

**Jaypee Institute of Information Technology, Sector 62, Noida**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**



## **INTRODUCTION TO DEVOPS**

### **PROJECT REPORT**

**TITLE: MICROSERVICES LIBRARY MANAGEMENT USING GITHUB-ACTIONS, DOCKER, KUBERNETES**

**Submitted By:**

Suyash Saraswat (22103261)

Batch: B9

**Submitted To:**

Dr. K. Rajalakshmi  
Assistant Professor  
(CSE Department)

## INTRODUCTION

This project presents a fully containerized and automated **Library Management System** designed to demonstrate modern **microservices architecture and DevOps practices**. The system features multiple Flask-based backend services responsible for managing books, users, borrowing, and cataloging, while the client-side functionality—adding, listing, deleting books and users, and borrowing/returning books—is implemented using **HTML, CSS, and JavaScript**.

Although simple in functionality, the project highlights real-world **DevOps workflows**, including containerization, CI/CD pipelines, and automated deployment. A **Dockerfile** for each service and the frontend enables efficient, production-ready container builds, while **GitHub Actions** orchestrates a CI/CD lifecycle consisting of building and pushing Docker images, and deploying services to **Kubernetes**. **Infrastructure as Code (IaC)** is implemented through Docker, Docker Compose, and Kubernetes manifests, ensuring reproducibility and consistency across environments.

Overall, the Library Management System serves not only as a functional application but also as a **portfolio-ready DevOps case study**, providing hands-on exposure to the full DevOps lifecycle—from development and testing to packaging, deployment, and service orchestration—demonstrating skills in microservices, containerization, and automated software delivery.

### GITHUB LINK

<https://github.com/suyashsaraswat2003/library-microservices.git>

## Problem Statement

Many entry-level developers understand software development but lack hands-on experience with **microservices architecture and DevOps practices** used in real companies. Traditional academic projects focus only on application logic and not on:

- CI/CD automation
- Testing frameworks
- Code quality enforcement
- Containerized deployment
- Security scanning
- Infrastructure as Code

This project bridges that gap by creating an actual **Library Management System** and equipping it with a complete DevOps pipeline that mirrors professional workflows.

---

## Objectives

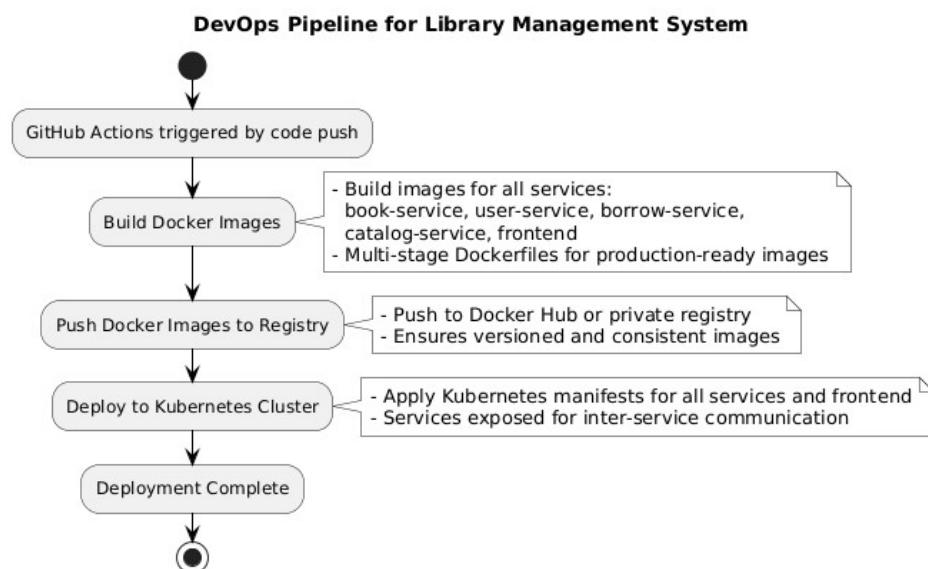
The primary objectives of this project are:

1. Build a functional, user-friendly **Library Management System**.
  2. Implement industry-grade **DevOps practices**.
  3. Develop a fully automated **CI/CD workflow**.
  4. Demonstrate **Docker containerization** for each microservice and frontend.
  5. Showcase **unit and integration testing** for backend APIs.
  6. Enforce **code quality and security standards**.
  7. Use **Infrastructure as Code (IaC)** through Docker, Docker Compose, and Kubernetes manifests.
-

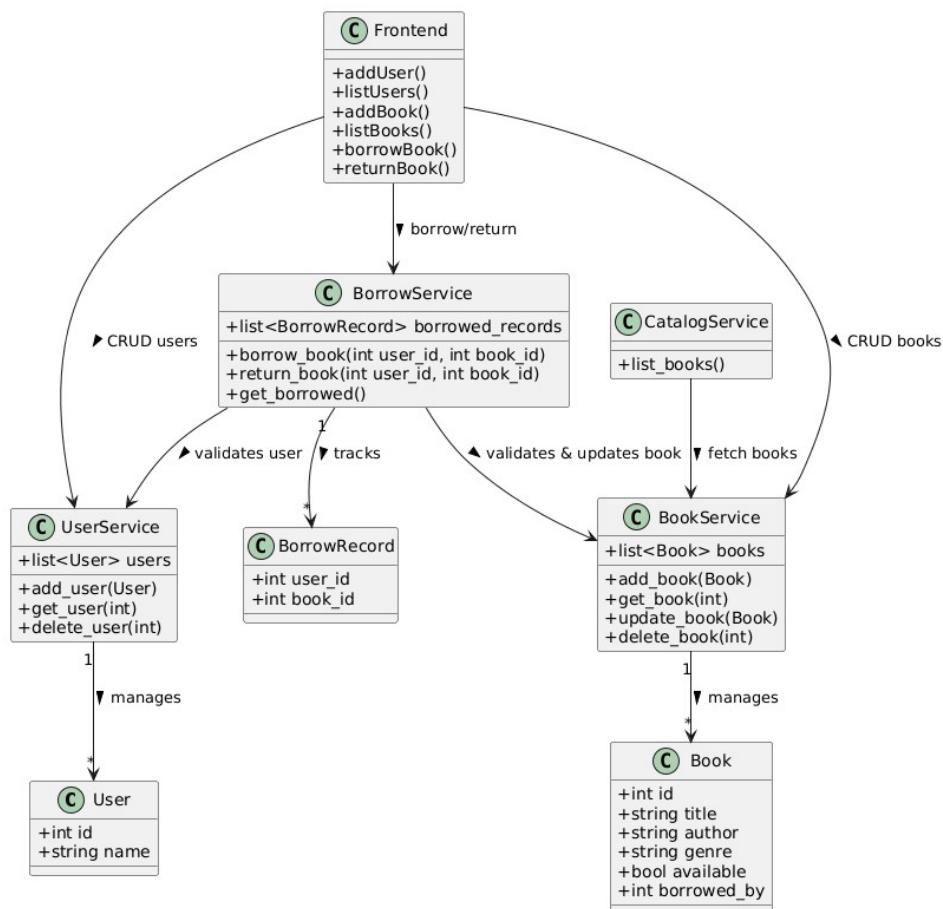
## FOLDER STRUCTURE

```
project/
└── services/
    ├── book-service/
    │   ├── main.py
    │   ├── requirements.txt
    │   └── Dockerfile
    ├── user-service/
    │   ├── main.py
    │   ├── requirements.txt
    │   └── Dockerfile
    ├── borrow-service/
    │   ├── main.py
    │   ├── requirements.txt
    │   └── Dockerfile
    └── catalog-service/
        ├── main.py
        ├── requirements.txt
        └── Dockerfile
    └── frontend/
        ├── index.html
        ├── script.js
        ├── style.css
        └── Dockerfile
    └── k8s/
        ├── book-deployment.yaml
        ├── user-deployment.yaml
        ├── borrow-deployment.yaml
        ├── catalog-deployment.yaml
        └── frontend-deployment.yaml
    └── docker-compose.yml
    .github/
        └── workflows/
            └── deploy.yml
```

## DEVOPS PIPELINE



## CLASS DIAGRAM



## DOCKER FILE

```

Dockerfile X
project > services > book-service > Dockerfile
1  FROM python:3.11-slim
2  WORKDIR /app
3  COPY requirements.txt .
4  RUN pip install --no-cache-dir -r requirements.txt
5  COPY . .
6  CMD ["python", "main.py"]
7
  
```

The Dockerfile for the book-service is located in the project's services directory. It specifies a Python 3.11-slim base image, sets the working directory to /app, copies the requirements.txt file, installs dependencies using pip, copies the local files, and runs the main.py script.

## Deploy.yaml

```
1   name: Build and Deploy Library Services
2
3   on:
4     push:
5       branches:
6         - main
7
8   jobs:
9     build-and-deploy:
10      runs-on: ubuntu-latest
11      env:
12        DEPLOY_TO_K8S: ${{ secrets.DEPLOY_TO_K8S }}
13
14      steps:
15        - name: Checkout repository
16          uses: actions/checkout@v3
17
18        - name: Set up Docker Buildx
19          uses: docker/setup-buildx-action@v3
20
21        - name: Log in to Docker Hub
22          uses: docker/login-action@v3
23          with:
24            username: ${{ secrets.DOCKERHUB_USERNAME }}
25            password: ${{ secrets.DOCKERHUB_TOKEN }}
26
27        - name: Build & Push Docker Images
28          run: |
29            docker build -t suyashsaraswat2003/book-service:latest ./services/book-service
30            docker push suyashsaraswat2003/book-service:latest
31            docker build -t suyashsaraswat2003/user-service:latest ./services/user-service
32            docker push suyashsaraswat2003/user-service:latest
33            docker build -t suyashsaraswat2003/borrow-service:latest ./services/borrow-service
34            docker push suyashsaraswat2003/borrow-service:latest
35            docker build -t suyashsaraswat2003/catalog-service:latest ./services/catalog-service
36            docker push suyashsaraswat2003/catalog-service:latest
37            docker build -t suyashsaraswat2003/frontend:latest ./frontend
38            docker push suyashsaraswat2003/frontend:latest
39
40        - name: Set up kubectl
41          if: ${{ env.DEPLOY_TO_K8S == 'true' }}
42          uses: azure/setup-kubectl@v3
43          with:
44            version: 'latest'
45
46        - name: Deploy to Kubernetes
47          if: ${{ env.DEPLOY_TO_K8S == 'true' }}
48          run: |
49            echo "${{ secrets.KUBE_CONFIG }}" | base64 --decode > kubeconfig.yaml
50            export KUBECONFIG=kubeconfig.yaml
51            kubectl apply -f k8s/
52            kubectl rollout status deployment/book-deployment
53            kubectl rollout status deployment/user-deployment
54            kubectl rollout status deployment/borrow-deployment
55            kubectl rollout status deployment/catalog-deployment
56            kubectl rollout status deployment/frontend-deployment
57
58        - name: Run Docker containers locally
59          if: ${{ env.DEPLOY_TO_K8S != 'true' }}
60          run: |
61            docker network create library-net || true
62            docker run -d --name book-service --network library-net suyashsaraswat2003/book-service:latest
63            docker run -d --name user-service --network library-net suyashsaraswat2003/user-service:latest
64            docker run -d --name borrow-service --network library-net suyashsaraswat2003/borrow-service:latest
65            docker run -d --name catalog-service --network library-net suyashsaraswat2003/catalog-service:latest
66            docker run -d -p 3000:3000 --name frontend --network library-net suyashsaraswat2003/frontend:latest
```

## System Architecture

### Client Layer

- HTML, CSS, JavaScript
- Manages user interactions for: add/list/delete users/books, borrow/return books
- Dynamic updates through API calls

### Server Layer (Flask Microservices)

- **Book Service** – manages books
- **User Service** – manages users
- **Borrow Service** – handles borrowing/returning books
- **Catalog Service** – queries books (optional)
- Each service exposes REST endpoints and is container-ready

### DevOps Layer

- Docker containers for all services and frontend
- **docker-compose** for local orchestration
- Kubernetes for deployment
- GitHub Actions for CI/CD pipelines

This modular architecture allows **easy maintenance and scalable deployment**.

## Conclusion

The **Library Management System with DevOps pipeline** demonstrates how modern DevOps practices enhance a microservices application by making it **reliable, maintainable, and production-ready**.

- Docker ensures **consistent deployment** across environments
- GitHub Actions automates **building and deployment**
- Kubernetes enables **scalable, modular service orchestration**

Overall, the project provides a **practical, portfolio-ready example** for learners to understand and implement **DevOps principles in real-world software projects**.