Project - High Level Design

on

ACADEMIA

Course Name: Devops

Institution Name: Medicaps University – Datagami Skill Based Course

Student Name(s) & Enrolment Number(s):

Sr no	Student Name	Enrolment Number
1.	Ujjawal Mehta	En22IT301114
2.	Ashish Rana	EN21CA301016
3.	Ravi Patidar	EN22IT301079
4.	Jay Patidar	EN22IT301040

Group Name: 06D12

Project Number: DO44

Industry Mentor Name:

University Mentor Name: Prof. Akshay Saksena

Academic Year: 2022-26

Table of Contents

1. Introduction

The Online Education App is a React-based web application designed to provide students with access to educational content through a simple and user-friendly interface. The application allows users to browse courses and explore learning materials efficiently.

This project focuses on deploying the application using Docker and Kubernetes (Minikube). Containerization ensures consistent deployment, while Kubernetes provides scalability, load balancing, and high availability. A key feature of the deployment is the implementation of rolling updates, which enables new versions of the application to be released without downtime. CI/CD tools such as Git and Jenkins/GitHub Actions are used to automate the build and deployment process.

1.1 Scope of the Document

This document presents the High-Level Design of deploying the Online Education App in a Kubernetes environment. It describes the system architecture, deployment workflow, key components, and integration between tools used for containerization and orchestration.

The document focuses on deployment and infrastructure design and does not include detailed source code or UI implementation.

1.2 Intended Audience

This document is intended for developers, DevOps engineers, faculty evaluators, and students who want to understand the deployment architecture and working of the Online Education App. It helps technical readers understand system components, workflows, and deployment strategies.

1.3 System Overview

The Online Education App is deployed as a containerized React application running inside a Kubernetes cluster. Users access the application through a web browser, and requests are routed via a Kubernetes Service to application pods.

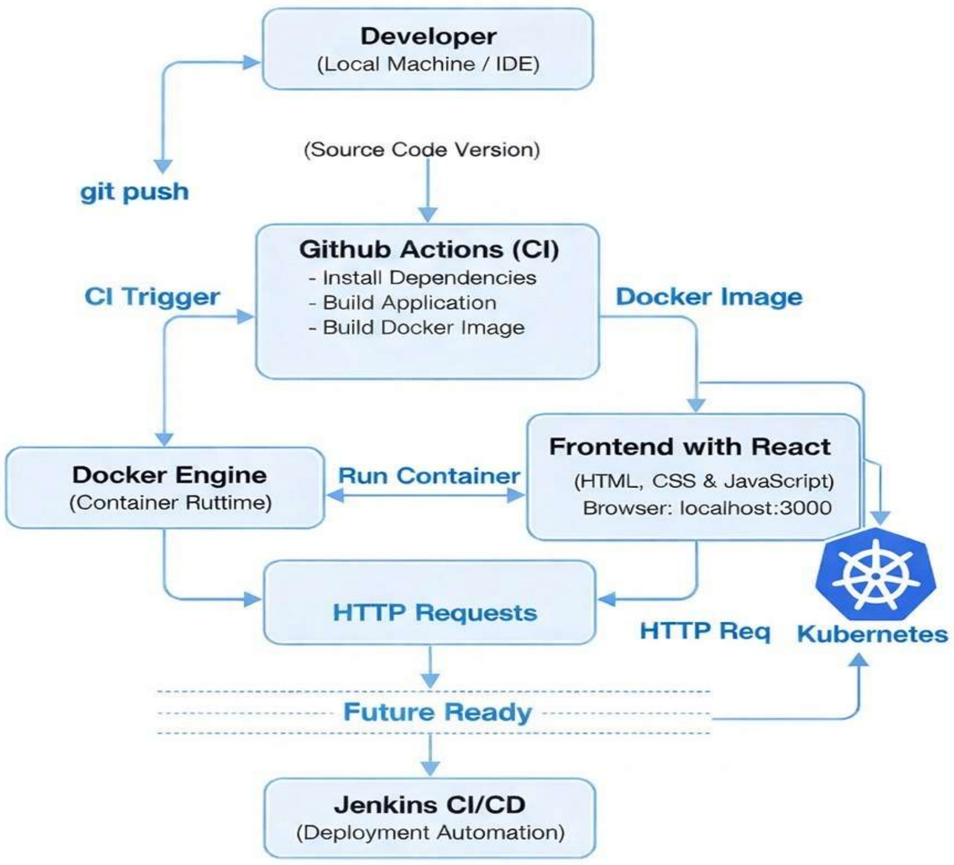
The system uses Git for version control, CI/CD tools for automated builds, Docker for containerization, and Kubernetes for managing deployment and scaling. Rolling updates ensure that application upgrades occur without interrupting user access.

2. System Design

The system design explains how the Online Education App is deployed and managed using containerization and Kubernetes orchestration. The React application is packaged into a Docker container and deployed inside a Kubernetes cluster using Minikube. A CI/CD pipeline automates the process of building and pushing container images whenever code changes are made. Kubernetes ensures application availability, load balancing, and rolling updates for seamless version upgrades.

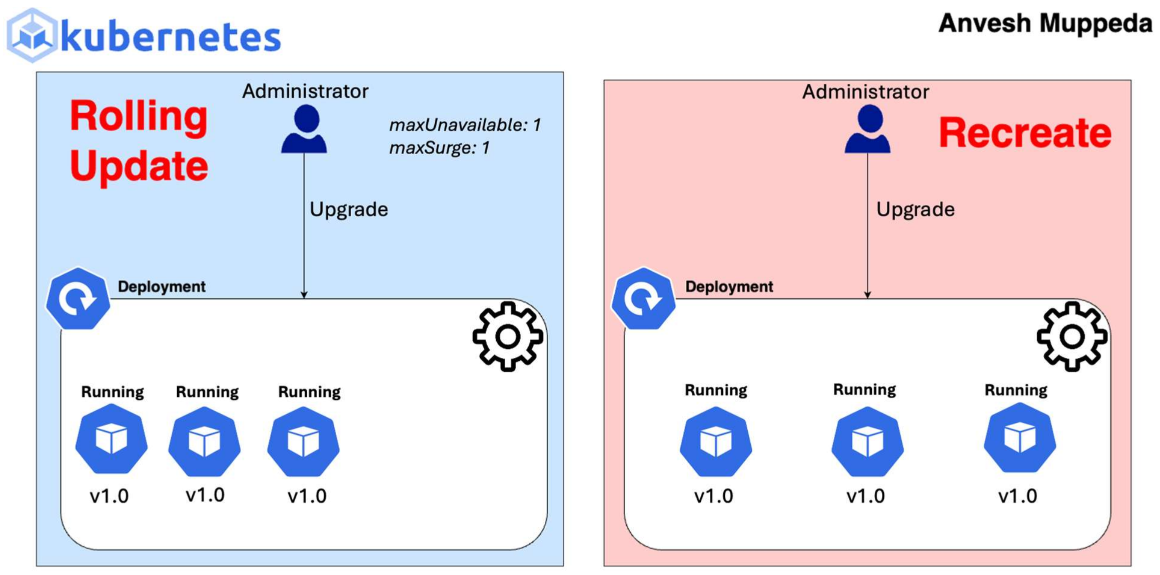
2.1 Application Design

The application is developed using React.js and provides an intuitive interface for accessing educational content. After building the React application, the optimized static files are served through an Nginx web server inside a Docker container. Docker ensures consistent runtime behavior across environments, while Kubernetes manages container deployment and ensures that the application remains available even if a pod fails.



2.2 Process Flow

The deployment process starts when a developer pushes code changes to the Git repository. The CI/CD pipeline automatically triggers the build process, compiles the React application, and creates a Docker image. The image is pushed to a container registry, from where Kubernetes pulls the updated version. Kubernetes then performs a rolling update, gradually replacing old pods with new ones to ensure uninterrupted user access.



Deployment Strategies

2.3 Information Flow

When users access the application through a browser, their requests are routed to the Kubernetes Service, which forwards traffic to one of the running pods hosting the application. The pod processes the request and returns the response to the user. During deployment, information flows from the developer to the Git repository, through the CI/CD pipeline, into the container registry, and finally to the Kubernetes cluster where the application is deployed.

2.4 Components Design

The system consists of multiple components working together to deploy and run the application efficiently. The React Education App provides the user interface. GitHub stores and manages the source code. Docker packages the application into portable containers. The CI/CD pipeline automates the build and deployment workflow. Kubernetes manages container orchestration, scaling, and availability through demonstrate Deployments and Services. Minikube provides the local cluster environment for development and testing.

2.5 Key Design Considerations

The design ensures high availability by running multiple pod replicas and leveraging Kubernetes self-healing capabilities. Rolling updates are used to prevent downtime during application upgrades. Scalability is supported through replica scaling, allowing the system to handle increased traffic. Automation through CI/CD improves maintainability and reduces manual errors. The containerized approach ensures portability and prepares the system for future cloud deployment.

2.6 API Catalogue

The current version of the Online Education App serves static frontend content and does not require external APIs. However, the architecture supports future integration with backend services such as authentication systems, course management APIs, and progress tracking services to enhance functionality.

3. Data Design

The current implementation of the Online Education App delivers static educational content through a React-based frontend. Since the application does not collect or store user-generated data, a traditional database is not required at this stage. All application resources such as HTML, CSS, JavaScript, and media files are served directly from the containerized web server.

However, the system architecture is designed to support future expansion. If dynamic features such as user login, course enrollment, and progress tracking are added, a database can be integrated seamlessly within the Kubernetes environment.

Key Points:

- Static content delivery through frontend
- No database required in current version
- Containerized architecture supports future database integration
- Scalable design for dynamic data support

3.1 Data Model

Currently, the application does not use a structured data model because it serves static content. The course information and educational materials are embedded within the frontend application.

In future versions, a structured data model may be implemented to manage users, courses, and learning progress. This will enable personalized learning experiences and progress tracking.

Key Points:

- No structured data model currently
- Content embedded within frontend
- Future entities may include:
    - Users
    - Courses
    - Enrollments
    - Progress records

3.2 Data Access Mechanism

At present, data access occurs through static file delivery. When a user opens the application, the browser downloads the required frontend assets served by the Nginx server running inside the Docker container.

If backend services are added later, the frontend can retrieve data using REST APIs, enabling dynamic communication between the user interface and the database.

Key Points:

- Static files served via Nginx container
- Browser loads assets to render UI
- Future support for REST API-based data access
- Kubernetes services can enable backend communication

3.3 Data Retention Policies

Since the current application does not store user information, no long-term data retention policies are required. However, container logs may be temporarily stored for monitoring and debugging purposes.

If user data storage is implemented in the future, retention policies will ensure secure storage, privacy protection, and compliance with data management standards.

Key Points:

- No user data stored currently
- Temporary storage of container logs
- Future policies may include:
    - Secure storage practices
    - Privacy protection
    - Data lifecycle management

3.4 Data Migration

Data migration is not required in the current system because no persistent data is stored. Application updates are handled through Docker image versioning and

Kubernetes rolling updates, which replace older application instances without affecting availability.

In future implementations involving databases, migration strategies such as schema versioning, backups, and rollback mechanisms can be implemented to ensure smooth upgrades.

Key Points:

- No migration required currently
- Updates handled via container versioning
- Rolling updates ensure seamless transitions
- Future database migration strategies may include:
  - Schema versioning
  - Backup & restore
  - Rollback support

4. Interfaces

The Online Education App interacts with users and system components through well-defined interfaces that enable smooth communication and reliable application delivery. Users access the application through a web browser interface, while deployment and infrastructure components communicate through automated pipelines and service integrations.

The user interface is a React-based web frontend that allows students to browse courses and access educational content from any device with internet access. On the system side, the application integrates with GitHub for version control, CI/CD tools for automated build and deployment, Docker for container image management, and Kubernetes for orchestration and service exposure.

Network communication is managed by Kubernetes Services, which route incoming user requests to the appropriate application pods and provide load balancing to ensure availability and performance.

The architecture is also designed to support future integration with external services such as authentication systems, cloud storage, or backend APIs to enhance functionality.

Key Points:

- Browser-based user interface for accessing the application
- Integration between GitHub, CI/CD pipeline, Docker registry, and Kubernetes
- Kubernetes Service manages routing and load balancing
- Enables automated deployment and system communication
- Supports future integration with external services

5. State and Session Management

The Online Education App follows a stateless architecture, which means each user request is processed independently without storing session data on the server. Since the current implementation delivers static educational content through a React frontend, user sessions are not required for accessing the application.

Application state related to navigation and user interactions is managed on the client side using React state management. This ensures smooth page transitions and responsive user experience without requiring server-side session storage.

Because the application runs inside containerized environments managed by Kubernetes, maintaining a stateless design improves scalability and reliability. Stateless services allow pods to be replaced, scaled, or updated without affecting user experience.

In future enhancements, if features such as user authentication or personalized learning are introduced, session management can be implemented using secure tokens (such as JWT) or external session stores.

Key Points:

- Stateless architecture ensures scalability and reliability
- No server-side session storage required currently
- React manages UI state on the client side
- Kubernetes benefits from stateless application design
- Future enhancements may use token-based authentication (JWT)

6. Caching

Caching is used to improve application performance and reduce loading time by storing frequently accessed resources temporarily. In the Online Education App, caching primarily occurs at the client (browser) level, where static assets such as HTML, CSS, JavaScript, and images are stored after the first load. This allows the application to load faster on subsequent visits and reduces server requests.

Since the application serves static content through an Nginx container, HTTP caching headers can be configured to enable efficient browser caching. This improves responsiveness and minimizes bandwidth usage.

The stateless architecture and containerized deployment allow caching mechanisms to be implemented without affecting scalability. In future deployments, advanced caching strategies such as Content Delivery Networks (CDNs) or reverse proxy caching can be introduced to further enhance performance and support large-scale traffic.

Key Points:

- Browser caching stores static assets for faster loading
- Reduces repeated server requests and bandwidth usage
- Nginx can enable HTTP caching headers
- Improves performance and user experience
- Future enhancements may include CDN or reverse proxy caching

7. Non-Functional Requirements

Non-functional requirements define the quality attributes of the Online Education App, ensuring that the system remains reliable, scalable, secure, and efficient. Since the application is deployed using Docker containers and Kubernetes orchestration, it benefits from high availability, fault tolerance, and automated recovery.

The stateless architecture improves scalability and allows the application to handle increased user traffic by scaling pods horizontally. CI/CD automation enhances maintainability and ensures consistent deployments. The system is designed to provide reliable access, fast response times, and smooth updates without service interruptions.

Key Points:

- High availability through Kubernetes orchestration
- Scalability using horizontal pod scaling
- Reliability via self-healing containers
- Maintainability through CI/CD automation
- Seamless updates using rolling deployment

7.1 Security Aspects

Security is an important consideration in the deployment of containerized applications. The system uses trusted Docker base images to reduce vulnerabilities and ensures secure image storage in the container registry. Access to the Kubernetes cluster can be controlled using role-based access control (RBAC) to prevent unauthorized operations.

Sensitive configuration data such as environment variables and credentials can be stored securely using Kubernetes Secrets. Network security can be enhanced through HTTPS and secure ingress configurations in future deployments.

Key Points:

- Use trusted and minimal Docker images
- Secure container registry access
- Kubernetes RBAC for access control
- Secure storage of secrets using Kubernetes Secrets
- HTTPS and secure ingress for future enhancements

7.2 Performance Aspects

Performance optimization ensures that the application loads quickly and provides a smooth user experience. Since the Online Education App serves static content, performance is improved through lightweight Docker images and efficient content delivery via the Nginx web server.

Kubernetes Services distribute incoming traffic across multiple pods, ensuring balanced load distribution and improved responsiveness. Browser caching further enhances performance by reducing repeated data transfers.

The architecture supports future performance improvements such as auto-scaling, CDN integration, and resource optimization.

Key Points:

- Lightweight container images improve speed
- Nginx efficiently serves static content
- Load balancing across pods improves responsiveness
- Browser caching reduces load time
- Future support for auto-scaling and CDN integration

8. References

This project design and deployment approach is based on industry-standard documentation and best practices for containerization, orchestration, and frontend application deployment.

The following resources were referred to for understanding the tools and technologies used in the project:

- Kubernetes Official Documentation – for cluster management, deployments, and rolling updates
- Docker Documentation – for containerization and image management
- React Official Documentation – for frontend development practices
- GitHub Guides – for version control and repository management
- CI/CD Documentation (Jenkins / GitHub Actions) – for automated build and deployment workflows
- DevOps and Kubernetes best practice guides for container orchestration and deployment strategies

Key Points:

- Based on official documentation and best practices
- References include containerization and orchestration tools
- Supports industry-standard deployment approach