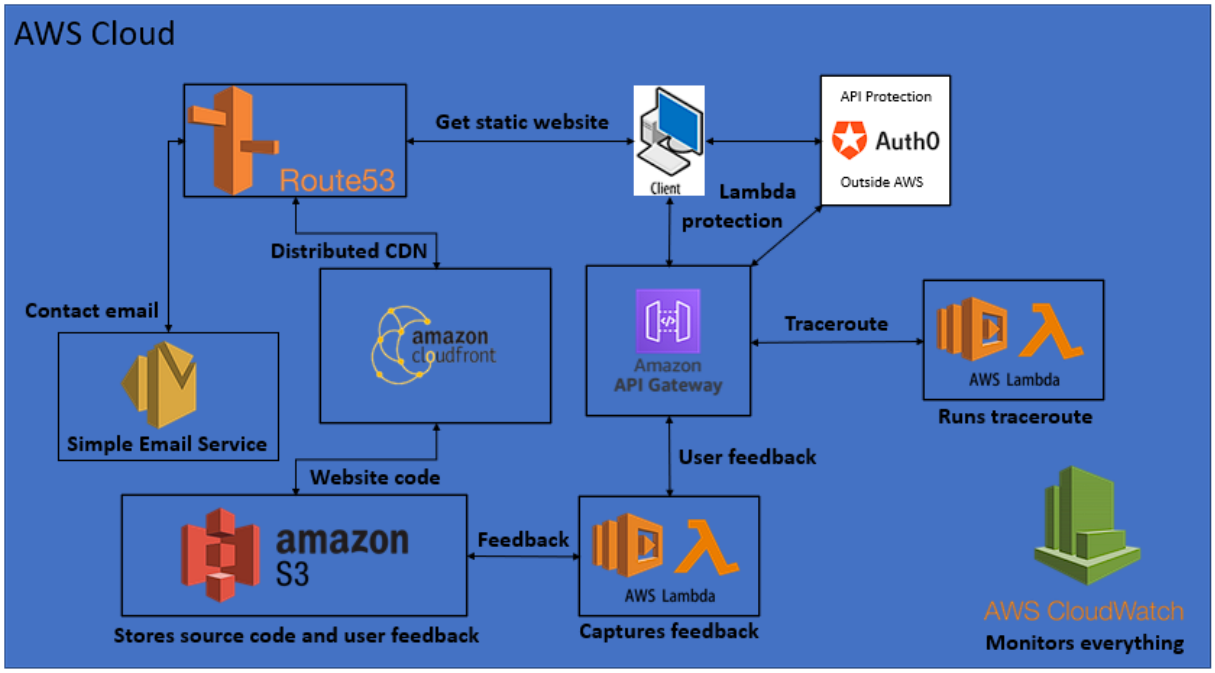
Project Summary:

For my term project I decided to implement a personal website, including some simple frontend networking tools, to gain hands-on experience in sone of the principles taught in class. The goal of the project was to set up a basic website frontend, host the site securely and inexpensively, and implement all of the extra frontend and backend feature ideas. This implementation report covers the final architecture of the project, the core functions and features of the website, the process that I followed to get the project to its end state, and the lessons I learned along the way.

Architecture Diagram:



Architecture Summary:

The user accesses the site by entering www.patrickmuradaz.com into their browser of choice. Their browser (client) starts by looking up the IP address of my website using DNS. The domain information for my website is stored in AWS Route53 and is provided to the client when a DNS lookup is performed. The client is then directed to the closest machine in the CloudFront CDN that I set up for the site. The CloudFront distribution caches versions of the original website source code (stored in S3) in many machines distributed around the US and EU. AWS CloudFront not only reduces physical distance (and thus propagation times) between

clients and my site, but it also allows the site to implement the https protocol using SSL certificates.

When the user wants to use the traceroute tool, or provide feedback on the site, their client first reaches out to the Auth0 API. I have set up an API through Auth0 to provide JWT Bearer tokens to clients accessing my site for use in requesting my AWS API. Once the client has used the Auth0 JWT to authenticate with the AWS API, the requested lambda function gets kicked off. The traceroute lambda function simply runs traceroute from my AWS VPC to the user's client IP. The feedback lambda function gathers client info and deposits it into either the positive or negative feedback S3 bucket (depending on user choice).

When the user wants to send an email to my contact address (contact@patrickmuradaz.com), the message is first sent to AWS, then forwarded to my personal Gmail. Rather than set up a full email server, I decided to move forward with AWS Simple Email Service. I set up the proper MX routing records in route53, so any email sent to my contact address is routed to my Simple Email Server. The SES then does some simple spam checking on the email and generates two new emails: one as a response to the user, letting them know I got their email; and the other to myself, with the contents of the user's email.

Aws CloudWatch monitors all of the systems involved in the architecture and I have set up a metrics dashboard to provide a "status-at-a-glance" for everything the site uses (including recent user feedback).

Functions and Features:

The site I put together for this project has five main features, described below:

1. Simple Cookies

   The site collects two kinds of information about the client and stores them as browser cookies. The first kind of data collected is when the user last visited the site. This info is then displayed to the user along with a welcome message. The second kind of data is the users IP and associated attributes (ISP, geographic location, known threat profile, etc.) which are gathered from a free API.

2. User Feedback

   The site allows users to provide simple positive/negative feedback on their experience with it, using buttons on the welcome popup. When a user provides feedback, their IP information is collected from the cookie and sent to either the positive or negative feedback S3 bucket, depending on their selection.

3. Simple Traceroute Tool

The site allows users to run a simple traceroute tool to see how many, and what kinds of, routers lie between the site and their client. The traceroute tool runs in AWS lambda and traces from my AWS VPC to the client.

4. User IP Information Display

   The site displays all of the aforementioned IP information to the user, including their approximate location on a google map plugin. All of the information gathered from the client IP comes from a free API that anyone can use (https://ipdata.co/). The displayed information comes with the warning "You should assume that every site on the internet knows at least this much about you".

5. Simple Contact Email Service

   The site has a contact section at the bottom of the page. This includes a contact email address that users can use to get in touch with me. Email to this address is routed to AWS SES and then forwarded to my personal email.

Design Flow:

The project implementation began as a series of frontend and backend ideas. The frontend ideas were: 1) build a tool to run traceroute between client and AWS host; 2) display the client's geolocation using their IP; 3) use cookies to show the last time that client visited the site; 4) allow the user to give feedback on the site and store that feedback in S3. The backend ideas were: 1) set up a contact email using the site domain; 2) set up some automated reporting system through AWS.

The initial implementation began on 10/8 when I set up the initial S3 bucket and domain records on route53. However, I had significant issues with the SSL certificate when setting up the CloudFront distribution. After banging my head into the wall for a while I deleted everything on 10/17 and got the initial site hosted and working with CloudFront.

On 10/21 I started setting up the contact email on the backend and started setting up the look of the tools on the frontend. By 11/2 I had the initial traceroute tool, client IP info display, and cookies system working. However, I noticed the traceroute tool seemed to be getting stuck at one of the outbound routers (which I would later handle by using Dr. Klein's suggestion of displaying a message like "your ISP is blocking my traffic" to the user). By 11/6 I had all four frontend tools working and formatted and began focusing exclusively on the backend.

Over the next few weeks, I worked on creating the automated CloudWatch monitoring system and dashboard, finishing the setup of the contact email, and protecting my AWS API using Auth0. By 11/14 I had finished the CloudWatch and SES work but was still having issues

with the API authentication. I was able to figure out the issue (the scopes between the AWS and Auth0 APIs were mismatched) and finish all backend work on 11/15. I then spent another few days on finishing touches and making sure everything worked “in production”.

Lessons Learned:

While working on this project I feel as though I was able to especially crystalize my understanding of DNS, network security, and the HTTP protocol. This deeper knowledge has already been extremely beneficial to my professional work.

I was able to essentially set up my own DNS table for my own domain using route53. The record set for my domain contains entries for all of the AWS authoritative domain name servers, my CloudWatch distribution servers, and my simple email server (inbox and outbox). I am now grappling with DNS tables in my work, and I feel much more equipped to handle that having first done this for my personal site.

I was also able to mess around with implementing SSL certificates for the website and JWTs for the backend API. Understanding how SSL certificate signature chains work, from original signing authority to application specific certs, has been extremely helpful to my job function recently. I have had to manually update SSL certs on several of my client’s servers in the past weeks and, again, being able to do something similar in the “low-stakes” setting of this project helped he immensely. I have not had the opportunity to take my JWT implementation knowledge to my work yet but I’m sure it’s just a matter of time.

Finally, through setting up my AWS API I became much more familiar with the HTTP header fields and request/response types. Once I set up JWT authentication the major headache that arose was a CORS error (which I had solved when the endpoint was unprotected). I learned that on Cross Origin Resource requests, most browsers will send an OPTIONS request first, to determine if the resource is configured to allow CORS. I had configured my AWS API to allow CORS from my domain. However, I had also configured it to accept and authenticate ALL incoming traffic. This was leading to an issue with the response headers on OPTIONS requests, which I learned was because of auth failure (since these are sent from the browser and don’t contain my JWTs). Thus, after much head scratching, I redesigned the AWS API to accept only OPTIONS and POST requests and to only authenticate POST requests (OPTIONS requests would simply reply immediately with an empty body and the access-control-allow-origin header included). This knowledge gained has helped me somewhat with my professional work so far, but I expect it to serve me very well as I progress in my career.

All in all, I feel as though I learned a tremendous amount in this class, through the textbook readings, class lectures, and projects especially. I’m looking forward to carrying this new knowledge with me through the rest of my time in the Vanderbilt program. I’m also

excited to grow my understanding of Computer Networking principles and technologies as they and I evolve during my career.