Jessica Kilbourne 2/26/21

CS 470 Final Reflection

Access to Project

I cannot upload my project to GitHub due to security risks involving the nature of cloud services and security features which create unique ID's. Please feel free to see a ten-minute presentation I made on my full stack application here: https://youtu.be/AvlRVQMoXeg

Strengths & Experiences

The full stack development course helped me to prepare for future opportunities in the management of migrating an application to a serverless cloud. I learned about and experienced the skills required for cloud software management, such as how to use docker to create images, how to use docker compose to orchestrate containerization and prepare my code for migration, and how to utilize cloud provider services such as Amazon Web Services (AWS) and configure their software to provide services to an organization. The course had a focus on serverless infrastructure using Functions as a Service (FaaS), but I understand that it is only for certain use-case scenarios such as being ideal for start-ups and small projects. Consideration to all requirements is needed to aptly suit an organization to an infrastructure.

As a developer my strengths lie in understanding various program languages and having the ability to fulfill a business need with those languages. I'm well versed in Java, Python, and C++ languages for various types of applications whether they are for the frontend, backend, middle or GUI related. Middleware, such as Node.js, Mongoose, Seedgoose and others, are also some tools in my toolbox that I feel comfortable with along with the Linux OS and commands. I am less acquainted but experienced with HTML structure, TypeScript and other various structure languages. I'm well versed in query languages such as relational database-oriented SQL and

MySQL, and also non-relational database notation such as MongoDB and DynamoDB's JavaScript Object Notation or JSON.

Roles which I have already been prepared for are that of developer, tester, or any role on a development scrum team. I know database and programming languages, I understand Linux OS's and commands, I know how to use cloud service providers and how to create microservices and applications with them. I only lack certification in information security which is easily overcome. I could also easily be a cloud administrator. I could learn C-Sharp or .NET if necessary. I am knowledgeable of one DevOps tool, Docker, and can learn others. I have a grasp on database configuration and can handle cloud infrastructure systems such as servers, storage, network, and visualization (I am familiar with visualization through my CS 330 course). I see these as beginner roles, where I will start small and prove myself.

One role that I could work my way up to is cloud engineer. A cloud engineer is the role most suited to my current skillset. The required languages match my knowledge of languages, as it requires Java, Python and C++. Although I do not know Ruby. Technical knowledge of Linux, cloud service providers and how to create RESTful services also matches my skillset. I also have some virtualization experience from my first full stack class, but I need further experience with optimizing VM's to reduce hardware units and, most likely, more certifications with service providers and general experience in the field.

Planning for Growth

AWS Lambda has what they call invocation scaling, as a function is invoked while an instance is already running, Lambda initializes another instance. My functions *concurrency* is the number of instances that serve requests at a given time, and as they reach the quota of 3000 (for US East) from the initial burst of traffic, the concurrency can continue to scale by 500 instances

each minute. The function scales until all requests are served or until a concurrency limit is reached. Additional requests will fail with a throttle error and return a 429 status if I hit the limit or if requests come in faster than my function can scale. One option is to submit a request to increase the limit for my account's concurrency. An additional option is to reserve concurrency by allocating capacity on a per-function basis, which allows me to serve a burst of traffic with very low latency. This can be taken a step further with Application Auto Scaling using a target tracking scalable policy which provides autoscaling for provisioned concurrency. (AWS Invocation-scaling, 2020)

When it comes to error handling, I can write and maintain error handling logic in each of my Lambda functions to handle API throttling, timeouts or memory issues. To avoid complications and lower the amount of error handling code needed I can use AWS Step Functions to build a workflow that bolsters function error handling. I can use conditional logic to configure the Step Function based on the error type that occurs, which separates my workflow logic from my business logic. Doing this allows me to modify how my workflow responds without changing the business logic of my Lambda function. (AWS handle-serverless-application-errors, 2020)

I can calculate costs, a critical task, using AWS simple Monthly calculator or AWS Cost Explorer but it can take time and requires knowledge of many system metrics to make accurate predictions. I can also run tests, but they require a day to run before you can analyze AWS cost/usage reports. I would rather start with a tool called AWS Near Real-Time Price Calculator, a tool by Ernesto Marquez, a certified AWS Solutions Architect. This tool saves time and is ideal for estimating AWS costs at scale. The tool has its limitations but fits my scenario. I can use it in my test environment while doing load tests at an expected volume. Not only does it consider response times and throughput, but it helps to quickly correlate system metrics and performance

with AWS costs. I can also use it in a production environment to quickly expose cost anomalies for my specific application or stack. (Marquez, 2020)

My project was built for a startup company that lacks the funds for on-premise servers or the technicians needed to maintain them. My application is also meant to be public, and according to engineer Gaurav “serverless computing and FPaaS (function platform as a service) are best suited for running applications in the public cloud and containers are useful in transforming on-premises hardware resources into a private cloud” (Yadav, 2018). Containers can be more effective in testing (thus more effective at predicting costs) when “the build task starts taking a large amount of time or testing involves long-running operations”. When this happens, it may indicate that serverless architecture is not for you (Yadav, 2018). I believe serverless is more cost predictable for my use case because it lowers overall costs, increases scalability, and decreases time to deployment and management needs from IT teams. Whereas containers increase overall costs and deployment time and are harder to scale. (Yadav 2018, Doerrfeld, 2017, Carlson, 2019). The best solution may be to start with cheap local containers (if possible), then run your tests and cost analysis and decide whether it’s right for you.

Due to the event-driven on-demand service that serverless provides I feel that serverless FaaS is currently the way to go. The pros to this service currently outweigh the cons. Pros to serverless are that it is very elastic: its ability to grow and shrink on demand generates zero waste, unlike its VM-based counterparts, due to its pay-as-you-go pricing model where you only pay for compute processing time. The time it takes to process a request is often less than one-third of a second and Lambda charges only point-two-millionths of a dollar per request. There’s no maintenance, your dev team doesn’t have to waste time managing resources and can instead be more productive creating code that produces products and thus revenue for a business. But its

downsides are that you have no control over underlying EC2 resources and configurations making it slightly harder to optimize performance. Another con is that it is slightly less secure than VM based counterparts.

Bibliography:

https://docs.aws.amazon.com/lambda/latest/dg/invocation-scaling.html

https://aws.amazon.com/blogs/architecture/scale-your-web-application-one-step-at-a-time/

https://aws.amazon.com/getting-started/hands-on/handle-serverless-application-errors-step-functions-lambda/

https://www.concurrencylabs.com/blog/calculate-near-realtime-pricing-serverless-applications/

https://containerjournal.com/topics/container-ecosystems/serverless-computing-vs-containers-which-to-choose-for-cost-and-benefits/

https://nordicapis.com/the-benefits-of-a-serverless-api-backend/#:~:text=But%20for%20developers%2C%20what%20serverless,No%20more%20over%20capacity%20issues&text=You%20don't%20pay%20for%20idle%20time

https://dis.co/blog/serverless-computing-vs-cloud-computing-whats-the-difference/