

# Project Documentation: Demo App With Docker

Presented to :

ENG / Ahmed Mourad

Presented By :

ENG/Mustafa Amer

ENG /Reham Ahmed

ENG/ Ahmed Sabry

ENG / Ali Morshed

DEPI Final Project

# 1 TABLE OF CONTENTS

---

2	Introduction and Overview .....	2
2.1	Application Components: .....	2
2.2	Docker for Containerization .....	2
2.3	Continuous Integration and Deployment.....	2
2.4	Target Audience.....	3
3	The main features of the project now include:.....	4
3.1	User-friendly Frontend Interface .....	4
3.2	Secure and Scalable Backend using Node.js and MongoDB.....	4
3.3	Integration with Mongo Express for Database Management through Docker .....	5
3.4	Containerized Application Allowing Easy Deployment and Scaling.....	5
3.5	Automated Build, Testing, and Deployment using Jenkins.....	6
3.6	Configuration Management and Automated Deployment using Ansible.....	6
4	Functional Workflows.....	7
4.1	Workflow 1: User Registration.....	7
4.2	Workflow 2: User Login.....	8
4.3	Workflow 3: Profile Update .....	9
4.4	Workflow 4: Data Retrieval and Display (User Dashboard) .....	10
4.5	Workflow 5: Database Management with Mongo Express .....	11
5	Screenshots and Descriptions .....	12
5.1	User Interface - Landing Page .....	12
5.2	Workflow - User Profile Creation .....	18
6	Technical Details .....	20
6.1	Dockerfile .....	20
6.2	Jenkinsfile.....	21
6.3	Docker Compose YAML .....	23
7	Challenges and Solutions.....	24
8	Conclusion .....	24

## 2 INTRODUCTION AND OVERVIEW

---

This project showcases a user profile application designed to demonstrate the integration of modern web technologies within a containerized environment using Docker. The primary goal of the project is to provide developers with a practical example of how to combine frontend and backend technologies while maintaining scalability and security.

### 2.1 APPLICATION COMPONENTS:

**Frontend:** Built with standard web technologies including HTML, CSS, and JavaScript, the frontend focuses on providing a clean, user-friendly interface. It is fully responsive, allowing users to access the application on various devices, from desktops to smartphones.

**Backend:** Powered by Node.js and Express, the backend is designed to handle requests from the frontend, process data, and interact with the MongoDB database. The RESTful API architecture ensures a clean separation of concerns, making it easier to maintain and extend.

**Database:** MongoDB, a NoSQL database, is used for storing user data in a flexible, schema-less format. This choice allows the application to scale horizontally, handling more users as the need arises without major overhauls to the database structure.

### 2.2 DOCKER FOR CONTAINERIZATION

Docker is used to containerize all components of the application (frontend, backend, and database), allowing the entire project to run in isolated environments. This ensures consistency across different deployment platforms, making it easy to replicate the application across development, staging, and production environments.

With Docker, developers no longer need to worry about differences in software environments, versioning, or dependencies. Once the application is containerized, it can be run anywhere Docker is supported, ensuring a high level of portability.

### 2.3 CONTINUOUS INTEGRATION AND DEPLOYMENT

The project integrates Jenkins for Continuous Integration (CI) and Ansible for configuration management and Continuous Deployment (CD). Jenkins automates the building, testing, and deployment process, ensuring that every change to the codebase is automatically tested and deployed without human intervention. This guarantees a streamlined workflow, reducing the chances of errors during deployment and increasing overall productivity.

Ansible is responsible for automating the deployment process, ensuring that the environment is configured correctly before deployment and that all services are running seamlessly. By leveraging Ansible, the project can be easily deployed across multiple environments, ensuring consistency and scalability.

## **2.4 TARGET AUDIENCE**

This project is aimed at developers and engineers who are looking to:

- Understand the integration of frontend and backend technologies.
- Learn how to use Docker for containerization and environment management.
- Automate their build, test, and deployment pipelines with tools like Jenkins and Ansible.

Through this project, users will gain hands-on experience with modern development and deployment practices, making it a valuable learning resource for both beginners and seasoned developers interested in Docker, Node.js, and scalable web applications.

### 3 THE MAIN FEATURES OF THE PROJECT NOW INCLUDE:

---

1. User-friendly frontend interface.
2. Secure and scalable backend using Node.js and MongoDB.
3. Integration with Mongo Express for database management through Docker.
4. Containerized application allowing easy deployment and scaling.
5. Automated build, testing, and deployment using Jenkins.
6. Configuration management and automated deployment using Ansible.

#### 3.1 USER-FRIENDLY FRONTEND INTERFACE

The user interface (UI) of the application is designed with the end-user in mind. The frontend uses HTML, CSS, and JavaScript, making it accessible to a wide range of devices and screen sizes. A minimalistic approach ensures that users are not overwhelmed with information while interacting with the app. Key UI principles include:

- **Simplicity:** Focused on ease of use, with clear buttons and organized content.
- **Responsiveness:** Media queries ensure that the application works seamlessly on mobile devices, tablets, and desktop computers. Fluid layouts dynamically adjust to various screen sizes.
- **Interactivity:** JavaScript is used to implement real-time form validation, providing instant feedback on user input. This enhances user experience by preventing form submission errors and ensuring smooth navigation.

The frontend aims to offer a smooth user journey, making tasks like registration, login, and profile updates intuitive and hassle-free. The interface is also optimized for performance, ensuring fast load times and efficient interactions.

#### 3.2 SECURE AND SCALABLE BACKEND USING NODE.JS AND MONGODB

The backend of the application is powered by Node.js and Express, providing a robust framework for handling HTTP requests and delivering dynamic content. MongoDB is used as the primary database, chosen for its flexibility in handling large datasets and its scalability. Key aspects of the backend include:

- **Security:** Sensitive user data is encrypted using bcrypt, ensuring that passwords are securely stored and never exposed in plaintext. Environment variables are used to manage database credentials securely, keeping them hidden from the codebase.
- **Scalability:** Node.js's event-driven, non-blocking I/O model ensures that the server can handle multiple concurrent requests efficiently. MongoDB's NoSQL architecture allows horizontal scaling, meaning as the application grows, the database can expand by adding more servers.
- **RESTful API Design:** The backend is structured as a REST API, with endpoints handling common operations such as creating, reading, updating, and deleting (CRUD) user profiles.

This combination of Node.js and MongoDB creates a scalable, high-performance backend that supports the growing demands of users while maintaining security and reliability.

### 3.3 INTEGRATION WITH MONGO EXPRESS FOR DATABASE MANAGEMENT THROUGH DOCKER

Mongo Express is a web-based interface used to manage the MongoDB database visually, integrated into the project using Docker. This interface makes database administration simpler, especially for developers and users not familiar with MongoDB's command line. Features of this integration include:

- **Ease of Use:** Mongo Express allows admins to browse through collections, documents, and databases directly from a browser. Users can perform actions like adding, updating, or deleting records without needing to access MongoDB's command-line interface.
- **Real-Time Management:** Changes made through Mongo Express reflect instantly in the application. For example, if a user's profile is updated directly via Mongo Express, the app reflects the changes in real-time.
- **Secure Environment:** Running Mongo Express within a Docker container ensures that the database management tool remains isolated and secure. It can be configured to only be accessible to authorized users, reducing the risk of unauthorized access to the database.

Mongo Express, through Docker, simplifies database operations while ensuring security and accessibility for admins managing the app.

### 3.4 CONTAINERIZED APPLICATION ALLOWING EASY DEPLOYMENT AND SCALING

Docker plays a central role in this project by containerizing the application, including the frontend, backend, and MongoDB database. By using containers, the app becomes highly portable and easier to manage. Key benefits of Docker include:

- **Simplified Deployment:** Docker allows developers to package the application, including all dependencies, into a container. Once packaged, the container can be deployed across various environments without worrying about system compatibility.
- **Scalability:** If the app experiences high traffic, new instances of the containers can be spun up quickly to handle the increased load. This is especially important for applications that may need to scale horizontally to meet demand.
- **Portability:** Docker ensures that the application behaves the same, regardless of where it's run—whether on a local machine, a cloud service, or a production server. This eliminates the “works on my machine” issue that often arises when moving between environments.
- **Isolation:** Each component of the application (frontend, backend, database) runs in its own isolated container, which prevents any interference between components and enhances security.

Docker containerization ensures that the application can be deployed quickly, scaled efficiently, and maintained consistently across different platforms.

### 3.5 AUTOMATED BUILD, TESTING, AND DEPLOYMENT USING JENKINS

Jenkins is used in the project for continuous integration (CI) and continuous deployment (CD). It automates the process of building, testing, and deploying the application, streamlining the development pipeline. Key Jenkins features include:

- **Build Automation:** Jenkins automatically triggers builds whenever code is pushed to the repository. This ensures that any new changes are compiled and built correctly.
- **Automated Testing:** Once a build is completed, Jenkins runs tests to verify the stability of the code. This ensures that any bugs or issues are caught early in the development cycle, improving overall quality.
- **Deployment:** After a successful build and test stage, Jenkins pushes the Docker images to Docker Hub and triggers the deployment process. This can be configured to deploy automatically to production or require manual approval for greater control.
- **Pipeline Management:** Jenkins allows developers to set up custom pipelines, defining each stage from code commit to production deployment.

The integration of Jenkins ensures that the project maintains a high level of quality and reliability by automating key parts of the development process.

### 3.6 CONFIGURATION MANAGEMENT AND AUTOMATED DEPLOYMENT USING ANSIBLE

Ansible is employed in the project for automating the configuration and deployment of the application. Ansible's role is to ensure that all services are deployed consistently across different environments. Its key features include:

- **Configuration Management:** Ansible's playbooks automate the process of setting up servers, databases, and application environments. This ensures that the same configuration is applied every time, reducing human error.
- **Automated Deployment:** Once the application is built and tested, Ansible is used to deploy the containers to production servers. Ansible handles the orchestration of services, ensuring that all components (frontend, backend, and database) are deployed and running correctly.
- **Scalability:** If more instances of the application are required, Ansible can automate the scaling process, ensuring that new containers are spun up with the same configuration as existing ones.
- **Security:** By automating the deployment process, Ansible ensures that all environment variables, credentials, and sensitive data are securely handled.

Ansible provides a powerful tool for managing the deployment and scaling of the application, ensuring consistency, security, and efficiency in the process.

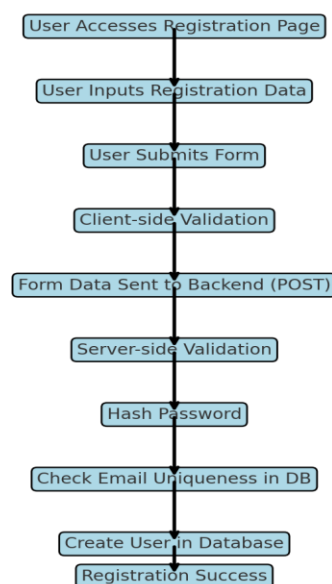
## 4 FUNCTIONAL WORKFLOWS

This section will detail the key functional workflows within the project. Each workflow will describe the steps involved in achieving a specific function and how the frontend, backend, and database interact to complete the process.

### 4.1 WORKFLOW 1: USER REGISTRATION

1. **User Input:** The user accesses the registration page from the frontend interface. The page contains a form where the user enters their personal details (name, email, password, etc.).
2. **Form Submission:** After filling in the required fields, the user clicks the “Register” button. The form data is validated client-side for basic errors (e.g., empty fields, invalid email format).
3. **Backend Processing:** The validated data is sent as a POST request to the Node.js backend via a secure API endpoint (/api/register). The backend receives the data and validates it further.
4. **Password Hashing:** The password is hashed using bcrypt to ensure it is stored securely in the database.
5. **Database Interaction:** The backend checks the MongoDB database to verify that the email is not already in use. If the email is unique, a new user record is created in the MongoDB collection.
6. **Success Response:** The backend sends a success response to the frontend, indicating that the registration was successful. The user is redirected to the login page or receives a confirmation

Workflow 1: User Registration



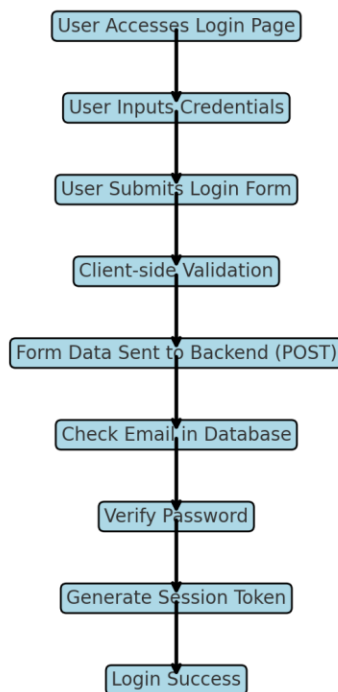
message.



## 4.2 WORKFLOW 2: USER LOGIN

1. **User Input:** The user accesses the login page and enters their credentials (email and password).
2. **Form Submission:** After inputting the credentials, the user clicks the “Login” button. The data is sent to the backend via a POST request to the `/api/login` endpoint.
3. **Backend Validation:** The backend receives the login request and searches the MongoDB database for a user record that matches the email.
4. **Password Verification:** The hashed password stored in the database is compared with the password provided by the user. If they match, the user is authenticated.
5. **Session or Token Creation:** After successful authentication, the backend generates a session token (or JWT) and sends it to the frontend.
6. **Login Success:** The user is logged in, and the session token is stored securely on the frontend. The user is redirected to their dashboard or profile page.

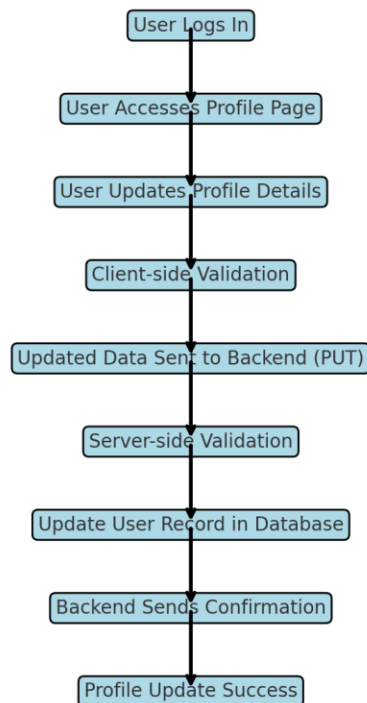
Workflow 2: User Login



### 4.3 WORKFLOW 3: PROFILE UPDATE

1. User Access: The logged-in user navigates to their profile page where they can view and edit their personal details.
2. Profile Changes: The user updates their details (e.g., name, address, etc.) and clicks “Save Changes.”
3. Data Validation: The frontend validates the updated data for completeness and correctness (e.g., ensuring all fields are filled in and have valid formats).
4. Backend Processing: The updated data is sent to the backend via a PUT request to the /api/profile endpoint.
5. Database Update: The backend processes the request by updating the user record in the MongoDB database.
6. Confirmation: The backend sends a success response to the frontend, and the updated profile information is displayed.

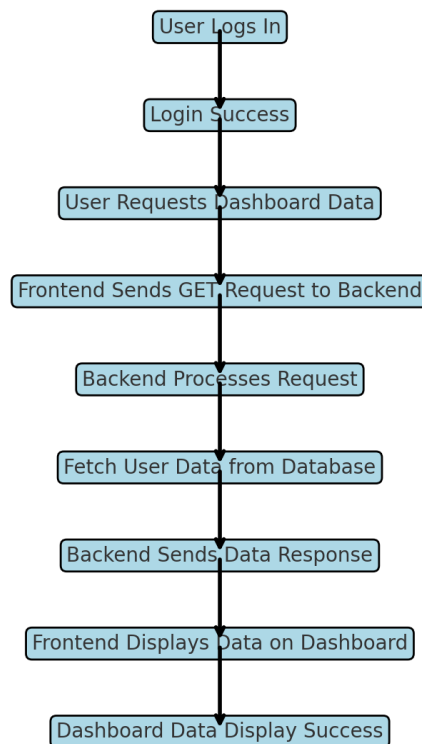
Workflow 3: Profile Update



#### 4.4 WORKFLOW 4: DATA RETRIEVAL AND DISPLAY (USER DASHBOARD)

1. User Login: The user logs into their account and is redirected to their dashboard.
2. API Request: Upon loading the dashboard, the frontend sends a GET request to the backend to fetch the user's data (e.g., profile info, recent activities).
3. Backend Processing: The backend queries the MongoDB database for the relevant user data.
4. Data Retrieval: The user's data is retrieved and formatted into a JSON response.
5. Display on Frontend: The frontend receives the data and dynamically updates the UI to display the user's profile information and other relevant content.

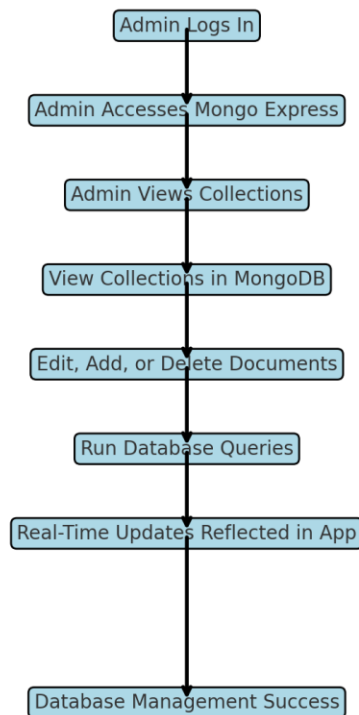
Workflow 4: Data Retrieval and Display (User Dashboard)



#### 4.5 WORKFLOW 5: DATABASE MANAGEMENT WITH MONGO EXPRESS

1. Admin Access: The administrator logs into Mongo Express to manage the MongoDB database.
2. Viewing Collections: The admin can view the list of collections in the MongoDB instance. Each collection corresponds to a specific part of the app's data (e.g., users, products).
3. Document Management: The admin can add, modify, or delete individual documents in any collection. For example, they can manually update a user's profile or delete inactive users.
4. Database Queries: Mongo Express allows the admin to run custom queries on the database. For example, the admin can filter user records to find those who registered within the last month.
5. Real-Time Updates: The changes made via Mongo Express are reflected immediately in the live app, allowing for real-time data management.

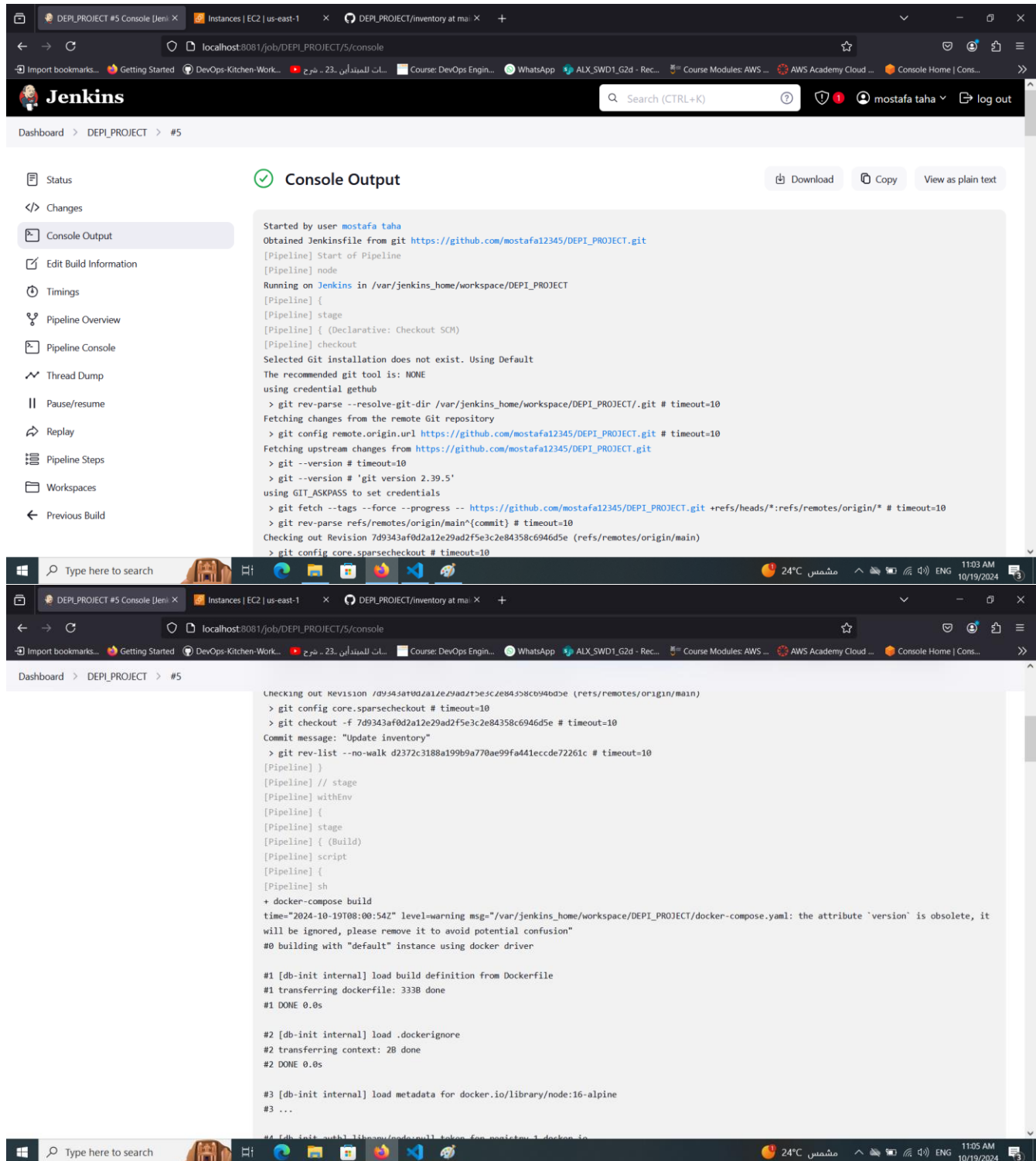
Workflow 5: Database Management with Mongo Express



## 5 SCREENSHOTS AND DESCRIPTIONS

### 5.1 USER INTERFACE - LANDING PAGE

This screenshot shows the landing page of the user profile app. It features a simple layout with user-friendly navigation.



## Project Documentation: Demo App with Docker

The screenshot shows a Docker build process in a console window. The build is for a Docker image named 'dep1\_project-db-init'. The build steps are as follows:

```
#4 [db-init auth] library/node:pull token for registry-1.docker.io
#4 DONE 0.0s

#3 [db-init internal] load metadata for docker.io/library/node:16-alpine
#3 DONE 1.9s

#5 [db-init 1/4] FROM docker.io/library/node:16-alpine@sha256:a1f9d027912b58a7c75be7716c97c-fbc6d3099f3a97ed84aa498be9dee20e787
#5 DONE 0.0s

#6 [db-init internal] load build context
#6 transferring context: 35B done
#6 DONE 0.0s

#7 [db-init 2/4] WORKDIR /usr/src/app
#7 CACHED

#8 [db-init 3/4] COPY init-mongo.js .
#8 CACHED

#9 [db-init 4/4] RUN npm install mongodb
#9 CACHED

#10 [db-init] exporting to image
#10 exporting layers done
#10 writing image sha256:a6d28282fc01e0e36fabd556da8dfcc1972b8a77403e05af612ebaca9f02c205 done
#10 naming to docker.io/library/dep1_project-db-init done
#10 DONE 0.0s

#11 [db-init] resolving provenance for metadata file
#11 DONE 0.0s

#12 [my-app internal] load build definition from Dockerfile
#12 transferring dockerfile: 403B done
#12 DONE 0.0s

#13 [my-app internal] load .dockerignore
#13 transferring context: 2B done
#13 DONE 0.0s

#14 [my-app internal] load metadata for docker.io/library/node:13-alpine
#14 DONE 0.2s

#15 [my-app 1/5] FROM docker.io/library/node:13-alpine@sha256:527c70f74817f6f6b5853588c28de33459178ab72421f1fb7b63a281ab670258
#15 DONE 0.0s

#16 [my-app internal] load build context
#16 transferring context: 348B done
#16 DONE 0.0s

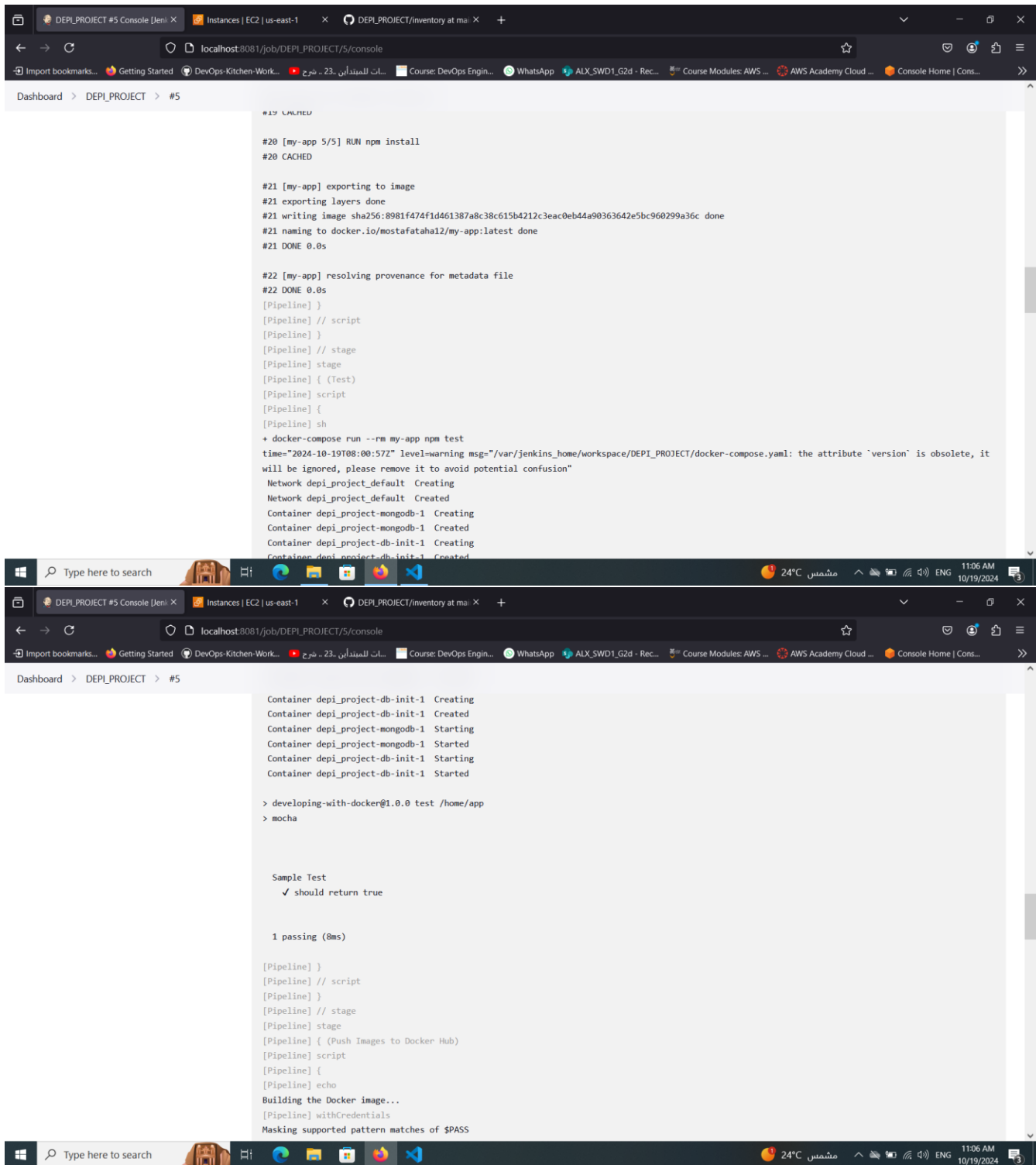
#17 [my-app 2/5] RUN mkdir -p /home/app
#17 CACHED

#18 [my-app 3/5] COPY ./app /home/app
#18 CACHED

#19 [my-app 4/5] WORKDIR /home/app
#19 CACHED
```

The console window shows the build progress and the resulting image. The build is successful and the image is named 'dep1\_project-db-init'.

## Project Documentation: Demo App with Docker



```
#19 CALLED

#20 [my-app 5/5] RUN npm install
#20 CACHED

#21 [my-app] exporting to image
#21 exporting layers done
#21 writing image sha256:8981f474f1d461387a8c38c615b4212c3eac0eb44a90363642e5bc960299a36c done
#21 naming to docker.io/mostafataha12/my-app:latest done
#21 DONE 0.0s

#22 [my-app] resolving provenance for metadata file
#22 DONE 0.0s

[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] script
[Pipeline] {
[Pipeline] sh
+ docker-compose run --rm my-app npm test
time="2024-10-19T08:00:57Z" level=warning msg="/var/jenkins_home/workspace/DEPI_PROJECT/docker-compose.yaml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
Network depi_project_default Creating
Network depi_project_default Created
Container depi_project-mongodb-1 Creating
Container depi_project-mongodb-1 Created
Container depi_project-db-init-1 Creating
Container depi_project-db-init-1 Created

Container depi_project-db-init-1 Creating
Container depi_project-db-init-1 Created
Container depi_project-mongodb-1 Starting
Container depi_project-mongodb-1 Started
Container depi_project-db-init-1 Starting
Container depi_project-db-init-1 Started

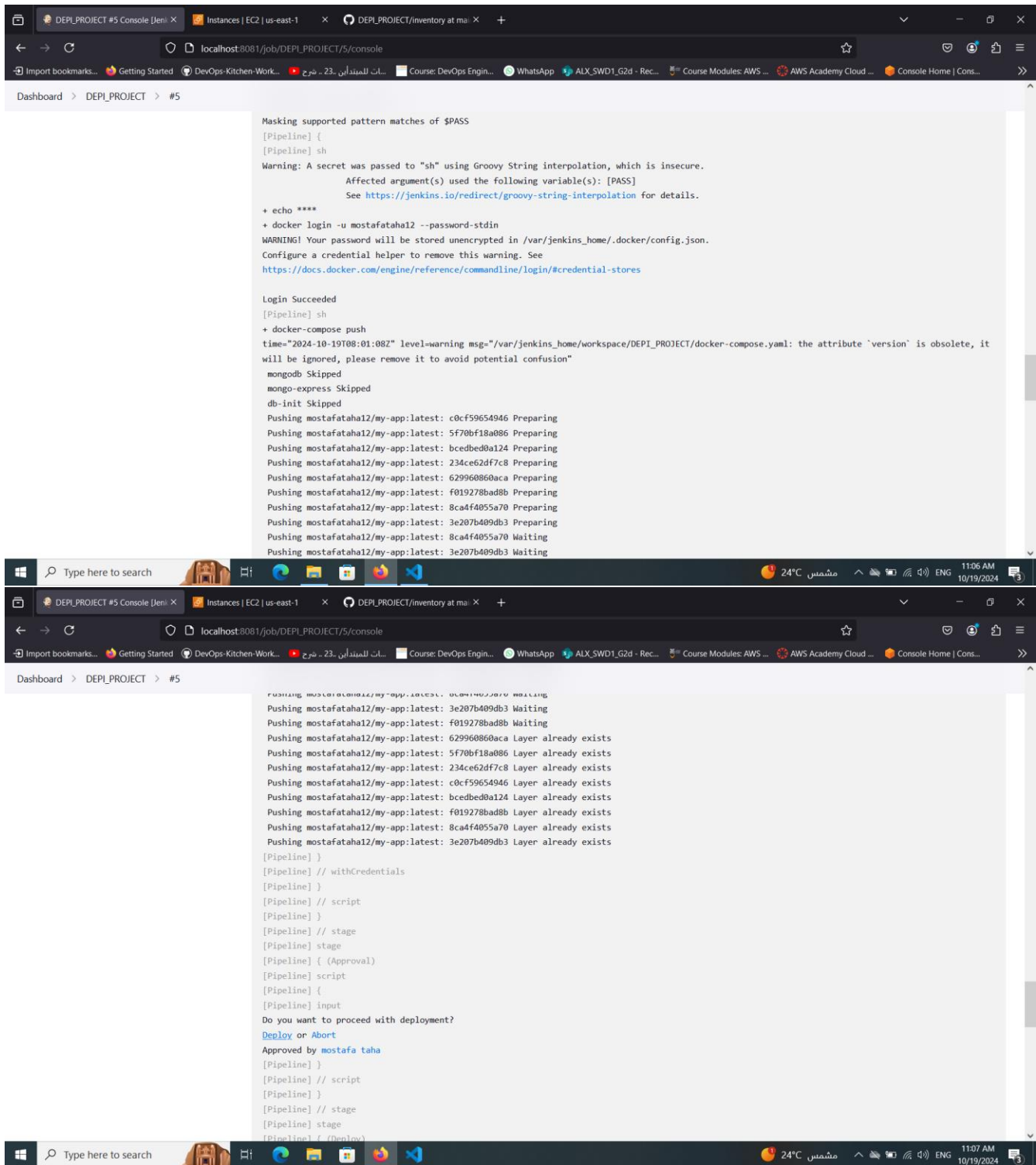
> developing-with-docker@1.0.0 test /home/app
> mocha

Sample Test
✓ should return true

1 passing (8ms)

[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Push Images to Docker Hub)
[Pipeline] script
[Pipeline] {
[Pipeline] echo
Building the Docker image...
[Pipeline] withCredentials
Masking supported pattern matches of $PASS
```

## Project Documentation: Demo App with Docker



```
Masking supported pattern matches of $PASS
[Pipeline] {
[Pipeline] sh
Warning: A secret was passed to "sh" using Groovy String interpolation, which is insecure.
Affected argument(s) used the following variable(s): [PASS]
See https://jenkins.io/redirect/groovy-string-interpolation for details.

+ echo ****
+ docker login -u mostafataha12 --password-stdin
WARNING! Your password will be stored unencrypted in /var/jenkins_home/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credential-stores

Login Succeeded
[Pipeline] sh
+ docker-compose push
time="2024-10-19T08:01:08Z" level=warning msg="/var/jenkins_home/workspace/DEPL_PROJECT/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
mongodb Skipped
mongo-express Skipped
db-init Skipped
Pushing mostafataha12/my-app:latest: c0cf59654946 Preparing
Pushing mostafataha12/my-app:latest: 5f70bf18a086 Preparing
Pushing mostafataha12/my-app:latest: bcedbed8a124 Preparing
Pushing mostafataha12/my-app:latest: 234ce62dfc8 Preparing
Pushing mostafataha12/my-app:latest: 629960860aca Preparing
Pushing mostafataha12/my-app:latest: f019278bad8b Preparing
Pushing mostafataha12/my-app:latest: 8ca4f4055a70 Preparing
Pushing mostafataha12/my-app:latest: 3e207b409db3 Preparing
Pushing mostafataha12/my-app:latest: 8ca4f4055a70 Waiting
Pushing mostafataha12/my-app:latest: 3e207b409db3 Waiting

Pushing mostafataha12/my-app:latest: 3e207b409db3 Waiting
Pushing mostafataha12/my-app:latest: 3e207b409db3 Waiting
Pushing mostafataha12/my-app:latest: f019278bad8b Waiting
Pushing mostafataha12/my-app:latest: 629960860aca Layer already exists
Pushing mostafataha12/my-app:latest: 5f70bf18a086 Layer already exists
Pushing mostafataha12/my-app:latest: 234ce62dfc8 Layer already exists
Pushing mostafataha12/my-app:latest: c0cf59654946 Layer already exists
Pushing mostafataha12/my-app:latest: bcedbed8a124 Layer already exists
Pushing mostafataha12/my-app:latest: f019278bad8b Layer already exists
Pushing mostafataha12/my-app:latest: 8ca4f4055a70 Layer already exists
Pushing mostafataha12/my-app:latest: 3e207b409db3 Layer already exists
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Approval)
[Pipeline] script
[Pipeline] {
[Pipeline] input
Do you want to proceed with deployment?
Deploy or Abort
Approved by mostafa taha
[Pipeline] }
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] }
```



## Project Documentation: Demo App with Docker

The screenshot displays a terminal window with a dark theme, showing the execution of an Ansible playbook. The browser tabs at the top include 'DEPL\_PROJECT #5 Console', 'Instances | EC2 | us-east-1', and 'DEPL\_PROJECT/inventory at mai'. The terminal output is as follows:

```
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] script
[Pipeline] {
[Pipeline] withCredentials
Masking supported pattern matches of $SSH_KEY_PATH
[Pipeline] {
[Pipeline] sh
+ ansible-playbook -i inventory --private-key **** playbook.yml

PLAY [Install Docker, Docker-Compose, and Setup EC2 for Deployment] *****

TASK [Gathering Facts] *****
ok: [44.208.28.25]

TASK [Install Docker] *****
ok: [44.208.28.25]

TASK [Start Docker service] *****
ok: [44.208.28.25]

TASK [Add ec2-user to the Docker group] *****
ok: [44.208.28.25]

TASK [Reconnect to the server session (apply Docker group changes)] *****

TASK [Install Docker Compose] *****
ok: [44.208.28.25]

PLAY [Copy Application and Docker Compose File to EC2] *****

TASK [Reconnect to the server session (apply Docker group changes)] *****

TASK [Install Docker Compose] *****
ok: [44.208.28.25]

PLAY [Copy Application and Docker Compose File to EC2] *****

TASK [Gathering Facts] *****
ok: [44.208.28.25]

TASK [Copy utility-app to EC2 instance] *****
ok: [44.208.28.25]

TASK [Copy the docker-compose.yaml file to EC2] *****
ok: [44.208.28.25]

PLAY [Deploy Application using Docker Compose] *****

TASK [Gathering Facts] *****
ok: [44.208.28.25]

TASK [Run docker-compose up] *****
changed: [44.208.28.25]

PLAY RECAP *****
44.208.28.25 : ok=10 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
```

The terminal window is part of a desktop environment. The taskbar at the bottom shows the Windows search bar, several application icons (including a web browser, file explorer, and various development tools), and system status icons on the right indicating a temperature of 24°C, network status, and the time 11:07 AM on 10/19/2024.

## Project Documentation: Demo App with Docker

The screenshot shows a Jenkins console window with the following output:

```
PLAY [Deploy Application using Docker Compose] *****

TASK [Gathering Facts] *****
ok: [44.208.28.25]

TASK [Run docker-compose up] *****
changed: [44.208.28.25]

PLAY RECAP *****
44.208.28.25      : ok=10   changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

REST API Jenkins 2.462.3

The screenshot shows an AWS EC2 Instance Connect terminal window with the following output:

```
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023

Last login: Sat Oct 19 08:02:45 2024 from 197.63.198.98
[ec2-user@ip-10-0-20-122 ~]$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED        STATUS        PORTS                               NAMES
a1829d1f38b4        mostafatahal2/my-app:latest  "/sbin/tini -- /dock..."  19 hours ago  Up 5 minutes  0.0.0.0:3000->3000/tcp, :::3000->3000/tcp  ec2-user-my-app-1
b4bc5fb4b476        mongo-express        "/sbin/tini -- /dock..."  19 hours ago  Up 6 minutes  0.0.0.0:8080->8081/tcp, :::8080->8081/tcp  ec2-user-mongo-express-1
67d62d3a2202        mongo                "/sbin/tini -- /dock..."  19 hours ago  Up 5 minutes  0.0.0.0:27017->27017/tcp, :::27017->27017/tcp  ec2-user-mongodb-1
```

i-02b0fd9f0025ac677 (DEPL\_PROJECT)  
Public IPs: 44.208.28.25 Private IPs: 10.0.20.122

## 5.2 WORKFLOW - USER PROFILE CREATION

This shows the profile creation workflow where users can input and submit their details.

The screenshot displays a web application interface for user profile creation. The browser window shows the URL `44.208.28.25:3000/`. The page title is "User profile". It features a profile picture of Jerry the mouse from Tom and Jerry. Below the picture, the user details are displayed:

- Name: **Tom Cat**
- Email: **tom.cat@example.com**
- Interests: **Jerry**

An "Edit Profile" button is located below the interests field. The browser's taskbar shows the Windows logo, a search bar, and several open applications including VS Code, Docker Desktop, and various web browsers.

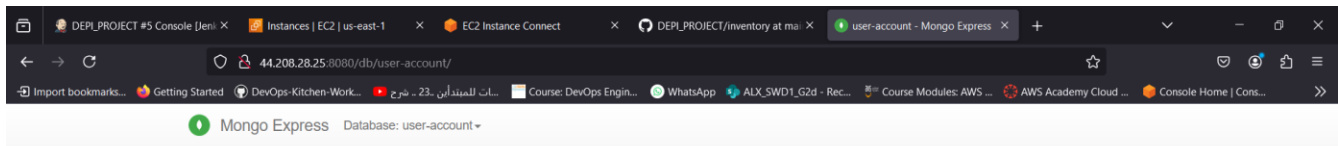
Below the user profile section, the Mongo Express interface is shown. The browser window displays the URL `44.208.28.25:8080`. The page title is "Mongo Express". The interface includes a "Databases" section with a "Create Database" button and a list of existing databases:

Database Name	View	Del
admin	<a href="#">View</a>	<a href="#">Del</a>
config	<a href="#">View</a>	<a href="#">Del</a>
local	<a href="#">View</a>	<a href="#">Del</a>
user-account	<a href="#">View</a>	<a href="#">Del</a>

Below the databases section, the "Server Status" section displays the following information:

Field	Value	Field	Value
Hostname	67d62d3a2202	MongoDB Version	8.0.1
Uptime	426 seconds	Node Version	18.20.3
Server Time	Sat, 19 Oct 2024 08:09:53 GMT	V8 Version	10.2.154.26-node.37

The browser's taskbar shows the Windows logo, a search bar, and several open applications including VS Code, Docker Desktop, and various web browsers.



## Viewing Database: user-account

**Collections**

Collection Name

+ Create collection

View

Export

[JSON]

Import

users

Del

**Database Stats**

Collections (incl. system.namespaces)	1
Data Size	72.0 Bytes
Storage Size	36.9 KB
Avg Obj Size #	36.0 Bytes
Objects #	2
Indexes #	1
Index Size	36.9 KB



## 6 TECHNICAL DETAILS

---

The technical stack used in this project includes the following:

- Frontend: HTML, JavaScript, CSS
- Backend: Node.js with Express.js
- Database: MongoDB
- Tools: Docker for containerization, Mongo Express for database management

This project is deployed and managed using Docker containers, providing a scalable and reliable solution.

### 6.1 DOCKERFILE

```
FROM node:13-alpine
```

```
ENV MONGO_DB_USERNAME=admin \  
    MONGO_DB_PWD=password
```

```
RUN mkdir -p /home/app
```

```
COPY ./app /home/app
```

```
# set default dir so that next commands executes in /home/app dir  
WORKDIR /home/app
```

```
# will execute npm install in /home/app because of WORKDIR  
RUN npm install
```

```
# no need for /home/app/server.js because of WORKDIR  
CMD ["node", "server.js"]
```

## 6.2 JENKINSFILE

This Jenkinsfile defines a multi-stage pipeline used for automating the build, test, and deployment processes. It integrates Docker for containerization and Ansible for configuration management. The stages include:

- **\*\*Build Stage\*\*:** The application is built using Docker Compose.
- **\*\*Test Stage\*\*:** Tests are run within the Docker container to ensure application stability.
- **\*\*Push Stage\*\*:** Docker images are pushed to Docker Hub.
- **\*\*Approval Stage\*\*:** Manual approval is required to proceed with deployment.
- **\*\*Deploy Stage\*\*:** Ansible is used to deploy the application on production servers, ensuring consistency and scalability.

```

pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        script {
          sh 'docker-compose build'
        }
      }
    }
    stage('Test') {
      steps {
        script {
          sh 'docker-compose run --rm my-app npm test'
        }
      }
    }
    stage('Push Images to Docker Hub') {
      steps {
        script {
          echo "Building the Docker image..."
          withCredentials([usernamePassword(credentialsId: 'docker-hub-repo', passwordVariable:
'PASS', usernameVariable: 'USER'))] {
            sh "echo $PASS | docker login -u $USER --password-stdin"
            sh "docker-compose push"
          }
        }
      }
    }
    stage('Approval') {
      steps {
        script {
          // Manual approval step
          input message: 'Do you want to proceed with deployment?', ok: 'Deploy'
        }
      }
    }
  }
}

```

```
}
stage('Deploy') {
    steps {
        script {
            withCredentials([sshUserPrivateKey(credentialsId: 'ec2-ssh-key', keyFileVariable:
'SSH_KEY_PATH', usernameVariable: 'SSH_USER')]) {
                sh '''
                    ansible-playbook -i inventory \
                    --private-key $SSH_KEY_PATH \
                    playbook.yml
                '''
            }
        }
    }
}
```

## 6.3 DOCKER COMPOSE YAML

```
version: '3'
services:
  my-app:
    build: .
    image: mostafataha12/my-app:latest
    ports:
      - 3000:3000
    depends_on:
      - "db-init"
  mongodb:
    image: mongo
    ports:
      - 27017:27017
    environment:
      - MONGO_INITDB_ROOT_USERNAME=admin
      - MONGO_INITDB_ROOT_PASSWORD=password
    volumes:
      - mongo-data:/data/db
  mongo-express:
    image: mongo-express
    restart: always
    ports:
      - 8080:8081
    environment:
      - ME_CONFIG_MONGODB_ADMINUSERNAME=admin
      - ME_CONFIG_MONGODB_ADMINPASSWORD=password
      - ME_CONFIG_MONGODB_SERVER=mongodb
    depends_on:
      - "mongodb"

  db-init:
    build:
      context: ./utility-app
      dockerfile: Dockerfile
    depends_on:
      - "mongodb"

volumes:
  mongo-data:
    driver: local
```



## 7 Challenges and Solutions

---

The primary challenges faced during this project were related to integrating Docker containers with MongoDB, ensuring seamless communication between the frontend and backend, and automating the deployment pipeline. These issues were resolved by:

- **1. \*\*Proper Docker Network Configuration\*\*:** Ensuring that the services (MongoDB, Express backend) could communicate securely through Docker's internal network using environment variables for flexibility.
- **2. \*\*Ansible for Configuration Management\*\*:** Ansible playbooks were used to automate tasks such as deploying and configuring services across environments, ensuring consistent deployment.
- **3. \*\*Jenkins Pipeline\*\*:** A Jenkins pipeline was created to automate the build, testing, and deployment processes. With clear stage-based execution, the pipeline ensures that code is properly built, tested, and deployed without manual intervention, reducing human error.

The primary challenges faced during this project were related to integrating the Docker containers with MongoDB and ensuring smooth communication between the frontend and backend. These issues were resolved by properly configuring the Docker network and using environment variables for container orchestration.

## 8 CONCLUSION

---

This project demonstrates how containerization, automation, and modular design can be combined to create a scalable, secure, and efficient web application. The use of Docker for consistent deployment, Jenkins for automated pipelines, and Ansible for configuration management ensures that the project can be easily maintained and expanded.

Potential improvements include:

- 1. Enhancing security measures by integrating more advanced user authentication mechanisms, such as OAuth.
- 2. Extending the frontend with additional features like multi-language support or real-time notifications.
- 3. Optimizing database queries for faster data retrieval as the application scales.

Future updates could include integrating more services, such as Redis for caching or Kubernetes for orchestrating container deployments.

This project showcases how containerization with Docker can simplify the deployment and management of web applications. The use of Node.js, MongoDB, and Docker provides a scalable solution that can be easily extended or modified in the future. Potential improvements include adding more frontend features and enhancing security measures.