Name: Isaac Escobar

Report on Microservices

1. Introduction

- Overview of software architectures and the evolution toward microservices.
- Definition of microservices architecture.
- Importance of microservices in modern application development.

2. What Are Microservices?

- Description of microservices as an architectural style.
- Key principles:
    - Decentralization.
    - Bounded context.
    - Lightweight communication.
- Contrast with monolithic architecture.

3. Key Characteristics of Microservices

- **Independent Deployability**: Services can be updated independently.
- **Technology Agnostic**: Each service can use different programming languages or frameworks.
- **Scalability**: Services can be scaled individually based on load.
- **Resilience**: Failures in one service don't bring down the entire system.
- **Small and Focused**: Each service focuses on a specific business capability.

4. Advantages of Microservices

- Enables faster development cycles.
- Simplifies debugging and testing due to service isolation.
- Supports innovation by allowing diverse tech stacks.
- Improves fault isolation.
- Facilitates team autonomy.

5. Challenges of Microservices

- Increased complexity in managing distributed systems.
- Challenges in ensuring consistent data across services.
- Requires advanced monitoring and logging systems.
- Overhead in communication between services.

- Dependency on robust DevOps practices.

6. Microservices Architecture

- Key components:

- API Gateways.
- Service Discovery.
- Database per service.
- Communication methods: REST, gRPC, or Message Brokers.

7. Tools and Technologies for Microservices

- **Development Frameworks**:
    - Spring Boot (Java).
    - Micronaut (Java).
    - Flask or FastAPI (Python).
- **Containerization**: Docker, Kubernetes.
- **Messaging Systems**: RabbitMQ, Kafka.
- **Monitoring**: Prometheus, Grafana.
- **CI/CD Pipelines**: Jenkins, GitLab CI/CD.

8. Best Practices for Microservices

- Design services around business capabilities.
- Keep services small and cohesive.
- Use API contracts for communication.
- Implement centralized logging and monitoring.
- Secure communication between services with encryption.
- Apply versioning for APIs.

9. Use Cases of Microservices

- **E-commerce**: Separate services for product catalog, inventory, and payment processing.
- **Streaming Platforms**: Independent services for content delivery, recommendations, and user profiles.
- **Financial Systems**: Modular services for account management, fraud detection, and transaction processing.

10. Microservices vs. Monolithic Architecture

Feature	**Microservices**	**Monolithic**
Scalability	Fine-grained, per service	Entire application
Deployment	Independent, faster	Entire app redeployed

Fault Isolation	High	Low
Complexity	High (distributed system)	Low
Technology Stack	Polyglot	Single tech stack

11. Conclusion

- Summary of the benefits and challenges of microservices.
- Importance of proper design, monitoring, and DevOps practices.
- Future trends in microservices: serverless computing, service mesh, etc.

12. References

1. **"Building Microservices" by Sam Newman**
    - Comprehensive guide to designing and deploying microservices.
2. **Microservices.io by Chris Richardson**
    - Link: https://microservices.io/
3. **Netflix Tech Blog**
    - Insights into Netflix's use of microservices.
    - Link: https://netflixtechblog.com/
4. **Kubernetes Documentation**
    - Managing microservices with Kubernetes.
    - Link: https://kubernetes.io/docs/home/