Mattia Danese
Final Project: AWS Tutorial
CS118 - Cloud Computing
Professor Sambasivan

Deploying a Serverless Web Application

Ever since the beginning of the internet, the procedure of navigating to a webpage on a browser revolved around the client-server interaction. This server-centric deployment involves a server machine that *hosts* a webpage (i.e. stores the HTML, CSS, and JS code) and awaits for a client machine to *request* for the webpage. Upon receiving a *request* from the client, the server sends the corresponding webpage files and data to the client so that the client can successfully see and interact with the webpage. For the vast majority of webpage requests over the course of the internet's life, this framework has worked very well; however, it is by no means perfect. The most notable issue with this server-centric deployment is that accessing webpages is very dependent on the state and capabilities of the server. If the server machine is down or is receiving too many requests (i.e. a DDos attack), then it will not be able to respond to the requests of client machines and the webpages it stores will be unreachable. To combat this, many webpages that have high traffic or must be accessible at all times are stored on multiple servers so that there is still a live server in the case that one or a few servers go down. This, of course, comes at the cost of storage and the real costs of maintaining more machines. Other potential issues of server-centric deployment regard the necessary upkeep, maintenance, and configuring of servers. All of these potential drawbacks are hassles for programmers and developers to deal with as their sole interest is to simply create a web application. As such, dealing with the necessary server upkeep has either added to the work of developers or manifested a whole new maintenance team which brings extra costs and complicates the overall workflow of the company, team, etc.

An emerging deployment technology that has various advantages over server-centric deployment is known as serverless deployment. Serverless deployment is a development model used to create and run applications without physically configuring and managing a server. Instead, servers are handled by a third-party cloud provider which allows developers to solely worry about writing the code for the application or webpage. In essence, the *server* aspect of deployment is now abstracted out of the usual deployment model that developers and companies follow. In many cases, serverless deployment results in greater scalability, flexibility, and faster deployment times than server-centric deployment. These advantages, coupled with a fairly reduced cost, makes serverless deployment a very attractive option for many developers.

Amazon Web Services (AWS) is a cloud-service provider that supports serverless deployment. The architecture of AWS serverless deployment involves the following AWS services: AWS Lambda, Amazon API Gateway, Amazon DynamoDB, Amazon Cognito, and AWS Amplify Console. AWS Amplify Console is responsible for physically storing the application or webpage files (HTML, CSS, JS, images). It is essentially a remote repository

which automatically builds and deploys the web application upon receiving new pushes (i.e. code updates). Amazon Cognito allows for user management and authentication in order to keep the backend API secure. This allows for users to securely create an account and log into the web application. Amazon DynamoDB provides a NoSQL database layer which stores data at the discretion of the backend API; most notably user credentials (i.e. usernames and emails), but can also store application-specific content. Finally, AWS Lambda and Amazon API Gateway work together to configure a backend API responsible for sending and receiving Javascript data to and from the client browser. In essence, both these services allow for user interaction with the web application and sustain communication between the client and the *server*.

AWS provides a very comprehensive and brief tutorial on how to configure serverless deployment. The first step (Host a Static Website) involves uploading the static webpage files to AWS Amplify Console and configuring Amplify to support continuous deployment. The second step (Manage Users) regards creating a user pool with Amazon Cognito in order to configure which credentials a user must provide when making an account (in this case it was email and password). The third step (Build a Serverless Backend) constitutes developing a backend process with AWS Lambda which handles client requests for the webpage. The fourth step (Deploy a RESTful API) masks the backend process created in the previous step as a public RESTful API so the client can interact seamlessly with functionality originating from ArcGIS. Finally, the last step (Terminate Resources) cleans up the resources used in the tutorial.

This tutorial was very intriguing and I definitely learned a lot. To begin with, I familiarized myself with many different AWS services that I did not even know existed beforehand. I am now more than comfortable using these services again and I even see myself able to explain these services and what they do to others. Something else that I learned, which is a bit surprising, is exactly the steps needed to deploy a web application. I have deployed many web applications in the past which all have used Node.js as the backend and, in my opinion, the modularity and organization of most Node.js backends is always a bit of a mess. With AWS, however, the different services provided the perfect amount of modularity and abstraction: I knew exactly what each service did (and they usually just did one thing only which is even better) and I was not overwhelmed with writing the fairly complex code needed for networking or authentication. Overall, this was a very fun experience and has inspired me to learn more about AWS and the countless services it offers.

The recording of the end result can be found here. One thing I would like to note about the recording is that the penultimate step of interacting with the ArcGIS map did not work. I tried to showcase this in the recording, but the necessary script files were not included in the S3 bucket given in the tutorial. The bucket can be found here and, based on the console error messages (see recording), the files `amazon-cognito-identity.min.js.map` and `aws-cognito-sdk.min.js.map` (the supposedly only two `.map` files) should be in the `/js/vendor/` directory but they are not. I tried to find them on the internet and push to the Amplify repository but had no luck.