CS 470 Final Reflection

https://youtu.be/Yn3NbwFB3_8

Sylvia Trynkin

12/16/2024

<u>Experiences and Strengths: Explain how this course will help you in reaching your professional goals.</u>

What skills have you learned, developed, or mastered in this course to help you become a more marketable candidate in your career field?

In this course, I have worked with AWS Lambda, DynamoDB, API Gateway, S3, Docker Compose and IAM. I gained experience in how to build a serverless application. While working with the services mentioned, I created scalable and secure APIs, S3 buckets, Docker containers, Lambda functions, DynamoDB tables and IAM policies. I believe I gained a solid foundation in cloud computing, though there is always more to learn. Working with cloud computing is essential for modern software development and therefore this course was of great value also for my future career.

The major skills I gained working on this project are:

- Lambda Functions: I created Lambda functions which can handle important CRUD operations on DynamoDB, but we did not have to manage any infrastructure.
- API Gateway Integration: I configured GET, POST, PUT and DELETE methods, configured access controls and permissions and integrated APIs with Lambda functions
- DynamoDB: I experienced working with a fast and scalable NoSQL database (querying data, handling data with Lambda functions).

- CORS Management: I learned what cross-origin resource sharing (CORS) in AWS API Gateway is and how to manage it, so applications can securely interact with APIs from different domains.
- Serverless Architecture: I learned about serverless design patterns and how events trigger a response (executing code) from AWS Lambda and how API Gateway brings it all together.

The skills I gained in this course make me a better and more marketable candidate for future job offerings especially in the cloud computing and Full-stack development field. This is an important building block especially when applying at companies utilizing serverless architectures or looking to do so in the future.

Describe your strengths as a software developer.

As a software developer, one of the strengths required is to be able to learn and adapt to changes (such as new technologies) quickly. I think this course required quick learning and understanding to be able to connect the dots how serverless web services and applications work together and how to build and deploy an entire cloud-based system. I have proven my ability in problem-solving, as well as utilizing cloud resources efficiently, which my presentation (see link on cover page) is proof of.

I understand the big picture of serverless architecture and utilize my knowledge to use the most suitable tools and solutions for our web application. This has been essential for working with microservices, Lambda functions, and DynamoDB in this project.

Identify the types of roles you are prepared to assume in a new job.

This course prepared me to assume roles like:

Full-Stack Developer: with the responsibility to integrate the backend (cloud services) with the front-end (web applications)

Cloud Engineer: being responsible for designing and maintaining serverless infrastructure as well as cloud-based APIs.

Software Architect: working on designing scalable and very efficient cloud-based architecture.

<u>Planning for Growth: Synthesize the knowledge you have gathered about cloud services.</u>

Identify various ways that microservices or serverless may be used to produce efficiencies of management and scale in your web application in the future. Consider the following:

How would you handle scale and error handling?

Scaling:

Based on the number of incoming requests, serverless architecture can automatically scale up or down. AWS Lambda also scales automatically based on event triggers (such as HTTP requests). When I utilize AWS services such as DynamoDB and API Gateway (just to name a few) my application handles increased or decreased workload on their own, without any necessary manual adjustments.

For microservices, scaling happens by adding more or reducing the number of containers and instances. In a microservice architecture different components of an application take care of the scaling independently.

Error Handling:

There are several fronts to tackle error handling in a serverless environment. For example, AWS Lambda offers tools such as retries, logging (CloudWatch), and alarms to handle errors. APIs need to have fallback procedures defined. We can also utilize the AWS Step Functions in case of errors.

In microservices, we can handle errors through centralized logging, retries, and circuit breakers.

How would you predict the cost?

With serverless (services like AWS Lambda) costs are based on the Pay-for-use model (pay for the number of executions and the duration of each execution). Therefore, cost is directly linked to elasticity (scalability) and though this means variability in cost, you can be sure you only pay for what you use.

Microservices especially if utilizing containers, are more predictable in terms of costs. You are paying for a fixed amount of compute resources (like EC2 instances or reserved clusters) to run your containers. Auto-scaling is possible and together with resource allocation, this can help reduce cost.

What is more cost predictable, containers or serverless?

In general, I think Containers are more predictable because you are paying a fixed amount for the compute infrastructure (regardless of any usage spikes).

In contrast, serverless is more cost-efficient (as you only pay what you use) but less predictable as charges are based on the actual number/duration of executions.

Explain several pros and cons that would be deciding factors in plans for expansion.

<u>Pros and Cons of Serverless:</u>

Pros:

- Uses Pay-as-you-go model (cost-effective most of the time)
- Automatic scaling – results in reliability
- Less administrative work (no infrastructure management)
- Great fit for event-driven applications.
- Faster deployments

Cons:

- Cold starts (initializing serverless functions) can take some time, so latency can be an issue
- Limited control over underlaying infrastructure.
- Possibly harder to monitor and debug serverless applications

Pros and Cons of Microservices (Containers):

Pros:

- More flexible regarding programming languages, DBs, frameworks etc.
- Increased resilience as the failure of one service does not automatically impact others
- More predictable costs with dedicated resources.
- Better for complex or long-running tasks.
- Allows independent scaling of services.

Cons:

- Infrastructure management is more complex (e.g., managing clusters, load balancing).
- more overhead for orchestration (more resources and infrastructure as multiple services are used).
- Possible communication issues between the several services used.

What roles do elasticity and pay-for-service play in decision making for planned future growth?

Elasticity is imperative for handling peaks and lows in workloads. It automatically scales up or down based on need. It is a great advantage of serverless systems (like AWS Lambda) and microservices architectures and can result in improved reliability and availability.

Pay-for-Service models, (especially in a serverless environment) are very beneficial as you only pay for what you actually use, especially for very unpredictable traffic patterns. But it is worth mentioning that for frequent high workloads, Microservices (Containers) might be more cost-effective in the long term.

<u>**Conclusion:**</u>

This course has prepared me well for working with serverless architectures and cloud-based services (like AWS Lambda, API Gateway, and DynamoDB). It gave me the opportunity to better positioning me in the professional world as a candidate for positions including cloud development and API integration. The invaluable knowledge I've gained about microservices and serverless computing in this course will be a big part in my future work to meet the growing demands of modern businesses.