Lab 2, Microservices at Amazon

Felix Wei

Amazon began as a monolithic app when its size and scope were small. However, as they grew, they needed to scale efficiently to fulfill their expanding business needs. Thus, they began the intensive process of transitioning to a microservice-based architecture.

Breaking down the larger systems into smaller, independent services allowed each one to focus on a single functionality. For instance, Amazon created separate microservices for the "Buy" button, tax calculators, and other site features.

The dev team responsibly for that service could more easily identify bugs, maintain, and upgrade code. Teams could specialize heavily into one specific area of knowledge. The new design also made for a more flexible structure where you could create more unique combinations of products by mixing and matching the best individual services.

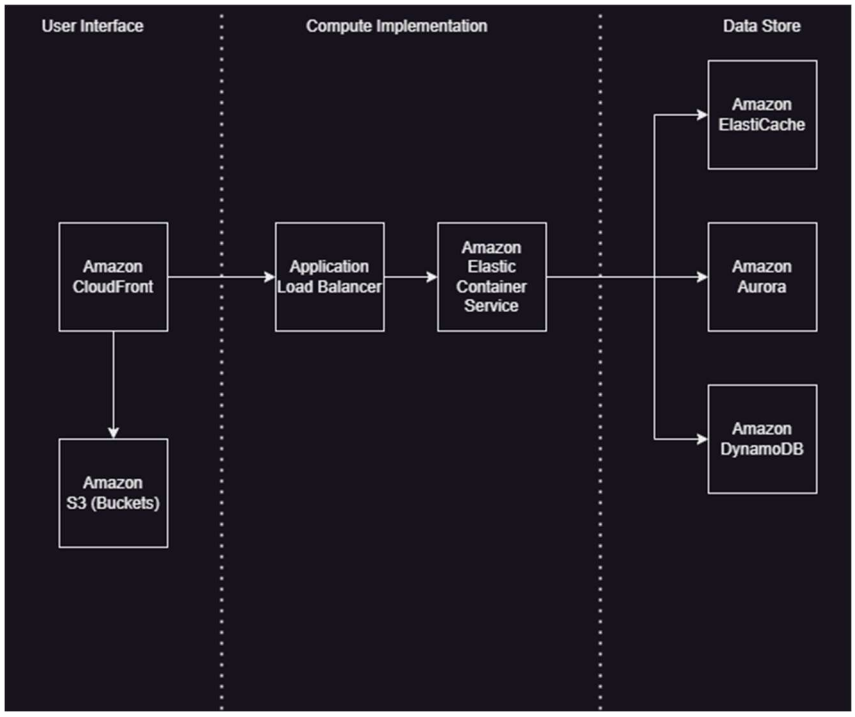

Figure 1. Typical microservice architecture design on AWS, (made in Draw.io)

Microservices' description:

- CloudFront → a content delivery network that caches static content for faster loading for nearby users and reducing load on other loading services
- S3 Buckets → stores user uploaded files
- Application Load Balancer → routes web traffic to other services
- Elastic Container Service → manages deployment of containerized apps
- ElastiCache / Aurora / DynamoDB → database management

Below is a summary of the pros and cons of microservice architecture for Amazon.

Pros:

- **Scalability**: Microservices can be scaled independently, allowing Amazon to handle high volumes of requests for specific services without affecting others.
- **Flexibility in Technology Choice**: Each microservice can use the best technology stack suited for its purpose. For example, the User Authentication service may use OAuth, while the Order Processing service might use a different technology stack.
- **Faster Development and Deployment**: Teams can focus on their specific microservice. Deployment can be done independently without affecting other services.
- **Resilience**: If one microservice fails (like notifications or chat) then the other services like checkout and payment remain unaffected.

Cons:

- **Poor cross-product compatibility:** If teams don't make their microservice very modular or flexible then it can be difficult to integrate with other services into a cohesive final product.
- **Poor Latency**: If services are located very far from each other and require a lot of cross-communication then the overall product could be slow.
- **Redundancies**: Some microservices may overlap a bit in their roles and responsibilities and be redundant like multiple logins required.

Considering how massive Amazon is, it is almost unimaginable that they could manage all their products and dev teams without using microservice architecture. Overall the benefits far outweigh the potential challenges.

References

1. "4 Microservices Examples: Amazon, Netflix, Uber, and Etsy": https://blog.dreamfactory.com/microservices-examples
2. "Simple microservices architecture on AWS": https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/simple-microservices-architecture-on-aws.html
3. "AWS re:Invent 2018: From Monolith to Microservices (And All the Bumps along the Way) (CON360-R1)": https://www.youtube.com/watch?v=gahR0_ZrYHs
4. "Simple microservices architecture on AWS" : https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/simple-microservices-architecture-on-aws.html