CS 470 Final Reflection

Scott Smith

Southern New Hampshire University

CS470: Full Stack Development II

Professor Watson

February 15, 2022

https://youtu.be/PfNhA3POQ8o

Experiences and Strengths

Back-to-back courses in full stack development have greatly influenced my understanding of various web technologies including: REST APIs, frontend/backend technologies such as Angular, Express, and Node.js, as well as the cloud development models of containerization using Docker, and serverless compute via Amazon Web Services (AWS). I have learned the intricacies of full stack development by implementing a MEAN Stack application locally, then built upon this knowledge by migrating a full stack application to the cloud using both models of cloud development.

As a software developer, I believe that being inquisitive toward the nuances of software development has been my biggest strength. I don't settle on piecing components together and hoping for the best. Working on complex applications such as a full stack web service requires composition between various technologies, using separate frameworks that change often during updates and vulnerability mitigations. Being able to conceptualize the connections and passageways of various frameworks, APIs, and processes has enabled me to adapt to this ever-changing atmosphere. For example, when accessing specific API endpoints, I like to analyze the sequence of operations step-by-step within the code, verifying that I'm fully aware of how data is being transferred starting from request to return as a response.

Full stack development is very intriguing to me, there are really no limits to what can be achieved through a well-performing web application. These concepts are not far off from most of the technology we use in our daily lives either, ranging from simple smart phone apps to Amazon Alexa. I have no preference of where I would see myself fitting in role-wise when it comes to being creative as a professional. Every aspect of development is crucial to the success of a software service, I’ll be happy in any of the various roles of the SDLC including planning, development, testing, deployment, maintenance, or security. I’m ready to get my hands dirty!

Planning for Growth

Hosting an application in a cloud environment is an efficient way to plan for future growth, but not without some challenges. There are two popular cloud development models to consider: microservices using containerization and serverless compute. In a microservices environment using containers, we deploy several lightweight applications which are linked to one another over a defined network. In essence, this creates a simpler engineering environment, having unique components of the application isolated from one another teams can individually focus on and ensure distinct services are processing inputs and producing outputs as required. In terms of error handling however, each service must be capable of reacting correctly to specific faults, retrying when facing issues of connectivity or system outages, or alerting using some sort of console logging for bugs and other long-term errors. In terms of scale, multiple instances of a service can easily be generated through Docker using the Docker Compose tool. We can also define a load balancer such as NGINX to evenly distribute incoming requests between the different instances.

Things change in the serverless environment. Scaling is performed automatically when using AWS. Resources are acquired as needed and released when no longer needed. We can also customize levels of resource consumption (memory, CPU, network) to verify our services are always operating as best as possible. Handling errors can be done using AWS Step Functions. Instead of embedding error handling code into Lambda (our compute service) functions, we can configure Step Functions to respond to workflow runtime errors, automatically trigger and track each step, retry executions (when necessary), as well as log the state of each step for debugging purposes.

AWS provides services for both approaches to cloud development, EC2 for containerization and Lambda for serverless compute. Each of these services integrates with AWS Cost Explorer which provides an intuitive interface for visualizing, understanding, and managing AWS costs over time. Cost Explorer can produce monthly reports with detailed service breakdowns as well as monitor usage at an hourly or resource level granularity. Savings plans can also be purchased for long term pricing models that offer substantial savings based on past usage patterns. These built-in analytic tools make both EC2 and Lambda valid choices for deployment and future growth by offering informative data for predicting future costs. However, developers should weigh the two based on the level of infrastructure control needed for a specific application. With EC2, we are paying for server instances, literally full operating systems complete with virtual copies of hardware, whereas Lambda only utilizes the system resources and dependencies needed to run a program. These factors will greatly influence the total cost overhead.

Deciding which approach is best comes down to how an organization chooses to operate moving forward. Using a container approach is not so different from using local infrastructure. Of course, servers must still be accessed via the internet, but all infrastructure management and provisioning are still in the hands of the developer. It is also easier to pack up a fully developed application using containers and ship it as needed for scale. Serverless requires more effort in rearchitecting an application to make best use of the serverless environment. The domain uses its own APIs and technologies to provide cutting-edge functionality that takes away the burden of server provisioning.

Local infrastructure has its benefits for protecting private data, but it’s often challenging to determine capacity requirements based on usage. AWS offers a level of elasticity that is difficult, if not impossible, to achieve locally. Resources will be allocated to match what is needed automatically and retracted when necessary, taking out the guesswork while planning capacity. We only pay for the compute time we use, and we don’t pay for anything when our services are not running. Each service has its own customizable payment model with on-demand options, we choose and pay only for exactly what we need. AWS integrations also help us optimize our costs by providing services to organize and track usage to enable better planning through budgeting and forecasting. For those of us that aren’t interesting in buying and selling mainframes like there’s no tomorrow, AWS is the easy and convenient alternative.