> Coursera Case Study :

Coursera is an educational technology company with a mission to provide universal access to the world's best curricula. The company partners with top universities and organizations worldwide to offer online courses for anyone to take, for free. Coursera has more than 13 million users from 190 countries enrolled and offers more than 1,000 courses from 119 institutions, everything from Programming in Python to Songwriting.

Amazon ECS mainly enabled Coursera to focus on releasing new software rather than spending time managing clusters.

The Challenge...

- Coursera had a large monolithic application for processing batch jobs that were difficult to run, deploy, and scale.
- A new thread was created whenever a new job needed to be completed, and each job took up different amounts of memory and CPU, continually creating inefficiencies.
- A lack of resource isolation allowed memory-limit errors to bring down the entire application.
- The infrastructure engineering team attempted to move to a microservices architecture using Docker containers, but they ran into problems as they tried to use Apache Mesos to manage the cluster and containers—Mesos was complicated to set up and Coursera didn't have the expertise or time required to manage a Mesos cluster.

Why Amazon Web Services...

- Docker containers on Amazon EC2 Container Service (ECS) enabled Coursera to easily move to a microservices-based architecture.
- Each job is created as a container and Amazon ECS schedules the container across the Amazon EC2 instance cluster.
- Amazon ECS handles all the cluster management and container orchestration, and containers provide the necessary resource isolation.

The Benefits:

- Ease of use: Because Amazon ECS setup is straightforward and it manages all of the details of the cluster, the team had a prototype up and running in under two months.
- Speed and agility: Time to deploy software changes went from hours to minutes, and each team can now develop and update its respective applications independently because the applications are resource isolated with no cross-dependencies.

- Scalable capacity: Auto Scaling groups allow the compute capacity to scale up to handle dynamic job loads.
- Operational efficiency: No extra infrastructure engineering time is spent installing software and maintaining a cluster—Amazon ECS handles everything from cluster management to container orchestration.

About Coursera...

Coursera is an educational technology company with a mission to provide universal access to the world's best curricula.

Benefits of AWS to Coursera Platform:

- Launched a prototype in less than two months
- Reduced time to deploy software changes from hours to minutes
- Reduced engineering time spent installing software and maintaining clusters

> Localytics Case Study :

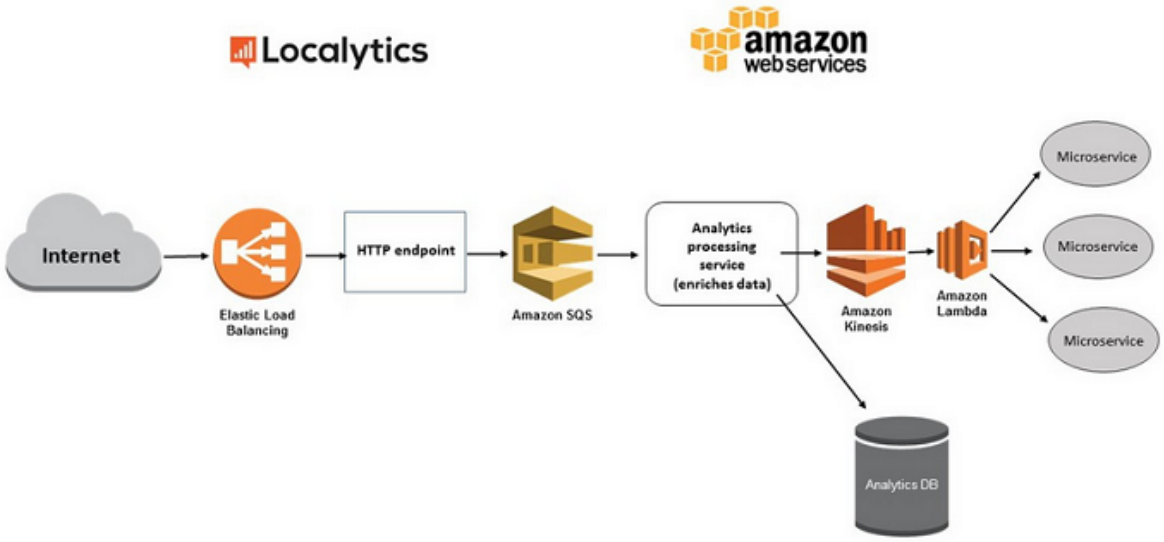
Localytics is a web and mobile app analytics and engagement company, with major brands such as ESPN, eBay, Fox, Salesforce, RueLaLa, and the New York Times using its marketing and analytics tools to understand how apps are performing and to engage with new and existing customers. The Boston-based company's software is used in more than 37,000 apps on more than three billion devices worldwide.

With AWS Lambda, various engineering teams can tap into a parallel data stream to create microservices independently from the main analytics application. It helps them get new services to the customers faster. For a startup, faster time to market is key as per Localytics Directors.

The Challenge...

- Supports pipeline with billions of data points uploaded every day from different mobile applications running Localytics analytics software.
- The engineering team needed to access subsets of data for creating new services, but this led to additional capacity planning, utilization monitoring, and infrastructure management.
- The platform team wanted to enable self-service for engineering teams.

Why Amazon Web Services...



- Uses AWS to send about 100 billion data points monthly through Elastic Load Balancing to Amazon Simple Queue Service, then to Amazon Elastic Compute Cloud, and finally into an Amazon Kinesis stream.
- For each new feature of the marketing software, a new microservice using AWS Lambda is created to access the Amazon Kinesis data stream. Each microservice can access the data stream in parallel with others.

The Benefits...

- Decouples product engineering efforts from the platform analytics pipeline, enabling the creation of new microservices to access the data stream without the need to be bundled with the main analytics application.
- Eliminates the need to provision and manage infrastructure to run each microservice.
- Lambda automatically scales up and down with load, processing tens of billions of data points monthly.
- Speeds time to market for new customer services, since each feature is a new microservice that can run and scale independently of every other microservice.

> Remind Case Study:

remind

Remind is a web and mobile application that enables teachers to send text messages to students and stay in touch with parents. Remind has 25 million users and more than 1.5 million teachers on its platform, and the app delivers 150 million messages per month.

Moving to Amazon ECS significantly improved the service performance of Remind application which has reduced the service response times in the 99th percentile by 50%.

The Challenge...

- Remind used a third-party cloud platform as a service (PaaS) to run its entire application infrastructure—message delivery engine, front-end API, web client, chat backend—as a monolithic application.
- Scaling issues prompted the move to a microservices-based architecture, but Remind wasn't satisfied with the limited visibility into CPU, memory, and network performance available from the PaaS provider.
- Remind wanted to use Docker on Amazon Elastic Compute Cloud (Amazon EC2) for better resource utilization and environment consistency, and the initial thought was to have an internal PaaS solution for developer efficiency.
- Remind started to build out their own PaaS on top of Linux-based CoreOS and fleet, but fleet and key-value store etc were unstable. The team also didn't want to spend the time to run and operate their own cluster-management system.

Why Amazon Web Services...

- Remind opted to use AWS directly in order to maintain operational simplicity; it was already using Amazon Redshift, Amazon DynamoDB, Amazon Simple Storage Service (Amazon S3), and Amazon CloudFront.
- The company decided to build its PaaS on top of Amazon EC2 Container Service (Amazon ECS) because the engineering team was small and didn't have the time or expertise necessary to operate and manage clusters.
- Remind open-sourced their PaaS solution on Amazon ECS as "Empire."
- Amazon ECS provides container scheduling and integration while Elastic Load Balancing (ELB) allows Empire to use DNS for service discovery.
- Empire provides a Heroku-compatible API and CLI that allows developers to easily deploy applications atop Amazon ECS.

The Benefits...

- Amazon ECS, a managed service, provides operational efficiency, allowing engineering resources to focus on developing and deploying applications instead of on operating and maintaining clusters.
- Moving to Amazon ECS provided large improvements in performance, including better stability and lower latency.
- Remind has seen a 2X decrease in response times in the 99th percentile, with less variance and fewer spikes.
- AWS provides control over security and routing through VPCs, plus clearer visibility into the performance of applications.

> Lyft Case Study:

Lyft is the fastest-growing rideshare company in the United States and is available in more than 200 cities, facilitating 14 million rides per month. Lyft uses AWS to move faster as a company and manage its exponential growth, leveraging AWS products to support more than 100 microservices that enhance every element of its customers' experience. Lyft launched on AWS and dramatically expanded its use of AWS products as they became available. It uses products such as Auto Scaling to manage up to eight times more riders during peak times and Amazon Redshift to gain customer insights that power the company's shared-ride product, Lyft Line. It uses Amazon Kinesis to funnel production events through the system and leverages the scalability of Amazon DynamoDB for multiple data stores, including a ride-tracking system that stores GPS coordinates for all rides. As part of their microservices infrastructure on AWS, Lyft relies on Amazon EC2 Container Registry (ECR) to durably store container images for their applications and reliably deliver these images to downstream test and deployment systems.

> Shippable Case Study:

Shippable is a platform providing hosted continuous integration, testing, and deployment from repositories such as GitHub and Bitbucket. The company helps developers and operations teams deliver software from application creation all the way to the end user with full automation and complete control over the process. Shippable's platform helps eliminate obstacles and challenges from a repeatable Continuous Integration/Continuous Delivery & workflow, so developers can focus on delivering high-quality code.

The Shippable platform consists of two parts:

- Continuous Integration (CI), enables developers to build and test their repositories for every code commit or pull request and get instant feedback.
- Continuous Delivery (CD) pipelines, automating the flow of an application from source control to production. Using these deployment pipelines, developers can easily deploy containerized applications to container services such as Amazon EC2 Container Service (Amazon ECS).
- Shippable integrates with popular tools and services, such as build/test tools, source control providers, Docker registries, cloud providers, and container services.

Amazon ECS practically eliminated the time the developers spent on ops-related tasks. The senior developers used to spend 80% of their time on back-end infrastructure management features, whereas now they spend 80% of their time on customer features.

The Challenge...

- The company built its CICD platform with a microservices architecture using Docker containers on Amazon EC2, which they managed using a custom-made cluster management tool.
- The initial solution used a hardcoded service discovery solution, making it hard to scale and manage the configuration.
- The engineering team spent 60-80 percent of its time maintaining the infrastructure and orchestrating containers, taking time away from implementing and delivering features for the platform itself.
- Managing the monitoring infrastructure with an extensive set of log aggregators and Elasticsearch was another time-consuming task.
- Shippable evaluated other open-source cluster management solutions such as Apache Mesos and Kubernetes but found that these solutions would still require a significant amount of engineering time to operate and scale the cluster.

Why Amazon Web Services...

- Started using Amazon ECS, a scalable, container management service that provides cluster management and container orchestration.
- Each microservice is defined as an ECS service with an Elastic Load Balancing load balancer; the Amazon ECS service scheduler manages multiple copies of each microservice across the ECS cluster.
- Amazon ECS and Amazon CloudWatch handle all of the infrastructure logging, with minimal effort from the engineering team.
- Shippable Docker images are stored on Amazon EC2 Container Registry (Amazon ECR), a fully-managed Docker container registry that makes it easy to store, manage, and deploy Docker container images.

The Benefits...

- Amazon ECS eliminates the need to install software and operate servers; this reduces overall management needs for running containers at scale and helps speed time to market.
- The engineering team now spends only 20 percent of its time maintaining the infrastructure, and has much more time to focus on implementing new features.

- Using Amazon ECS enables Shippable to deliver features to its customers faster than before, increasing the number of deployments from once a week to multiple times a day.
- With AWS Identity Access Management (IAM) policies for Amazon ECR, the company is able to add a layer of security that ensures only approved individuals can push and pull production images in the production cluster.

> Netflix Case Study:

Netflix is one of the most popular online media service provider and production companies in the world right now. What started out as a small subscription-based DVD provider about 20 years ago, has now evolved into a company which revolutionized online media streaming as we know it today. One of the reasons why Netflix is an interesting company to talk about is that they are considered one of the role models for modern cloud-based companies through the use of Microservices architecture.

Why are Microservices associated with Netflix?

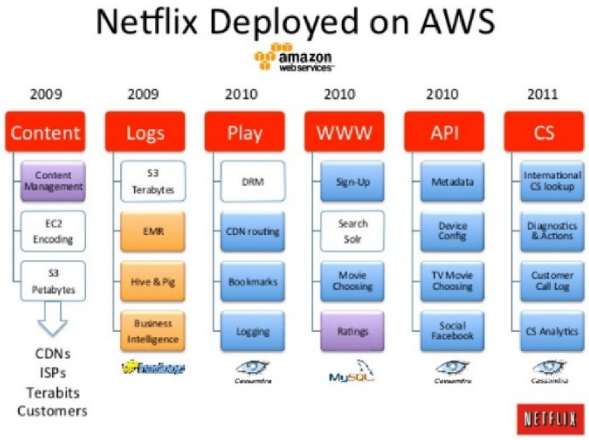
Netflix is one of the first companies to have successfully migrated from a traditional monolithic to cloud-based microservices architecture. In fact, Netflix implemented this architecture long before the term microservices was even introduced. It took more than two years for Netflix to achieve complete migration to the cloud. Not only did Netflix perfect the use of microservices but it also managed to open source many of the tools which were used to build it. The Netflix OSS (Open Source Software Center) has a lot of tools and technologies which can be used by other companies to create microservice architecture on the cloud.

Reason for this Migration

When Netflix announced its big move to the cloud it faced a lot of criticism as no one believed such a feat was possible at that time. The main reason why Netflix had decided to move to the cloud was due to its rapidly increasing data and user information that was hard to store it in its current data centers, this caused a huge deal of problems. The solution was accomplished using Amazon Web Service (AWS), which promised to provide large computing resources and data

centers with guaranteed security and reliability. With AWS scaling can be done within a couple of minutes without any user involvement.

While moving to the cloud, Netflix managed to split its single monolithic application into hundreds of small loosely coupled services. Today Netflix has over 1000 microservices, each managing a separate part of the site.



The Move...

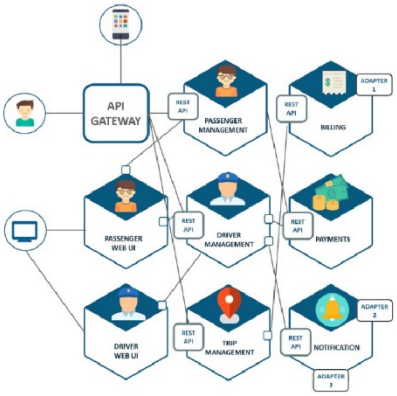
According to the then-cloud architect of Netflix, the migration process began in 2009. It started out by moving movie encoding, a noncustomer-facing application. By 2010 Netflix started moving the remaining pieces to the cloud such as account signup, movie selections, and other configurations. By December 2011, Netflix had successfully migrated its entire operation to the cloud – from a monolithic to a then-unknown territory of microservice architecture.

Unlike how it looks, Netflix had to face a lot of problems during this big move. First of all, during the entire migration process, Netflix had to keep both its cloud servers and its in-house servers running to ensure smooth working during this transition period. Also moving to the cloud meant replicating all the data from the local data centers to the cloud data centers, this involved having to deal with a huge amount of data. During the migration of customer-facing applications, Netflix had to deal with a lot of latency issues while serving the web pages. Netflix also faced a number of other issues such as load increases, instance failures, and other performance issues.

What people considered a crazy move by Netflix turned out to be a game-changer in the field of cloud computing today. Today almost all the big companies like Google, Twitter, IBM, and Amazon have already moved to the cloud while other companies have slowly started their migration. Microservice architecture has since become one of the important approaches to the cloud. When it comes to cloud computing today it is no surprise that Netflix has managed to come out as a technology leader.

> Uber Case Study:

The next case in discussion is Uber. This taxi-hailing cab originated in San Francisco and was meant to serve the people in the city. Soon the service spread to other cities and countries. Problems began showing up when Uber wanted to launch new features or fix bugs or upgrade their services to global levels. Moreover, all the developers needed extensive knowledge of the existing system even to make minor changes.



Uber's Monolithic structure:-

A REST API connected the Uber drivers and passengers. Three adapters with embedded API served functions such as billing, payments and chats. All features including a MySQL database were contained within the monolithic structure.

To work through these obstacles, Uber decides to shift to cloud-based services. They soon built microservices for the different functionalities such as the management of trips and passengers. The services communicated with the outside world via API gateways.

Talking about the advantages that microservices brought to Uber's existing monolithic interdependent system, each development has clear ownership and boundaries of the functionalities to be developed. This improved the speed of development and the quality.

When the individual teams concentrated on the services that required scaling, all the work related to scaling took place very fast. There was no downtime whatsoever when specific services were up for maintenance. The system fault tolerance was more reliable.

However, it was well understood that merely getting into the microservices architecture was not the end of troubles for the organization. The need of the hour was a standardization strategy to prevent any loss of control. Here is how they went about the same.

All of the 1300 microservices first adopted local standards for every microservices. This was because Uber's microservices could not latch with the microservices of other organizations because of differing standards. Maintenance became an issue when developers altered a specific microservice. Service outages became common. It was impossible to coordinate the standards for all the microservices after maintenance.

Finally, Uber decided to develop global standards for all of the microservices. For this, they set up quantifiable standards for principals such as documentation, reliability, stability, fault tolerance, etc. These they measured based on business metrics like webpage views, etc. With global standards that were developed, the services were of the highest quality.

> Spotify Case Study:

Spotify is a global music company that serves more than 170 million users. Serving these subscribers and looking after the computing, storage, and networking resources is a major task in itself. To be the best music service in the world, the organization started to feel that maintaining data centers did not contribute to this goal in any manner.

Further, they also wanted to free up developers from the jobs of provisioning resources and maintaining them. Moreover, Spotify wanted to derive advantages from Google Cloud's latest innovations (BigQuery- data warehouse, Pub/Sub - messaging, DataFlow – streaming data).

Spotify formulated the migration plan in 2015 and they split the job into two parts, services and data.

Services migration:-

By forming a team of engineers and Google experts, Spotify first created a live visualization of the migration process so that the engineers could easily track progress on their own. The visualization was a set of red (data center) and green (Google Cloud) bubbles and the size represented the number of machines that were involved.

The team then mapped dependencies as every microservice would rely on a minimum of 10 to 15 other microservices for just servicing a customer request. Therefore, all-at-once migration was ruled out because customers expect constant uptime from Spotify.

The engineers were therefore tasked with moving the systems in just a 2-week window, and in this period, they temporarily put on hold all the product development activity. They pushed everything unnecessary out of their way. They used the Virtual Private Cloud option to their advantage.

Another problem the team faced was VPN latency. Shifting 1200 microservices to the loud ones took up a lot of VPN bandwidth. The team then collaborated with Google to get over this problem. Then they started moving the user traffic to the cloud. Soon after this, they could decouple the service migration from user traffic.

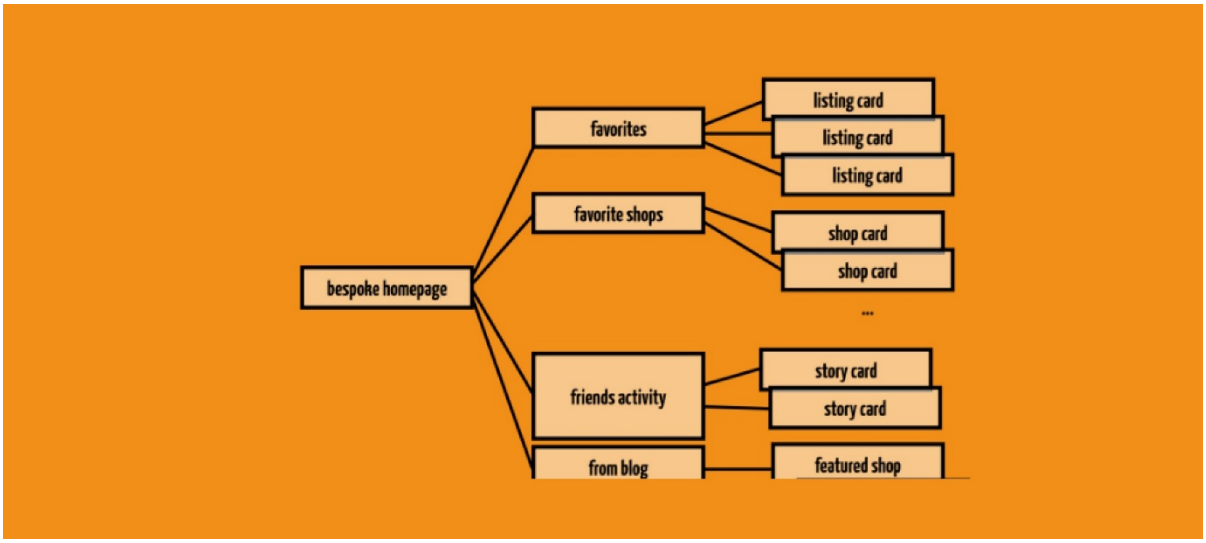
The central migration team induced failures which were kind of dry runs to see how the teams reacted to and managed the new microservices architecture. By 2017, the traffic was routed through Google Cloud and they soon closed their on-premise data centers.

Data Migration:-

This task involved moving Europe's biggest on-premise Hadoop clusters to the cloud. Following the dependency map, the task was to move close to 20 000 data jobs to Google Cloud without any downstream disruptions.

They wanted to copy all the data to the cloud and restart the required operations. Even if the speed was 160 Gb/s, the job would have taken two months to complete. To handle operations without disruptions they also decided to copy the moved data to the on-premise cluster till the job got done completely. The entire job lasted for almost a year. Spotify now runs its data on BigQuery while running close to 10 million queries per month.

> Etsy Case Study:



Poor server processing time was what prompted the eCommerce platform to think of microservices. Performance issues started cropping up regularly and soon the development team began to set goals to reduce processing time to 1000-ms time-to-glass. This is the amount of time required for the screen to update values on the user's device.

The development team also realized that concurrent transactions are the only way that would improve processing time. They had a PHP-based system and this ruled out concurrent API calls.

Etsy's earlier system was based on sequential transaction execution and this made it very sluggish. The development team also wanted to extend the Etsy mobile app capabilities. The API team decided to move ahead with a new approach where the development teams could be familiar with and access the APIs.

Etsy's approach...

With inspiration from Netflix's migration to microservices architecture, Etsy first decided to establish a two-tier API with meta endpoints with each of them aggregating additional endpoints. With this arrangement, low-level general-purpose resources on the server side were transformed into device-specific resources.

This action created a multilevel tree and the customer end of the website and Etsy's mobile app were composed into a custom view using one layer of concurrent meta endpoints (atomic component endpoints). The non-meta-endpoints (those at the lowest level) are the ones that will communicate with the database.

Even at this stage, a lack of concurrency was limiting the processing speed of the eCommerce application. Though all these actions generated a bespoke version of the mobile app and website, the sequential processing of the many meta endpoints was an obstacle to reaching the performance goals.

This problem was ultimately solved by the development team when they used cURL for parallel HTTP calls that helped them achieve API concurrency. They had to make a few more associated changes before Etsy's new structure went live in 2016. Etsy went on to create general-purpose tools that favored the developers and helped them speed up the adoption of the 2-layer API. Now their microservices structure supports faster upgrades, concurrent processing, easy scaling, and innovation as and when required.

Conclusion:-

Microservices challenges and solutions are discussed with gusto, Netflix, etc. by most large corporations that have migrated successfully from the monolith behemoth they tried to sustain earlier on. Though the challenges were overwhelming, the results have brought important benefits to these businesses. Microservices and the advantages that they can bring are very evident from the success stories of these large corporations. Are microservices the right choice for your business? You must be aware that the shift requires time and brings along its challenges. However, at the end of it all, the advantages of microservices can help your business scale to newer heights.

References

- https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith
- https://aws.amazon.com/compare/the-difference-between-monolithic-and-microservices-architecture/
- https://www.geeksforgeeks.org/monolithic-vs-microservices-architecture/
- https://www.linkedin.com/pulse/microservices-architecture-case-study-from-various-suryawanshi/