**MICROSERVICES CASE STUDIES WITH SOLUTIONS**

**1. Netflix: Handling Scalability and High Availability**

**Problem:**  
Netflix needed a scalable and resilient system to handle millions of users globally. Their monolithic architecture could not scale efficiently, leading to outages and slow updates.

**Solution:**  
Netflix adopted a **microservices architecture** deployed on **AWS cloud infrastructure**. They used:

* **Spring Boot & Spring Cloud** for service development
* **Eureka for service discovery**
* **Zuul for API Gateway**
* **Hystrix for fault tolerance**
* **Kafka for event-driven communication**

**Outcome:**

* Faster deployment cycles
* Improved fault tolerance and uptime
* Better scalability to support millions of users

**2. Uber: Overcoming Monolithic Bottlenecks**

**Problem:**  
Uber started with a monolithic backend, but as it expanded to new cities, the system became hard to scale and maintain. New feature rollouts were slow, and downtime affected user experience.

**Solution:**  
Uber migrated to a **microservices architecture** with:

* **Go, Node.js, and Python** for services
* **Apache Kafka for event-driven communication**
* **Cassandra and MySQL for data storage**
* **Docker and Kubernetes for containerization and orchestration**

**Outcome:**

* Faster feature rollouts
* Better system reliability
* Scalability for a global user base

**3. Amazon: Optimizing E-commerce Performance**

**Problem:**  
Amazon’s monolithic system was inefficient in handling peak loads (e.g., Black Friday sales). A single failure could bring down the entire application.

**Solution:**  
Amazon transitioned to microservices, using:

* **AWS Lambda for serverless execution**
* **DynamoDB and RDS for database services**
* **API Gateway for request management**
* **SQS and SNS for event-driven messaging**

**Outcome:**

* Improved performance during high-traffic periods
* More efficient resource utilization
* Reduced downtime

**4. PayPal: Enhancing Payment Processing**

**Problem:**  
PayPal’s monolithic system led to **slow deployments and frequent downtimes** during feature releases.

**Solution:**

* Adopted **Spring Boot-based microservices**
* Used **GraphQL** for optimized API responses
* Deployed on **Kubernetes** for scalability
* Implemented **CI/CD pipelines** for faster deployments

**Outcome:**

* Faster transaction processing
* Improved system resilience
* Accelerated feature releases

**5. Spotify: Improving Music Streaming Performance**

**Problem:**  
Spotify needed a scalable architecture for **real-time music streaming**. Their monolithic system was causing **latency and poor user experience**.

**Solution:**

* Migrated to **microservices with event-driven architecture**
* Used **Apache Kafka for real-time event streaming**
* Adopted **Docker and Kubernetes for service orchestration**
* Implemented **CDN caching** for better performance

**Outcome:**

* Faster music playback
* Better scalability for global users
* Efficient API request handling