Use Case on Automating Node.js and MongoDB Application Deployment and Testing Using Docker

Introduction:

TechFlex, a SaaS startup, is launching a real-time collaboration tool. To ensure a seamless user experience, the backend services powered by Node.js and MongoDB need to be scalable and reliable. With high traffic expected during the product launch, the DevOps team has been tasked with automating the deployment and testing of the application using Docker Compose. This automation ensures consistent, efficient deployments and verifies the functionality of the services in a controlled environment.

Background:

In previous launches, manual deployments of the Node.js application and MongoDB instances led to inconsistencies in environment configuration and downtime. These issues resulted in customer dissatisfaction and revenue loss. To resolve this, the DevOps team plans to containerize the application and database, automating the deployment and testing process to ensure both services operate reliably under load.

This use case demonstrates how automating the deployment of Node.js and MongoDB services transforms TechFlex's infrastructure, enhancing scalability and robustness while reducing downtime during critical periods.

Objective:

To automate the deployment of a Node.js application and MongoDB database using Docker Compose and verify their functionality through Python-based test scripts.

Pre-requisites:

- Docker installed on your machine.
- Docker Compose installed.
- Python 3.x installed along with docker and requests libraries.
- Git installed and configured.
- Access to a GitHub repository for version control.

---

1. Prerequisites

Before starting, ensure you have the following installed:

- **Docker** and **Docker Compose**: For containerized deployment.
- **Node.js** and **npm** For local development and testing.

---

2. Understand the Files

- **Dockerfile**: Builds the Docker image for the Node.js application.
- **docker-compose.yml**: Sets up services (app, database) with Docker Compose.
- **server.js**: Entry point for the Node.js backend.
- **routes/**: Handles routes for authentication and polls.
- **public/**: Contains frontend HTML files for the app.

3.You are requested to create a node js application connected with mongo db with the locatl environment .

Ensure the following folder structure is in place after the files:

Create the folder poll-app

```
poll-app/
├── Dockerfile
├── docker-compose.yml
├── package.json
├── server.js
├── routes/
│   ├── auth.js
│   └── polls.js
├── public/
│   ├── index.html
│   ├── login.html
│   └── poll.html
```

---

4 .Database schema

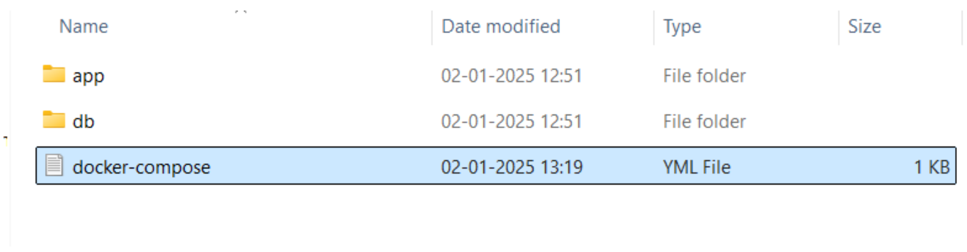
Make sure the username and password is kept static

Username : admin

Password : password123

After the application is done the backend must be able to authenticate using the username and password

5 Create the docker file in yml format



Name	Date modified	Type	Size
app	02-01-2025 12:51	File folder	
db	02-01-2025 12:51	File folder	
docker-compose	02-01-2025 13:19	YML File	1 KB

The yml file should have the configuration for the application

Create the Docker Environment

Create app directory

WORKDIR /usr/src/app

Build and Start the Application

Run the following commands in the poll-app/ directory:

1. **Build and Start the Containers**:

use the docker compose to create the docker application

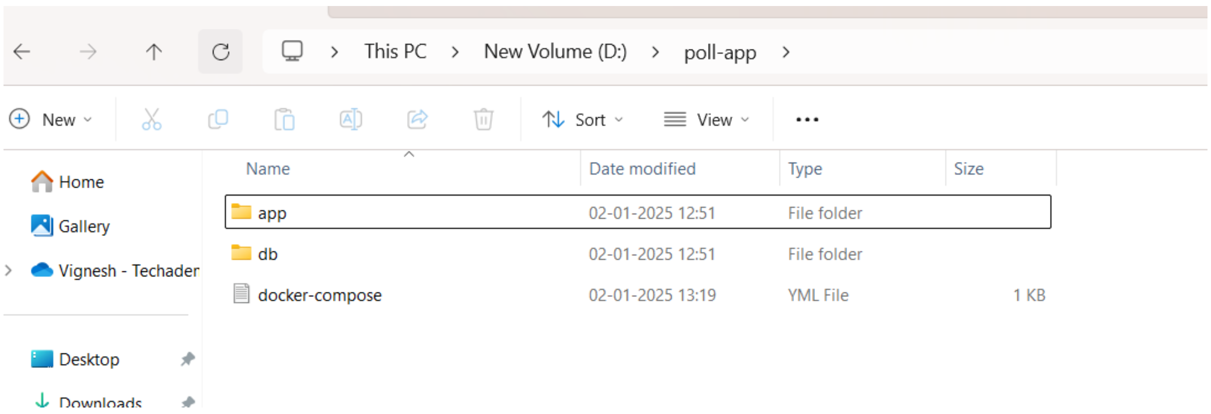
2. **docker-compose up --build**

3. **Check Running Containers**: Verify that the app and db containers are running:

   Using this command **docker ps**

Poll application specification

As you can see from the screenshot you need to create the respective directories



Name	Date modified	Type	Size
app	02-01-2025 12:51	File folder	
db	02-01-2025 12:51	File folder	
docker-compose	02-01-2025 13:19	YML File	1 KB

Verify the Build

Once the build process is complete, verify that the images were created:

docker images

You should see the Docker images for:

- The Node.js backend service.
- The MongoDB service (if it was pulled from Docker Hub)

Access the Application

- Open your browser and navigate to:
    - **Frontend**: http://localhost:3000
    - **Database : MongoDB**: localhost:27017

Test the Application

- Test the following:
    - **Login Page**: http://localhost:3000/login.html
    - **Poll Page**: http://localhost:3000/poll.html

After you have built the application

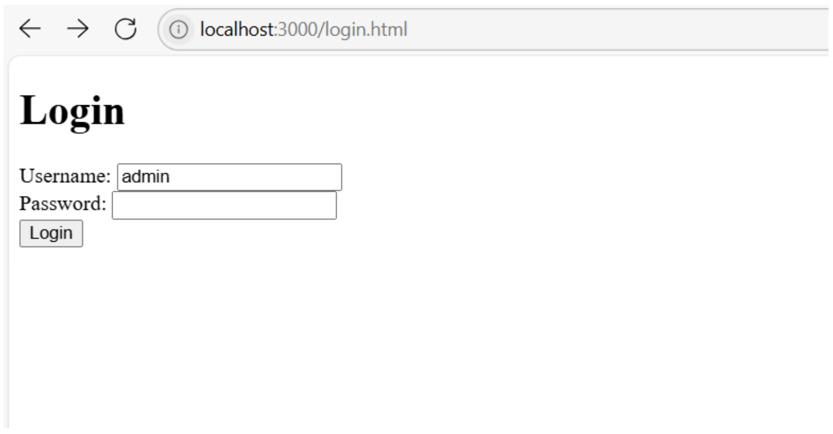
Task to complete

1. **Create app directory and create a poll application**
2. **Create the DB schema and connect with front and backend of the application**
3. **Create the compose file docker file**
4. **Launch the application using the browser**
5. **Expose the application with the correct ports**
6. **Run the testcases and push the source files**

Run and push the code

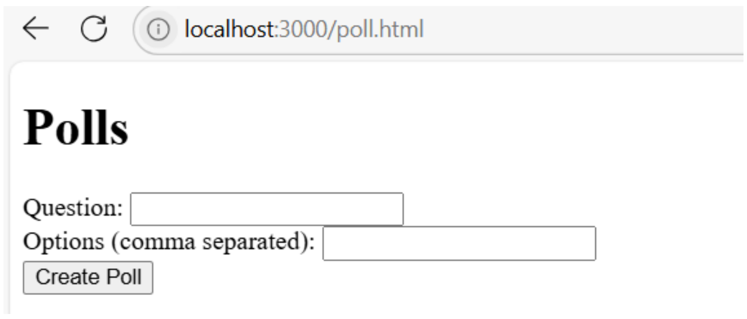
In the project folder you need to click on the run the **push.exe** to run the testcase and push the code

Sample screen shot of application for reference


localhost:3000/login.html

Login

Username: admin
Password:
Login


localhost:3000/poll.html

Polls

Question:

Options (comma separated):