



Serving Dynamic Tiles

1.A - Tile Serving Architecture

- 1

User requests **tiles.maphub.dev** and is served the site's static content from a Cloudfront Distribution
- 2

tiles.maphub.dev makes calls to the tiles API at **api.maphub.dev**. This API sits behind an Application Load Balancer that distributes traffic across several service instances. Each request takes the form of:

https://api.maphub.dev/{layer}/{z}/{x}/{y}

Where **{layer}** represents the base layer of the request and **{x}**, **{y}**, **{z}** represent the location of the tile on Earth.
- 3

The ALB routes traffic to one of several instances. Each instance is attached to our ECS cluster and runs 4 containers. A core API container (Golang), a X-Ray Agent, Nginx, and a cache (Redis). r6g instances are chosen to support Redis' memory use and allow for caching tens of thousands of tiles.
- 4

The core API container checks the TileCache for this layer, if the layer is present in the cache, the content is returned to the user as a vector tile. [0 - 10ms]
- 5

If the tile is not available, then the core API gets the IP of available DB instances from CloudMap**
- 6

The core API formats a request for a vector tile to the database and waits for the tile to be generated. [20-500ms]
- 7

The core API caches a successful tile request to the local TileCache. Subsequent requests for this layer and tile will be fulfilled by fetching from the cache
- 8

The vector tile is returned to the user and rendered in their browser.
- 4.1

Requests to the core API are sampled at random by the local XRay Agent and traces are written to AWS XRay

**As of writing, there are no database replica instances. Instead of service discovery via Cloud Map to find a reader node, the API "resolves" DNS names by calling a fixed parameter in AWS Parameter Store.

Serving Dynamic Tiles

1.B - Building OSM Database

- B.1

OSM data can be ingested to PostGIS using a tool called **osm2pgsql**, but it is very memory intensive.

A spot instance with 32-64GB RAM is purchased for ingesting OSM data into a database. To expedite the build, we place the data directory of the PostgreSQL database on the ephemeral NVME disk attached to m6gd instances.
- B.2

OSM data is downloaded from a mirror of the Open Street Map project, **geofabrik.de**. This download is a compressed extract of the OSM DB and is about 50GB compressed.
- B.3

osm2pgsql loads the OSM data to PostGIS, creates its geospatial indexes, and adds hstore tags. Depending on the exact settings used to load the DB, this process results in a new schema with OSM point, OSM line, OSM road, and OSM polygon tables with a total size of ~600GB.
- B.4

The spot instance's DB dumps the data to the main application's PostGIS instance and can then be terminated.
- B.5

[NOT SHOWN] - **osm2pgsql** includes a module that allows a DB maintainer to receive periodic updates from the OSM replication server (or a mirror). I launch a ECS task that runs on a fixed interval to poll updates from this server and push them to the main DB.