Andrew Bach  
6/30/2024  
Module 6.2

In chapter 13 of the DevOps Handbook, we dived into a case study surrounding Amazon and changes to their architecture in the early 2000s. In this report, I will be discussing monolithic and microservice architectures and how Amazons changed to microservices from a monolithic system resulted in several lessons learned.

In the case study “Evolutionary Architecture at Amazon (2002)” we learn a little bit about the architecture Amazon was originally built on. In 1996, Amazon was built on a monolithic architecture. This meant that all of Amazon's business logic was tightly tied together in a single application. This application, called Obidos, grew to a point where things were too tangles and scalability was suffering due to the complexity of maintaining the monolith.

To address this issue, Amazon was one of the first companies to research and change into a microservice architecture. This meant that, instead of all the logic being tied up in a single project, Amazon broke up the logic into many related entities. These services handle a smaller amount of business logic and allow the program to mostly function even when one service is down. This made their code more manageable and scalable moving forward.

This case study teaches us a couple interesting lessons. The first being that microservices allow us to achieve a higher level of ownership over a process and provide more control. Second, by moving database access to microservices, the client no longer has direct database access through the client. This resulted in increased scalability and reliability of the platform. If 1 microservice were to crash, the site could be build to attempt to handle the downed service instead of crashing. The final lesson learned was that microservices enable us to develop and make more commits. In 2011, Amazon was doing around 15000 deployments, but after switching to microservices and dedicated teams for each service, Amazon now makes 136,000 deployments a day.

In conclusion, mircoservices allow us to break up business logic into smaller pieces making them more maintainable and scalable, while also increasing the pace at which we can deploy changes.