Team Asteroid: EV Charging Stations Map

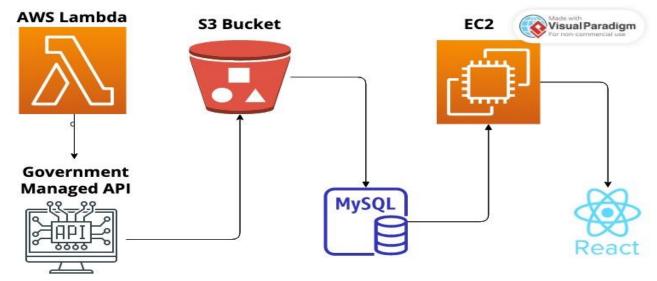
Vaishnavi Manikonda, Nathan Leverence, Michael Hangsterfer, Anthony Alagna

Abstract

Forbes, Ernst & Young, *Wired* agree on one thing: data is the new gold. The need for data has spurred innovations in data engineering (i.e. Snowflake, Kafka, Airflow, etc.) working to get data into a usable format in the least expensive way possible. We wanted to create a project that leveraged common data engineering tools in order to build a useful web application. With this idea we decided to use a common data engineering stack, AWS S3, RDS, Lambda, EventBridge, and StepFunctions to display how you can inexpensively get data into a usable format to power a web application. This displays how clean data powers not only data analysis but other use cases as well. The project pulls from the National Renewable Energy Laboratory API for electric vehicle fuel stations. We then use a front end to prompt the user for a zip code and display the locations and addresses of charging stations located in the zip code. This is done with a web application built in React which leverages Google Maps API.

Architecture

Two lambda functions, APItoS3 and S3toRDS are run using StepFunctions which is triggered by EventBridge Scheduler every 24 hours. APItoS3 calls and moves the data into S3. Then S3toRDS grabs the data from the S3 buckets, changes it to tabular formatting using Python's pandas library and then moves it into MySQL using SQLAlchemy. No version control is used since we only want the most up to date data from the API. The EC2 instance uses nginx for the routing to run the web application using data from the MySQL table. Node.js is used for the backend and API of the web application.



Features

In this section, we describe the features which we used to build our project which has the ability to be used as a pipeline, web application deployment service, or both as we have done with our application.

- 1. The data pipeline component flows through the following AWS Services: EventBridge → Step Functions → Lambda Function → S3 → Lamba Function → MySQL. This flow can be leveraged for a variety of other use cases. For instance, if we wanted to scale our application to not only show customers EV charging station locations but also other information about the surrounding area, we can use the same steps for other data sources. Our code has the potential to ingest CSV or JSON data. This can also expand to other data types if needed. S3 is used as a dumping ground for all data in its raw format and provides cheap storage should we need it. Finally, the data is cleaned using Python and placed into a MySQL database.
- 2. The most important piece of this portion of the project is the hosting in the cloud. Once the database is in RDS, the web application can make use of the data in the EC2 instance in a variety of ways, from data analysis, to creating APIs, to targeted advertising, trend prediction and so much more. Hosting a web application on EC2 allows our application to be more failure resistant, cheaper through auto-scaling, and have complete control over instances and who accesses them. The web application part of the code can take a large database of electric vehicle charger locations and return the closest locations through zip code, and can easily be updated to provide even more targeted data through more endpoints, change the data source to a completely different project, or display the data in a different manner.
- 3. In order to fully make use of the tools described above, a user would need to set up a VPC with access to the RDS from the EC2. Once this is complete, an individual can implement this type of service for their use case.
- 4. This project can be used with an API to get current, up-to-date data that refreshes daily and only returns the desired fields. Here, we use an API of Electric Vehicle location data to create a web application that displays the closest EV locations to the user.

How it Works

- 1. Backend
 - a. **APItoS3**: The first step in order to begin the project is to create two lambda functions APItoS3 and S3toRDS. These functions leverage the classes Store.py, Ingest.py, and Process.py in order to run. APItoS3 uses an API key from the National Renewable Energy Laboratory and sets parameters to bring in EV charging stations in the USA. A main piece in making this work is

using boto3 to make the connection to the S3 bucket after the API call has completed. Our function needed 1024 MB of memory in order to complete both of these calls because of the large amount of data (~70,000 rows with ~80 columns) being brought in. We also set the timeout to 2 ½ minutes. We also used Python 3.10 as the runtime for the functions as well as the layers.

- b. **S3toRDS** leverages boto3 as well as pandas and SQLAlchemy in order to take the data, clean it, and move it to a MySQL database. We included these dependencies as Lambda Layers. We also only included the needed nine columns for the web application to run effectively when moving the data to the MySQL database. These are the columns involving latitude, longitude, and address. We decided to overwrite the database as opposed to using version control because we only want the data provided by the most recent API call.
- c. The actual code for these functions can be accessed via the public GitHub repo provided at the end of the post.
- 2. Frontend
 - a. EC2 instance hosts web application
 - i. An EC2 instance is created for the web application. We used an Ubuntu 20.04 instance with t2.micro.
 - ii. The client portion of the React app was built and pushed to the private repository, and was modified to run from starting the backend server with express
 - iii. pm2, npm, nginx used
 - Pm2, a Production Process Manager with a built-in Load Balancer, is used to keep the application running indefinitely. We use it to start our updated backend.
 - 2. Npm is used to install react dependencies
 - 3. Nginx is used as a proxy server that handles the routing
 - iv. VPC needed between RDS and EC2 instance so that the application can query data from the database
 - b. React, Node.js, and express used for web application
 - i. Backend
 - 1. The Node.js and express backend uses express to serve the static build files.
 - 2. An endpoint queries the database with a SQL query using a mysql connection, taking a zip code as a parameter.
 - ii. Frontend/Client in typescript

- 1. When a user enters a zipcode into the input bar and submits, the event is handled and a function that fetches from the endpoint using the zipcode as a parameter is called.
 - a. The URL for the fetch changes depending on the production environment
- 2. The Google maps api is given the formatted values from the list of data returned and then we create the markers, format and concatenate the data for the info window that shows up when the marker is clicked, create a link that takes the user to the same location on Google Maps, and center the map around the list of EV charger locations.

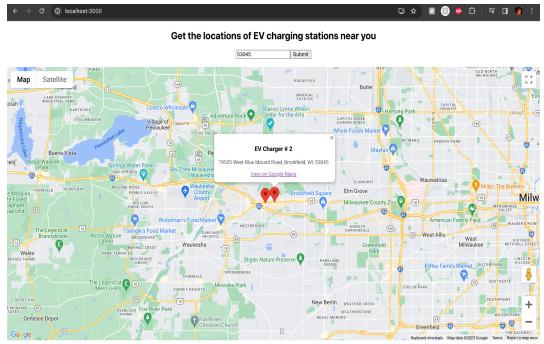
Running the React App locally

1. Once you have the database detailed above, check to make sure you can access it from MySQLWorkbench. You should have over 60 thousand records. You will need this database for the React app to run.

SCHEMAS	44		i 🕕 🚯 💿 6		on't Limit							
		🖿 🖬 🖗 🖗 🖗	i 🕐 🔂 🔗 🤅	9 🙋 🖻	on't Limit		74	🥑 🔍 ┨ 🗉				
Q Filter objects		1 • SELECT	* FROM evloco.C	arData;								
🗸 📄 evloco												
🗸 📷 Tables												
🗸 📰 CarData	a											
> 📷 Colur												
🔤 Index	kes											
> 📲 Forei		100% 🗘 30:1										
> 🏹 Trigg	jers				_			_				
> 🗾 tester		Result Grid 👖 🐧	👌 Filter Rows: 🔍 S		Export: 🙀 F	etch ro	ws:	 •				
> 🗾 TestTable		latitude	longitude	city	intersection_directions	plus4	state	street_address	zip	country		Result Grid
📷 Views		34.248319152719	3 -118.3879713743439	Sun Valley	NULL	NULL	CA	11797 Truesdale St	91352	US		
Stored Procedures		34.040539	-118.271387	Los Angeles	West hall and South hall	NULL	CA	1201 S Figueroa St	90015	US		
		34.059133	-118.248589	Los Angeles	Across Hope	NULL	CA	111 N Hope St	90012	US		
		33.759802	-118.096665	Long Beach	NULL	NULL	CA	6801 E 2nd St	90803	US		
Object Info	Session	33.770508	-118.265628	Wilmington	At B St	NULL	CA	161 N Island Ave	90744	US		
Table: CarData		34.30309	-118.480505	Sylmar	NULL	NULL	CA	13201 Sepulveda Blvd				
Table: Carbata		34.066801	-118.227605	Los Angeles	NULL	NULL	CA	1630 N Main St	90012			
Columns:		34.036777	-118.368841	Los Angeles	NULL	NULL	CA	2311 S Fairfax Ave	90016			
latitude		34.06872	-118.064	El Monte	NULL	NULL	CA	9530 Telstar Ave	91731			
		34.039736	-118.236065	Los Angeles	NULL	NULL	CA	1212 Palmetto St	90013			
		32.89947	-117.243	La Jolla	Patient Parking Structu	NULL	CA	10666 N Torrey Pine	92037			
longitude double		34.221665	-118.468371	Sepulveda	NULL NULL	NULL	CA	15421 Roscoe Blvd	91343			
		33.909914	-117.459053	Riverside	NULL	NULL		1299 Galleria at Tyler	92503			
city		34.14762	-118.147111	Pasadena	NULL	NULL	CA	150 E Holly St	92503			
text		34.14762	-118.152655		NULL	NULL	CA		91103			
intersection_directions text				Pasadena				45 De Lacey Ave				
		34.145119	-118.150133	Pasadena	NULL	NULL	CA	33 E Green St	91105 91030			
		34.118433	-118.149597	South Pas	NULL	NULL		606 Fair Oaks Ave				
plus4		33.7637	-118.192	Long Beach	NULL	NULL	CA	200 S Pine Ave	90802			Plan
		33.94325	-118.403342	Los Angeles	Central Terminal Area (NULL	CA	1 World Way	90045			
state		34.2387162	-118.554686	Northridge	NULL	NULL	CA	9301 Tampa Ave	91324			
		34.07122	-117.575	Ontario	NULL	NULL	CA	700 N Haven Ave	91764			
street_address		34.228315	-119.170865	Oxnard	NULL	NULL	CA	600 E Esplanade Dr	93030			
		33.824004	-116.543589	Palm Sprin	NULL	NULL	CA	400 E Tahquitz Cany	92262			
zip		34.0189	-117.38	Riverside	NULL	NULL	CA	5425 Wilson St	92509			
text		38.69005	-121.58889	Sacramento	NULL	NULL	CA	Lindbergh Dr	95837			
		34.017081	-118.44704	Santa Mon		NULL	CA	3223 Donald Dougla	90405			
country		34.010021	-118.49583	Santa Mon	NULL	NULL	CA	200 Santa Monica Pier	90405	US		
text		CarData 3									Read Only	
		Action Output 0										
		Time	Action					Response			Du	uration / Fetch Time
		3 20:13:55	SELECT * FROM evid	nco CarData				68446 row(s)	returne		0.0	60 sec / 0.660 sec
		20113133		Joo.ou Data				00440104(3)	Toturne		0.0	00.000, 0.000 300

2. The React app can be downloaded and run on your local machine by creating a terminal for the client and backend folder. Navigate to the server.js file in the backend folder and update the mysql connection variable with the ip address of the server running mysql, as well as the username, password, and name of the database. Cd into the respective folders in two separate terminals, run npm install, and then npm run start for both of them. <u>Get a Google Maps api key</u> and put it in a .env file.

3. When the React app starts, you will see a search bar and submit button, and when you enter and submit a zip code, a map will show up and be populated with the EV charging station closest to you.



Try it Yourself

Below you will find a link to our public github repo which can be used to download our code as well as instructions with the best way to leverage the code to create a similar application on your own!

A user can adjust the code to run on their local machine quite easily by making use of an .env file. However, since this is for a Cloud Computing course, we will focus on how to use the code to operate on AWS.

- 1. Download Code from GitHub repo and/or clone it to also make sure you can run the React app.
- 2. Create API keys for both for the National Renewable Energy Laboratory (<u>https://developer.nrel.gov/signup/</u>) as well as Google Maps API.
- 3. Create Access Keys and Secret keys for AWS account in order to use boto3 and access S3 buckets
- 4. Create S3 buckets and MySQL database by chosen names (our code points to the same S3 bucket object each time).
- 5. Create Lambda functions and put in Environment variables as described in the code for each lambda function.
- 6. Add in Lambda Layers via the zip files included in the GitHub repo.

- 7. Create Access Keys for the EC2 instance.
- 8. Update the environment variables in backend/server.js in order to connect to the MySQL database.
- 9. Create a .env file with your Google Maps API key.
- 10. Run npm run build in the client folder.
- 11. Push to your updated version of the git repository.
- 12. Create an EC2 instance in the same VPC as MySQL database and connect the instance to the database.
- 13. Install nginx, pm2, and node.js. Make sure you can see the nginx web page when you are at your instance's ipv4 address.
- 14. Clone your updated git repository into the instance.
- 15. Update the .conf file. Check to make sure the syntax is valid.
- 16. Use pm2 to start the server.js file. Check the logs using pm2 and make sure you can see the log saying that the port is running.
- 17. Install React code onto EC2 instance and update environment variables in order to connect to the MySQL database.

Link to public GitHub repo: https://github.com/mhangsterfer24/Team-Asteroid