Multi-Environment Application Deployment Assessment([LI nmNK](https://github.com/mohanDevOps-arch/MultienvApp.git))

**Total Time: 1 hour  
Total Marks: 50 (Assessment: 45 marks + VIVA: 5 marks)**

Problem Statement

**You are provided with a multi-environment ticket management application that needs to be deployed on your local computer. The application consists of two Flask services: development backend, production backend, and a frontend service (React) that interacts with both environments.**

Task Overview

**Your task is to:**

1. **Create necessary Docker configurations for each service**
2. **Deploy the entire application locally**
3. **Ensure proper communication between all services**
4. **Document your deployment process**
5. **Demonstrate working application in VIVA**

Requirements

1. Infrastructure Setup (8 marks)

* **Launch appropriate EC2 instance (or use local environment)**
* **Configure security groups (if using EC2)**
* **Setup necessary networking components**
* **Document environment specifications**

2. Docker Configuration (12 marks)

* **Create Dockerfile for backend development environment**
* **Create Dockerfile for backend production environment**
* **Create Dockerfile for frontend service**
* **Create docker-compose.yml file**
* **Ensure proper environment variable configuration**

3. Application Deployment (15 marks)

* **Install required tools (Docker, Docker Compose)**
* **Deploy application using Docker Compose**
* **Ensure all services are running correctly**
* **Verify connectivity between services**

4. Documentation & Testing (10 marks)

* **Document deployment steps**
* **Provide testing evidence with screenshots**
* **List any assumptions made**
* **Document access URLs and ports**

5. VIVA Assessment (5 marks)

* **Demonstrate running application**
* **Explain Docker configurations and deployment process**
* **Answer questions about troubleshooting and modifications**

Application Structure

|  |
| --- |
| multiEnv/ ├── docker-compose.yml ├── backend/ │   ├── dev/ │   │   ├── app.py │   │   ├── requirements.txt │   │   ├── Dockerfile │   │   └── .env │   └── prod/ │       ├── app.py │       ├── requirements.txt │       ├── Dockerfile │       └── .env └── frontend/     ├── src/     ├── public/     ├── Dockerfile     └── package.json |

Marking Criteria

Infrastructure Setup (8 marks)

* **EC2 Instance setup with appropriate size OR local environment configuration**
* **Security group configuration OR local network setup**
* **Network configuration and documentation**

Docker Configuration (12 marks)

* **Backend development Dockerfile with proper configuration**
* **Backend production Dockerfile with proper configuration**
* **Frontend Dockerfile with optimized build process**
* **Docker Compose configuration with proper service definitions**

Application Deployment (15 marks)

* **Successful installation and configuration of required tools**
* **Successful application deployment with all services running**
* **Proper service connectivity and inter-service communication**
* **Error handling, logging, and troubleshooting evidence**

Documentation & Testing (10 marks)

* **Comprehensive deployment documentation with clear steps**
* **Testing evidence with screenshots of running services**
* **Clear documentation of assumptions and limitations**
* **Complete access information including URLs and ports**

VIVA Assessment (5 marks)

* **Live demonstration of fully functional application**
* **Clear explanation of Docker configurations and deployment choices**
* **Ability to answer questions about troubleshooting and potential modifications**

**Deliverables**

1. Technical Files

* **All Docker configuration files (Dockerfiles for all services)**
* **docker-compose.yml with proper service definitions**
* **Environment configuration files**

2. Documentation Package

* **Deployment Guide including:**
  + **Environment setup steps (EC2 )**
  + **Security group configurations (if applicable)**
  + **Docker installation and configuration**
  + **Step-by-step deployment commands**
  + **Testing procedures and verification steps**
  + **Access URLs and port configurations**

3. Evidence Package

* **Screenshots of:**
  + **Running Docker containers (docker ps)**
  + **Application access on all endpoints**
  + **EC2 configuration (if used)**
  + **Service logs demonstrating proper functionality**
  + **Network connectivity tests**

Important Notes

Port Configuration

**The application uses the following ports:**

* **Frontend: 3000**
* **Development Backend: 3001**
* **Production Backend: 3002**

Required URLs after deployment:

* **Development Environment: http://[IP]:3000/dev**
* **Production Environment: http://[IP]:3000/prod**
* **Frontend Dashboard: http://[IP]:300**

Evaluation Guidelines

Full Marks Criteria

* **All services running successfully in containers**
* **Frontend successfully connects to both backend environments**
* **Application accessible via configured URLs**
* **All environment variables properly set**
* **Complete and clear documentation**
* **Successful live demonstration in VIVA**

Success Criteria

Minimum Passing Requirements

1. **At least 2 out of 3 services running in containers**
2. **Basic documentation of deployment process**
3. **Evidence of deployment attempts with troubleshooting notes**

Excellence Criteria

1. **All services running in containers with proper networking**
2. **Frontend successfully connecting to both backend environments**
3. **Application accessible via all required URLs**
4. **Comprehensive documentation with troubleshooting guides**
5. **Clear demonstration of understanding during VIVA**
6. **Proper error handling and logging implementation**

Submission Guidelines

Repository Submission

* **Create a public GitHub repository for your project**
* **Repository name format: multienv-deployment-[Name]**
* **Push all project files to the repository following the specified directory structure**
* **Include a comprehensive README.md with:**
  + **Quick start instructions**
  + **Prerequisites and dependencies**
  + **Step-by-step deployment guide**
  + **Troubleshooting section**
  + **Screenshots and evidence**

Final Submission Process

1. **Complete your repository with all files and documentation**

**Create a text file named submission-info.txt in the submission/ folder containing:**

* **Student Name: [Your Full Name]**
* **Repository URL: [Your GitHub Repository Link]**
* **Deployment Status: [Working/Partial/Issues]**
* **Access URLs:**
* **- Development: http://[IP]:3000/dev**
* **- Production: http://[IP]:3000/prod**
* **- Frontend: http://[IP]:3000**

1. **Date of Submission: [DD-MM-YYYY]**
2. **Submit the submission-info.txt file through the designated submission link provided by instructor**
3. **Ensure repository is public and accessible for evaluation**

Time Management Recommendations

* **Infrastructure Setup: 10 minutes**
* **Docker Configuration: 20 minutes**
* **Application Deployment: 20 minutes**
* **Testing & Verification: 5 minutes**
* **Documentation: 5 minutes**

**Note: Prioritize getting a working deployment over perfect documentation. A partially documented working application scores higher than a perfectly documented non-working application.**

------------------------------------------------------------------------------------------------------------------------**1. Task Breakdown**

**You need to:**

* Containerize each backend (dev/prod) and the frontend with Dockerfiles.
* Use Docker Compose to orchestrate all services (including MongoDB).
* Ensure all services communicate correctly (networking, environment variables).
* Document every step, including setup, deployment, and testing.
* Prepare for a live demonstration and Q&A.

**2. Key Requirements**

**Infrastructure Setup**

* Local environment or EC2 (if EC2, configure security groups and networking).
* Document environment specs (OS, Docker version, etc.).

**Docker Configuration**

* Dockerfile for each backend (backend/dev/Dockerfile, backend/prod/Dockerfile).
* Dockerfile for frontend (frontend/Dockerfile).
* docker-compose.yml at the root to define all services and networks.
* Use .env files for environment variables (especially MongoDB URI).

**Application Deployment**

* Install Docker and Docker Compose.
* Deploy with docker-compose up.
* Verify all services are running and can communicate (frontend can call both backends, both backends connect to MongoDB).
* Check logs and handle errors.

**Documentation & Testing**

* Step-by-step deployment guide.
* Screenshots of running containers, endpoints, logs.
* List assumptions and limitations.
* Document all URLs and ports.

**VIVA**

* Be ready to explain Dockerfiles, Compose, networking, troubleshooting, and possible improvements.

**3. Application Structure (as required)**

Your project already matches the required structure:

* [dev](vscode-file://vscode-app/c:/Users/sneha/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) and [prod](vscode-file://vscode-app/c:/Users/sneha/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) each with [app.py](vscode-file://vscode-app/c:/Users/sneha/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), [requirements.txt](vscode-file://vscode-app/c:/Users/sneha/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), and will need a Dockerfile and .env.
* [frontend](vscode-file://vscode-app/c:/Users/sneha/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) with Dockerfile, [package.json](vscode-file://vscode-app/c:/Users/sneha/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o "), etc.
* Root-level docker-compose.yml.

**4. Service Ports & URLs**

* Frontend: 3000
* Dev backend: 3001
* Prod backend: 3002
* URLs:
  + Dev: http://[IP]:3000/dev
  + Prod: http://[IP]:3000/prod
  + Dashboard: http://[IP]:3000

**5. Next Steps**

1. **Write Dockerfiles** for each service.
2. **Write docker-compose.yml** to define all services and networks.
3. **Create .env files** for each backend with the correct MONGO\_URI.
4. **Test locally**: docker-compose up --build
5. **Document** every step, including troubleshooting.
6. **Prepare screenshots** for evidence.

**6. Assumptions**

* MongoDB will be run as a container in Compose.
* All services will be on the same Docker network.
* Environment variables will be passed via .env files or Compose.