Table of Contents

Project Title: Centralization of All Projects on One Server

Objective:

In modern development environments, managing multiple projects across different servers is complex due to variations in configurations and dependencies. This project aims to centralize all projects onto a single server using Docker and Kubernetes to create a unified platform for efficient management, deployment, and monitoring.

By centralizing projects, this approach:

1. Simplifies infrastructure management by consolidating multiple projects.
2. Enhances resource utilization by optimizing server usage.
3. Improves security through centralized updates and monitoring.
4. Reduces operational costs by minimizing redundant resource allocation.

This documentation explains the setup, technologies used, and the structure of the project, making it easy for anyone—regardless of experience—to understand and continue the work.

Technologies Used:

1. Frontend: React.js
2. Backend: Node.js
3. Containerization: Docker, Docker Compose
4. Database: MongoDB
5. Version Control: GitHub

GitHub Repository:
Project Repository: https://github.com/hshdacs/ACS_Projects

System Architecture and Design

Architecture Overview:

The project follows a microservices-based architecture, where each sub-project is an independent service running inside a Docker container. These containers communicate through Docker Compose or Kubernetes(if scaling is required in the future).

Directory Structure:

The project follows a well-organized directory structure:

```
project-root/ - Centralized Dashboard
 ├── docker-compose.yml  (Defines how services run together)
 ├── java-app/
|    ├── Dockerfile  (java project container configuration)
|    ├── src/
|   └── pom.xml
 ├── react-app/
|    ├── Dockerfile  (React project container configuration)
|    ├── src/
|    ├── public/
|   └── package.json
 ├── angular app/
|    ├── Dockerfile  (Angular project container configuration)
|    ├── src/
|   └── package.json
└── other-sub-project/
    ├── Dockerfile  (Other projects)
    ├── src/
   └── package.json
```

Components:

Frontend:
Built using React.js, providing an interactive dashboard.
Displays project cards, each representing a different service.
Allows starting, stopping, and viewing project details dynamically.

Backend:
Developed with Node.js, handling server requests.
Executes Docker commands to manage project containers.
Connects to MongoDB for storing project metadata.

Docker Containers:
Each project runs in a separate, isolated container.
This ensures environment consistency across different systems.

Design Decisions:

1. Microservices Model: Each project runs independently, improving scalability and maintainability.
2. Dockerized Deployment: Containers ensure consistent environments for all projects.
3. MongoDB for Configuration Storage: Stores metadata about each project, making the system dynamically configurable.

Setup and Implementation

Prerequisites:

Before running the project, install the following:

1. Node.js (v14+): Required for backend and frontend development.
2. Docker Desktop: To run containerized services.
3. MongoDB: For storing project metadata.
4. Npm: To manage JavaScript dependencies.
5. Git: For cloning the project repository.

Setup Steps:

Clone the Repository
```sh
git clone <repository_url>
cd project-root
```
Install Dependencies

Backend:
```sh
cd backend
npm install
```
Frontend:
```sh
cd ../react-app
npm install
```

Configure Environment Variables
Create a `.env` file in the backend directory with the following details:
```
MONGODB_URI=mongodb://<your-mongodb-url>:<port>/<database-name>
PORT=5000
```

Start MongoDB
If using a local MongoDB server:
```sh
mongod
```
If using MongoDB Atlas, ensure the connection string is set in `.env`.

Running the Project

Option 1: Using Docker Compose (Recommended)
Run all services using Docker Compose:
```sh
docker-compose up --build
```
Access the application:
Frontend: http://localhost:3000 (http://localhost:3000)
Backend API: http://localhost:5000 (http://localhost:5000)

Option 2: Running Without Docker
If Docker is unavailable, start each service manually:
```sh
Start backend
cd backend
npm run dev

Start frontend
cd ../react-app
npm start
```

Configuration Management

MongoDB Collections:

1. **Configuration Collection**

Stores configuration details for each project.
```json
{
  "projectId": "ObjectId",
  "serviceName": [
    { "name": "frontend", "port": 3000, "type": "React" },
    { "name": "backend", "port": 5000, "type": "Node.js" }
  ],
  "dockerFiles": [
    { "fileName": "Dockerfile", "content": "Dockerfile content here" }
  ]
}
```

2. **Project Collection**

Holds general project information.
```json
{
  "projectTitle": "Centralized Dashboard",
  "createdDate": "2024-02-10",
  "longDescription": "A centralized dashboard for managing projects.",
  "technologiesUsed": ["React", "Node.js", "MongoDB"],
  "githubUrl": "https://github.com/hshdacs/ACS_Projects"
}
```

Additional Notes

1. Port Conflicts: Ensure defined ports are free before starting services.
2. Updating Dependencies: Use `npm update` to keep packages up to date.
3. Security Best Practices: Avoid hardcoding sensitive credentials in `.env` files.
4. Error Handling: Implement error logging in the backend for debugging issues.

Conclusion

This documentation provides all necessary details to set up, configure, and maintain the centralized project management system. By following this guide, any developer—regardless of prior experience—can continue the development and maintenance of this project seamlessly.