# Techdome Application Documentation

**Overview**

The application is a multi-container setup that consists of a frontend, backend, and database. The primary goals of the application are to facilitate a streamlined build, deployment, and management process using Docker and Kubernetes in a local development environment.

## **1. Application Architecture**

### **Components**

1. **Frontend (React Application)**
   * Hosted in a Docker container and served over port 3000.
   * Connects to the backend via a specified API URL for accessing data and performing actions.
2. **Backend (Node.js Application)**
   * Hosted in a Docker container with an Express server.
   * Listens on a dynamic port specified in an environment variable, connecting to the MongoDB database to handle data operations.
3. **MongoDB Database**
   * Containerized to simplify the local setup.
   * Stores application data and can be configured through environment variables.

### **Networking**

The application components communicate via an internal Docker network (app-network), enabling seamless interaction between the frontend, backend, and database.

## **2. Deployment Strategy**

The deployment leverages Docker Compose for container orchestration. This strategy allows for straightforward deployment on a AWS and ensures each service is contained within its respective environment.

### Docker Compose Configuration

A Docker Compose file (docker-compose.yml) orchestrates the multi-container setup. Key configurations include:

* **Frontend**: Specifies build context, Dockerfile, port mappings, and environment variables.
* **Backend**: Specifies build context, Dockerfile, dynamic port mapping from the .env file, and necessary environment variables for database connection.
* **Database**: Runs a MongoDB instance with data persistence (if configured).



### Dockerfile Customizations

Each component has a dedicated Dockerfile:

* **Dockerfile.frontend** for the React frontend.
* **Dockerfile.backend** for the Node.js backend.

## **3. Instructions for Building, Deploying, and Managing the Application**

### Prerequisites

Ensure Docker and Docker Compose are installed.

### Step-by-Step Guide

#### 1. **Clone the Repository**

git clone  Backend: https://github.com/Anand-1432/Techdome-backend

git clone  Frontend: <https://github.com/Anand-1432/Techdome-frontend>

#### 2. **Set Up Environment Variables**

Create an .env file in the project root with the following variables:

PORT=5000

DB=mongodb://mongo:27017/mydatabase

#### 3. **Build and Deploy the Application**

Use the following command to build the application without using cache:

docker-compose up --build --no-cache

This command will:

* Rebuild each service (frontend, backend, MongoDB) from scratch.
* Deploy containers on the AWS machine, accessible at http://publicip:3000 for the frontend.

## **4. Challenges Faced**

### Module Errors and Build Caching

* **Issue**: Encountered MODULE\_NOT\_FOUND error indicating missing dependencies due to Docker build cache.
* **Resolution**: **Used --no-cache option with docker-compose up --build to ensure a clean build.** Upgraded Docker Compose version to v2.5.0 to support this flag.

### Storage Constraints

* **Issue**: Limited disk space led to an unknown flag: --no-cache error and stalled deployments.
* **Resolution**: Cleaned up Docker images, containers, volumes, and unnecessary files using the following commands:

docker system prune -a --volumes

sudo apt-get clean && sudo apt-get autoremove

### Snap Package Storage

* **Issue**: Snap packages occupied excessive disk space, causing root filesystem saturation.
* **Resolution**: Removed old, unused Snap packages to free space

### Frontend 404 Error: Unable to Connect to Backend

* **Issue**: The frontend displayed a 404 error when attempting to connect to the backend. This was due to a misconfigured API URL environment variable in the frontend service.
* **Resolution**: Corrected the API URL in the Docker Compose configuration to point to http://backend:5000, aligning it with the backend service name and port within the Docker network.

