**1. Introduction**

* **Background & Motivation**
  + Why microservices architecture is important in modern applications.
  + The challenges in microservices, particularly in **configuration management**.
  + Why centralized configuration management (Spring Cloud Config Server) is essential.
* **Problem Statement**
  + Issues with local per-service configuration management.
  + Security, scalability, and maintainability concerns.
* **Research Objectives**
  + Evaluate the benefits of centralized configuration management over local configuration.
  + Assess the impact on **scalability, security, and maintainability**.
* **Research Question**
  + *How does centralized configuration management with Spring Cloud Config Server improve scalability, security, and maintainability in microservices compared to local per-service configurations?*

**2. Literature Review**

* **Microservices Architecture**
  + Basics of microservices (advantages, challenges).
  + Transition from monolith to microservices.
* **Configuration Management in Microservices**
  + Local per-service configuration challenges.
  + Benefits of centralized configuration using Spring Cloud Config Server.
* **Security in Microservices Configuration Management**
  + Sensitive data storage (database credentials, API keys).
  + OAuth2, OpenID Connect, and encryption mechanisms.
* **Scalability and Performance Considerations**
  + Horizontal scaling with Kubernetes.
  + Configuration consistency across multiple instances.
* **Observability and Maintainability**
  + Monitoring configuration changes with **Prometheus, Grafana, and OpenTelemetry**.

**3. Methodology**

* **Overview of the Research Approach**
  + Comparative study between:
    1. Microservices with **local configurations**.
    2. Microservices using **Spring Cloud Config Server**.
  + Optional comparison with **monolithic configuration management** (if time allows).
* **System Architecture & Implementation**
  + **Technology Stack:**
    1. **Spring Boot & Spring Cloud** (Microservices development).
    2. **Docker & Kubernetes** (Containerization & orchestration).
    3. **Spring Security, OAuth2, Keycloak** (Security enforcement).
    4. **Kafka & RabbitMQ** (Event-driven architecture).
* **Evaluation Criteria**
  + **Performance Testing:** Deployment speed, latency, load balancing.
  + **Security Assessment:** Authentication, encryption, secrets management.
  + **Maintainability & Observability:** Ease of configuration updates, rollback mechanisms.

**4. System Design and Implementation**

* **System Architecture Diagram** (similar to the one you uploaded).
* **Microservices Implementation**
  + Spring Boot-based microservices (Accounts, Loans, Cards).
  + Configuration Management (Spring Cloud Config Server).
  + Service Discovery & Routing (Eureka & Spring Cloud Gateway).
* **Security Integration**
  + OAuth2 & Keycloak for authentication.
  + Secure storage of sensitive configurations.
* **Deployment Process**
  + Containerization with Docker.
  + Orchestration with Kubernetes.
* **Monitoring & Logging**
  + Metrics collection using **Prometheus & Grafana**.

**5. Performance Evaluation**

* **Scalability Tests**
  + How well services scale under different loads.
* **Security Assessment**
  + Encryption of configurations, token-based authentication.
* **Maintainability & Reliability**
  + Ease of configuration updates and rollback mechanisms.

**6. Results and Discussion**

* **Comparison of Local vs. Centralized Configuration Management**
* **Key Findings**
* **Challenges & Limitations**

**7. Conclusion and Future Work**

* **Summary of Findings**
* **Future Research Directions**

**8. References & Appendix**

* **IEEE-formatted references**
* **Additional documentation, diagrams, and implementation details**