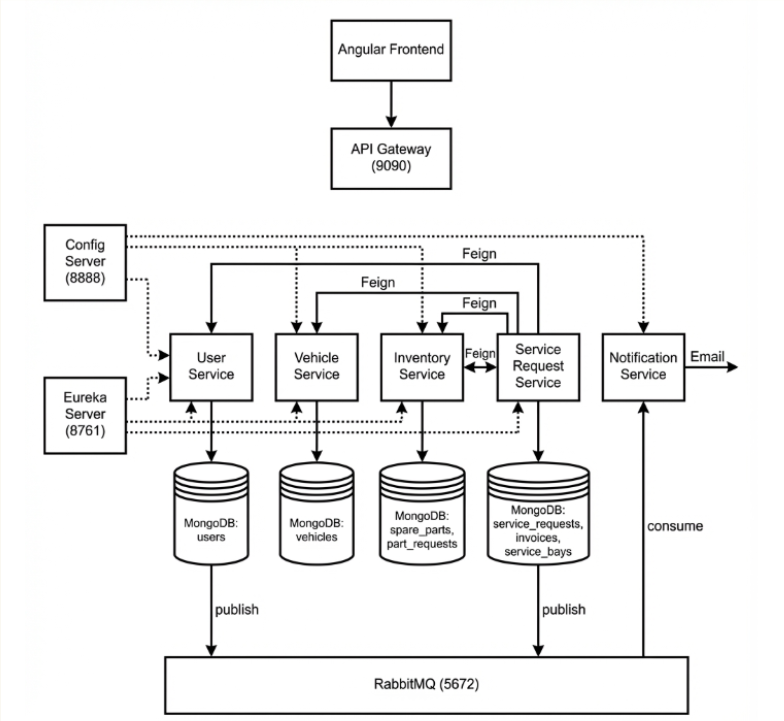
**SOFTWARE DESIGN DOCUMENT (SDD)**

## Vehicle Service Management System



## 1. DOCUMENT CONTROL

| **Item** | **Details** |
| --- | --- |
| Project Name | Vehicle Service Management System |
| Version | 1.0 |
| Author | (Podila Sravya) |
|  |  |
| Status | Final |

## 2. PURPOSE OF THE DOCUMENT

This document describes the **system architecture, design decisions, component structure, APIs, data models, and non-functional aspects** of the Vehicle Service Management System.

It is intended for:

* Developers
* Reviewers
* Interview discussions
* Maintenance & enhancement planning

## 3. SYSTEM OVERVIEW

**Business Objective**

Provide a scalable, secure, and maintainable system to:

* Manage vehicle service requests end-to-end
* Assign technicians and track service progress
* Manage spare parts inventory
* Generate invoices and track payments
* Send real-time email notifications
* Provide role-based dashboards and reports

**High-Level Features**

* User registration with role-based approval (Customer auto-approved, Staff needs admin approval)
* Vehicle registration and management
* Service request lifecycle (Pending → Assigned → In Progress → Completed → Closed)
* Technician workload management
* Spare parts inventory and part requests
* Automated billing and invoice generation
* Email notifications via RabbitMQ
* Admin reports and analytics

## 

## 

## 4. ARCHITECTURE OVERVIEW (HLD)

**Architecture Style**

* **Microservices Architecture** with 8 independent services
* **REST-based** synchronous communication via Feign Clients
* **Event-driven** asynchronous communication via RabbitMQ
* **Containerized deployment** using Docker

**Core Components**

1. Config Server (Centralized Configuration)
2. Eureka Server (Service Discovery)
3. API Gateway (JWT Authentication & Routing)
4. User Service
5. Vehicle Service
6. Inventory Service
7. Service Request Service
8. Notification Service
9. Angular Frontend
10. MongoDB (per service)
11. RabbitMQ (Messaging)
12. CI/CD Pipeline (Jenkins + SonarCloud + Docker)

## 5. TECHNOLOGY STACK

**Backend**

* Java 21
* Spring Boot
* Spring Web
* Spring Cloud
* Spring Security + JWT
* Netflix Eureka
* Spring Cloud Config
* OpenFeign
* RabbitMQ
* Resilience4j
* Spring Data MongoDB

**Frontend**

* Angular 21
* TypeScript
* CSS
* HttpClient
* Interceptors
* Route Guards

**Database**

* MongoDB

**DevOps**

* Docker
* Docker Compose
* Jenkins
* JaCoCo
* SonarQube

**Testing**

* JUnit 5
* Mockito
* Spring Boot Test

## 6. MICROSERVICES DESIGN

**6.1 Config Server**

**Port: 8888**

**Responsibilities:**

* **Centralized configuration management**
* **Git-based configuration repository**

**Config Source:**

spring.cloud.config.server.git.uri=https://github.com/psravya-10/vsms-config-repo

**6.2 Eureka Server**

**Port: 8761**

**Responsibilities:**

* **Service discovery and registration**

**6.3 API Gateway**

**Port: 9090**

**Responsibilities:**

* **Single entry point for all requests**
* **JWT token validation**
* **Role-based route authorization**
* **Request routing to microservices**

### 6.4 User Service

**Responsibilities:**

* User registration (Customer auto-approved, Staff pending)
* Authentication and JWT generation
* Role management (ADMIN, MANAGER, TECHNICIAN, CUSTOMER)
* Staff approval/rejection by Admin
* Technician availability management
* Password change

**APIs**

POST /api/auth/register/customer → Register customer (auto-approved)

POST /api/auth/register/staff → Register technician/manager (pending approval)

POST /api/auth/login → Login, returns JWT token

GET /api/auth/profile → Get my profile

PUT /api/auth/change-password → Change password

GET /api/admin/staff → Get all staff (filter by status, role)

GET /api/admin/users → Get all users

PUT /api/admin/users/{id}/approval → Approve/reject user

GET /api/manager/technicians → Get all technicians

GET /api/manager/technicians/available → Get available technicians

**Database**

users collection

**RabbitMQ Events Published:**

* REGISTRATION\_APPROVED
* REGISTRATION\_REJECTED

**6.5 Vehicle Service**

**Responsibilities:**

* **Vehicle registration**
* **Vehicle CRUD operations**
* **Vehicle history**

**APIs**

POST /api/vehicles

GET /api/vehicles

GET /api/vehicles/{id}

GET /api/vehicles/user/{userId}

PUT /api/vehicles/{id}

DELETE /api/vehicles/{id}

**Database**

vehicles collection

**6.6 Inventory Service**

**Responsibilities:**

* **Spare parts catalog management**
* **Part request creation by technicians**
* **Part request approval by manager**
* **Stock management**

**APIs**

POST /api/inventory/parts → Add spare part

GET /api/inventory/parts → Get all parts

GET /api/inventory/parts/low-stock → Get low stock parts

PUT /api/inventory/parts/{id}/restock → Restock part

PUT /api/inventory/parts/{id}/threshold → Update threshold

POST /api/inventory/requests

GET /api/inventory/requests

PUT /api/inventory/requests/{id}/approve

PUT /api/inventory/requests/{id}/reject

**Database**

spare\_parts collections

part\_requests collections

**Feign Clients**

* Service Request Service (to add used parts)

### 6.7 Service Request Service

**Responsibilities:**

* Service request creation
* Technician assignment
* Status management
* Service bay allocation
* Invoice generation
* Billing management

API s:

# Customer

POST /api/customer/service-requests

GET /api/customer/service-requests

GET /api/customer/service-requests/{id}

GET /api/customer/service-requests/vehicle/{vehicleId}

# Manager

GET /api/manager/service-requests

GET /api/manager/service-requests/pending

PUT /api/manager/service-requests/{id}/assign

PUT /api/manager/service-requests/{id}/close

GET /api/manager/technicians/workload

# Technician

GET /api/technician/service-requests

PUT /api/technician/service-requests/{id}/status

# Billing

POST /api/billing/invoices/generate/{serviceRequestId}

GET /api/billing/invoices/{id}

PUT /api/billing/invoices/{id}/pay

GET /api/billing/admin/invoices/monthly

# Service Bays

GET /api/bays

POST /api/bays

**Database:**

service\_requests collections

invoices collections

 service\_bays collections

**Feign Clients:**

* **User Service (technician availability)**
* **Vehicle Service (vehicle details)**
* **Inventory Service (check pending requests)**

**RabbitMQ Events Published:**

* **SERVICE\_STARTED**
* **SERVICE\_COMPLETED**
* **INVOICE\_GENERATED**

### 6.8 Notification Service

**Responsibilities:**

* Consume events from RabbitMQ
* Send email notifications

**Events Consumed:**

* REGISTRATION\_APPROVED
* REGISTRATION\_REJECTED
* SERVICE\_STARTED
* SERVICE\_COMPLETED
* INVOICE\_GENERATED

**Database:** None

## 7. DATA DESIGN (LLD)

**User Document**

{

"id": "u101",

"name": "John Doe",

"email": "john@test.com",

"password": "encrypted",

"role": "TECHNICIAN",

"status": "APPROVED",

"availability": BUSY,

"phone": "9876543210",

"pincode": "500001"

}

**Vehicle Document**

{

"id": "12345678201",

"userId": "125148101",

"registrationNumber": "TS09AB1234",

"brand": "Honda",

"model": "City",

"vehicleType": "CAR",

"manufactureYear": 2022

}

**Service Request Document**

{

"id": "sr301",

"userId": "u101",

"vehicleId": "v201",

"technicianId": "t501",

"bayId": "b001",

"issueDescription": "Engine noise",

"status": "IN\_PROGRESS",

"priority": "NORMAL",

"remarks": "Checking engine",

"labourCharges": 500.0,

"partsUsed": [

{ "partId": "p101", "partName": "Oil Filter", "quantity": 1, "unitPrice": 250.0 }

],

"partsTotal": 250.0

}

**Invoice Document**

{

"id": "inv401",

"serviceRequestId": "sr301",

"userId": "u101",

"vehicleId": "v201",

"labourCharges": 500.0,

"partsUsed": [...],

"partsTotal": 250.0,

"totalAmount": 750.0,

"paymentStatus": "PENDING",

"createdAt": "2026-01-07T10:00:00",

"paidAt": null

}

**Spare Part Document**

{

"id": "p101",

"name": "Oil Filter",

"category": "Filters",

"availableQuantity": 50,

"unitPrice": 250.0,

"lowStockThreshold": 5

}

**Part Request Document**

{

"id": "pr601",

"serviceRequestId": "sr301",

"technicianId": "t501",

"partId": "p101",

"quantity": 2,

"status": "PENDING",

"approvedBy": null,

"remarks": "Need for service"

}

**Service Bay Document**

{

"id": "b001",

"bayCode": "BAY-01",

"status": "OCCUPIED"

}

## 8. API DESIGN & VALIDATION

* RESTful principles
* JSON request/response
* Bean Validation at DTO layer
* Service-level business rule enforcement
* Standard HTTP status codes

## 9. ERROR HANDLING STRATEGY

**Global Exception Handling**

* Centralized using @ControllerAdvice
* Standard error response format
* {
* "timestamp": "2026-01-07T10:00:00",
* "status": 400,
* "error": "Business Exception",
* "message": "Email already registered"
* }

## 10. SECURITY DESIGN

* Password encryption (BCrypt)
* Role-based access control
* JWT authentication (optional enhancement)
* Secure API access
* Route Guards

## 11. NON-FUNCTIONAL REQUIREMENTS

| **Area** | **Design Decision** |
| --- | --- |
| Scalability | Stateless services, Docker |
| Performance | Pagination, async calls |
| Availability | Independent services |
| Maintainability | POM-like layered backend |
| Security | Validation, encryption |
| Observability | Logging & monitoring |
|  |  |

## 

## 12. TESTING STRATEGY

**Backend**

* Unit tests (Service layer) - JUnit 5, Mockito
* Controller tests (MockMvc)
* Minimum 90% coverage for controllers & services

**Frontend**

* Component tests
* Service tests

**Quality Gates**

* SonarCloud enforced
* JaCoCo coverage reports

## 13. CI/CD DESIGN

**Pipeline Flow**

1. **Git Commit**
2. **Jenkins Checkout**
3. **Maven Build & Test**
4. **JaCoCo Coverage Report**
5. **SonarCloud Scan**
6. **Docker Compose Build**
7. **Docker Compose Up**

**Jenkinsfile:**

pipeline {

agent any

tools { maven 'Maven' }

environment { SONAR\_TOKEN = credentials('sonar-token') }

stages {

stage('Checkout') {

steps { git branch: 'main', url: 'https://github.com/psravya-10/vehicle-service-management-system-backend.git' }

}

stage('Build & Test') {

steps { bat 'mvn clean verify' }

}

stage('SonarCloud Analysis') {

steps { bat 'mvn sonar:sonar -Dsonar.token=%SONAR\_TOKEN%' }

}

stage('Docker Compose Build') {

steps { bat 'docker-compose build' }

}

stage('Docker Compose Up') {

steps { bat 'docker-compose up -d' }

}

}

}

## 14. DEPLOYMENT DESIGN

* Docker image per microservice
* docker-compose for orchestration
* Environment-specific configs via Config Server

## 15. ASSUMPTIONS & CONSTRAINTS

**Assumptions**

* Services communicate over REST
* MongoDB available
* Docker environment present

**Constraints**

* No distributed transactions
* Event-driven architecture out of scope

## 16. FUTURE ENHANCEMENTS

* SMS Notifications
* Payment Gateway Integration
* Mobile Application
* Kubernetes Deployment
* Centralized Logging (ELK)

## 17. CONCLUSION

This design ensures:  
✔ Clean separation of concerns (8 microservices)

✔ Scalability & maintainability

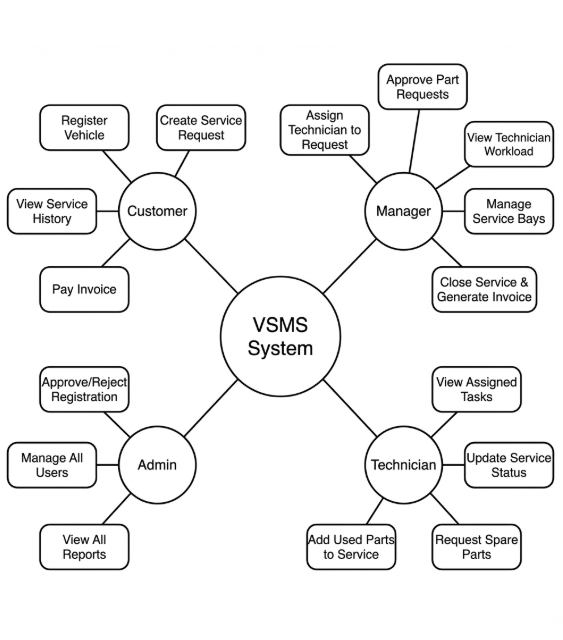
✔ Testability (90%+ coverage)

✔ CI/CD readiness (Jenkins + Docker + SonarCloud)

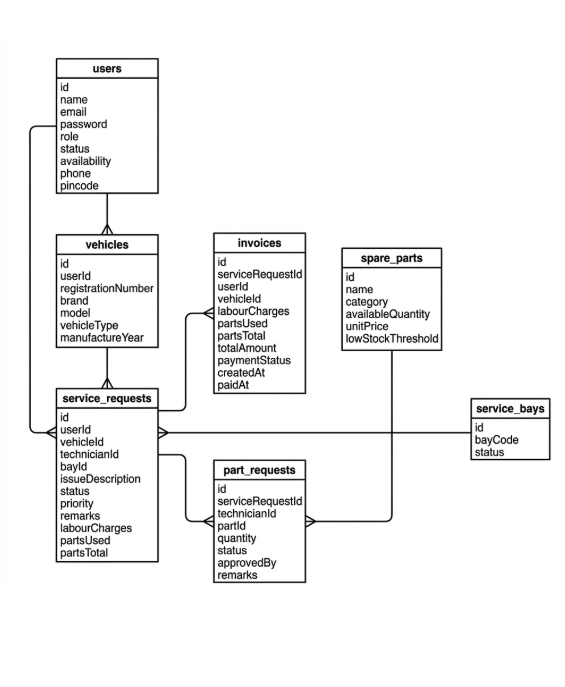
✔ Security (JWT + RBAC)

✔ Real-time notifications (RabbitMQ)

**USE CASE DIAGRAM**

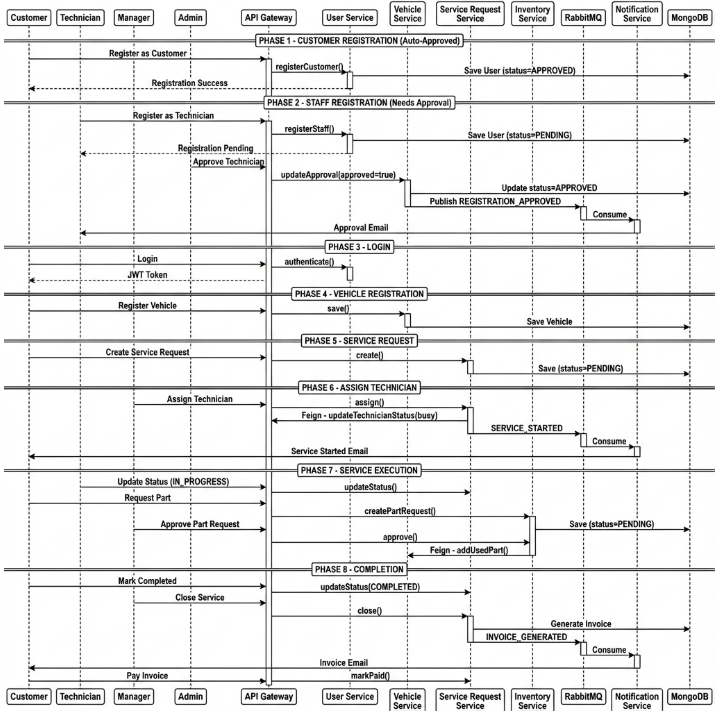


# ER DIAGRAM :



# 

# SEQUENCE DIAGRAMS — VEHICLE SERVICE MANAGEMENT SYSTEM



**1. Customer Registration (Auto-Approved)**

Flow:

1. Customer enters details in Angular UI
2. Angular sends POST /api/auth/register/customer
3. User Service validates request
4. Password encrypted with BCrypt
5. User saved with status=APPROVED
6. Success response returned

**Participants:**

Customer

Angular UI

API Gateway

User Service

MongoDB

**2. Staff Registration (Needs Approval)**

Flow:

1. Technician/Manager enters details
2. Angular sends POST /api/auth/register/staff
3. User Service validates request
4. User saved with status=PENDING
5. Admin approves via PUT /api/admin/staff/{id}/approval
6. User Service publishes REGISTRATION\_APPROVED to RabbitMQ
7. Notification Service sends email

**Participants: Staff → Angular UI → API Gateway → User Service → MongoDB → RabbitMQ → Notification Service → Email**

**3. User Login**

Flow:

1. User submits login form
2. Angular calls POST /api/auth/login
3. User Service fetches user from MongoDB
4. Password verified
5. JWT token generated
6. Token returned to Angular
7. Angular stores token and navigates

Participants: User → Angular UI → API Gateway → User Service → MongoDB

**4. Service Request Creation**

Flow:

1. Customer selects vehicle and describes issue
2. Angular sends POST /api/customer/service-requests
3. Service Request Service creates request with status=PENDING
4. Saved to MongoDB

Participants: Customer → Angular UI → API Gateway → Service Request Service → MongoDB

**5. Technician Assignment**

Flow:

1. Manager views pending requests
2. Manager assigns technician and bay
3. PUT /api/manager/service-requests/{id}/assign
4. Service Request Service updates status=ASSIGNED
5. Feign call to User Service to update technician availability
6. RabbitMQ event: SERVICE\_STARTED
7. Notification Service sends email to customer

Participants: Manager → Angular UI → API Gateway → Service Request Service → User Service (Feign) → MongoDB → RabbitMQ → Notification Service → Customer Email

**5. Technician Assignment (MOST IMPORTANT)**

**Flow:**

1. Manager views pending requests
2. Manager assigns technician and bay
3. PUT /api/manager/service-requests/{id}/assign
4. Service Request Service updates status=ASSIGNED
5. Feign call to User Service to update technician availability
6. RabbitMQ event: SERVICE\_STARTED
7. Notification Service sends email to customer

**Participants:**

 Manager → Angular UI → API Gateway → Service Request Service → User Service (Feign) → MongoDB → RabbitMQ → Notification Service → Customer Email

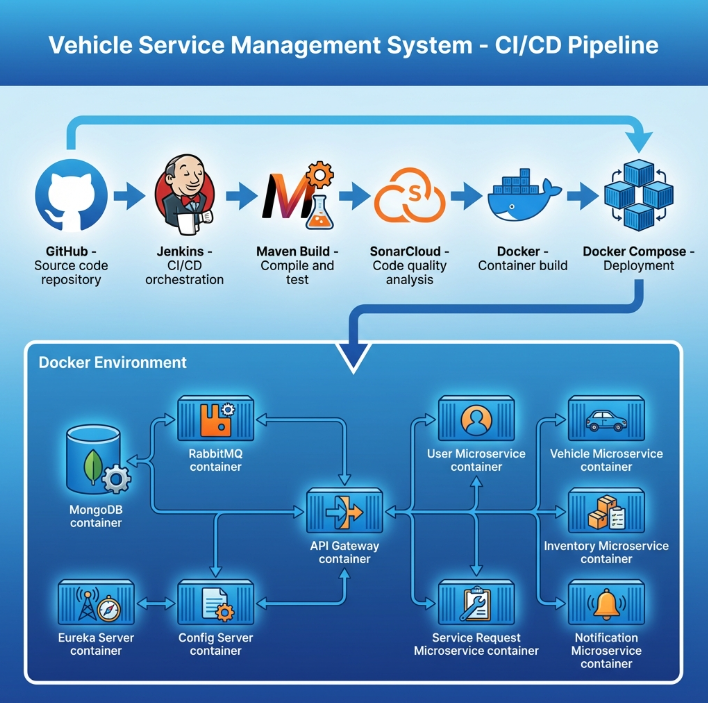
**7. Service Completion & Invoice**

**Flow:**

1. Technician updates status=COMPLETED
2. Manager closes service via PUT /api/manager/service-requests/{id}/close
3. Invoice auto-generated (labour + parts)
4. RabbitMQ event: INVOICE\_GENERATED
5. Notification Service sends invoice email
6. Customer pays via PUT /api/billing/invoices/{id}/pay

**Participants:** Technician → Manager → API Gateway → Service Request Service → MongoDB → RabbitMQ → Notification Service → Customer

## 6. CI/CD Pipeline — Sequence Diagram



**Scenario**

Developer pushes code to Git repository.

**Flow**

1. Developer pushes code
2. Jenkins pipeline triggered
3. Jenkins runs unit tests
4. Jenkins runs SonarQube scan
5. Quality gate checked
6. Docker images built
7. Docker Compose deploys services

**Participants**

* Developer
* Git
* Jenkins
* SonarQube
* Docker

## 7. Error Handling — Sequence Diagram

**Scenario**

Invalid request sent to backend.

**Flow**

1. Angular sends invalid request
2. Controller validation fails
3. Global exception handler triggered
4. Standard error response returned

**Key Point**

* Consistent error structure across services

**Docker-compose.yml:**

services:

  mongo:

    image: mongo:6

    container\_name: mongo

    restart: always

    ports:

      - "27017:27017"

    volumes:

      - mongo\_data:/data/db

  rabbitmq:

    image: rabbitmq:3-management

    container\_name: rabbitmq

    restart: always

    ports:

      - "5672:5672"

      - "15672:15672"

    volumes:

      - rabbitmq\_data:/var/lib/rabbitmq

  eureka:

    build: ./eureka-server

    container\_name: eureka

    restart: always

    ports:

      - "8761:8761"

  config-server:

    build: ./config-server

    container\_name: config-server

    restart: always

    ports:

      - "8888:8888"

    depends\_on:

      - eureka

  api-gateway:

    build: ./api-gateway

    container\_name: api-gateway

    restart: always

    ports:

      - "9090:8080"

    depends\_on:

      - eureka

      - config-server

    environment:

      - SPRING\_PROFILES\_ACTIVE=docker

      - SPRING\_CONFIG\_IMPORT=optional:configserver:http://config-server:8888

  user-service:

    build: ./user-service

    container\_name: user-service

    restart: always

    depends\_on:

      - mongo

      - eureka

      - config-server

    environment:

      - SPRING\_PROFILES\_ACTIVE=docker

      - SPRING\_CONFIG\_IMPORT=optional:configserver:http://config-server:8888

  vehicle-service:

    build: ./vehicle-service

    container\_name: vehicle-service

    restart: always

    depends\_on:

      - mongo

      - eureka

      - config-server

    environment:

      - SPRING\_PROFILES\_ACTIVE=docker

      - SPRING\_CONFIG\_IMPORT=optional:configserver:http://config-server:8888

  inventory-service:

    build: ./inventory-service

    container\_name: inventory-service

    restart: always

    depends\_on:

      - mongo

      - eureka

      - config-server

    environment:

      - SPRING\_PROFILES\_ACTIVE=docker

      - SPRING\_CONFIG\_IMPORT=optional:configserver:http://config-server:8888

  service-request-service:

    build: ./service-request-service

    container\_name: service-request-service

    restart: always

    depends\_on:

      - mongo

      - rabbitmq

      - eureka

      - config-server

    environment:

      - SPRING\_PROFILES\_ACTIVE=docker

      - SPRING\_CONFIG\_IMPORT=optional:configserver:http://config-server:8888

  notification-service:

    build: ./notification-service

    container\_name: notification-service

    restart: always

    depends\_on:

      - rabbitmq

      - eureka

      - config-server

    environment:

      - SPRING\_PROFILES\_ACTIVE=docker

      - SPRING\_CONFIG\_IMPORT=optional:configserver:http://config-server:8888

volumes:

  mongo\_data:

  rabbitmq\_data: