Deployment Problems I faced with AWS

Deployment to AWS with my Minimum Viable Project led me to many dead ends. So I'm going to try and log the ones I remember. I should have made this from the beginning, but I didn't, so there will definitely be pieces that I don't remember or that I can't remember the exact problem I worked around

I originally started my project with a frontend and a backend. The frontend would be made with React and Javascript, while the backend would be made in Flask. The plan was to have the user upload an image to the frontend, and as a post request that image would be sent to the backend, with bounding boxes being returned to the frontend.

The backend would find these bounding boxes by importing a pretrained model like Faster_RCNN from PyTorch and run a prediction task on the input image. It worked on my machine, so I dockerized the flask app, and I built the React app separately. From an article I found walking through deployment on AWS, I planned to upload the React app bundle to an S3 bucket, which would then be hosted on CloudFront. So far this all worked.

The issue came when I tried to upload the docker image to the Elastic Container Registry (ECR). The hope was that I could upload the docker image to ECR, then use Elastic Beanstalk (EB) to host the backend. Unfortunately, in the beginning, my docker image was over 7 GB in size, with one layer of the image being 5 GB. As each layer was being uploaded to ECR, it would constantly stall and restart sending the largest layer, then eventually complain that the connection had timed out, or an unexpected EOF was encountered.

I spent some time trying to reduce the size of that docker image, eventually relying on having pip only install the cpu version of PyTorch, which reduced the largest layer size to roughly 1 GB, with the total size of the image being around 2 GB. This was capable of being pushed to ECR, but it would take several tries each time, which wasn't very consistent. I didn't like this solution much, as it means I wouldn't ever be able to publish an app that relies on PyTorch GPU functionality, and I feel like this issue isn't discussed much online, so I feel I may have being doing this process wrong from the beginning, but since it worked I tried hosting it with EB anyways.

Unfortunately, it would successfully connect to the docker image, but would have a health status of SEVERE, which seems like a problem, and I ran into many different issues with trying to send images between the hosted frontend and backend. Sometimes it would respond with an HTTP gateway timeout issue, and sometimes it would complain that a CORS policy wasn't allowing the data transfer to complete. Having gone through weeks of this with no progress really being made, I decided to change my approach to the project for now. At least until I get a working version hosted on AWS.

So I transitioned into using Tensorflow.js, where I could hopefully cut out Flask entirely, and do all the inference on the frontend. No more complications with sending images over HTTP. After fighting with some npm dependency errors and a frankly confusing bit of documentation for how to use the pretrained models, I was successfully able to get a prediction on an input image, and overlay the image with the resulting bounding boxes and labels.

With this approach, hosting the web app on AWS became as simple as uploading the build package to S3 and hosting the index.html file on CloudFront. Which was the part of my previously attempted hosting trials that

I knew worked. So this was a considerably easier and faster method for deployment. While I'm not fond of the user having to download the model files in order to get predictions, I'd rather that be done on the backend in an AWS cluster after all, it seems like models trained in Python can be converted to a format that tensorflow.js can understand. With some experimentation, I had trouble getting that to work for me, but theoretically it would allow model exploration in a Python notebook to be transferred directly to the web application, which is nice.