CS 470 Final Reflection

Ian R. Hefflefinger

Friday, April 2022

Southern New Hampshire University

https://www.youtube.com/watch?v=3fxWQ8empts

Experiences and Strengths

Throughout this course, I learned a great deal about a variety of DevOps processes. I learned how to use Docker to create containers and in doing so a way to mobilize application parts. I learned how to use Docker Compose, an orchestration tool, to tie together multiple containers that rely on each other to form a unified application. I also learned how to use Amazon: S3 to store front-end files, Lambda to process logic, DynamoDB to store data, and API Gateway to handle incoming API requests/routing/security.

One of my strengths as a software developer is utilizing various resources to find information on how to use a new technology. The software engineering landscape is changing rapidly and requires one to learn fast - this is something I learned how to do well in IT and has carried over into my development skills. I also have a strong strength in problem-solving - solving puzzles might seem like busy-work, but it develops real problem-solving skills that are applicable in the business world.

With a history working in IT, a Computer Science degree with a focus in Software Engineering, and the work I did in this course, I am easily qualified for many DevOps roles. I am, however, looking for a software development role and I feel I am prepared to assume a junior developer role in web, server, or desktop application engineering.

Planning for Growth

Micro-services allow for software to be segmented into individual parts in an effort to separate concerns - an idea that allows for software to be more compartmentalized and simpler.

Micro-services can be used to handle specific tasks such as authentication, data-processing, and file transfer.

Server-less design is ideal for situations where you cannot or should not be managing your own servers for an application. This type of architecture benefits companies that cannot manage constantly upgrading/downgrading to scale their applications and companies who shouldn't concern themselves with developer operations.

Server-less tends to lead to a more predictable cost as it scales dynamically and you are typically charged for the services you use - in a container managed setup, you pay for the tier of CPU and storage whether you use it or not. Containers are more efficient than a locally managed setup, but server-less is even more efficient than containers. The benefit of using containers, though, is more control and mobility with your application.

Scaling can more-or-less be ignored with many server-less designs as the scaling is abstracted away and handled by the platform. Containerized applications can be scaled manually by adding more containers and configuration or by using an orchestration tool like Kubernetes to manage the load.

In deciding which solution to use, control plays a huge role. A server-less setup handles scaling for you, but you give up control. On the other hand, containerization gives you plenty of control and some options for scaling, but scaling is a little less consistent and requires more management.

Elasticity is extremely important because it prevents you from paying for resources that you are not using and prevents your application from crumbling under a load it cannot handle with its current resources. An elastic solution will ensure you are always using only what you need.

Pay-per-use is an excellent model in that it is fair and efficient. AWS, Azure, and other cloud service providers are excellent at making sure their resources are managed well, so only the bare minimum resources go to waste - this is akin to riding a bus instead of everyone driving their own cars and wasting portions of their own resources. This model is also fair in that you pay for only what you use and what you don't use can be utilized by other customers for other reasons - also, much of the "wasted" resources can be used for processing that is low-fidelity and can be done in chunks when resources are available as opposed to when needed (ie data processing for relatively unimportant things).

Choosing a solution can be difficult, but there are many options out there. Local managed systems give total control, but are inefficient. Containers allow more efficient resource utilization and allow for mobility, but are no the most efficient. Server-less solutions are typically the most efficient, but you give up control over your application. In the end, no one solution fits all and each should be addressed on a case-by-case basis.