon Location Service, and submit it through the Amazon Location Tracker API. You can create this function through the AWS Lambda console, or you can use the AWS Command Line Interface (AWS CLI) or the AWS Lambda APIs.

To create a Lambda function that publishes position updates to Amazon Location using the console:

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. From the left navigation, choose **Functions**.
3. Choose **Create Function**, and make sure that **Author from scratch** is selected.
4. Fill out the following boxes:

   - **Function name** – Enter a unique name for your function. Valid entries include alphanumeric characters, hyphens, and underscores with no spaces. For example, *MyLambda*.
   - **Runtime** – Choose *Python 3.8*.

5. Choose **Create function**.
6. Choose the **Code** tab to open the editor.
7. Overwrite the placeholder code in `lambda_function.py` with the following, replacing the value assigned to `TRACKER_NAME` with the name of the tracker that you created as a prerequisite.

```
from datetime import datetime
import json
import os

import boto3
```

```
# Update this to match the name of your Tracker resource
TRACKER_NAME = "MyTracker"

"""
This Lambda function receives a payload from AWS IoT Core and publishes device
 updates to
Amazon Location Service via the BatchUpdateDevicePosition API.

Parameter 'event' is the payload delivered from AWS IoT Core.

In this sample, we assume that the payload has a single top-level key 'payload' and
 a nested key
'location' with keys 'lat' and 'long'. We also assume that the name of the device
 is nested in
the payload as 'deviceid'. Finally, the timestamp of the payload is present as
 'timestamp'. For
example:

>>> event
{ 'payload': { 'deviceid': 'thing123', 'timestamp': 1604940328,
  'location': { 'lat': 49.2819, 'long': -123.1187 },
  'accuracy': {'Horizontal': 20.5 },
  'positionProperties': {'field1':'value1','field2':'value2'} }
}

If your data doesn't match this schema, you can either use the AWS IoT Core rules
 engine to
format the data before delivering it to this Lambda function, or you can modify the
 code below to
match it.
"""
def lambda_handler(event, context):
  update = {
      "DeviceId": event["payload"]["deviceid"],
      "SampleTime": datetime.fromtimestamp(event["payload"]
["timestamp"]).strftime("%Y-%m-%dT%H:%M:%SZ"),
      "Position": [
        event["payload"]["location"]["long"],
        event["payload"]["location"]["lat"]
        ]
    }
  if "accuracy" in event["payload"]:
      update["Accuracy"] = event["payload"]['accuracy']
  if "positionProperties" in event["payload"]:
```

```
      update["PositionProperties"] = event["payload"]['positionProperties']

  client = boto3.client("location")
  response = client.batch_update_device_position(TrackerName=TRACKER_NAME,
 Updates=[update])

  return {
    "statusCode": 200,
    "body": json.dumps(response)
  }
```

8. Choose **Deploy** to save the updated function.
9. Choose the **Configuration** tab.
10. In the **Permissions** section, choose the hyperlinked Role name to grant Amazon Location Service permissions to your Lambda function.
11. From your role's **Summary** page, choose **Add permissions**, and then from the dropdown list, select **Create inline policy**.
12. Choose the **JSON** tab, and overwrite the policy with the following document. This allows your Lambda function to update device positions managed by all tracker resources across all Regions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "WriteDevicePosition",
      "Effect": "Allow",
      "Action": "geo:BatchUpdateDevicePosition",
      "Resource": "arn:aws:geo:*:*:tracker/*"
    }
  ]
}
```

13. Choose **Review policy**.
14. Enter a policy name. For example, *AmazonLocationTrackerWriteOnly*.
15. Choose **Create policy**.

You can modify this function code, as necessary, to adapt to your own device message schema.

Create an AWS IoT Core rule

Next, create an AWS IoT Core rule to forward your devices' positional telemetry to the AWS Lambda function for transformation and publication to Amazon Location Service. The example rule provided assumes that any necessary transformation of device payloads is handled by your Lambda function. You can create this rule through the AWS IoT Core console, the AWS Command Line Interface (AWS CLI), or the AWS IoT Core APIs.

> **Note**
>
> While the AWS IoT console handles the permission necessary to allow AWS IoT Core to invoke your Lambda function, if you are creating your rule from the AWS CLI or SDK, you must configure a policy to grant permission to AWS IoT.

To create an AWS IoT Core using the console

1. Sign in to the AWS IoT Core console at https://console.aws.amazon.com/iot/.
2. In the left navigation, expand **Act**, and choose **Rules**.
3. Choose **Create a rule** to start the new rule wizard.
4. Enter a name and description for your rule.
5. For the **Rule query statement**, update the `FROM` attribute to refer to a topic where at least one device is publishing telemetry that includes location. If you are testing the solution, no modification is needed.

   ```
   SELECT * FROM 'iot/topic'
   ```
6. Under **Set one or more actions** , choose **Add action**.
7. Select **Send a message to a lambda function**.
8. Choose **Configure action**.
9. Find and select your Lambda function from the list.
10. Choose **Add action**.
11. Choose **Create rule**.

Test your AWS IoT Core rule in the console

If no devices are currently publishing telemetry that includes location, you can test your rule and this solution using the AWS IoT Core console. The console has a test client where you can publish a sample message to verify the results of the solution.

1. Sign in to the AWS IoT Core console at https://console.aws.amazon.com/iot/.
2. In the left navigation, expand **Test**, and choose **MQTT test client**.
3. Under **Publish to a topic**, set the **Topic name** to *iot/topic* (or the name of the topic that you set up in your AWS IoT Core rule, if different), and provide the following for the **Message payload**. Replace the timestamp *1604940328* with a valid timestamp within the last 30 days (any timestamps older than 30 days are ignored).

```
{
  "payload": {
    "deviceid": "thing123",
    "timestamp": 1604940328,
    "location": { "lat": 49.2819, "long": -123.1187 },
    "accuracy": { "Horizontal": 20.5 },
    "positionProperties": { "field1": "value1", "field2": "value2" }
  }
}
```

4. Choose **Publish** to topic to send the test message.
5. To validate that the message was received by Amazon Location Service, use the following AWS CLI command. If you modified them during setup, replace the tracker name and device id with the ones that you used.

```
aws location batch-get-device-position --tracker-name MyTracker --device-ids
 thing123
```

Managing your geofence collection resources

Manage your geofence collections using the Amazon Location console, the AWS CLI, or the Amazon Location APIs.

List your geofence collection resources

You can view your geofence collection list using the Amazon Location console, the AWS CLI, or the Amazon Location APIs:

Console

To view a list of geofence collections using the Amazon Location console

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. Choose **Geofence collections** from the left navigation pane.
3. View a list of your geofence collections under **My geofence collections**.

API

Use the `ListGeofenceCollections` operation from the Amazon Location Geofences APIs.

The following example is an API request to get a list of geofence collections in the AWS account.

```
POST /geofencing/v0/list-collections
```

The following is an example response for `ListGeofenceCollections`:

```
{
    "Entries": [
    {
        "CollectionName": "ExampleCollection",
        "CreateTime": 2020-09-30T22:59:34.142Z,
        "Description": "string",
        "UpdateTime": 2020-09-30T23:59:34.142Z
    },
    "NextToken": "1234-5678-9012"
}
```

CLI

Use the `list-geofence-collections` command.

The following example is an AWS CLI to get a list of geofence collections in the AWS account.

```
aws location list-geofence-collections
```

Get geofence collection details

You can get details about any geofence collection resource in your AWS account using the Amazon Location console, the AWS CLI, or the Amazon Location APIs:

Console

To view the details of a geofence collection using the Amazon Location console

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. Choose **Geofence collections** from the left navigation pane.
3. Under **My geofence collections**, select the name link of the target geofence collection.

API

Use the `DescribeGeofenceCollection` operation from the Amazon Location Geofences APIs.

The following example is an API request to get the geofence collection details for *`ExampleCollection`*.

```
GET /geofencing/v0/collections/ExampleCollection
```

The following is an example response for `DescribeGeofenceCollection`:

```
{
    "CollectionArn": "arn:aws:geo:us-west-2:123456789012:geofence-collection/
GeofenceCollection",
    "CollectionName": "ExampleCollection",
    "CreateTime": 2020-09-30T22:59:34.142Z,
    "Description": "string",
    "KmsKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "Tags": {
        "Tag1" : "Value1"
    },
    "UpdateTime": 2020-09-30T23:59:34.142Z
}
```

CLI

Use the `describe-geofence-collection` command.

The following example is an AWS CLI to get the geofence collection details for *ExampleCollection*.

```
aws location describe-geofence-collection \
    --collection-name "ExampleCollection"
```

Delete a geofence collection

You can delete a geofence collection from your AWS account using the Amazon Location console, the AWS CLI, or the Amazon Location APIs.

Console

To delete a geofence collection using the Amazon Location console

> **Warning**
>
> This operation deletes the resource permanently.

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. Choose **Geofence collections** from the left navigation pane.
3. Under **My geofence collection**, select the target geofence collection.
4. Choose **Delete geofence collection**.

API

Use the `DeleteGeofenceCollection` operation from the Amazon Location APIs.

The following example is an API request to delete the geofence collection *ExampleCollection*.

```
DELETE /geofencing/v0/collections/ExampleCollection
```

The following is an example response for `DeleteGeofenceCollection`:

```
HTTP/1.1 200
```

CLI

Use the `delete-geofence-collection` command.

The following example is an AWS CLI command to delete the geofence collection *ExampleCollection*.

```
aws location delete-geofence-collection \
    --collection-name "ExampleCollection"
```

List stored geofences

You can list geofences stored in a specified geofence collection using the Amazon Location console, the AWS CLI, or the Amazon Location APIs.

Console

To view a list of geofences using the Amazon Location console

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. Choose **Geofence collections** from the left navigation pane.
3. Under **My geofence collection**, select the name link of the target geofence collection.
4. View geofences in the geofence collection under **Geofences**

API

Use the `ListGeofences` operation from the Amazon Location Geofences APIs.

The following example is an API request to get a list of geofences stored in the geofence collection *ExampleCollection*.

```
POST /geofencing/v0/collections/ExampleCollection/list-geofences
```

The following is an example response for `ListGeofences`:

```
{
   "Entries": [
      {
         "CreateTime": 2020-09-30T22:59:34.142Z,
```

```
            "GeofenceId": "geofence-1",
            "Geometry": {
                "Polygon": [
                    [-5.716667, -15.933333,
                    [-14.416667, -7.933333],
                    [-12.316667, -37.066667],
                    [-5.716667, -15.933333]
                ]
            },
            "Status": "ACTIVE",
            "UpdateTime": 2020-09-30T23:59:34.142Z
        }
    ],
    "NextToken": "1234-5678-9012"
}
```

CLI

Use the `list-geofences` command.

The following example is an AWS CLI to get a list of geofences stored in the geofence collection *ExampleCollection*.

```
aws location list-geofences \
    --collection-name "ExampleCollection"
```

Get geofence details

You can get the details of a specific geofence, such as the create time, update time, geometry, and status, from a geofence collection using the Amazon Location console, AWS CLI, or the Amazon Location APIs.

Console

To view the status of a geofence using the Amazon Location console

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. Choose **Geofence collections** from the left navigation pane.
3. Under **My geofence collection**, select the name link of the target geofence collection.
4. Under **Geofences**, you'll be able to view the status of your geofences.

API

Use the GetGeofence operation from the Amazon Location Geofences APIs.

The following example is an API request to get the geofence details from a geofence collection *ExampleCollection*.

```
GET /geofencing/v0/collections/ExampleCollection/geofences/ExampleGeofence1
```

The following is an example response for GetGeofence:

```
{
   "CreateTime": 2020-09-30T22:59:34.142Z,
   "GeofenceId": "ExampleGeofence1",
   "Geometry": {
      "Polygon": [
          [-1,-1],
          [1,-1],
          [0,1],
          [-1,-1]
      ]
   },
   "Status": "ACTIVE",
   "UpdateTime": 2020-09-30T23:59:34.142Z
}
```

CLI

Use the get-geofence command.

The following example is an AWS CLI to get the geofence collection details for *ExampleCollection*.

```
aws location get-geofence \
    --collection-name "ExampleCollection" \
    --geofence-id "ExampleGeofence1"
```

Delete geofences

You can delete geofences from a geofence collection using the Amazon Location console, the AWS CLI, or the Amazon Location APIs.

Console

To delete a geofence using the Amazon Location console

> **Warning**
>
> This operation deletes the resource permanently.

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. Choose **Geofence collections** from the left navigation pane.
3. Under **My geofence collection**, select the name link of the target geofence collection.
4. Under **Geofences**, select the target geofence.
5. Choose **Delete geofence**.

API

Use the `BatchDeleteGeofence` operation from the Amazon Location Geofences APIs.

The following example is an API request to delete geofences from the geofence collection *ExampleCollection*.

```
POST /geofencing/v0/collections/ExampleCollection/delete-geofences
Content-type: application/json

{
   "GeofenceIds": [ "ExampleGeofence11" ]
}
```

The following is an example success response for `BatchDeleteGeofence`.

```
HTTP/1.1 200
```

CLI

Use the `batch-delete-geofence` command.

The following example is an AWS CLI command to delete geofences from the geofence collection *ExampleCollection*.

```
aws location batch-delete-geofence \
    --collection-name "ExampleCollection" \
    --geofence-ids "ExampleGeofence11"
```

Managing your tracker resources

You can manage your trackers using the Amazon Location console, the AWS CLI, or the Amazon Location APIs.

List your trackers

You can view your trackers list using the Amazon Location console, the AWS CLI, or the Amazon Location APIs:

Console

To view a list of existing trackers using the Amazon Location console

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. Choose **Trackers** from the left navigation.
3. View a list of your tracker resources under **My trackers**.

API

Use the `ListTrackers` operation from the Amazon Location Trackers APIs.

The following example is an API request to get a list of trackers in your AWS account.

```
POST /tracking/v0/list-trackers
```

The following is an example response for `ListTrackers`:

```
{
   "Entries": [
      {
         "CreateTime": 2020-10-02T19:09:07.327Z,
         "Description": "string",
         "TrackerName": "ExampleTracker",
         "UpdateTime": 2020-10-02T19:10:07.327Z
```

```
        }
    ],
    "NextToken": "1234-5678-9012"
}
```

CLI

Use the `list-trackers` command.

The following example is an AWS CLI to get a list of trackers in your AWS account.

```
aws location list-trackers
```

Disconnecting a tracker from a geofence collection

You can disconnect a tracker from a geofence collection using the Amazon Location console, the AWS CLI, or the Amazon Location APIs:

Console

To disassociate a tracker from an associated geofence collection using the Amazon Location console

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. Choose **Trackers** from the left navigation pane.
3. Under **My trackers**, select the name link of the target tracker.
4. Under **Linked Geofence Collections**, select a geofence collection with a **Linked** status.
5. Choose **Unlink**.

API

Use the `DisassociateTrackerConsumer` operation from the Amazon Location Trackers APIs.

The following example is an API request to disassociate a tracker from an associated geofence collection.

```
DELETE /tracking/v0/trackers/ExampleTracker/consumers/arn:aws:geo:us-
west-2:123456789012:geofence-collection/ExampleCollection
```

The following is an example response for DisassociateTrackerConsumer:

```
HTTP/1.1 200
```

CLI

Use the disassociate-tracker-consumer command.

The following example is an AWS CLI command to disassociate a tracker from an associated geofence collection.

```
aws location disassociate-tracker-consumer \
    --consumer-arn "arn:aws:geo:us-west-2:123456789012:geofence-collection/
ExampleCollection" \
    --tracker-name "ExampleTracker"
```

Get tracker details

You can get details about any tracker in your AWS account by using the Amazon Location console, the AWS CLI, or the Amazon Location APIs.

Console

To view tracker details by using the Amazon Location console

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. Choose **Trackers** from the left navigation.
3. Under **My trackers**, select the name link of the target tracker.
4. View the tracker details under **Information**.

API

Use the DescribeTracker operation from the Amazon Location Tracker APIs.

The following example is an API request to get the tracker details for *ExampleTracker*.

```
GET /tracking/v0/trackers/ExampleTracker
```

The following is an example response for DescribeTracker:

```
{
   "CreateTime": 2020-10-02T19:09:07.327Z,
   "Description": "string",
   "EventBridgeEnabled": false,
   "KmsKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
   "PositionFiltering": "TimeBased",
   "Tags": {
      "Tag1" : "Value1"
   },
   "TrackerArn": "arn:aws:geo:us-west-2:123456789012:tracker/ExampleTracker",
   "TrackerName": "ExampleTracker",
   "UpdateTime": 2020-10-02T19:10:07.327Z
}
```

CLI

Use the describe-tracker command.

The following example is an AWS CLI command to get tracker details for *ExampleTracker*.

```
aws location describe-tracker \
    --tracker-name "ExampleTracker"
```

Delete a tracker

You can delete a tracker from your AWS account using the Amazon Location console, the AWS CLI, or the Amazon Location APIs:

Console

To delete an existing map resource using the Amazon Location console

> **Warning**
>
> This operation deletes the resource permanently. If the tracker resource is in use, you may encounter an error. Make sure that the target resource isn't a dependency for your applications.

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. Choose **Trackers** from the left navigation pane.
3. Under **My trackers**, select the target tracker.
4. Choose **Delete tracker**.

API

Use the `DeleteTracker` operation from the Amazon Location Tracker APIs.

The following example is an API request to delete the tracker *ExampleTracker*.

```
DELETE /tracking/v0/trackers/ExampleTracker
```

The following is an example response for `DeleteTracker`:

```
HTTP/1.1 200
```

CLI

Use the `delete-tracker` command.

The following example is an AWS CLI command to delete the tracker *ExampleTracker*.

```
aws location delete-tracker \
    --tracker-name "ExampleTracker"
```

Sample Geofencing and Tracking mobile application

This topic covers tutorials designed to demonstrate the key features of using the Amazon Location geofences and trackers in a mobile application. The applications demonstrate how a tracker and geofence interact using a combination of Lambda, AWS IoT and Amazon Location features. There are two tutorials available.

- Sample tracking and geofencing application for Android, and you can clone the project files from GitHub: https://github.com/aws-geospatial/amazon-location-samples-android/tree/main/tracking-with-geofence-notifications.

- Sample tracking and geofencing application for iOS, and you can clone the project files from GitHub: https://github.com/aws-geospatial/amazon-location-samples-ios/tree/main/tracking-with-geofence-notifications.

Sample tracking and geofence application for Android

This topic covers the Android tutorial designed to demonstrate the key features of using the Amazon Location geofences and trackers in a mobile application. The applications demonstrate how a tracker and geonfence interact using a combination of Lambda, AWS IoT and Amazon Location features.

Topics

- Create Amazon Location resources for your app
- Create a Geofence Collection
- Link a tracker to a geofence collection
- Using AWS Lambda with MQTT
- Set up the sample app code
- Using the sample app

Create Amazon Location resources for your app

To begin you will need to create the required Amazon Location resources. These resources will be essential for the functionality of the application and executing the provided code snippets.

> **Note**
>
> If you haven't created an AWS account, follow the instructions in the AWS account managment user guide.

To begin you will need to create a Amazon Cognito Identity Pool Id, use the following procedure:

1. Open the Amazon Cognito console and select **Identity pools** from the left side menu, then select **Create Identity pool**.
2. Make sure **Guest Access** is checked, and press **Next** to contiue.
3. Next create a new IAM role or Use an existing IAM role.

4. Enter an Identity pool name, and make sure Identity Pool has access to Amazon Location (geo)resources for the map and tracker you will be creating nin the next procedure.

Next you need to create and style a map in the AWS Amazon Location console, use the following procedure:

1. Navigate to the Maps section of the Amazon Location console and select **Create Map**.
2. Give the new map a **Name** and **Description**. Record the name you assign, as it is used later in the tutorial.
3. When choosing a map style, consider the map data provider. Refer to section 82 of the AWS service terms for more details.
4. Accept the Amazon Location Terms and Conditions, then select **Create Map**, to finish the map creation process.

Next you need to create a tracker in the Amazon Location console, use the following procedure:

1. Open the Maps section in the Amazon Location console.
2. Choose **Create tracker**.
3. Fill in the required fields. Make note of the tracker's **Name** as it will be refrenced throughout this tutoiral.
4. Under the **Position filtering** field, choose the option that best fits how you intend to use your tracker resource. If you do not set Position filtering, the default setting is `TimeBased`. For more information, see Trackers, and PositionFiltering in the Amazon Location API Reference.
5. Choose **Create tracker** to finish creating the tracker.

Create a Geofence Collection

Now will you create a geofence collection. You can use either the console, API or CLI. The following procedures walk you through each option.

- Create a geofence collection using the Amazon Location console:

  1. Open the Geofence Collections section of the Amazon Location console.
  2. Choose **Create geofence collection**.
  3. Provide a name and description for the collection.

4. Under the EventBridge rule with Amazon CloudWatch as a target, you can create an optional EventBridge rule to get started reacting to geofence events. This enables Amazon Location to publish events to Amazon CloudWatch Logs.
5. Press the **Create geofence collection** to finish creating the collection.

- Create a geofence collection using the Amazon Location API:

  Use the CreateGeofenceCollection operation from the Amazon Location Geofences APIs. The following example uses an API request to create a geofence collection called *GEOCOLLECTION_NAME*.

```
POST /geofencing/v0/collections
Content-type: application/json

{
   "CollectionName": "GEOCOLLECTION_NAME",
   "Description": "Geofence collection 1 for shopping center",
   "Tags": {
      "Tag1" : "Value1"
   }
}
```

- Create a geofence collection using AWS CLI commands:

  Use the `create-geofence-collection` command. The following example uses an AWS CLI to create a geofence collection called *GEOCOLLECTION_NAME*. For more information on using the AWS CLI, see the AWS Command Line Interface Documentation.

```
aws location \
  create-geofence-collection \
  --collection-name "ExampleGeofenceCollection" \
  --description "Shopping center geofence collection" \
  --tags Tag1=Value1
```

Link a tracker to a geofence collection

To link a tracker to a geofence collection you can use either the console, API, or CLI. The following procedures walk you through each option.

Link a tracker resource to a geofence collection using the Amazon Location Service console:

1. Open the Amazon Location console.
2. In the left navigation pane, choose **Trackers**.
3. Under **Device Trackers**, select the name link of the target tracker.
4. Under **Linked Geofence Collections**, choose **Link Geofence Collection**.
5. In the **Linked Geofence Collection window**, select a geofence collection from the dropdown menu.
6. Choose **Link**.
7. After you link the tracker resource, it will be assigned an Active status.

Link a tracker resource to a geofence collection using the Amazon Location APIs:

Use the `AsssociateTrackerConsumer` operation from the Amazon Location Trackers APIs. The following example uses an API request that associates an `ExampleTracker` with a geofence collection using its Amazon Resource Name (ARN).

```
        POST /tracking/v0/trackers/ExampleTracker/consumers
        Content-type: application/json
        {
           "ConsumerArn": "arn:aws:geo:us-west-2:123456789012:geofence-
collection/GOECOLLECTION_NAME"
            }
```

Link a tracker resource to a geofence collection using AWS CLI commands:

Use the `associate-tracker-consumer` command. The following example uses an AWS CLI to create a geofence collection called *GOECOLLECTION_NAME*.

```
aws location \
associate-tracker-consumer \
    --consumer-arn "arn:aws:geo:us-west-2:123456789012:geofence-
collection/GOECOLLECTION_NAME" \
    --tracker-name "ExampleTracker"
```

Using AWS Lambda with MQTT

In order to create a connection between AWS IoT and Amazon Location, you need a Lambda function to process messages forwarded by EventBridge CloudWatch events. This function will extract any positional data, format it for Amazon Location, and submit it through the Amazon Location Tracker API.

The following procedure shows you how to create this function through the Lambda console:

1. Open the console.
2. From the left navigation, choose **Functions**.
3. Then choose **Create Function**, and make sure that the **Author from scratch** option is selected.
4. provide a **Function name**, and for the **Runtime** option, choose Node.js 16.x.
5. Choose **Create function**.
6. Open the **Code tab** to access the editor.
7. Overwrite the placeholder code in the `index.js` file with the following:

```
            const AWS = require('aws-sdk')
            const iot = new AWS.Iot();
            exports.handler =  function(event) {
              console.log("event===>>>", JSON.stringify(event));
              var param = {
                endpointType: "iot:Data-ATS"
              };
              iot.describeEndpoint(param, function(err, data) {
                if (err) {
                  console.log("error===>>>", err, err.stack); // an error occurred
                } else {
                  var endp = data['endpointAddress'];
                  const iotdata = new AWS.IotData({endpoint: endp});
                  const trackerEvent = event["detail"]["EventType"];
                  const src = event["source"];
                  const time = event["time"];
                  const gfId = event["detail"]["GeofenceId"];
                  const resources = event["resources"][0];
                  const splitResources = resources.split(".");
                  const geofenceCollection = splitResources[splitResources.length -
1];
                  const coordinates = event["detail"]["Position"];
```

```
                    const deviceId = event["detail"]["DeviceId"];
                    console.log("deviceId===>>>", deviceId);
                    const msg =  {
                        "trackerEventType" : trackerEvent,
                        "source" : src,
                        "eventTime" : time,
                        "geofenceId" : gfId,
                        "coordinates": coordinates,
                        "geofenceCollection": geofenceCollection
                      };
                    const params = {
                      topic: `${deviceId}/tracker`,
                      payload: JSON.stringify(msg),
                      qos: 0
                    };
                    iotdata.publish(params, function(err, data) {
                        if (err) {
                          console.log("error===>>>", err, err.stack); // an error
occurred
                        } else {
                          console.log("Ladmbda triggered===>>>", trackerEvent);  //
successful response
                        }
                    });
                  }
                });
            }
```

8. Press the **Deploy** to save the updated function.
9. Next open the **Configuration** tab.
10. In the **Triggers** section, press the **Add Trigger** button.
11. Select **EventBridge (CloudWatch Events)** in Source field.
12. Select the **Existing Rules** option.
13. Enter the rule name, for example `AmazonLocationMonitor-GEOFENCECOLLECTION_NAME`.
14. Press the **Add** button.
15. This will also attach **Resource-based policy statements** in the permissions tab

Now you will set up the MQTT Test Client using AWS IoT, use the following procedure:

1. Open the https://console.aws.amazon.com/iot/.

2. In the left navigation pane, select the **MQTT test client**.
3. You'll see a section titled **MQTT test client** where you can configure your MQTT connection.
4. After configuring the necessary settings, click on the **Connect** button to establish a connection to the MQTT broker using the provided parameters.
5. Record endpoint, as it is used later in the tutoiral.

   Once connected to the test client, you can subscribe to MQTT topics or publish messages to topics using the respective input fields provided in the MQTT test client interface. Next you will create an AWS IoT policy.
6. On the left side menu, under **Manage** expand **Security** option and click on **Policies**.
7. Click on **Create Policy** button.
8. Enter a policy name.
9. On **Policy Document** select **JSON** tab.
10. Copy paste the policy shown below, but make sure to update all elements with your *REGION* and *ACCOUNT_ID*:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:REGION:ACCOUNT_ID:client/${cognito-
identity.amazonaws.com:sub}",
        "arn:aws:iot:REGION:ACCOUNT_ID:topic/${cognito-
identity.amazonaws.com:sub}",
        "arn:aws:iot:REGION:ACCOUNT_ID:topicfilter/${cognito-
identity.amazonaws.com:sub}/*",
        "arn:aws:iot:REGION:ACCOUNT_ID:topic/${cognito-identity.amazonaws.com:sub}/
tracker"
      ],
      "Effect": "Allow"
    }
  ]
```

```
}
```

11. Select the **Create** button to finish.

After completing the previous procedure, you will now update the permissions for the guest role as follows:

1. Navigate to Amazon Cognito and open your identity pool. Then, proceed to user access and select the guest role.
2. Click on permission policies to enable editing.

```
{
    'Version': '2012-10-17',
    'Statement': [
        {
            'Action': [
                'geo:GetMap*',
                'geo:BatchUpdateDevicePosition',
                'geo:BatchEvaluateGeofences',
                'iot:Subscribe',
                'iot:Publish',
                'iot:Connect',
                'iot:Receive',
                'iot:AttachPrincipalPolicy',
                'iot:AttachPolicy',
                'iot:DetachPrincipalPolicy',
                'iot:DetachPolicy'
            ],
            'Resource': [
                'arn:aws:geo:us-east-1:{USER_ID}:map/{MAP_NAME}',
                'arn:aws:geo:us-east-1:{USER_ID}:tracker/{TRACKER_NAME}',
                'arn:aws:geo:us-east-1:{USER_ID}:geofence-collection/
{GEOFENCE_COLLECTION_NAME}',
                'arn:aws:iot:us-east-1:{USER_ID}:client/${cognito-
identity.amazonaws.com:sub}',
                'arn:aws:iot:us-east-1:{USER_ID}:topic/${cognito-
identity.amazonaws.com:sub}',
                'arn:aws:iot:us-east-1:{USER_ID}:topicfilter/${cognito-
identity.amazonaws.com:sub}/*',
                'arn:aws:iot:us-east-1:{USER_ID}:topic/${cognito-
identity.amazonaws.com:sub}/tracker'
            ],
```

```
            'Effect': 'Allow'
        },
        {
            'Condition': {
                'StringEquals': {
                    'cognito-identity.amazonaws.com:sub': '${cognito-
identity.amazonaws.com:sub}'
                }
            },
            'Action': [
                'iot:AttachPolicy',
                'iot:DetachPolicy',
                'iot:AttachPrincipalPolicy',
                'iot:DetachPrincipalPolicy'
            ],
            'Resource': [
                '*'
            ],
            'Effect': 'Allow'
        }
    ]
}
```

3. With the above policy changes, all necessary AWS resources are now configured appropriately for the application.

Set up the sample app code

1. Clone this repository: https://github.com/aws-geospatial/amazon-location-samples-android/tree/main/tracking-with-geofence-notifications to your local machine.
2. Open the `AmazonSampleSDKApp` project in Android Studio.
3. Build and run the app on your Android device or emulator.

Using the sample app

To use the sample follow these procedures:

- **Create a `custom.properties`**:

  To configure your `custom.properties` file, follow these steps:

1. Open your preferred text editor or IDE.
2. Create a new file.
3. Save the file with the name `custom.properties`.
4. Update the `custom.properties` with the following code sample, and replace the `MQTT_END_POINT`, `POLICY_NAME`, `GEOFENCE_COLLECTION_NAME`, and `TOPIC_TRACKER` with your resoucre names:

```
MQTT_END_POINT=YOUR_END_POINT.us-east-1.amazonaws.com
POLICY_NAME=YOUR_POLICY
GEOFENCE_COLLECTION_NAME=YOUR_GEOFENCE
TOPIC_TRACKER=YOUR_TRACKER
```

5. Clean and Rebuild the project. After this, you can run the project.

- **Sign In**:

  To sign in to the application, follow the below steps:

  1. Press the **Sign In** button.
  2. Provide an **Identity Pool Id**, **Tracker name**, and a **Map name**.
  3. Press **Sign In** again to finish.

- **Manage Filters**:

  Open the configuration screen, and perform the following:

  1. Toggle filters on or off using the switch UI.
  2. Update Time and Distance filters when needed.

- **Tracking Operations:**

  Open the tracking screen and perform the following:

  - You can start and stop tracking in foreground, background, or in battery-saver mode by pressing the respective buttons.

Sample tracking and geofencing application for iOS

This topic covers the iOS tutorial designed to demonstrate the key features of using the Amazon Location geofences and trackers in a mobile application. The applications demonstrate how a

tracker and geonfence interact using a combination of Lambda, AWS IoT and Amazon Location features.

Topics

- Create Amazon Location resources for your app
- Create a Geofence Collection
- Link a tracker to a geofence collection
- Using AWS Lambda with MQTT
- Setting up sample app code
- Using the sample app

Create Amazon Location resources for your app

To begin you will need to create the required Amazon Location resources. These resources will be essential for the functionality of the application and executing the provided code snippets.

> **Note**
>
> If you haven't created an AWS account, follow the instructions in the AWS account managment user guide.

To begin you will need to create a Amazon Cognito Identity Pool Id, use the following procedure:

1. Open the Amazon Cognito console and select **Identity pools** from the left side menu, then select **Create Identity pool**.
2. Make sure **Guest Access** is checked, and press **Next** to contiue.
3. Next create a new IAM role or Use an existing IAM role.
4. Enter an Identity pool name, and make sure Identity Pool has access to Amazon Location (geo)resources for the map and tracker you will be creating nin the next procedure.

Next you need to create and style a map in the AWS Amazon Location console, use the following procedure:

1. Navigate to the Maps section of the Amazon Location console and select **Create Map**.

2. Give the new map a **Name** and **Description**. Record the name you assign, as it is used later in the tutorial.
3. When choosing a map style, consider the map data provider. Refer to section 82 of the AWS service terms for more details.
4. Accept the Amazon Location Terms and Conditions, then select **Create Map**, to finish the map creation process.

Next you need to create a tracker in the Amazon Location console, use the following procedure:

1. Open the Maps section in the Amazon Location console.
2. Choose **Create tracker**.
3. Fill in the required fields. Make note of the tracker's **Name** as it will be refrenced throughout this tutoiral.
4. Under the **Position filtering** field, choose the option that best fits how you intend to use your tracker resource. If you do not set Position filtering, the default setting is `TimeBased`. For more information, see Start tracking, and PositionFiltering in the Amazon Location API Reference.
5. Choose **Create tracker** to finish creating the tracker.

Create a Geofence Collection

Now will you create a geofence collection. You can use either the console, API or CLI. The following procedures walk you through each option.

- Create a geofence collection using the Amazon Location console:
  1. Open the Geofence Collections section of the Amazon Location console.
  2. Choose **Create geofence collection**.
  3. Provide a name and description for the collection.
  4. Under the EventBridge rule with Amazon CloudWatch as a target, you can create an optional EventBridge rule to get started reacting to geofence events. This enables Amazon Location to publish events to Amazon CloudWatch Logs.
  5. Press the **Create geofence collection** to finish creating the collection.
- Create a geofence collection using the Amazon Location API:

Use the CreateGeofenceCollection operation from the Amazon Location Geofences APIs. The following example uses an API request to create a geofence collection called *GEOCOLLECTION_NAME*.

```
POST /geofencing/v0/collections
Content-type: application/json

{
   "CollectionName": "GEOCOLLECTION_NAME",
   "Description": "Geofence collection 1 for shopping center",
   "Tags": {
      "Tag1" : "Value1"
   }
}
```

- Create a geofence collection using AWS CLI commands:

  Use the `create-geofence-collection` command. The following example uses an AWS CLI to create a geofence collection called *GEOCOLLECTION_NAME*. For more information on using the AWS CLI, see the AWS Command Line Interface Documentation.

```
aws location \
  create-geofence-collection \
  --collection-name "ExampleGeofenceCollection" \
  --description "Shopping center geofence collection" \
  --tags Tag1=Value1
```

Link a tracker to a geofence collection

To link a tracker to a geofence collection you can use either the console, API, or CLI. The following procedures walk you through each option.

Link a tracker resource to a geofence collection using the Amazon Location Service console:

1. Open the Amazon Location console.
2. In the left navigation pane, choose **Trackers**.
3. Under **Device Trackers**, select the name link of the target tracker.
4. Under **Linked Geofence Collections**, choose **Link Geofence Collection**.

5. In the **Linked Geofence Collection window**, select a geofence collection from the dropdown menu.
6. Choose **Link**.
7. After you link the tracker resource, it will be assigned an Active status.

Link a tracker resource to a geofence collection using the Amazon Location APIs:

Use the `AsssociateTrackerConsumer` operation from the Amazon Location Trackers APIs. The following example uses an API request that associates ExampleTracker with a geofence collection using its Amazon Resource Name (ARN).

```
POST /tracking/v0/trackers/ExampleTracker/consumers
Content-type: application/json
    {
       "ConsumerArn": "arn:aws:geo:us-west-2:123456789012:geofence-
collection/GEOCOLLECTION_NAME"
    }
```

Llink a tracker resource to a geofence collection using AWS CLI commands:

Use the `associate-tracker-consumer` command. The following example uses an AWS CLI to create a geofence collection called *GEOCOLLECTION_NAME*.

```
aws location \
    associate-tracker-consumer \
        --consumer-arn "arn:aws:geo:us-west-2:123456789012:geofence-
collection/GEOCOLLECTION_NAME" \
        --tracker-name "ExampleTracker"
```

Using AWS Lambda with MQTT

In order to create a connection between AWS IoT and Amazon Location, you need a Lambda function to process messages forwarded by EventBridge CloudWatch events. This function will extract any positional data, format it for Amazon Location, and submit it through the Amazon Location Tracker API.

The following procedure shows you how to create this function through the Lambda console:

1. Open the console.
2. From the left navigation, choose **Functions**.

3. Then choose **Create Function**, and make sure that the **Author from scratch** option is selected.
4. provide a **Function name**, and for the **Runtime** option, choose Node.js 16.x.
5. Choose **Create function**.
6. Open the **Code tab** to access the editor.
7. Overwrite the placeholder code in the `index.js` file with the following:

```
            const AWS = require('aws-sdk')
            const iot = new AWS.Iot();
            exports.handler =  function(event) {
              console.log("event===>>>", JSON.stringify(event));
              var param = {
                endpointType: "iot:Data-ATS"
              };
              iot.describeEndpoint(param, function(err, data) {
                if (err) {
                  console.log("error===>>>", err, err.stack); // an error occurred
                } else {
                  var endp = data['endpointAddress'];
                  const iotdata = new AWS.IotData({endpoint: endp});
                  const trackerEvent = event["detail"]["EventType"];
                  const src = event["source"];
                  const time = event["time"];
                  const gfId = event["detail"]["GeofenceId"];
                  const resources = event["resources"][0];
                  const splitResources = resources.split(".");
                  const geofenceCollection = splitResources[splitResources.length -
1];
                  const coordinates = event["detail"]["Position"];

                  const deviceId = event["detail"]["DeviceId"];
                  console.log("deviceId===>>>", deviceId);
                  const msg =  {
                      "trackerEventType" : trackerEvent,
                      "source" : src,
                      "eventTime" : time,
                      "geofenceId" : gfId,
                      "coordinates": coordinates,
                      "geofenceCollection": geofenceCollection
                    };
                  const params = {
                    topic: `${deviceId}/tracker`,
```

```
                    payload: JSON.stringify(msg),
                    qos: 0
                };
                iotdata.publish(params, function(err, data) {
                    if (err) {
                      console.log("error===>>>", err, err.stack); // an error
occurred
                    } else {
                      console.log("Ladmbda triggered===>>>", trackerEvent);  //
successful response
                    }
                });
              }
            });
          }
```

8. Press the **Deploy** to save the updated function.
9. Next open the **Configuration** tab.
10. In the **Triggers** section, press the **Add Trigger** button.
11. Select **EventBridge (CloudWatch Events)** in Source field.
12. Select the **Existing Rules** option.
13. Enter the rule name, for example `AmazonLocationMonitor-GEOFENCECOLLECTION_NAME`.
14. Press the **Add** button.
15. This will also attach **Resource-based policy statements** in the permissions tab

Now you will set up the AWS IoT MQTT Test Client, use the following procedure:

1. Open the https://console.aws.amazon.com/iot/.
2. In the left navigation pane, select the **MQTT test client**.
3. You'll see a section titled **MQTT test client** where you can configure your MQTT connection.
4. After configuring the necessary settings, click on the **Connect** button to establish a connection to the MQTT broker using the provided parameters.
5. Record endpoint, as it is used later in the tutoiral.

   Once connected to the test client, you can subscribe to MQTT topics or publish messages to topics using the respective input fields provided in the MQTT test client interface. Next you will create an AWS IoT policy.

6. On the left side menu, under **Manage** expand **Security** option and click on **Policies**.
7. Click on **Create Policy** button.
8. Enter a policy name.
9. On **Policy Document** select **JSON** tab.
10. Copy paste the policy shown below, but make sure to update all elements with your *REGION* and *ACCOUNT_ID*:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:REGION:ACCOUNT_ID:client/${cognito-
identity.amazonaws.com:sub}",
        "arn:aws:iot:REGION:ACCOUNT_ID:topic/${cognito-
identity.amazonaws.com:sub}",
        "arn:aws:iot:REGION:ACCOUNT_ID:topicfilter/${cognito-
identity.amazonaws.com:sub}/*",
        "arn:aws:iot:REGION:ACCOUNT_ID:topic/${cognito-identity.amazonaws.com:sub}/
tracker"
      ],
      "Effect": "Allow"
    }
  ]
}
```

11. Select the **Create** button to finish.

Setting up sample app code

In order to setup the sample code you must have the following tools installed:

- Git
- XCode 15.3 or Later

- iOS Simulator 16 or later

Use this procedure to set up the sample app code:

1. Clone the git repository from this URL: https://github.com/aws-geospatial/amazon-location-samples-ios/tree/main/tracking-with-geofence-notifications.
2. Open the `AWSLocationSampleApp.xcodeproj` project file.
3. Wait for the package resolution process to finish.
4. On the project navigation menu rename `ConfigTemplate.xcconfig` to `Config.xcconfig` and fill in the following values:

```
IDENTITY_POOL_ID = `YOUR_IDENTITY_POOL_ID`
MAP_NAME = `YOUR_MAP_NAME`
TRACKER_NAME = `YOUR_TRACKER_NAME`
WEBSOCKET_URL = `YOUR_MQTT_TEST_CLIENT_ENDPOINT`
GEOFENCE_ARN = `YOUR_GEOFENCE_COLLECTION_NAME`
```

Using the sample app

After setting up the sample code you can now run the app on an iOS simulator or a physical device.

1. Build and run the app.
2. The app will ask you for location and notification permissions. You need to allow them.
3. Press the **Cognito Configuration** button.
4. Save the configuration.
5. You can now see the Filter options for time, distance and accuracy. Use them as per your need.
6. Go to **Tracking** tab in the app and you will see the map and **Start Tracking** button.
7. If you have installed the app on a simulator you may want to simulate location changes. This can be done in **Features** under the **Location** menu option. For example select **Features**, then **Location**, then **Freeway Drive**.
8. Press the **Start Tracking** button. You should see the tracking points on the map.
9. The app is also tracking the locations in the background. So, when you move the app in the background it will ask for your permission to continue tracking in background mode.
10. You can stop the tracking by tapping on **Stop Tracking** button.

Tagging your Amazon Location Service resources

Use resource tagging in Amazon Location to create tags to categorize your resources by purpose, owner, environment, or criteria. Tagging your resources helps you manage, identify, organize, search, and filter your resources.

For example, with AWS Resource Groups, you can create groups of AWS resources based on one or more tags or portions of tags. You can also create groups based on their occurrence in an AWS CloudFormation stack. Using Resource Groups and Tag Editor, you can consolidate and view data for applications that consist of multiple services, resources, and Regions in one place. For more information on Common Tagging Strategies, see the *AWS General Reference*.

Each tag is a label consisting of a key and value that you define:

- **Tag key** – A general label that categorizes the tag values. For example, `CostCenter`.
- **Tag value** – An optional description for the tag key category. For example, `MobileAssetTrackingResourcesProd`.

This topic helps you get started with tagging by reviewing tagging restrictions. It also shows you how to create tags and use tags to track your AWS cost for each active tag by using cost allocation reports.

Topics

- Tagging restrictions
- Grant permission to tag resources
- Add a tag to an Amazon Location Service resource
- Track resource cost by tag
- Control access to Amazon Location Service resources using tags
- Learn more

Tagging restrictions

The following basic restrictions apply to tags:

- Maximum tags per resource – 50
- For each resource, each tag key must be unique, and each tag key can have only one value.

> **Note**
>
> If you add a new tag with the same tag key as an existing tag, the new tag overwrites the existing tag.

- Maximum key length – 128 Unicode characters in UTF-8
- Maximum value length – 256 Unicode characters in UTF-8
- The allowed characters across services are: letters, numbers, and spaces representable in UTF-8, and the following characters: + - = . _ : / @.
- Tag keys and values are case-sensitive.
- The `aws:` prefix is reserved for AWS use. If a tag has a tag key with this prefix, then you can't edit or delete the tag's key or value. Tags with the `aws:` prefix don't count against your tags per resource limit.

Grant permission to tag resources

You can use IAM policies to control access to your Amazon Location resources and grant permission to tag a resource on creation. In addition to granting permission to create resources, the policy can include `Action` permissions to allow tagging operations:

- `geo:TagResource` – Allows a user to assign one or more tags to a specified Amazon Location resource.

- `geo:UntagResource` – Allows a user to remove one or more tags from a specified Amazon Location resource.
- `geo:ListTagsForResource` – Allows a user to list all the tags assigned to an Amazon Location resource.

The following is a policy example to allow a user to create a geofence collection and tag resources:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowTaggingForGeofenceCollectionOnCreation",
```

```
            "Effect": "Allow",
            "Action": [
                "geo:CreateGeofenceCollection",
                "geo:TagResource"
            ],
            "Resource": "arn:aws:geo:region:accountID:geofence-collection/*"
    ]
}
```

Add a tag to an Amazon Location Service resource

You can add tags when creating your resources using the Amazon Location console, the AWS CLI, or the Amazon Location APIs:

- Create a map resource
- Create a place index resource
- Create a route calculator resource
- Create a geofence collection
- Create a tracker resource

To tag existing resources, edit or delete tags

1. Open the Amazon Location console at https://console.aws.amazon.com/location/.
2. In the left navigation pane, choose the resource you want to tag. For example, **Maps**.
3. Choose a resource from the list.
4. Choose **Manage tags** to add, edit, or delete your tags.

Track resource cost by tag

You can use tags for cost allocation to track your AWS cost in detail. After you activate the cost allocation tags, AWS uses the cost allocation tags to organize your resource billing on your cost allocation report. This helps you categorize and track your usage costs.

There are two types of cost allocation tags you can activate:

- AWS-generated – These tags are generated by AWS. AWS tags use the `aws:` prefix, for example, `aws:createdBy`.

- User-defined – These are custom tags that you create. The user-defined tags use the `user:` prefix, for example, `user:CostCenter`.

You must activate each tag type individually. After your tags are activated, you can enable AWS Cost Explorer or view your monthly cost allocation report.

AWS-generated tags

To activate AWS-generated tags

1. Open the Billing and Cost Management console at https://console.aws.amazon.com/billing/.
2. In the left navigation pane, choose **Cost Allocation Tags**.
3. Under the **AWS-Generated Cost Allocation Tags** tab, select the tag keys that you want to activate.
4. Choose **Activate**.

User-defined tags

To activate user-defined tags

1. Open the Billing and Cost Management console at https://console.aws.amazon.com/billing/.
2. In the left navigation pane, choose **Cost Allocation Tags**.
3. Under the **User-Defined Cost Allocation Tags** tab, select the tag keys you want to activate.
4. Choose **Activate**.

After you activate your tags, AWS generates a monthly Cost Allocation Report for your resource usage and cost. This cost allocation report includes all of your AWS costs for each billing period, including tagged and untagged resources. For more information, see Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

Control access to Amazon Location Service resources using tags

AWS Identity and Access Management (IAM) policies support tag-based conditions, which enables you to manage authorization for your resources based on specific tags key and values. For

example, an IAM role policy can include conditions to limit access to specific environments, such as development, test, or production, based on tags.

For more information, see the topic on control resource access based on tags.

Learn more

For more information about:

- Tagging best practices, see Tagging AWS resources in the *AWS General Reference*.
- Using tags to control access to AWS resources, see Controlling access to AWS resources using tags in the *AWS Identity and Access Management User Guide*.

Granting access to Amazon Location Service

To use Amazon Location Service, a user must be granted access to the resources and APIs that make up Amazon Location. There are three strategies you can use to grant access to your resources.

- **Use IAM** – To grant access to users authenticated with AWS IAM Identity Center or AWS Identity and Access Management (IAM), create an IAM policy that allows access to the resources that you want. For more information about IAM and Amazon Location, see Identity and Access Management for Amazon Location Service.
- **Use API keys** – To grant access to unauthenticated users, you can create API Keys that give read-only access to your Amazon Location Service resources. This is useful in a case where you do not want to authenticate every user. For example, a web application. For more information about API keys, see Allowing unauthenticated guest access to your application using API keys.
- **Use Amazon Cognito** – An alternative to API keys is to use Amazon Cognito to grant anonymous access. Amazon Cognito allows you to create a richer authorization with policy to define what can be done by the unauthenticated users. For more information about using Amazon Cognito, see Allowing unauthenticated guest access to your application using Amazon Cognito.

> **Note**
>
> You can also use Amazon Cognito to use your own authentication process or to combine multiple authentication methods, using Amazon Cognito Federated Identities. For more information, see Getting Started with Federated Identities in the *Amazon Cognito Developer Guide*.

Topics

- Allowing unauthenticated guest access to your application using API keys
- Allowing unauthenticated guest access to your application using Amazon Cognito

Allowing unauthenticated guest access to your application using API keys

When you call Amazon Location Service APIs in your applications, you typically make this call as an *authenticated user* who is authorized to make the API calls. However, there are some cases where you do not want to authenticate every user of your application. For example, you might want a web application that shows your business location to be available to anyone using the website, whether they are logged in or not. In this case, one alternative is to use API keys to make the API calls.

API keys are a key value that is associated with specific Amazon Location Service resources in your AWS account, and specific actions that you can perform on those resources. You can use an API key in your application to make unauthenticated calls to the Amazon Location APIs for those resources. For example, if you associate an API key with the map resource *myMap*, and the `GetMap*` actions, then an application that uses that API key will be able to view maps created with that resource, and your account will be charged as any other usage from your account. That same API key would not give permissions to change or update the map resource—only using the resource is allowed.

> **Note**
>
> API keys are available to use only with **map**, **place**, and **route** resources, and you cannot modify or create those resources. If your application needs access to other resources or actions for unauthenticated users, you can use Amazon Cognito to provide access along with, or instead of, API keys. For more information, see Allowing unauthenticated guest access to your application using Amazon Cognito.

API keys include a plaintext *value* that gives access to one or more resources in your AWS account. If someone copies your API key, they can access those same resources. To avoid this, you can specify the domains where the API key can be used when you create the key. These domains are called referers. If needed, you can also create short term API Keys by setting expiration times for your API Keys.

Topics

- API keys compared to Amazon Cognito
- Creating API keys
- Using an API key to call an Amazon Location API
- Using an API key to render a map
- Managing API key lifetimes

API keys compared to Amazon Cognito

API keys and Amazon Cognito are used in similar ways for similar scenarios, so why would you use one over the other? The following list highlights some of the differences between the two.

- API keys are available only for map, place, and route resources, and only for certain actions. Amazon Cognito can be used to authenticate access to most Amazon Location Service APIs.
- The performance of map requests with API keys is typically faster than similar scenarios with Amazon Cognito. Simpler authentication means fewer round trips to the service and cached requests when getting the same map tile again in a short time period.
- With Amazon Cognito, you can use your own authentication process or combine multiple authentication methods, using Amazon Cognito Federated Identities. For more information, see Getting Started with Federated Identities in the *Amazon Cognito Developer Guide*.

Creating API keys

You can create an API key, and associate it with one or more resources in your AWS account.

You can create an API key using the Amazon Location Service console, the AWS CLI, or the Amazon Location APIs.

Console

To create an API key using the Amazon Location Service console

1. In the **Amazon Location console**, choose **API keys** from the left menu.
2. On the **API keys** page, choose **Create API key**.
3. On the **Create API key** page, fill in the following information:

- **Name** – A name for your API key, such as `MyWebAppKey`.
- **Description** – An optional description for your API key.
- **Resources** – Choose the Amazon Location resources to give access to with this API key from the dropdown. You can add more than one resource by choosing **Add resource**.
- **Actions** – Specify the actions you want to authorize with this API key. You must select at least one action to match each resource type you have selected. For example, if you selected a place resource, you must select at least one of the choices under **Places Actions**.
- **Expiration time** – Optionally, add an expiration date and time for your API key. For more information, see Managing API key lifetimes.
- **Referers** – Optionally, add one or more domains where you can use the API key. For example, if the API key is to allow an application running on the website `example.com`, then you could put `*.example.com/` as an allowed referer.
- **Tags** – Optionally, add tags to the API key.

4. Choose **Create API key** to create the API key.
5. On the detail page for the API key, you can see information about the API key that you have created. Choose **Show API key** to see the key value that you use when calling Amazon Location APIs. The key value will have the format `v1.public.a1b2c3d4...`. For more information about using the API key to render maps, see Using an API key to render a map.

API

To create an API key using the Amazon Location APIs

Use the `CreateKey` operation from the Amazon Location APIs.

The following example is an API request to create an API key called *ExampleKey* with no expiration date, and access to a single map resource.

```
POST /metadata/v0/keys HTTP/1.1
Content-type: application/json

{
   "KeyName": "ExampleKey"
   "Restrictions": {
      "AllowActions": [
```

```
            "geo:GetMap*"
        ],
        "AllowResources": [
            "arn:aws:geo:region:map/mapname"
        ]
    },
    "NoExpiry": true
    }
}
```

The response includes the API key value to use when accessing resources in your applications. The key value will have the format `v1.public.a1b2c3d4...`. To learn more about using the API key to render maps, see Using an API key to render a map.

You can also use the `DescribeKey` API to find the key value for a key at a later time.

AWS CLI

To create an API key using AWS CLI commands

Use the `create-key` command.

The following example creates an API key called *ExampleKey* with no expiration date, and access to a single map resource.

```
aws location \
  create-key \
  --key-name ExampleKey \
  --restrictions '{"AllowActions":["geo:GetMap*"],"AllowResources":
["arn:aws:geo:region:map/mapname"]}' \
  --no-expiry
```

The response includes the API key value to use when accessing resources in your applications. The key value will have the format `v1.public.a1b2c3d4...`. To learn more about using the API key to render maps, see Using an API key to render a map. The response to `create-key` looks like the following.

```
{
    "Key": "v1.public.a1b2c3d4...",
    "KeyArn": "arn:aws:geo:region:accountId:api-key/ExampleKey",
    "KeyName": "ExampleKey",
```

```
    "CreateTime": "2023-02-06T22:33:15.693Z"
}
```

You can also use `describe-key` to find the key value at a later time. The following example shows how to call `describe-key` on an API key named *`ExampleKey`*.

```
aws location describe-key \
    --key-name ExampleKey
```

Using an API key to call an Amazon Location API

After you create an API key, you can use the key value to make calls to Amazon Location APIs in your application.

The APIs that support API keys have an additional parameter that takes the API key value. For example, if you call the `GetPlace` API, you can fill in the key parameter, as follows

```
GET /places/v0/indexes/IndexName/places/PlaceId?key=KeyValue
```

If you fill in this value, you do not need to authenticate the API call with AWS Sig v4 as you normally would.

For JavaScript developers, you can use the Amazon Location JavaScript Authentication helper to help with authenticating API operations with API keys.

For mobile developers, you can use the following Amazon Location mobile authentication SDKs:

- Amazon Location Service Mobile Authentication SDK for iOS
- Amazon Location Service Mobile Authentication SDK for Android

For AWS CLI users, when you use the `--key` parameter, you should also use the `--no-sign-request` parameter, to avoid signing with Sig v4.

> **Note**
>
> If you include both a `key` and and an AWS Sig v4 signature in a call to Amazon Location Service, only the API key is used.

Using an API key to render a map

You can use the API key value to render a map in your application using MapLibre. This is a little bit different than using the API keys in other Amazon Location APIs that you are calling directly, because MapLibre makes those calls for you.

The following sample code shows using the API key to render a map in a simple webpage by using the MapLibre GL JS map control. For this code to work properly, replace the *v1.public.your-api-key-value*, *us-east-1*, and *ExampleMap* strings with values that match your AWS account.

```
<!-- index.html -->
<html>
  <head>
    <link href="https://unpkg.com/maplibre-gl@1.14.0/dist/maplibre-gl.css"
 rel="stylesheet" />
    <style>
      body { margin: 0; }
      #map { height: 100vh; }
    </style>
  </head>
  <body>
    <!-- Map container -->
    <div id="map" />
    <!-- JavaScript dependencies -->
    <script src="https://unpkg.com/maplibre-gl@1.14.0/dist/maplibre-gl.js"></script>
    <script>
      const apiKey = "v1.public.your-api-key-value"; // API key
      const region = "us-east-1"; // Region
      const mapName = "ExampleMap"; // Map name
      // URL for style descriptor
      const styleUrl = `https://maps.geo.${region}.amazonaws.com/maps/v0/maps/
${mapName}/style-descriptor?key=${apiKey}`;
      // Initialize the map
      const map = new maplibregl.Map({
        container: "map",
        style: styleUrl,
        center: [-123.1187, 49.2819],
        zoom: 11,
      });
      map.addControl(new maplibregl.NavigationControl(), "top-left");
    </script>
  </body>
```

```
</html>
```

Managing API key lifetimes

You can create API keys that work indefinitely. However, if you want to create a temporary API key, rotate API keys on a regular basis, or revoke an existing API key, you can use *API key expiration*.

When creating a new API key, or updating an existing one, you can set the expiration time for that API key.

- When an API key reaches its expiration time, the key is automatically deactivated. Inactive keys can no longer be used to make maps requests.
- You can delete an API key 90 days after deactivating it.
- If you have an inactive key that you haven't yet deleted, you can restore it by updating the expiration time to a future time.
- To create a permanent key, you can remove the expiration time.
- If you attempt to deactivate an API key that has been used within the last 7 days, you'll be prompted to confirm that you want to make the change. If you are using the Amazon Location Service API, or the AWS CLI, you will receive an error, unless you set `ForceUpdate` parameter to true.

Allowing unauthenticated guest access to your application using Amazon Cognito

You can use Amazon Cognito authentication as an alternative to directly using AWS Identity and Access Management (IAM) with both frontend SDKs and direct HTTPS requests.

You may want to use this form of authentication for the following reasons:

- **Unauthenticated users** – If you have a website with anonymous users, you can use Amazon Cognito identity pools. For more information, see the section on the section called “Using Amazon Cognito”.
- **Your own authentication** – If you would like to use your own authentication process, or combine multiple authentication methods, you can use Amazon Cognito Federated Identities. For more information, see Getting Started with Federated Identities in the *Amazon Cognito Developer Guide*.

Amazon Cognito provides authentication, authorization, and user management for web and mobile apps. You can use Amazon Cognito unauthenticated identity pools with Amazon Location as a way for applications to retrieve temporary, scoped-down AWS credentials.

For more information, see Getting Started with User Pools in the *Amazon Cognito Developer Guide*.

> **Note**
>
> For mobile developers, Amazon Location provides mobile authentication SDKs for both iOS and Android, see the following github repositories for more information:
>
> - Amazon Location Service Mobile Authentication SDK for iOS
> - Amazon Location Service Mobile Authentication SDK for Android

Create an Amazon Cognito identity pool

You can create Amazon Cognito identity pools to allow unauthenticated guest access to your application through the Amazon Cognito console, the AWS CLI, or the Amazon Cognito APIs.

> **Important**
>
> The pool that you create must be in the same AWS account and AWS Region as the Amazon Location Service resources that you're using.

You can use IAM policies associated with unauthenticated identity roles with the following actions:

- `geo:GetMap*`
- `geo:SearchPlaceIndex*`
- `geo:GetPlace`
- `geo:CalculateRoute*`
- `geo:GetGeofence`
- `geo:ListGeofences`
- `geo:PutGeofence`
- `geo:BatchDeleteGeofence`
- `geo:BatchPutGeofence`

- `geo:BatchEvaluateGeofences`
- `geo:GetDevicePosition*`
- `geo:ListDevicePositions`
- `geo:BatchDeleteDevicePositionHistory`
- `geo:BatchGetDevicePosition`
- `geo:BatchUpdateDevicePosition`

Including other Amazon Location actions will have no effect, and unauthenticated identities will be unable to call them.

Example

To create an identity pool using the Amazon Cognito console

1. Go to the Amazon Cognito console.
2. Choose **Manage Identity Pools.**
3. Choose **Create new identity pool**, then enter a name for your identity pool.
4. From the **Unauthenticated identities** collapsible section, choose **Enable access to unauthenticated identities**.
5. Choose **Create Pool**.
6. Choose which IAM roles you want to use with your identity pool.
7. Expand **View Details**.
8. Under **Unauthenticated identities**, enter a role name.
9. Expand the **View Policy Document** section, then choose **Edit** to add your policy.
10. Add your policy to grant access to your resources.

    The following are policy examples for Maps, Places, Trackers, and Routes. To use the examples for your own policy, replace the *`region`* and *`accountID`* placeholders:

    Maps policy example

    The following policy grants read-only access to a map resource named *`ExampleMap`*.

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Sid": "MapsReadOnly",
      "Effect": "Allow",
      "Action": [
        "geo:GetMapStyleDescriptor",
        "geo:GetMapGlyphs",
        "geo:GetMapSprites",
        "geo:GetMapTile"
      ],
      "Resource": "arn:aws:geo:region:accountID:map/ExampleMap"
    }
  ]
}
```

Adding an IAM condition that matches `aws:referer` lets you limit browser access to your resources to a list of URLs or URL prefixes. The following example allows access to a map resource named `RasterEsriImagery` from only the website `example.com`:

> **Warning**
>
> While `aws:referer` can limit access, it is not a secuirty mechanism. It is dangerous to include a publicly known referer header value. Unauthorized parties can use modified or custom browsers to provide any `aws:referer` value that they choose. As a result, `aws:referer` should not be used to prevent unauthorized parties from making direct AWS requests. It is offered only to allow customers to protect their digital content, such as content stored in Amazon S3, from being referenced on unauthorized third-party sites. For more infromation, see AWS:referer.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "geo:GetMap*",
            "Resource": "arn:aws:geo:us-west-2:111122223333:map/
RasterEsriImagery",
            "Condition": {
                "StringLike": {
                    "aws:referer": [
```

```
                        "https://example.com/*",
                        "https://www.example.com/*"
                    ]
                }
            }
        }
    ]
}
```

If you're using Tangram to display a map, it doesn't use the style descriptors, glyphs, or sprites returned by the Maps API. Instead, it's configured by pointing to a .zip file that contains style rules and necessary assets. The following policy grants read-only access to a map resource named *ExampleMap* for the `GetMapTile` operation.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "MapsReadOnly",
      "Effect": "Allow",
      "Action": [
        "geo:GetMapTile"
      ],
      "Resource": "arn:aws:geo:region:accountID:map/ExampleMap"
    }
  ]
}
```

Places policy example

The following policy grants read-only access to a place index resource named *ExamplePlaceIndex* to search for places by text or positions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PlacesReadOnly",
      "Effect": "Allow",
      "Action": [
          "geo:SearchPlaceIndex*",
          "geo:GetPlace"
```

```
      ],
      "Resource": "arn:aws:geo:region:accountID:place-index/ExamplePlaceIndex"
    }
  ]
}
```

Adding an IAM condition that matches `aws:referer` lets you limit browser access to your resources to a list of URLs or URL prefixes. The following example denies access to a place index resource named *`ExamplePlaceIndex`* from all referring websites, except `example.com`.

> **Warning**
>
> While `aws:referer` can limit access, it is not a secuirty mechanism. It is dangerous to include a publicly known referer header value. Unauthorized parties can use modified or custom browsers to provide any `aws:referer` value that they choose. As a result, `aws:referer` should not be used to prevent unauthorized parties from making direct AWS requests. It is offered only to allow customers to protect their digital content, such as content stored in Amazon S3, from being referenced on unauthorized third-party sites. For more infromation, see AWS:referer.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "geo:*",
            "Resource": "arn:aws:geo:us-west-2:111122223333:place-
index/ExamplePlaceIndex",
            "Condition": {
                "StringLike": {
                    "aws:referer": [
                        "https://example.com/*",
                        "https://www.example.com/*"
                    ]
                }
            }
        }
    ]
```

```
}
```

Trackers policy example

The following policy grants access to a tracker resource named *ExampleTracker* to update device positions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "UpdateDevicePosition",
      "Effect": "Allow",
      "Action": [
        "geo:BatchUpdateDevicePosition"
      ],
      "Resource": "arn:aws:geo:region:accountID:tracker/ExampleTracker"
    }
  ]
}
```

Adding an IAM condition that matches `aws:referer` lets you limit browser access to your resources to a list of URLs or URL prefixes. The following example denies access to a tracker resource named *ExampleTracker* from all referring websites, except `example.com`.

> **Warning**
>
> While `aws:referer` can limit access, it is not a secuirty mechanism. It is dangerous to include a publicly known referer header value. Unauthorized parties can use modified or custom browsers to provide any `aws:referer` value that they choose. As a result, `aws:referer` should not be used to prevent unauthorized parties from making direct AWS requests. It is offered only to allow customers to protect their digital content, such as content stored in Amazon S3, from being referenced on unauthorized third-party sites. For more infromation, see AWS:referer.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": "geo:GetDevice*",
            "Resource": "arn:aws:geo:us-
west-2:111122223333:tracker/ExampleTracker",
            "Condition": {
                "StringLike": {
                    "aws:referer": [
                        "https://example.com/*",
                        "https://www.example.com/*"
                    ]
                }
            }
        }
    ]
}
```

Routes policy example

The following policy grants access to a route calculator resource named *ExampleCalculator* to calculate a route.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RoutesReadOnly",
      "Effect": "Allow",
      "Action": [
        "geo:CalculateRoute"
      ],
      "Resource": "arn:aws:geo:region:accountID:route-
calculator/ExampleCalculator"
    }
  ]
}
```

Adding an IAM condition that matches `aws:referer` lets you limit browser access to your resources to a list of URLs or URL prefixes. The following example denies access to a route calculator named *ExampleCalculator* from all referring websites, except `example.com`.

Warning

While `aws:referer` can limit access, it is not a secuirty mechanism. It is dangerous to include a publicly known referer header value. Unauthorized parties can use modified or custom browsers to provide any `aws:referer` value that they choose. As a result, `aws:referer` should not be used to prevent unauthorized parties from making direct AWS requests. It is offered only to allow customers to protect their digital content, such as content stored in Amazon S3, from being referenced on unauthorized third-party sites. For more infromation, see AWS:referer.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "geo:*",
            "Resource": "arn:aws:geo:us-west-2:111122223333:route-
calculator/ExampleCalculator",
            "Condition": {
                "StringLike": {
                    "aws:referer": [
                        "https://example.com/*",
                        "https://www.example.com/*"
                    ]
                }
            }
        }
    ]
}
```

Note

While unauthenticated identity pools are intended for exposure on unsecured internet sites, note that they will be exchanged for standard, time-limited AWS credentials. It's important to scope the IAM roles associated with unauthenticated identity pools appropriately.

11. Choose **Allow** to create your identity pools.

The resulting identity pool follows the syntax `<region>:<GUID>`.

For example:

```
us-east-1:1sample4-5678-90ef-aaaa-1234abcd56ef
```

For more policy examples specific to Amazon Location, see the section called “Identity-based policy examples”.

Using the Amazon Cognito identity pools in JavaScript

The following example exchanges the unauthenticated identity pool that you've created for credentials that are then used to fetch the style descriptor for your map resource *ExampleMap*.

```
const AWS = require("aws-sdk");

const credentials = new AWS.CognitoIdentityCredentials({
   IdentityPoolId: "<identity pool ID>" // for example, us-east-1:1sample4-5678-90ef-
aaaa-1234abcd56ef
});

const client = new AWS.Location({
   credentials,
   region: AWS.config.region || "<region>"
});

console.log(await client.getMapStyleDescriptor("ExampleMap").promise());
```

> **Note**
>
> Retrieved credentials from unauthenticated identities are valid for **one hour**.

The following is an example of a function that automatically renews credentials before they expire.

```
async function refreshCredentials() {
  await credentials.refreshPromise();
  // schedule the next credential refresh when they're about to expire
  setTimeout(refreshCredentials, credentials.expireTime - new Date());
```

```
}
```

To simplify this work, you can use the Amazon Location JavaScript Authentication helper. This is in place of both getting the credentials, and refreshing them. This example uses the AWS SDK for JavaScript v3.

```
import { LocationClient, GetMapStyleDescriptorCommand } from "@aws-sdk/client-
location";
import { withIdentityPoolId } from "@aws/amazon-location-utilities-auth-helper";

const identityPoolId = "<identity pool ID>"; // for example, us-
east-1:1sample4-5678-90ef-aaaa-1234abcd56ef

// Create an authentication helper instance using credentials from Cognito
const authHelper = await withIdentityPoolId(identityPoolId);

const client = new LocationClient({
  region: "<region>", // The region containing both the identity pool and tracker
 resource
  ...authHelper.getLocationClientConfig(), // Provides configuration required to make
 requests to Amazon Location
});

const input = {
  MapName: "ExampleMap",
};

const command = new GetMapStyleDescriptorCommand(input);

console.log(await client.send(command));
```

Next steps

- To modify your roles, go to the IAM console.
- To manage your identity pools, go to the Amazon Cognito console.

Monitoring Amazon Location Service

When using Amazon Location Service, you can monitor your usage and resources over time by using:

- **Amazon CloudWatch**. Monitors your Amazon Location Service resources, and provides metrics with statistics in near-real time.
- **AWS CloudTrail**. Provides event tracking of all calls to Amazon Location Service APIs.

This section provides information about using these services.

Topics

- Monitoring Amazon Location Service with Amazon CloudWatch
- Logging and monitoring with AWS CloudTrail

Monitoring Amazon Location Service with Amazon CloudWatch

Amazon CloudWatch monitors your AWS resources and the applications that you run on AWS in near-real time. You can monitor Amazon Location resources using CloudWatch, which collects raw data and processes metrics into meaningful statistics in near-real time. You can view historical information for up to 15 months, or search metrics to view in the Amazon CloudWatch console for more perspective about your Amazon Location resources. You can also set alarms by defining thresholds, and send notifications or take actions when those thresholds are met.

For more information, see the Amazon CloudWatch User Guide

Topics

- Amazon Location Service metrics exported to Amazon CloudWatch
- View Amazon Location Service metrics
- Create CloudWatch alarms for Amazon Location Service metrics
- Using CloudWatch to monitor usage against quotas
- CloudWatch metric examples for Amazon Location Service

Amazon Location Service metrics exported to Amazon CloudWatch

Metrics are time-ordered data points that are exported to CloudWatch. A dimension is a name/value pair that identifies the metric. For more information, see Using CloudWatch metrics and CloudWatch dimensions in the Amazon CloudWatch User Guide.

The following are metrics that Amazon Location Service exports to CloudWatch in the `AWS/Location` namespace.

Metric	Description
`CallCount`	The number of calls made to a given API endpoint. Valid Dimensions: Amazon Location Service API names Valid Statistic: Sum Units: Count
`ErrorCount`	The number of error responses from calls made to a given API endpoint. Valid Dimensions: Amazon Location Service API names Valid Statistic: Sum Units: Count
`SuccessCount`	The number of successful calls made to a given API endpoint. Valid Dimensions: Amazon Location Service API names Valid Statistic: Sum Units: Count
`CallLatency`	The amount of time the operation takes to process and return a response when a call is made to a given API endpoint. Valid Dimensions: Amazon Location Service API names Valid Statistic: Average Units: Milliseconds

View Amazon Location Service metrics

You can view metrics for Amazon Location Service on the Amazon CloudWatch console or by using the Amazon CloudWatch API.

To view metrics using the CloudWatch console

Example

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. On the **All metrics tab**, choose the Amazon Location namespace.
4. Select the type of metric to view.
5. Select the metric and add it to the chart.

For more information, see View Available Metrics in the *Amazon CloudWatch User Guide*.

Create CloudWatch alarms for Amazon Location Service metrics

You can use CloudWatch to set alarms on your Amazon Location Service metrics. For example, you can create an alarm in CloudWatch to send an email whenever an error count spike occurs.

The following topics give you a high-level overview of how to set alarms using CloudWatch. For detailed instructions, see Using Alarms in the *Amazon CloudWatch User Guide*.

To set alarms using the CloudWatch console

Example

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarm**.
3. Choose **Create Alarm**.
4. Choose **Select metric**.
5. On the **All metrics** tab, select the Amazon Location namespace.
6. Select a metric category.
7. Find the row with the metric you want to create an alarm for, then select the check box next to this row.
8. Choose **Select metric**.
9. Under **Metric**, fill in the values .
10. Specify the alarm **Conditions** .
11. Choose **Next**.

12. If you want to send a notification when the alarm conditions are met:
    - Under **Alarm state trigger**, select the alarm state to prompt a notification to be sent.
    - Under **Select an SNS topic**, choose **Create new topic** to create a new Amazon Simple Notification Service (Amazon SNS) topic. Enter the topic name and the email to send the notification to.
    - Under **Send a notification to** enter additional email addresses to send the notification to.
    - Choose **Add notification**. This list is saved and appears in the field for future alarms.
13. When done, choose **Next**.
14. Enter a name and description for the alarm, then choose **Next**.
15. Confirm the alarm details, then choose **Next**.

> **Note**
>
> When creating a new Amazon SNS topic, you must verify the email address before a notification can be sent. If the email is not verified, the notification will not be received when an alarm is initiated by a state change.

For more information about how to set alarms using the CloudWatch console, see Create an Alarm that Sends Email in the *Amazon CloudWatch User Guide*.

Using CloudWatch to monitor usage against quotas

You can create Amazon CloudWatch alarms to notify you when your utilization of a given quota exceeds a configurable threshold. This enables you to recognize when you are close to your quota limits, and either adapt your utilization to avoid cost overruns, or request a quota increase, if needed. For information about how to use CloudWatch to monitor quotas, see Visualizing your service quotas and setting alarms in the *Amazon CloudWatch User Guide*.

CloudWatch metric examples for Amazon Location Service

You can use the GetMetricData API to retrieve metrics for Amazon Location.

- For example, you can monitor `CallCount` and set an alarm for when a drop in number occurs.

  Monitoring the `CallCount` metrics for `SendDeviceLocation` can help give you perspective on tracked assets. If the `CallCount` drops, it means that tracked assets, such as a fleet of trucks,

have stopped sending their current locations. Setting an alarm for this can help notify you an issue has occurred.

- For another example, you can monitor `ErrorCount` and set an alarm for when a spike in number occurs.

  Trackers must be associated with geofence collections in order for device locations to be evaluated against geofences. If you have a device fleet that requires continuous location updates, seeing the `CallCount` for `BatchEvaluateGeofence` or `BatchPutDevicePosition` drop to zero indicates that updates are no longer flowing.

The following is an example output for GetMetricData with the metrics for `CallCount` and `ErrorCount` for creating map resources.

```
{
  "StartTime": 1518867432,
  "EndTime": 1518868032,
  "MetricDataQueries": [
    {
      "Id": "m1",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/Location",
          "MetricName": "CallCount",
          "Dimensions": [
            {
              "Name": "SendDeviceLocation",
              "Value": "100"
            }
          ]
        },
        "Period": 300,
        "Stat": "SampleCount",
        "Unit": "Count"
      }
    },
    {
      "Id": "m2",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/Location",
          "MetricName": "ErrorCount",
```

```
          "Dimensions": [
            {
              "Name": "AssociateTrackerConsumer",
              "Value": "0"
            }
          ]
        },
        "Period": 1,
        "Stat": "SampleCount",
        "Unit": "Count"
      }
    }
  ]
}
```

Logging and monitoring with AWS CloudTrail

AWS CloudTrail is a service that provides a record of actions taken by a user, role, or an AWS service. CloudTrail records all API calls as events. You can use Amazon Location Service with CloudTrail to monitor your API calls, which include calls from the Amazon Location Service console and AWS SDK calls to the Amazon Location Service API operations.

When you create a trail, you can enable continuous delivery of CloudTrail events to an S3 bucket, including events for Amazon Location Service. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon Location Service, the IP address from which the request was made, who made the request, when it was made, and additional details.

For more information about CloudTrail, see the AWS CloudTrail User Guide.

Topics

- Amazon Location Service Information in CloudTrail
- Understanding Amazon Location Service Log File Entries

Amazon Location Service Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon Location Service, that activity is recorded in a CloudTrail event along with other AWS

service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Amazon Location Service, create a trail. A *trail* enables CloudTrail to deliver log files to an S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs.

For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All Amazon Location Service actions are logged by CloudTrail and are documented in the Amazon Location Service API references. For example, calls to the `CreateTracker`, `UpdateTracker` and `DescribeTracker` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine whether the request was made:

- With root or AWS Identity and Access Management (IAM) user credentials.
- With temporary security credentials for a role or federated user.
- By another AWS service.

For more information, see the CloudTrail userIdentity Element.

Understanding Amazon Location Service Log File Entries

A trail is a configuration that enables delivery of events as log files to an S3 bucket that you specify, or to Amazon CloudWatch Logs. For more information, see Working with CloudTrail log files in the *AWS CloudTrail User Guide*.

CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested operation, the date and time of the operation, request parameters, and so on.

> **Note**
>
> CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order. To determine the order of operations, use `eventTime`.

The following example shows a CloudTrail log entry that demonstrates the `CreateTracker` operation, which creates a tracker resource.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "123456789012",
        "arn": "arn:aws:geo:us-east-1:123456789012:tracker/ExampleTracker"
        "accountId": "123456789012",
        "accessKeyId": "123456789012",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "123456789012",
                "arn": "arn:aws:geo:us-east-1:123456789012:tracker/ExampleTracker",
                "accountId": "123456789012",
                "userName": "exampleUser",
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-10-22T16:36:07Z"
            }
        }
    },
    "eventTime": "2020-10-22T17:43:30Z",
    "eventSource": "geo.amazonaws.com",
    "eventName": "CreateTracker",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0/24—TEST-NET-1",
```

```
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.864
 Linux/4.14.193-110.317.amzn2.x86_64 OpenJDK_64-Bit_Server_VM/11.0.8+10-LTS java/11.0.8
 kotlin/1.3.72 vendor/Amazon.com_Inc. exec-env/AWS_Lambda_java11",
    "requestParameters": {
        "TrackerName": "ExampleTracker",
        "Description": "Resource description"
    },
    "responseElements": {
        "TrackerName": "ExampleTracker",
        "Description": "Resource description"
        "TrackerArn": "arn:partition:service:region:account-id:resource-id",
        "CreateTime": "2020-10-22T17:43:30.521Z"
    },
    "requestID": "557ec619-0674-429d-8e2c-eba0d3f34413",
    "eventID": "3192bc9c-3d3d-4976-bbef-ac590fa34f2c",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012",
}
```

The following shows a log entry for the `DescribeTracker` operation, which returns the details of a tracker resource.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "123456789012",
        "arn": "arn:partition:service:region:account-id:resource-id",
        "accountId": "123456789012",
        "accessKeyId": "123456789012",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "123456789012",
                "arn": "arn:partition:service:region:account-id:resource-id",
                "accountId": "123456789012",
                "userName": "exampleUser",
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-10-22T16:36:07Z"
```

```
            }
        }
    },
    "eventTime": "2020-10-22T17:43:33Z",
    "eventSource": "geo.amazonaws.com",
    "eventName": "DescribeTracker",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0/24–TEST-NET-1",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.864
 Linux/4.14.193-110.317.amzn2.x86_64 OpenJDK_64-Bit_Server_VM/11.0.8+10-LTS java/11.0.8
 kotlin/1.3.72 vendor/Amazon.com_Inc. exec-env/AWS_Lambda_java11",
    "requestParameters": {
        "TrackerName": "ExampleTracker"
    },
    "responseElements": null,
    "requestID": "997d5f93-cfef-429a-bbed-daab417ceab4",
    "eventID": "d9e0eebe-173c-477d-b0c9-d1d8292da103",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012",
}
```

Creating Amazon Location Service resources with AWS CloudFormation

Amazon Location Service is integrated with AWS CloudFormation, a service that helps you to model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all the AWS resources that you want (such as Amazon Location resources), and AWS CloudFormation provisions and configures those resources for you.

When you use AWS CloudFormation, you can reuse your template to set up your Amazon Location resources consistently and repeatedly. Describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

Amazon Location and AWS CloudFormation templates

To provision and configure resources for Amazon Location and related services, you must understand AWS CloudFormation templates. Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your AWS CloudFormation

stacks. If you're unfamiliar with JSON or YAML, you can use AWS CloudFormation Designer to help you get started with AWS CloudFormation templates. For more information, see What is AWS CloudFormation Designer? in the *AWS CloudFormation User Guide*.

Amazon Location supports creating the following resource types in AWS CloudFormation:

- AWS::Location::Map
- AWS::Location::PlaceIndex
- AWS::Location::RouteCalculator
- AWS::Location::Tracker
- AWS::Location::TrackerConsumer
- AWS::Location::GeofenceCollection

For more information, including examples of JSON and YAML templates for Amazon Location resources, see the Amazon Location Service resource type reference in the *AWS CloudFormation User Guide*.

Learn more about AWS CloudFormation

To learn more about AWS CloudFormation, see the following resources:

- AWS CloudFormation
- AWS CloudFormation User Guide
- AWS CloudFormation API Reference
- AWS CloudFormation Command Line Interface User Guide

Security in Amazon Location Service

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to Amazon Location Service, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon Location. The following topics show you how to configure Amazon Location to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon Location resources.

Topics

- Data protection in Amazon Location Service
- Identity and Access Management for Amazon Location Service
- Incident Response in Amazon Location Service
- Compliance validation for Amazon Location Service
- Resilience in Amazon Location Service
- Infrastructure security in Amazon Location Service
- Configuration and vulnerability analysis in Amazon Location
- Cross-service confused deputy prevention
- Security best practices for Amazon Location Service
- Best practices for Amazon Location Service

Data protection in Amazon Location Service

The AWS shared responsibility model applies to data protection in Amazon Location Service. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Amazon Location or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Data privacy

With Amazon Location Service, you retain control of your organization's data. Amazon Location anonymizes all queries sent to data providers by removing customer metadata and account information.

Amazon Location doesn't use data providers for tracking and geofencing. This means your sensitive data remains in your AWS account. This helps shield sensitive location information, such as facility, asset, and personnel location, from third parties, protect user privacy, and reduce your application's security risk.

For additional information, see the AWS Data Privacy FAQ.

Data retention in Amazon Location

The following characteristics relate to how Amazon Location collects and stores data for the service:

- **Amazon Location Service Trackers** – When you use the Trackers APIs to track the location of entities, their coordinates can be stored. Device locations are stored for 30 days before being deleted by the service.
- **Amazon Location Service Geofences** – When you use the Geofences APIs to define areas of interest, the service stores the geometries you provided. They must be explicitly deleted.

> **Note**
>
> Deleting your AWS account delete all resources within it. For additional information, see the AWS Data Privacy FAQ.

Data encryption at rest for Amazon Location Service

Amazon Location Service provides encryption by default to protect sensitive customer data at rest using AWS owned encryption keys.

- **AWS owned keys** — Amazon Location uses these keys by default to automatically encrypt personally identifiable data. You can't view, manage, or use AWS owned keys, or audit their use. However, you don't have to take any action or change any programs to protect the keys that

encrypt your data. For more information, see AWS owned keys in the *AWS Key Management Service Developer Guide*.

Encryption of data at rest by default helps reduce the operational overhead and complexity involved in protecting sensitive data. At the same time, it enables you to build secure applications that meet strict encryption compliance and regulatory requirements.

While you can't disable this layer of encryption or select an alternate encryption type, you can add a second layer of encryption over the existing AWS owned encryption keys by choosing a customer managed key when you create your tracker and geofence collection resources:

- **Customer managed keys** — Amazon Location supports the use of a symmetric customer managed key that you create, own, and manage to add a second layer of encryption over the existing AWS owned encryption. Because you have full control of this layer of encryption, you can perform such tasks as:
  - Establishing and maintaining key policies
  - Establishing and maintaining IAM policies and grants
  - Enabling and disabling key policies
  - Rotating key cryptographic material
  - Adding tags
  - Creating key aliases
  - Scheduling keys for deletion

  For more information, see customer managed key in the *AWS Key Management Service Developer Guide*.

The following table summarizes how Amazon Location encrypts personally identifiable data.

Data type	AWS owned key encryption	Customer managed key encryption (Optional)
`Position` A point geometry containing the device position details.	Enabled	Enabled

Data type	AWS owned key encryption	Customer managed key encryption (Optional)
`PositionProperties` A set of key-value pairs associated with the position update.	Enabled	Enabled
`GeofenceGeometry` A polygon geofence geometry representing the geofenced area.	Enabled	Enabled
`DeviceId` The device identifier specified when uploading a device position update to a tracker resource.	Enabled	Not supported
`GeofenceId` An identifier specified when storing a geofence geometry, or a batch of geofences in a given geofence collection.	Enabled	Not supported

Note

Amazon Location automatically enables encryption at rest using AWS owned keys to protect personally identifiable data at no charge.
However, AWS KMS charges apply for using a customer managed key. For more information about pricing, see the AWS Key Management Service pricing.

For more information on AWS KMS, see What is AWS Key Management Service?

How Amazon Location Service uses grants in AWS KMS

Amazon Location requires a grant to use your customer managed key.

When you create a tracker resource or geofence collection encrypted with a customer managed key, Amazon Location creates a grant on your behalf by sending a CreateGrant request to AWS KMS. Grants in AWS KMS are used to give Amazon Location access to a KMS key in a customer account.

Amazon Location requires the grant to use your customer managed key for the following internal operations:

- Send DescribeKey requests to AWS KMS to verify that the symmetric customer managed KMS key ID entered when creating a tracker or geofence collection is valid.
- Send GenerateDataKeyWithoutPlaintext requests to AWS KMS to generate data keys encrypted by your customer managed key.
- Send Decrypt requests to AWS KMS to decrypt the encrypted data keys so that they can be used to encrypt your data.

You can revoke access to the grant, or remove the service's access to the customer managed key at any time. If you do, Amazon Location won't be able to access any of the data encrypted by the customer managed key, which affects operations that are dependent on that data. For example, if you attempt to get device positions from an encrypted tracker that Amazon Location can't access, then the operation would return an `AccessDeniedException` error.

Create a customer managed key

You can create a symmetric customer managed key by using the AWS Management Console, or the AWS KMS APIs.

To create a symmetric customer managed key

Follow the steps for Creating symmetric customer managed key in the *AWS Key Management Service Developer Guide*.

Key policy

Key policies control access to your customer managed key. Every customer managed key must have exactly one key policy, which contains statements that determine who can use the key and how

they can use it. When you create your customer managed key, you can specify a key policy. For more information, see Managing access to customer managed keys in the *AWS Key Management Service Developer Guide*.

To use your customer managed key with your Amazon Location resources, the following API operations must be permitted in the key policy:

- `kms:CreateGrant` – Adds a grant to a customer managed key. Grants control access to a specified KMS key, which allows access to grant operations Amazon Location requires. For more information about Using Grants, see the *AWS Key Management Service Developer Guide*.

  This allows Amazon Location to do the following:

  - Call `GenerateDataKeyWithoutPlainText` to generate an encrypted data key and store it, because the data key isn't immediately used to encrypt.
  - Call `Decrypt` to use the stored encrypted data key to access encrypted data.
  - Set up a retiring principal to allow the service to `RetireGrant`.
- `kms:DescribeKey` – Provides the customer managed key details to allow Amazon Location to validate the key.

The following are policy statement examples you can add for Amazon Location:

```
"Statement" : [
  {
    "Sid" : "Allow access to principals authorized to use Amazon Location",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "*"
    },
    "Action" : [
      "kms:DescribeKey",
      "kms:CreateGrant"
    ],
    "Resource" : "*",
    "Condition" : {
      "StringEquals" : {
        "kms:ViaService" : "geo.region.amazonaws.com",
        "kms:CallerAccount" : "111122223333"
      }
  },
  {
```

```
      "Sid": "Allow access for key administrators",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
       },
      "Action" : [
        "kms:*"
       ],
      "Resource": "arn:aws:kms:region:111122223333:key/key_ID"
    },
    {
      "Sid" : "Allow read-only access to key metadata to the account",
      "Effect" : "Allow",
      "Principal" : {
        "AWS" : "arn:aws:iam::111122223333:root"
      },
      "Action" : [
        "kms:Describe*",
        "kms:Get*",
        "kms:List*",
        "kms:RevokeGrant"
      ],
      "Resource" : "*"
    }
  ]
```

For more information about specifying permissions in a policy, see the *AWS Key Management Service Developer Guide*.

For more information about troubleshooting key access, see the *AWS Key Management Service Developer Guide*.

Specifying a customer managed key for Amazon Location

You can specify a customer managed key as a second layer encryption for the following resources:

- Tracker resource
- Geofence collection

When you create a resource, you can specify the data key by entering a **KMS ID**, which Amazon Location uses to encrypt the identifiable personal data stored by the resource.

- **KMS ID** — A key identifier for an AWS KMS customer managed key. Enter a key ID, key ARN, alias name, or alias ARN.

Amazon Location Service encryption context

An encryption context is an optional set of key-value pairs that contain additional contextual information about the data.

AWS KMS uses the encryption context as additional authenticated data to support authenticated encryption. When you include an encryption context in a request to encrypt data, AWS KMS binds the encryption context to the encrypted data. To decrypt data, you include the same encryption context in the request.

Amazon Location Service encryption context

Amazon Location uses the same encryption context in all AWS KMS cryptographic operations, where the key is `aws:geo:arn` and the value is the resource Amazon Resource Name (ARN).

Example

```
"encryptionContext": {
    "aws:geo:arn": "arn:aws:geo:us-west-2:111122223333:geofence-collection/SAMPLE-
GeofenceCollection"
}
```

Using encryption context for monitoring

When you use a symmetric customer managed key to encrypt your tracker or geofence collection, you can also use the encryption context in audit records and logs to identify how the customer managed key is being used. The encryption context also appears in logs generated by AWS CloudTrail or Amazon CloudWatch Logs.

Using encryption context to control access to your customer managed key

You can use the encryption context in key policies and IAM policies as `conditions` to control access to your symmetric customer managed key. You can also use encryption context constraints in a grant.

Amazon Location uses an encryption context constraint in grants to control access to the customer managed key in your account or region. The grant constraint requires that the operations that the grant allows use the specified encryption context.

Example

The following are example key policy statements to grant access to a customer managed key for a specific encryption context. The condition in this policy statement requires that the grants have an encryption context constraint that specifies the encryption context.

```
{
    "Sid": "Enable DescribeKey",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/ExampleReadOnlyRole"
     },
     "Action": "kms:DescribeKey",
     "Resource": "*"
},
{
     "Sid": "Enable CreateGrant",
     "Effect": "Allow",
     "Principal": {
         "AWS": "arn:aws:iam::111122223333:role/ExampleReadOnlyRole"
     },
     "Action": "kms:CreateGrant",
     "Resource": "*",
     "Condition": {
         "StringEquals": {
             "kms:EncryptionContext:aws:geo:arn": "arn:aws:geo:us-
west-2:111122223333:tracker/SAMPLE-Tracker"
          }
     }
}
```

Monitoring your encryption keys for Amazon Location Service

When you use an AWS KMS customer managed key with your Amazon Location Service resources, you can use AWS CloudTrail or Amazon CloudWatch Logs to track requests that Amazon Location sends to AWS KMS.

The following examples are AWS CloudTrail events for `CreateGrant`, `GenerateDataKeyWithoutPlainText`, `Decrypt`, and `DescribeKey` to monitor KMS operations called by Amazon Location to access data encrypted by your customer managed key:

CreateGrant

When you use an AWS KMS customer managed key to encrypt your tracker or geofence collection resources, Amazon Location sends a `CreateGrant` request on your behalf to access the KMS key in your AWS account. The grant that Amazon Location creates are specific to the resource associated with the AWS KMS customer managed key. In addition, Amazon Location uses the `RetireGrant` operation to remove a grant when you delete a resource.

The following example event records the `CreateGrant` operation:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-04-22T17:02:00Z"
            }
        },
        "invokedBy": "geo.amazonaws.com"
    },
    "eventTime": "2021-04-22T17:07:02Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "172.12.34.56",
    "userAgent": "ExampleDesktop/1.0 (V1; OS)",
    "requestParameters": {
        "retiringPrincipal": "geo.region.amazonaws.com",
        "operations": [
```

```
            "GenerateDataKeyWithoutPlaintext",
            "Decrypt",
            "DescribeKey"
        ],
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "granteePrincipal": "geo.region.amazonaws.com"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE"
    },
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333"
}
```

GenerateDataKeyWithoutPlainText

When you enable an AWS KMS customer managed key for your tracker or geofence collection resource, Amazon Location creates a unique table key. It sends a `GenerateDataKeyWithoutPlainText` request to AWS KMS that specifies the AWS KMScustomer managed key for the resource.

The following example event records the `GenerateDataKeyWithoutPlainText` operation:

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "geo.amazonaws.com"
```

```
    },
    "eventTime": "2021-04-22T17:07:02Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKeyWithoutPlaintext",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "172.12.34.56",
    "userAgent": "ExampleDesktop/1.0 (V1; OS)",
    "requestParameters": {
        "encryptionContext": {
            "aws:geo:arn": "arn:aws:geo:us-west-2:111122223333:geofence-collection/
SAMPLE-GeofenceCollection"
        },
        "keySpec": "AES_256",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333",
    "sharedEventID": "57f5dbee-16da-413e-979f-2c4c6663475e"
}
```


Decrypt

When you access an encrypted tracker or geofence collection,Amazon Location calls the `Decrypt` operation to use the stored encrypted data key to access the encrypted data.

The following example event records the `Decrypt` operation:


```
{
    "eventVersion": "1.08",
```

```
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "geo.amazonaws.com"
    },
    "eventTime": "2021-04-22T17:10:51Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "172.12.34.56",
    "userAgent": "ExampleDesktop/1.0 (V1; OS)",
    "requestParameters": {
        "encryptionContext": {
            "aws:geo:arn": "arn:aws:geo:us-west-2:111122223333:geofence-collection/
SAMPLE-GeofenceCollection"
        },
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE",
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333",
    "sharedEventID": "dc129381-1d94-49bd-b522-f56a3482d088"
}
```


DescribeKey

Amazon Location uses the `DescribeKey` operation to verify if the AWS KMS customer managed key associated with your tracker or geofence collection exists in the account and region.

The following example event records the `DescribeKey` operation:


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-04-22T17:02:00Z"
            }
        },
        "invokedBy": "geo.amazonaws.com"
    },
    "eventTime": "2021-04-22T17:07:02Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "172.12.34.56",
    "userAgent": "ExampleDesktop/1.0 (V1; OS)",
    "requestParameters": {
        "keyId": "00dd0db0-0000-0000-ac00-b0c000SAMPLE"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
```

```
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333"
}
```

Learn more

The following resources provide more information about data encryption at rest.

- For more information about AWS Key Management Service basic concepts, see the *AWS Key Management Service Developer Guide*.
- For more information about Security best practices for AWS Key Management Service, see the *AWS Key Management Service Developer Guide*.

Data in transit encryption for Amazon Location Service

Amazon Location protects data in transit, as it travels to and from the service, by automatically encrypting all inter-network data using the Transport Layer Security (TLS) 1.2 encryption protocol. Direct HTTPS requests sent to the Amazon Location Service APIs are signed by using the AWS Signature Version 4 Algorithm to establish a secure connection.

Identity and Access Management for Amazon Location Service

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon Location resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating with identities

- Managing access using policies
- How Amazon Location Service works with IAM
- How Amazon Location Service works with unauthenticated users
- Identity-based policy examples for Amazon Location Service
- Troubleshooting Amazon Location Service identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon Location.

Service user – If you use the Amazon Location service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon Location features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon Location, see Troubleshooting Amazon Location Service identity and access.

Service administrator – If you're in charge of Amazon Location resources at your company, you probably have full access to Amazon Location. It's your job to determine which Amazon Location features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon Location, see How Amazon Location Service works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon Location. To view example Amazon Location identity-based policies that you can use in IAM, see Identity-based policy examples for Amazon Location Service.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on

authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider

in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.

- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile

that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed

policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.

- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon Location Service works with IAM

Before you use IAM to manage access to Amazon Location, learn what IAM features are available to use with Amazon Location.

IAM features you can use with Amazon Location Service

IAM feature	Amazon Location support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes

IAM feature	Amazon Location support
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Principal permissions	No
Service roles	No
Service-linked roles	No

To get a high-level view of how Amazon Location and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Amazon Location

Supports identity-based policies	Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Amazon Location

To view examples of Amazon Location identity-based policies, see Identity-based policy examples for Amazon Location Service.

Resource-based policies within Amazon Location

Supports resource-based policies	No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for Amazon Location

Supports policy actions	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Amazon Location actions, see Actions Defined by Amazon Location Service in the *Service Authorization Reference*.

Policy actions in Amazon Location use the following prefix before the action:

```
geo
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "geo:action1",
      "geo:action2"
         ]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Get`, include the following action:

```
"Action": "geo:Get*"
```

To view examples of Amazon Location identity-based policies, see Identity-based policy examples for Amazon Location Service.

Policy resources for Amazon Location

Supports policy resources	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of Amazon Location resource types and their ARNs, see Resources Defined by Amazon Location Service in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon Location Service.

To view examples of Amazon Location identity-based policies, see Identity-based policy examples for Amazon Location Service.

Policy condition keys for Amazon Location

Supports service-specific policy condition keys	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of Amazon Location condition keys, see Condition Keys for Amazon Location Service in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions Defined by Amazon Location Service.

Amazon Location supports condition keys to allow you to allow or deny access to specific geofences or devices in your policy statements. The following condition keys are available:

- `geo:GeofenceIds` for use with Geofence actions. The type is `ArrayOfString`.
- `geo:DeviceIds` for use with Tracker actions. The type is `ArrayOfString`.

The following actions can be used with `geo:GeofenceIds` in your IAM policy:

- `BatchDeleteGeofences`
- `BatchPutGeofences`
- `GetGeofence`
- `PutGeofence`

The following actions can be used with `geo:DeviceIds` in your IAM policy:

- `BatchDeleteDevicePositionHistory`
- `BatchGetDevicePosition`
- `BatchUpdateDevicePosition`
- `GetDevicePosition`
- `GetDevicePositionHistory`

> **Note**
>
> You can't use these condition keys with the `BatchEvaluateGeofences`, `ListGeofences`, or `ListDevicePosition` actions.

To view examples of Amazon Location identity-based policies, see Identity-based policy examples for Amazon Location Service.

ACLs in Amazon Location

Supports ACLs	No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Amazon Location

Supports ABAC (tags in policies)	Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see What is ABAC? in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

For more information about tagging Amazon Location resources, see Tagging your Amazon Location Service resources.

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see Control resource access based on tags.

Using temporary credentials with Amazon Location

Supports temporary credentials	Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for Amazon Location

Supports forward access sessions (FAS)	No

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for Amazon Location

Supports service roles	No

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break Amazon Location functionality. Edit service roles only when Amazon Location provides guidance to do so.

Service-linked roles for Amazon Location

Supports service-linked roles	No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

How Amazon Location Service works with unauthenticated users

Many scenarios for using Amazon Location Service, including showing maps on the web or in a mobile application, require allowing access to users who haven't signed in with IAM. For these unauthenticated scenarios, you have two options.

- **Use API keys** – To grant access to unauthenticated users, you can create API Keys that give read-only access to your Amazon Location Service resources. This is useful in a case where you do not want to authenticate every user. For example, a web application. For more information about API keys, see Allowing unauthenticated guest access to your application using API keys.
- **Use Amazon Cognito** – An alternative to API keys is to use Amazon Cognito to grant anonymous access. Amazon Cognito allows you to create a richer authorization with IAM policy to define what can be done by the unauthenticated users. For more information about using Amazon Cognito, see Allowing unauthenticated guest access to your application using Amazon Cognito.

For an overview of providing access to unauthenticated users, see Granting access to Amazon Location Service.

Identity-based policy examples for Amazon Location Service

By default, users and roles don't have permission to create or modify Amazon Location resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Creating IAM policies in the *IAM User Guide*.

For details about actions and resource types defined by Amazon Location, including the format of the ARNs for each of the resource types, see Actions, Resources, and Condition Keys for Amazon Location Service in the *Service Authorization Reference*.

Topics

- Policy best practices
- Using the Amazon Location console
- Allow users to view their own permissions
- Using Amazon Location Service resources in policy
- Permissions for updating device positions
- Read-only policy for tracker resources
- Policy for creating geofences
- Read-only policy for geofences
- Permissions for rendering a map resource
- Permissions to allow search operations
- Read-only policy for route calculators
- Control resource access based on condition keys
- Control resource access based on tags

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon Location resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see IAM Access Analyzer policy validation in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the Amazon Location console

To access the Amazon Location Service console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon Location resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum

required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can use the Amazon Location console, attach the following policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

The following policy gives access to the Amazon Location Service console, to be able to create, delete, list and view details about Amazon Location resources in your AWS account.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GeoPowerUser",
      "Effect": "Allow",
      "Action": [
        "geo:*"
      ],
      "Resource": "*"
    }
  ]
}
```

Alternatively, you can grant read-only permissions to facilitate read-only access. With read-only permissions, an error message will appear if the user attempts write actions such as creating or deleting resources. As an example, see the section called “Read-only policy for trackers”

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
```

```
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Using Amazon Location Service resources in policy

Amazon Location Service uses the following prefixes for resources:

Amazon Location resource prefix

Resource	Resource prefix
Map resources	`map`
Place resources	`place-index`
Route resources	`route-calculator`
Tracking resources	`tracker`

Resource	Resource prefix
Geofence Collection resources	`geofence-collection`

Use the following ARN syntax:

```
arn:Partition:geo:Region:Account:ResourcePrefix/ResourceName
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

Examples

- Use the following ARN to allow access to a specified map resource.

  ```
  "Resource": "arn:aws:geo:us-west-2:account-id:map/map-resource-name"
  ```

- To specify access to all `map` resources that belong to a specific account, use the wildcard (*):

  ```
  "Resource": "arn:aws:geo:us-west-2:account-id:map/*"
  ```

- Some Amazon Location actions, such as those for creating resources, can't be performed on a specific resource. In those cases, you must use the wildcard (*).

  ```
  "Resource": "*"
  ```

To see a list of Amazon Location resource types and their ARNs, see Resources Defined by Amazon Location Service in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon Location Service.

Permissions for updating device positions

To update device positions for multiple trackers, you'll want to grant a user access to one or more of your tracker resources. You will also want to allow the user to update a batch of device positions.

In this example, in addition to granting access to the *Tracker1* and *Tracker2* resources, the following policy grants permission to use the `geo:BatchUpdateDevicePosition` action against the *Tracker1* and *Tracker2* resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "UpdateDevicePositions",
      "Effect": "Allow",
      "Action": [
        "geo:BatchUpdateDevicePosition"
      ],
      "Resource": [
        "arn:aws:geo:us-west-2:account-id:tracker/Tracker1",
        "arn:aws:geo:us-west-2:account-id:tracker/Tracker2"
      ]
    }
  ]
}
```

If you want to limit the user to only be able to update device positions for a specific device, you can add a condition key for that device id.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "UpdateDevicePositions",
      "Effect": "Allow",
      "Action": [
        "geo:BatchUpdateDevicePosition"
      ],
      "Resource": [
        "arn:aws:geo:us-west-2:account-id:tracker/Tracker1",
        "arn:aws:geo:us-west-2:account-id:tracker/Tracker2"
      ],
      "Condition":{
        "ForAllValues:StringLike":{
          "geo:DeviceIds":[
            "deviceId"
          ]
        }
      }
    }
  ]
```

```
}
```

Read-only policy for tracker resources

To create a read-only policy for all tracker resources in your AWS account, you'll need to grant access to all tracker resources. You'll also want to grant a user access to actions that allow them to get the device position for multiple devices, get the device position from a single device and get the position history.

In this example, the following policy grants permission to the following actions:

- `geo:BatchGetDevicePosition` to retrieve the position of multiple devices.
- `geo:GetDevicePosition` to retrieve the position of a single device.
- `geo:GetDevicePositionHistory` to retrieve the position history of a device.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GetDevicePositions",
      "Effect": "Allow",
      "Action": [
        "geo:BatchGetDevicePosition",
        "geo:GetDevicePosition",
        "geo:GetDevicePositionHistory"
      ],
      "Resource": "arn:aws:geo:us-west-2:account-id:tracker/*"
    }
  ]
}
```

Policy for creating geofences

To create a policy to allow a user to create geofences, you'll need to grant access to specific actions that allow users to create one or more geofences on a geofence collection.

The policy below grants permission to the following actions on *`Collection`*:

- `geo:BatchPutGeofence` to create multiple geofences.

- `geo:PutGeofence` to create a single geofence.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CreateGeofences",
      "Effect": "Allow",
      "Action": [
        "geo:BatchPutGeofence",
        "geo:PutGeofence"
      ],
      "Resource": "arn:aws:geo:us-west-2:account-id:geofence-collection/Collection"
    }
  ]
}
```

Read-only policy for geofences

To create a read-only policy for geofences stored in a geofence collection in your AWS account, you'll need to grant access to actions that read from the geofence collection storing the geofences.

The policy below grants permission to the following actions on *`Collection`*:

- `geo:ListGeofences` to list geofences in the specified geofence collection.
- `geo:GetGeofence` to retrieve a geofence from the geofence collection.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GetGeofences",
      "Effect": "Allow",
      "Action": [
        "geo:ListGeofences",
        "geo:GetGeofence"
      ],
      "Resource": "arn:aws:geo:us-west-2:account-id:geofence-collection/Collection"
    }
  ]
```

```
}
```

Permissions for rendering a map resource

To grant sufficient permissions to render maps, you'll need to grant access to map tiles, sprites, glyphs, and the style descriptor:

- `geo:GetMapTile` retrieves map tiles used to selectively render features on a map.
- `geo:GetMapSprites` retrieves the PNG sprite sheet and corresponding JSON document describing offsets within it.
- `geo:GetMapGlyphs` retrieves the glyphs used for displaying text.
- `geo:GetMapStyleDescriptor` retrieves the map's style descriptor, containing rendering rules.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GetTiles",
      "Effect": "Allow",
      "Action": [
        "geo:GetMapTile",
        "geo:GetMapSprites",
        "geo:GetMapGlyphs",
        "geo:GetMapStyleDescriptor"
      ],
      "Resource": "arn:aws:geo:us-west-2:account-id:map/Map"
    }
  ]
}
```

Permissions to allow search operations

To create a policy to allow search operations, you'll first need to grant access to the place index resource in your AWS account. You'll also want to grant access to actions that let the user search using text by geocoding and search using a position by reverse geocoding.

In this example, in addition to granting access to *PlaceIndex*, the following policy also grants permission to the following actions:

- `geo:SearchPlaceIndexForPosition` allows you to search for places, or points of interest near a given position.
- `geo:SearchPlaceIndexForText` allows you to search for an address, name, city or region using free-form text.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Search",
      "Effect": "Allow",
      "Action": [
        "geo:SearchPlaceIndexForPosition",
        "geo:SearchPlaceIndexForText"
      ],
      "Resource": "arn:aws:geo:us-west-2:account-id:place-index/PlaceIndex"
    }
  ]
}
```

Read-only policy for route calculators

You can create a read-only policy to allow a user access to a route calculator resource to calculate a route.

In this example, in addition to granting access to *ExampleCalculator*, the following policy grants permission to the following operation:

- `geo:CalculateRoute` calculates a route given a departure position, destination positon, and a list of waypoint positions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RoutesReadOnly",
      "Effect": "Allow",
      "Action": [
        "geo:CalculateRoute"
```

```
      ],
      "Resource": "arn:aws:geo:us-west-2:accountID:route-calculator/ExampleCalculator"
    }
  ]
}
```

Control resource access based on condition keys

When you create an IAM policy to grant access to use geofences or device positions, you can use Condition operators for more precise control over which geofences or devices a user can access. You can do this by including the geofence id or device id in the `Condition` element of your policy.

The following example policy shows how you might create a policy that allows a user to update device positions for a specific device.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "UpdateDevicePositions",
      "Effect": "Allow",
      "Action": [
        "geo:BatchUpdateDevicePosition"
      ],
      "Resource": [
        "arn:aws:geo:us-west-2:account-id:tracker/Tracker"
      ],
      "Condition":{
        "ForAllValues:StringLike":{
          "geo:DeviceIds":[
            "deviceId"
          ]
        }
      }
    }
  ]
}
```

Control resource access based on tags

When you create an IAM policy to grant access to use your Amazon Location resources, you can use attribute-based access control for better control over which resources a user can modify, use, or

delete. You can do this by including tag information in the `Condition` element of your policy to control access based on your resource tags.

The following example policy shows how you might create a policy that allows a user to create geofences. This grants the permission to the following actions to create one or more geofences on a geofence collection called *`Collection`*:

- `geo:BatchPutGeofence` to create multiple geofences.
- `geo:PutGeofence` to create a single geofence.

However, this policy uses the `Condition` element to grant the permission only if the *`Collection`* tag, `Owner`, has the value of that user's user name.

- For example, if a user named `richard-roe` attempts to view an Amazon Location *`Collection`*, the *`Collection`* must be tagged `Owner=richard-roe` or `owner=richard-roe`. Otherwise the user is denied access.

> **Note**
>
> The condition tag key `Owner` matches both `Owner` and `owner` because condition key names are not case-sensitive. For more information, see IAM JSON Policy Elements: Condition in the *IAM User Guide*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CreateGeofencesIfOwner",
      "Effect": "Allow",
      "Action": [
        "geo:BatchPutGeofence",
        "geo:PutGeofence"
      ],
      "Resource": "arn:aws:geo:us-west-2:account-id:geofence-collection/Collection",
      "Condition": {
                "StringEquals": {"geo:ResourceTag/Owner": "${aws:username}"}
      }
    }
  ]
```

```
}
```

For a tutorial about how to define permissions to access AWS resources based on tags, see the *AWS Identity and Access Management User Guide*.

Troubleshooting Amazon Location Service identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon Location and IAM.

Topics

- I am not authorized to perform an action in Amazon Location
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my Amazon Location resources

I am not authorized to perform an action in Amazon Location

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `geo:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 geo:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `geo:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Amazon Location.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon Location. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my Amazon Location resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon Location supports these features, see How Amazon Location Service works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Incident Response in Amazon Location Service

Security is the highest priority at AWS. As part of the AWS Cloud shared responsibility model, AWS manages a data center and network architecture that meets the requirements of the most security-sensitive organizations. As an AWS customer, you share a responsibility for maintaining security in the cloud. This means you control the security you choose to implement from the AWS tools and features you have access to.

By establishing a security baseline that meets the objectives for your applications running in the cloud, you're able to detect deviations that you can respond to. Since security incident response can be a complex topic, we encourage you to review the following resources so that you are better able to understand the impact that incident response (IR) and your choices have on your corporate goals: AWS Security Incident Response Guide, AWS Security Best Practices whitepaper, and the AWS Cloud Adoption Framework (AWS CAF).

Logging and Monitoring in Amazon Location Service

Logging and monitoring are an important part of incident response. It lets you establish a security baseline to detect deviations that you can investigate and respond to. By implementing logging and monitoring for Amazon Location Service, you're able to maintain the reliability, availability, and performance for your projects and resources.

AWS provides several tools that can help you log and collect data for incident response:

AWS CloudTrail

Amazon Location Service integrates with AWS CloudTrail, which is a service that provides a record of actions taken by a user, role or AWS service. This includes actions from the Amazon Location Service console, and programmatic calls to Amazon Location API operations. These records of action are called events. For more information, see Logging and monitoring Amazon Location Service with AWS CloudTrail.

Amazon CloudWatch

You can use Amazon CloudWatch to collect and analyze metrics related to your Amazon Location Service account. You can enable CloudWatch alarms to notify you if a metric meets certain conditions, and has reached a specified threshold. When you create an alarm, CloudWatch sends a notification to an Amazon Simple Notification Service that you define. For more information, see the Monitoring Amazon Location Service with Amazon CloudWatch.

AWS Health Dashboards

Using AWS Health Dashboards, you can verify the status of the Amazon Location Service service. You can also monitor and view historical data about any events or issues that might affect your AWS environment. For more information, see the AWS Health User Guide.

Compliance validation for Amazon Location Service

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

> **Note**
>
> Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).

- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in Amazon Location Service

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, Amazon Location offers several features to help support your data resiliency and backup needs.

Infrastructure security in Amazon Location Service

As a managed service, Amazon Location Service is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Amazon Location through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Configuration and vulnerability analysis in Amazon Location

Configuration and IT controls are a shared responsibility between AWS and you, our customer. For more information, see the AWS shared responsibility model.

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

Amazon Location Service does not act as a calling service on your behalf to other AWS services, so you do not need to add these protections in this case. To learn more about confused deputy, see The confused deputy problem in the *AWS Identity and Access Management User Guide*.

Security best practices for Amazon Location Service

Amazon Location Service provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don’t represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

Detective security best practices for Amazon Location Service

The following best practices for Amazon Location Service can help detect security incidents:

Implement AWS monitoring tools

Monitoring is critical to incident response and maintains the reliability and security of Amazon Location Service resources and your solutions. You can implement monitoring tools from the several tools and services available through AWS to monitor your resources and your other AWS services.

For example, Amazon CloudWatch allows you to monitor metrics for Amazon Location Service and enables you to setup alarms to notify you if a metric meets certain conditions you've set and has reached a threshold you've defined. When you create an alarm, you can set CloudWatch to sent a notification to alert using Amazon Simple Notification Service. For more information, see the section called “Logging and Monitoring”.

Enable AWS logging tools

Logging provides a record of actions taken by a user, role or an AWS service in Amazon Location Service. You can implement logging tools such as AWS CloudTrail to collect data on actions to detect unusual API activity.

When you create a trail, you can configure CloudTrail to log events. Events are records of resource operations performed on or within a resource such as the request made to Amazon Location, the IP address from which the request was made, who made the request, when the request was made, along with additional data. For more information, see Logging Data Events for Trails in the AWS CloudTrail User Guide.

Preventive security best practices for Amazon Location Service

The following best practices for Amazon Location Service can help prevent security incidents:

Use secure connections

Always use encrypted connections, such as those that begin with `https://` to keep sensitive information secure in transit.

Implement least privilege access to resources

When you create custom policies to Amazon Location resources, grant only the permissions required to perform a task. It's recommended to start with a minimum set of permissions and grant additional permissions as needed. Implementing least privilege access is essential to reducing the risk and impact that could result from errors or malicious attacks. For more information, see the section called “Identity and Access Management”.

Use globally-unique IDs as device IDs

Use the following conventions for device IDs.

- Device IDs must be unique.
- Device IDs should not be secret, because they can be used as foreign keys to other systems.
- Device IDs should not contain personally-identifiable information (PII), such as phone device IDs or email addresses.
- Device IDs should not be predictable. Opaque identifiers like UUIDs are recommended.

Do not include PII in device position properties

When sending device updates (for example, using DevicePositionUpdate), do not include personally-identifiable information (PII) such as phone number or email address in the `PositionProperties`.

Best practices for Amazon Location Service

This topic provides best practices to help you use Amazon Location Service. While these best practices can help you take full advantage of the Amazon Location Service, they do not represent a complete solution. You should follow only the recommendations that are applicable for your environment.

Topics

- Security
- Resource management
- Billing and cost management
- Quotas and usage

Security

To help manage or even avoid security risks, consider the following best practices:

- Use identity federation and IAM roles to manage, control, or limit access to your Amazon Location resources. For more information, see IAM Best Practices in the *IAM User Guide*.
- Follow the Principle of Least Privilege to grant only the minimum required access to your Amazon Location Service resources. For more information, see the section called "Managing access using policies".
- For Amazon Location Service resources used in web applications, restrict access using an `aws:referer` IAM condition, limiting use by sites other than those included in the allow-list.
- Use monitoring and logging tools to track resource access and usage. For more information, see the section called "Logging and Monitoring" and Logging Data Events for Trails in the AWS CloudTrail User Guide.
- Use secure connections, such as those that begin with `https://` to add security and protect users against attacks while data is being transmitted between the server and browser.

For more information about detective and preventive security best practices, see the topic on the section called "Security best practices".

Resource management

To help effectively manage your location resources in Amazon Location Service, consider the following best practices:

- Use regional endpoints that are central to your expected user base to improve their experience. For information about region endpoints, see Amazon Location Regions and endpoints.
- For resources that use data providers, such as map resources and place index resources, make sure to follow the terms of use agreement of the specific data provider. For more information, see Data providers.
- Minimize the creation of resources by having one resource for each configuration of map, place index, or routes. Within a region, you typically need only one resource per data provider or map style. Most applications use existing resources, and do not create resources at run time.
- When using different resources in a single application, such as a map resource and a route calculator, use the same data provider in each resource to ensure that the data matches. For

example, that a route geometry you create with your route calculator aligns with the streets on the map drawn using the map resource.

Billing and cost management

To help manage your costs and billing, consider the following best practice:

- Use monitoring tools, such as Amazon CloudWatch, to track your resource usage. You can set alerts that notify you when usage is about to exceed your specified limits. For more information, see Creating a Billing Alarm to Monitor Your Estimated AWS Charges in the *Amazon CloudWatch User Guide*.

Quotas and usage

You AWS account includes quotas that set a default limit your usage amount. You can set up alarms to alert you when your usage is getting close to your limit, and you can request a raise to a quota, when you need it. For information about how to work with quotas, see the following topics.

- Amazon Location Service quotas
- Using CloudWatch to monitor usage against quotas
- Visualizing your service quotas and setting alarms in the *Amazon CloudWatch User Guide*.

You can create alarms to give you advance warning when you are close to exceeding your limits. We recommend setting alarms for each quota in each AWS Region where you use Amazon Location. For example, you can monitor your use of the `SearchPlaceIndexForText` operation, and create an alarm when you exceed 80 percent of your current quota.

When you get an alarm warning about your quota, you must decide what to do. You might be using additional resources because your customer base has grown. In that case you may want to request an increase to your quota, such as a 50 percent increase in the quota for an API call in that Region. Or, maybe there's an error in your service that causes you to make additional unnecessary calls to Amazon Location. In that case you'd want to solve the problem in your service.

Document history

The following table describes the documentation for Amazon Location Service. For notification about updates you can subscribe to an RSS feed.

Change	Description	Date
Amazon Location Service releases a new SDK for JavaScript	To make developing Amazon Location applications easier with in web front ends, Amazon Location adds a new open source SDK that supports AWS SDK for JavaScript v3, simplifying authentication and using GeoJSON. For mor informati on, see Amazon Location SDK.	July 6, 2023
Amazon Location Service releases API keys to general availability	Amazon Location adds support for place and route and announce general availability of the API keys feature. For more informati on, see Using API keys.	July 6, 2023
Amazon Location Service adds Amazon EventBridge events for position updates	Amazon Location adds support for sending tracker position update events to EventBridge. For more information, including how to enable the events for a tracker, see Reacting to events with EventBridge.	July 6, 2023
Amazon Location adds metadata to geofences	Using the Amazon Location API, you can now add metadata properties to	June 15, 2023

	your geofences. These are stored with your geofence, and included in events related to the geofence in Amazon EventBridge. For more information, see Draw geofences and Create event rules.	
Amazon Location adds categories for places	Amazon Location add categories in place search results, and filtering results by category. For more informati on, see Categories and filtering.	June 15, 2023
Amazon Location introduces political views	Amazon Location adds political views to certain map styles. For more information, see Political views.	May 23, 2023
Amazon Location introduces new demo and samples site	Amazon Location announces a new web site that gives you access to Amazon Location demos and samples. For more information, see Amazon Location demo site.	May 3, 2023
Amazon Location introduce s longer routes in Calculate RouteMatrix	Amazon Location now allows unlimited length routes for route matrix routes created with the HERE data provider. For more information, see Longer route planning.	April 24, 2023

Amazon Location documentation adds feature differences by data provider	Amazon Location documentation has been updated with information regarding the differences between each data provider in Maps, Places search, and routing. For more information, see Features by data provider.	March 30, 2023
Amazon Location Open Data maps general availability	General availability of Amazon Location Service data provider and style, based on OpenStreetMap's Daylight maps. For more information, see Open Data.	March 7, 2023
Amazon Location adds new authorization method in preview	Amazon Location Service adds API keys as a new authorization method for anonymous users, in preview mode. For more information, see Allowing unauthenticated guest access to your application using API keys.	February 23, 2023
Amazon Location documentation updated with latest IAM best practices	Amazon Location Service documentation has been updated to meet the most recent AWS Identity and Access Management best practices. For more information, see Security in Amazon Location Service.	January 26, 2023

Amazon Location Service adds GrabMaps as a data provider in Southeast Asia	Amazon Location introduces GrabMaps as a data provider in Southeast Asia. For more information, see GrabMaps.	January 10, 2023
Amazon Location Service Open Data maps in preview	New Amazon Location data provider and style added in public preview, based on OpenStreetMap's Daylight maps. For more information, see Open Data (Preview).	December 15, 2022
New HERE satellite imagery styles	Two new map styles have been added for maps using HERE as a data provider, HERE Satellite Imagery and HERE Hybrid map styles. For more information, see HERE map styles.	October 25, 2022
Units in addresses	Amazon Location Service now supports units in addresses , for example "123 Main St, *Apartment 3B*, Anytown, USA".	September 20, 2022
Get places by ID	Amazon Location Service now includes support finding the exact location suggested by the `SearchPlaceIndexForSuggestions` operation using the `GetPlace` operation. See Using autocomplete.	September 20, 2022

Additional condition keys for IAM policy	Amazon Location Service now supports additional condition keys that let you set access for specific geofences or devices in IAM policy. See Condition keys.	August 23, 2022
Circle geofences	Amazon Location Service now supports geofences defined as a circle with a center point and a radius, to get events when devices are within a certain distance of a location. See Adding circular geofences .	August 11, 2022
Combined API reference	Amazon Location Service now has a single API reference guide, rather than separate guides for each subservic e. For more information about the APIs, see Amazon Location APIs.	July 7, 2022
Service Quotas integration	Amazon Location is now integrated with Service Quotas allowing you to view and manage your quotas through the AWS Managemen t Console or using the AWS CLI.	July 6, 2022
Updated concepts documenta tion chapter	The Amazon Location concepts chapter has been updated with more informati on for users of Amazon Location.	April 22, 2022

New Android quick start tutorial	A new quick start tutorial for Android development using Kotlin has been added to get developers up and running quickly.	April 15, 2022
New HERE Map styles	Two new map styles have been added for maps using HERE as a data provider. For more information, see HERE map styles.	March 15, 2022
Restructure documentation with added code examples and tutorials	This developer guide has been restructured to find topics more easily, including new Quick start and Code examples chapters.	February 25, 2022
Accuracy-based position filtering for trackers	You can now use accuracy-based filters when creating a tracker resource.	December 7, 2021
Autocomplete for place indexes	You can now use autocomplete when searching place indexes.	December 6, 2021
New Amplify tutorial for using maps	A new tutorial is available showing how to use AWS Amplify to display maps in a web application. The tutorial is available at Using the Amplify library with Amazon Location Service.	November 24, 2021

Place query extensions	Amazon Location Service now supports setting a preferred language for results when geocoding, or reverse geocoding, and adds the time zone and other informati on to the results. For more information about geocoding and reverse geocoding, see Geocoding, reverse geocoding , and searching.	November 16, 2021
Tracker position filtering	Amazon Location Service adds a new position filtering feature to trackers that can help you control costs. This feature filters out some position updates on devices before the updates are stored or evaluated against geofences. For more informati on about position filtering, see Trackers.	October 5, 2021
Update operations	The following operation s have been added to the Amazon Location Service API References : UpdateMap, UpdatePla ceIndex, UpdateRouteCalcula tor, UpdateGeofenceColl ection, and UpdateTracker.	July 19, 2021

Tutorial update: Amazon Aurora PostgreSQL user-defined functions	A new tutorial has been added for how to use Amazon Aurora PostgreSQL user-defined functions with Amazon Location to validate, clean, and enrich geospatial data.	July 19, 2021
AWS CloudFormation resources	Amazon Location now supports creating the following resource types in AWS CloudFormation resources: `AWS::Location::Map` , `AWS::Location::PlaceIndex` , `AWS::Location::RouteCalculator` , `AWS::Location::Tracker` , `AWS::Location::TrackerConsumer` , and `AWS::Location::GeofenceCollection` .	June 7, 2021
Tagging resources	You can now add tags to your Amazon Location resources to help manage, identify, organize, search, and filter your resources.	June 1, 2021
General availability	General availability release of the Amazon Location Service developer documentation: Region and endpoints and service quotas updated.	June 1, 2021

Esri Imagery	Amazon Location now supports the use of Esri map style: Esri Imagery. For more information, see Esri World Imagery on the Esri website.	June 1, 2021
Calculating routes	You can now use Amazon Location route calculators to calculate routes and estimate travel time based on up-to-date road network and live traffic information from your chosen data provider.	June 1, 2021
AWS KMS customer managed key encryption for data at rest	Amazon Location now supports the use of a symmetric customer managed key that you create, own, and manage to add a second layer of encryption over the existing AWS owned encryption.	June 1, 2021
Public preview release	Initial release of the public preview documentation.	December 16, 2020
Tutorial update: Displaying maps	Tutorials for displaying maps using MapLibre for Android and iOS have been updated to use the MapLibre native SDK.	March 17, 2020

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.