

SCD Project 25

Docker Deployment Report

Submitted By:

Zain Ali Khan

22i-2624

BS Software Engineering

Batch 22

Institution:

FAST NUCES Islamabad

December 7, 2025

Contents

1 Part 3: Building Features into a Provided Project	4
1.1 Step 1: Clone the Repository	4
1.1.1 Commands Executed:	4
1.2 Step 2: Examine Project Structure	4
1.3 Step 3: Run the Application Locally	5
1.4 Step 4: Create a Feature Branch	5
1.4.1 Commands Executed:	6
1.5 Git Versioning Strategy	6
2 Feature Implementations	7
2.1 Feature 1: Search Functionality	7
2.2 Feature 2: Sorting Capability	7
2.3 Feature 3: Export Vault Data to Text File	8
2.4 Feature 4: Automatic Backup System	10
2.5 Feature 5: Display Data Statistics	10
2.6 Updated Menu Structure	11
2.7 Git Commit for Features 1-5	12
2.8 Step 5: Merge Feature Branch into Main	12
3 Part 4: Containerize the Application	13
3.1 Step 2: Create Dockerfile	13
3.2 Step 3: Commit Dockerfile	14
3.3 Step 4: Build Docker Image	14
3.4 Step 5: Create Docker Network	14
3.5 Step 6: Run MongoDB Container	14
3.6 Step 7: Run NodeVault Container	15
3.7 Step 10: Publish to Docker Hub	15
3.8 Step 11: Commit and Merge	16
3.9 Summary - Part 4	17
4 Part 5: Deploy Containers Manually	17
4.1 Overview	17
4.2 Step 1: Clean Up Existing Containers	17
4.3 Step 2: Create Private Docker Network	18
4.4 Step 3: Create Docker Volume for MongoDB Persistence	19
4.5 Step 4: Run MongoDB Container with Volume	19
4.6 Step 5: Run NodeVault Backend Container	21
4.7 Step 6: Verify Network Isolation (Proof of Private Network)	21
4.8 Step 7.1: Add Data to the Application	22
4.9 Step 7.2: Destroy and Recreate Containers	22
4.10 Step 7.3: Relaunch Containers	23
4.11 Complete List of Docker Commands Used	23
4.12 Difficulties in Manual Container Setup	24
4.13 Time and Effort Analysis	24

5 Part 6: Simplifying with Docker Compose	25
5.1 Overview	25
5.2 Step 1: Create docker-compose.yml	25
5.3 Step 2: Update .env File	26
5.4 Step 3: Stop Existing Containers	27
5.5 Step 4: Start Services with Docker Compose	27
5.6 Step 5: Verify Services are Running	27
5.7 Step 7: Test the Application	28
5.8 Step 8: Verify Network and Volumes	28
5.9 Step 9: Stop Services	28
5.10 Docker Compose vs Manual Deployment Comparison	28
5.11 Benefits of Docker Compose	28
5.12 Summary - Part 6	30
6 Part 7: Update Project Repo to include Docker Compose	30
6.1 Step 1: Update docker-compose.yml to Build from Dockerfile	30
6.2 Step 2: Clean Slate - Remove All Docker Images	31
6.3 Step 3: Build and Run with Docker Compose	31
6.4 Step 4: Verify Application is Working	32
6.5 Step 5: Create README.md	32
6.6 Step 6: Commit and Push to GitHub	33
6.7 Step 7: Verify on GitHub	34
6.8 Files Committed	35
6.9 Issues Encountered and Solutions	35
6.10 Summary - Part 7	35
7 Final Project Summary	35
7.1 All Parts Completed	35
7.2 Technologies Used	35
7.3 Key Learnings	36

1 Part 3: Building Features into a Provided Project

1.1 Step 1: Clone the Repository

1.1.1 Commands Executed:

```
1 # Create working directory
2 mkdir -p ~/scd-project
3 cd ~/scd-project
4
5 # Clone the repository
6 git clone https://github.com/LaibaImran1500/SCDProject25.git
7
8 # Navigate to the project
9 cd SCDProject25
10
11 # View project structure
12 ls -la
```

SCREENSHOT 1: Repository Cloned



Figure 1: Repository Cloned

1.2 Step 2: Examine Project Structure

```
1 # View project structure
2 ls -la
3
4 # View all files recursively
5 find . -type f -name "*.js" -o -name "*.json" | head -20
```

Project Structure:

```
1 SCDProject25/
2   +- main.js                      # Main application entry point
3   +- data/
4     |   +- vault.json              # In-memory database (JSON file)
5     +- db/
6       |   +- index.js               # Database operations (CRUD)
7       |   +- file.js                # File read/write operations
8       |   +- record.js              # Record validation and ID generation
9     +- events/
10       +- index.js                # Event emitter
11       +- logger.js               # Event logging
```

Application Overview:

- This is a NodeVault application - a CLI-based CRUD application
- Uses an in-memory JSON file database (`data/vault.json`)
- Has event-driven logging for record operations
- Current menu options: Add, List, Update, Delete, Exit

1.3 Step 3: Run the Application Locally

```
1 # Run the application
2 node main.js
```

Current Menu:

```
1 ===== NodeVault =====
2 1. Add Record
3 2. List Records
4 3. Update Record
5 4. Delete Record
6 5. Exit
7 ======
```

SCREENSHOT 2: Application Running Locally

```
27  ls -la
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
npm ERR! A complete log of this run can be found in:
npm ERR!     /home/zainAliKhan/.npm/_logs/2025-12-06T21_32_57_264Z-debug-0.log
• ZainAliKhan@Ubuntu:~/Desktop/SCD Projects cd SCDProject25
• ZainAliKhan@Ubuntu:~/Desktop/SCD Project/SCDProject25$ node main.js

===== NodeVault =====
1. Add Record
2. List Records
3. Update Record
4. Delete Record
5. Exit
=====

Choose option: 1
Enter name: Zain Ali Khan
Enter value: zainalik157@gmail.com
[EVENT] Record added: ID 1765056827612, Name: Zain Ali Khan
✓ Record added successfully!

===== NodeVault =====
1. Add Record
2. List Records
3. Update Record
4. Delete Record
5. Exit
=====

Choose option: 2
ID: 1765056827612 | Name: Zain Ali Khan | Value: zainalik157@gmail.com

===== NodeVault =====
1. Add Record
2. List Records
3. Update Record
4. Delete Record
5. Exit
=====
```

Figure 2: Application Running Locally

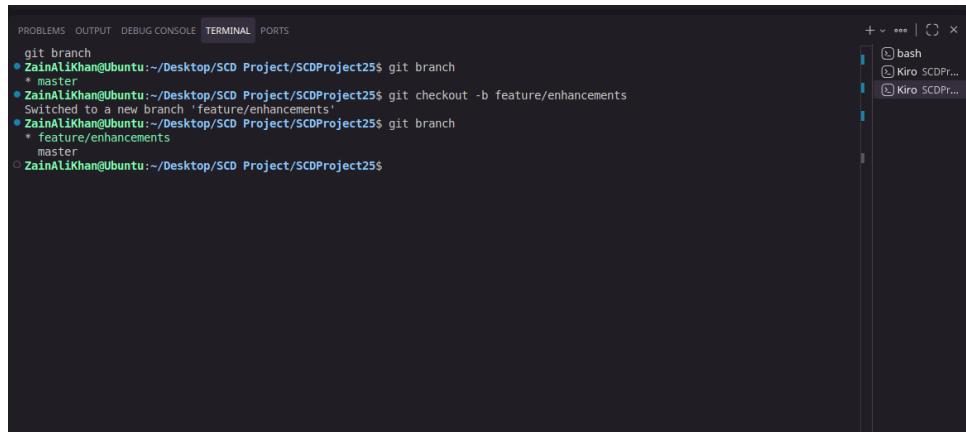
1.4 Step 4: Create a Feature Branch

Before making any modifications, we create a new branch from the main branch. All changes will be made in this feature branch.

1.4.1 Commands Executed:

```
1 # Navigate to project directory
2 cd ~/scd-project/SCDProject25
3
4 # Check current branch
5 git branch
6
7 # Create and switch to feature branch
8 git checkout -b feature/enhancements
9
10 # Verify branch switch
11 git branch
```

SCREENSHOT 3: Feature Branch Created



The screenshot shows a terminal window with the following command history:

```
git branch
* master
ZainAliKhan@Ubuntu:~/Desktop/SCD Project/SCDProject25$ git branch
* master
ZainAliKhan@Ubuntu:~/Desktop/SCD Project/SCDProject25$ git checkout -b feature/enhancements
Switched to a new branch 'feature/enhancements'
ZainAliKhan@Ubuntu:~/Desktop/SCD Project/SCDProject25$ git branch
* feature/enhancements
  master
ZainAliKhan@Ubuntu:~/Desktop/SCD Project/SCDProject25$
```

The terminal is part of a dark-themed IDE interface, with a sidebar on the right showing open files: 'bash', 'Kiro SCDPr...', and 'Kiro SCDPr...'. The terminal tab is labeled 'TERMINAL'.

Figure 3: Feature Branch Created

1.5 Git Versioning Strategy

For every major change, we will create a version tag:

Feature	Version Tag	Command
Search Functionality	v1.0	git tag -a v1.0 -m "Added search functionality"
Sorting Capability	v1.1	git tag -a v1.1 -m "Added sorting capability"
Export to Text File	v1.2	git tag -a v1.2 -m "Added export functionality"
Automatic Backup	v1.3	git tag -a v1.3 -m "Added automatic backup"
Data Statistics	v1.4	git tag -a v1.4 -m "Added vault statistics"
MongoDB Setup	v1.5	git tag -a v1.5 -m "MongoDB integration"
Env File Setup	v2.0	git tag -a v2.0 -m "Environment variables setup"

Table 1: Git Versioning Strategy

2 Feature Implementations

2.1 Feature 1: Search Functionality

Description: Allows users to search for existing records by name or ID (case-insensitive).

Implementation:

```
1 // Function to search records (case-insensitive)
2 function searchRecords(keyword) {
3     const records = db.listRecords();
4     const searchTerm = keyword.toLowerCase();
5
6     return records.filter(r =>
7         r.name.toLowerCase().includes(searchTerm) ||
8         r.id.toString().includes(searchTerm) ||
9         (r.value && r.value.toLowerCase().includes(searchTerm))
10    );
11 }
```

Menu Option Added: Option 5 - “Search Records”

SCREENSHOT 4: Search Functionality

```
Dec 7 02:53
File Edit Selection View Go Run Terminal Help
Docker Deployment Report.mnd SCDProject25_Report.mnd main.js env Index.js Preview Docker_Deployment_Report.mnd New...
EXPLORER-SCD PROJECT Docker Deployment Report.mnd SCDProject25_Report.mnd main.js env Index.js Preview Docker_Deployment_Report.mnd New...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
8. View Vault Statistics
9. Exit
=====
Choose option: 1
Enter name: Talha Saleem
Enter value: Talha.saleem@gmail.com
Enter ID: 6934a56a88451e32dd75c4e0
[EVENT] Record added: ID 6934a56a88451e32dd75c4e0, Name: Talha Saleem
Record added successfully!
Backup created: backup_2025-12-06T21-51-38.json
=====
1. Add Record
2. List Records
3. Update Record
4. Delete Record
5. Search Records
6. Sort Records
7. Export Data
8. View Vault Statistics
9. Exit
=====
Choose option: 1
Enter name: Malik Sohail
Enter value: malik.sohail@gmail.com
[EVENT] Record added: ID 6934a57d88451e32dd75c4e3
Record added successfully!
Backup created: backup_2025-12-06T21-51-57.json
=====
1. Add Record
2. List Records
3. Update Record
4. Delete Record
5. Search Records
6. Sort Records
7. Export Data
8. View Vault Statistics
9. Exit
=====
Choose option: 5
Enter search keyword: Talha
Found 1 matching record(s):
1. ID: 6934a56a88451e32dd75c4e0 | Name: Talha Saleem | Created: 2025-12-06
```

Figure 4: Search Functionality

2.2 Feature 2: Sorting Capability

Description: Allows users to sort records by Name or Creation Date in Ascending or Descending order.

Implementation:

```
1 // Function to sort records
2 function sortRecords(field, order) {
3     const records = [...db.listRecords()]; // Clone to avoid
        modifying original
```

```

4 records.sort((a, b) => {
5     let valA, valB;
6
7     if (field === 'name') {
8         valA = a.name.toLowerCase();
9         valB = b.name.toLowerCase();
10    } else if (field === 'date' || field === 'id') {
11        valA = a.id; // ID is timestamp-based
12        valB = b.id;
13    }
14
15    if (order === 'asc') {
16        return valA > valB ? 1 : valA < valB ? -1 : 0;
17    } else {
18        return valA < valB ? 1 : valA > valB ? -1 : 0;
19    }
20
21 });
22
23 return records;
24
}

```

Menu Option Added: Option 6 - “Sort Records”

Expected Output:

```

1 Choose field to sort by: Name
2 Choose order: Ascending
3 Sorted Records:
4 1. ID: 104 | Name: Adeel
5 2. ID: 110 | Name: Bilal
6 3. ID: 108 | Name: Zain

```

SCREENSHOT 5: Sorting Capability

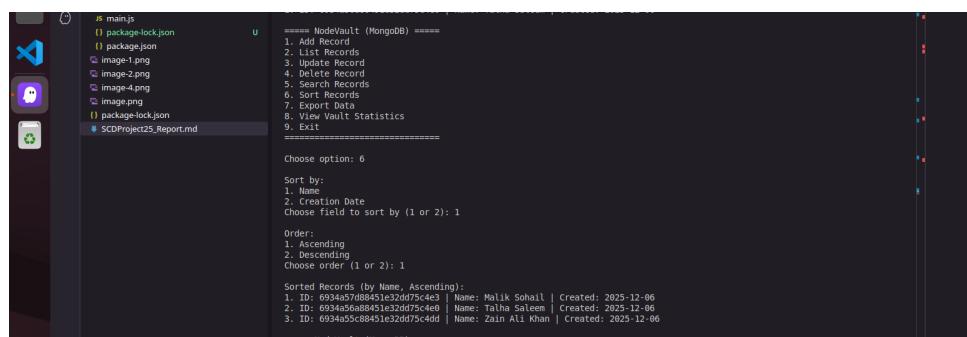


Figure 5: Sorting Capability

2.3 Feature 3: Export Vault Data to Text File

Description: Exports all records to a human-readable export.txt file.

Implementation:

```

1 // Function to export data to text file
2 function exportData() {
3     const records = db.listRecords();
4     const now = new Date();
5     const dateStr = now.toISOString().replace('T', ' ').slice(0,
6         19);
7
8     let content = '=====\\n';
9     content += '          NODEVAULT DATA EXPORT\\n';
10    content += '=====\\n\\n';
11    content += `Export Date/Time: ${dateStr}\\n`;
12    content += `Total Records: ${records.length}\\n`;
13    content += `File Name: export.txt\\n`;
14    // ... rest of formatting
15
16    const exportPath = path.join(__dirname, 'export.txt');
17    fs.writeFileSync(exportPath, content);
18    return exportPath;
}

```

Menu Option Added: Option 7 - “Export Data”
SCREENSHOT 6: Export Functionality

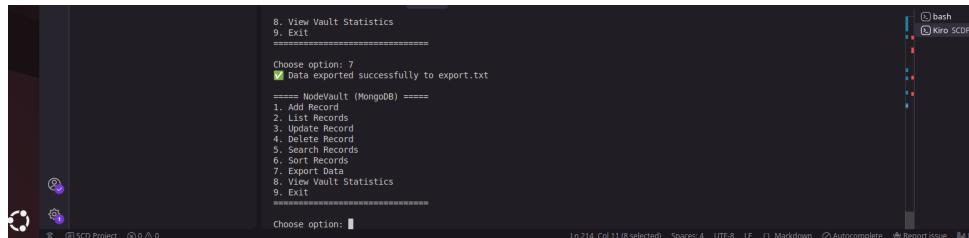


Figure 6: Export Functionality - Part 1

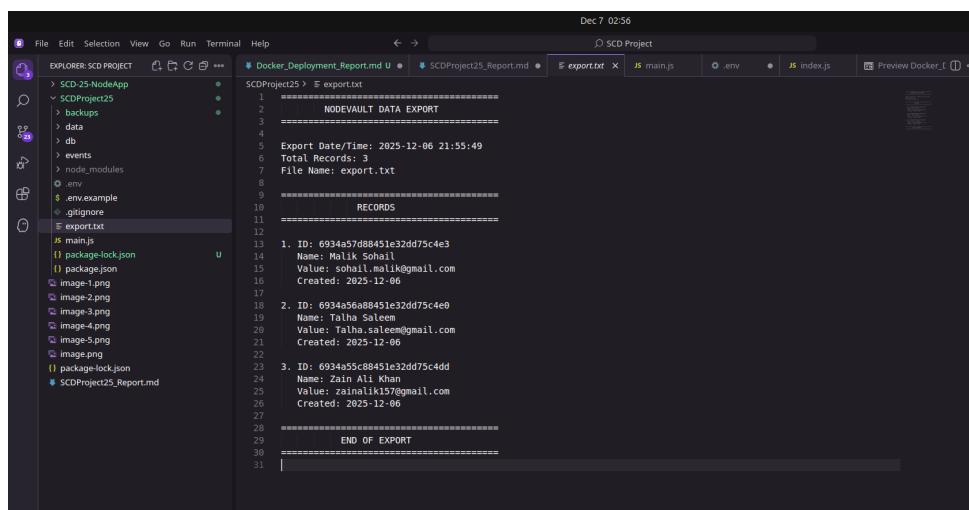


Figure 7: Export Functionality - Part 2

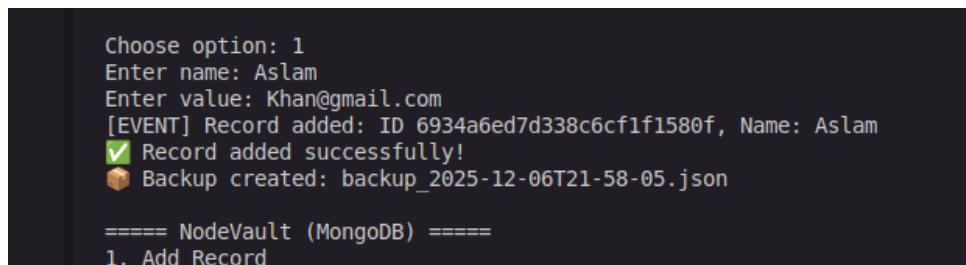
2.4 Feature 4: Automatic Backup System

Description: Automatically creates a backup whenever a record is added or deleted.

Implementation:

```
1 // Backup directory
2 const backupDir = path.join(__dirname, 'backups');
3 if (!fs.existsSync(backupDir)) fs.mkdirSync(backupDir);
4
5 // Function to create automatic backup
6 function createBackup() {
7   const now = new Date();
8   const timestamp = now.toISOString().replace(/[:.]/g,
9     '_').slice(0, 19);
10  const backupFileName = 'backup_${timestamp}.json';
11  const backupPath = path.join(backupDir, backupFileName);
12
13  const records = db.listRecords();
14  fs.writeFileSync(backupPath, JSON.stringify(records, null, 2));
15  console.log('Backup created: ${backupFileName}');
}
```

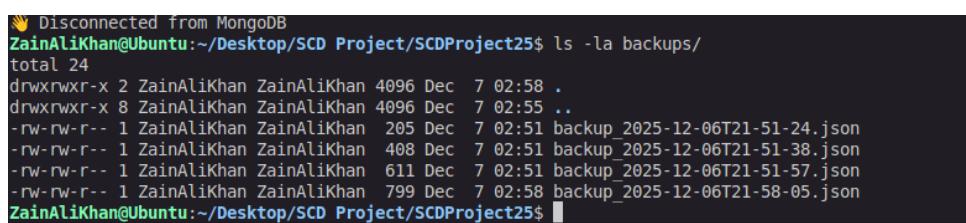
Backup Location: /backups/backup_YYYY-MM-DD_HH-MM-SS.json
SCREENSHOT 7: Automatic Backup



```
Choose option: 1
Enter name: Aslam
Enter value: Khan@gmail.com
[EVENT] Record added: ID 6934a6ed7d338c6cf1f1580f, Name: Aslam
✓ Record added successfully!
📦 Backup created: backup_2025-12-06T21-58-05.json

===== NodeVault (MongoDB) =====
1. Add Record
```

Figure 8: Automatic Backup - Part 1



```
Disconnected from MongoDB
ZainAliKhan@Ubuntu:~/Desktop/SCD Project/SCDProject25$ ls -la backups/
total 24
drwxrwxr-x 2 ZainAliKhan ZainAliKhan 4096 Dec  7 02:58 .
drwxrwxr-x 8 ZainAliKhan ZainAliKhan 4096 Dec  7 02:55 ..
-rw-rw-r-- 1 ZainAliKhan ZainAliKhan  205 Dec  7 02:51 backup_2025-12-06T21-51-24.json
-rw-rw-r-- 1 ZainAliKhan ZainAliKhan  408 Dec  7 02:51 backup_2025-12-06T21-51-38.json
-rw-rw-r-- 1 ZainAliKhan ZainAliKhan  611 Dec  7 02:51 backup_2025-12-06T21-51-57.json
-rw-rw-r-- 1 ZainAliKhan ZainAliKhan  799 Dec  7 02:58 backup_2025-12-06T21-58-05.json
ZainAliKhan@Ubuntu:~/Desktop/SCD Project/SCDProject25$
```

Figure 9: Automatic Backup - Part 2

2.5 Feature 5: Display Data Statistics

Description: Displays useful statistics about the vault data.

Implementation:

```
1 // Function to display vault statistics
2 function getVaultStatistics() {
```

```

3  const records = db.listRecords();
4  const vaultPath = path.join(__dirname, 'data', 'vault.json');
5
6  const stats = {
7    totalRecords: records.length,
8    lastModified: 'N/A',
9    longestName: 'N/A',
10   longestNameLength: 0,
11   earliestRecord: 'N/A',
12   latestRecord: 'N/A'
13 };
14
15 // Get file modification time
16 if (fs.existsSync(vaultPath)) {
17   const fileStat = fs.statSync(vaultPath);
18   stats.lastModified =
19     fileStat.mtime.toISOString().replace('T', ' ').slice(0,
20     19);
21 }
22
23 // ... calculate other statistics
24
25 return stats;
}

```

Menu Option Added: Option 8 - “View Vault Statistics”

Expected Output:

```

1 Vault Statistics:
2 -----
3 Total Records: 5
4 Last Modified: 2025-11-04 15:20:32
5 Longest Name: Muhammad Abdullah (18 characters)
6 Earliest Record: 2025-09-12
7 Latest Record: 2025-11-02
8 -----

```

SCREENTHOT 8: Vault Statistics

2.6 Updated Menu Structure

```

1 ===== NodeVault =====
2 1. Add Record
3 2. List Records
4 3. Update Record
5 4. Delete Record
6 5. Search Records      (NEW)
7 6. Sort Records        (NEW)
8 7. Export Data         (NEW)
9 8. View Vault Statistics (NEW)
10 9. Exit
11 =====

```

```

v2.0
○ ZainAliKhan@Ubuntu:~/Desktop/SCD Project/SCDProject25$ node main.js
✖ Starting NodeVault...
✓ Connected to MongoDB successfully!

===== NodeVault (MongoDB) =====
1. Add Record
2. List Records
3. Update Record
4. Delete Record
5. Search Records
6. Sort Records
7. Export Data
8. View Vault Statistics
9. Exit
=====
Choose option: 8

Vault Statistics:
-----
Total Records: 4
Last Modified: 2025-12-06 22:01:19
Longest Name: Zain Ali Khan (13 characters)
Earliest Record: 2025-12-06
Latest Record: 2025-12-06
-----

===== NodeVault (MongoDB) =====
1. Add Record
2. List Records
3. Update Record
4. Delete Record
5. Search Records
6. Sort Records
7. Export Data
8. View Vault Statistics
9. Exit
=====
Choose option: █

```

Figure 10: Vault Statistics

2.7 Git Commit for Features 1-5

```

1 # Stage all changes
2 git add .
3
4 # Commit with message
5 git commit -m "Added search, sort, export, backup, and statistics
   features"
6
7 # Create version tag
8 git tag -a v3.0 -m "Version 1.0: Core feature enhancements"

```

2.8 Step 5: Merge Feature Branch into Main

```

1 # Switch to main/master branch
2 git checkout master
3
4 # Merge feature branch
5 git merge feature/enhancements
6
7 # Push to remote (if applicable)
8 git push origin master
9
10 # Push tags

```

```
11 | git push --tags
```

SCREENSHOT 14: Merge to Main

Figure 11: Merge to Main

3 Part 4: Containerize the Application

3.1 Step 2: Create Dockerfile

Dockerfile:

```
1 # Use Node.js 18 Alpine as base image
2 FROM node:18-alpine
3
4 # Set working directory
5 WORKDIR /app
6
7 # Copy package files
8 COPY package*.json ./
```

9

```
10 # Install dependencies
11 RUN npm install --production
12
13 # Copy application source code
14 COPY . .
15
16 # Create backups directory
17 RUN mkdir -p /app/backups
18
19 # Set environment to production
20 ENV NODE_ENV=production
```

```
21 # Expose port (if needed for future HTTP API)
22 EXPOSE 3000
23
24
25 # Command to run the application
26 CMD ["node", "main.js"]
```

.dockerignore:

```
1 node_modules
2 npm-debug.log
3 .git
4 .gitignore
5 .env
6 *.md
7 backups/*
8 export.txt
9 data/vault.json
```

3.2 Step 3: Commit Dockerfile

```
1 # Stage changes
2 git add .
3
4 # Commit
5 git commit -m "Added Dockerfile for containerization"
6
7 # Create version tag
8 git tag -a v3.0 -m "Version 3.0: Docker containerization"
```

3.3 Step 4: Build Docker Image

```
1 # Build the Docker image
2 docker build -t nodevault:v1 .
3
4 # Verify image was created
5 docker images | grep nodevault
```

SCREENSHOT 17: Docker Build Process

3.4 Step 5: Create Docker Network

```
1 # Create a network for the containers
2 docker network create nodevault-network
```

3.5 Step 6: Run MongoDB Container

```
[+] ZainAliKhan@Ubuntu:~/Desktop/SCD Project/SCDProject2$ docker build -t nodevault:v1 .
[+] Building 13.5s (12/12) FINISHED                                            docker:default
=> [internal] load build definition from Dockerfile                         0.1s
=> => transferring dockerfile: 501B                                         0.0s
=> [internal] load metadata for docker.io/library/node:18-alpine           2.1s
=> [auth] library/node:pull token for registry-1.docker.io                  0.0s
=> [internal] load .dockerignore                                           0.2s
=> => transferring context: 130B                                         0.0s
=> [1/6] FROM docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e 0.1s
=> => resolve docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e 0.1s
=> [internal] load build context                                         0.1s
=> => transferring context: 30.12kB                                       0.0s
=> => CACHED [2/6] WORKDIR /app                                         0.0s
=> [3/6] COPY package*.json /                                         0.2s
=> [4/6] RUN npm install --production                                    6.5s
=> [5/6] COPY .                                                       0.6s
=> [6/6] RUN mkdir -p /app/backups                                     0.6s
=> exporting to image                                                 2.6s
=> => exporting layers                                              1.8s
=> => exporting manifest sha256:76b1bca71f41af4d99fa3f1615725fd2e332296ffd10650510a0260be6d960c1 0.2s
=> => exporting config sha256:5978c08afc6587fc1404f5dd8b8a21285f6bd3e3ba73303dc816dc3692a1992 0.0s
=> => exporting attestation manifest sha256:8222d433e0b2428c14c29bc0524d3c890b43a0750ded17ea04135be738165bfdf 0.1s
=> => exporting manifest list sha256:ac793a9b817c43f2d426le8c0adc6fb413ec127efcd585b0be1512022133cfb 0.0s
=> => naming to docker.io/library/nodevault:v1                           0.0s
=> => unpacking to docker.io/library/nodevault:v1                      0.4s
● ZainAliKhan@Ubuntu:~/Desktop/SCD Project/SCDProject2$ cd "/home/ZainAliKhan/Desktop/SCD Project" && docker images | grep node
node
nodevault
WARNING: This output is designed for human readability. For machine-readable output, please use --format.
```

Figure 12: Docker Build Process

```
1 # Run MongoDB container (if not already running)
2 docker run -d \
3   --name mongodb \
4   --network nodevault-network \
5   -p 27017:27017 \
6   mongo:latest
```

3.6 Step 7: Run NodeVault Container

```
1 # Run the NodeVault container
2 docker run -it \
3   --name nodevault-app \
4   --network nodevault-network \
5   -e MONGODB_URI=mongodb://mongodb:27017/nodevault \
6   nodevault:v1
```

SCREENSHOT 18: Container Running

```
-name: command-not-found
ZainAliKhan@Ubuntu-:~/Desktop/SCD Project$ docker run -it \
--name nodevault-app \
--network nodevault-network \
-e MONGODB_URI=mongodb://mongodb:27017/nodevault \
nodevault:vl
```
 ↗ Starting NodeVault...
 ✓ Connected to MongoDB successfully!
 ===== NodeVault (MongoDB) =====
 1. Add Record
 2. List Records
 3. Update Record
 4. Delete Record
 5. Search Records
 6. Sort Records
 7. Export Data
 8. View Vault Statistics
 9. Exit
 =====
 Choose option: 1
```

Figure 13: Container Running

### 3.7 Step 10: Publish to Docker Hub

```

1 # Login to Docker Hub
2 docker login
3
4 # Tag the image for Docker Hub
5 # Replace YOUR_DOCKERHUB_USERNAME with your actual username
6 docker tag nodevault:v1 zainalik157/nodevault:v1
7 docker tag nodevault:v1 zainalik157/nodevault:latest
8
9 # Push to Docker Hub
10 docker push zainalik157/nodevault:v1
11 docker push zainalik157/nodevault:latest

```

SCREENSHOT 21: Docker Push

```

Dec 7 03:21
File Edit Selection View Go Run Terminal Help
EXPLORER: SCD PROJECT Docker_Development_Report.md SCProject25_Report.md image-8.png main.js .env index.js Preview D ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
mongod
④ Zainalikhkhan@Ubuntu:~/Desktop/SCD Projects# SC Project 25 - Docker Deployment Report
Error from daemon: container 39fab4b24bd83467d675b62d2aefc79cd0d8263c0c988d843d7c5f6fae9471 is not running
④ Zainalikhkhan@Ubuntu:~/Desktop/SCD Projects# docker run -it \
 -name nodevault-app \
 -network nodevault-network \
 -e MONGODB_URL=mongodb://:27017/nodevault \
 nodevault
docker: Error response from daemon: Conflict. The container name "/nodevault-app" is already in use by container "39fab4b24bd83467d675b62d2aefc79cd0d8263c0c988d843d7c5f6fae9471". You have to remove (or rename) that container to be able to reuse that name.

Run 'docker run --help' for more information
④ Zainalikhkhan@Ubuntu:~/Desktop/SCD Projects# docker login
Authenticating with existing credentials... [Username: zainalik157]
Info - To login with a different account, run 'docker logout' followed by 'docker login'

Login Succeeded
④ Zainalikhkhan@Ubuntu:~/Desktop/SCD Projects# docker tag nodevault:v1 zainalik157/nodevault:v1
④ Zainalikhkhan@Ubuntu:~/Desktop/SCD Projects# docker tag nodevault:v1 zainalik157/nodevault:latest
④ Zainalikhkhan@Ubuntu:~/Desktop/SCD Projects# docker push zainalik157/nodevault:v1
The push refers to repository [docker.io/zainalik157/nodevault]
579f20d105cf: Pushed
4f4fb7006f54: Pushed
16594c89ce65: Mounted from zainalik157/scd-nodeapp
ec9dbd37c446: Mounted from zainalik157/scd-nodeapp
3660b1cb15f6: Pushed
370f20d105cf: Mounted from zainalik157/scd-nodeapp
25f2da83641: Mounted from zainalik157/scd-nodeapp
f18232174bc9: Mounted from zainalik157/scd-nodeapp
41c374341180: Pushed
495de74ab082: Pushed
v1: digest: sha256:aac793a9b817c43f24d2d1ef8c0adc6f8413ec127efcd5850be1512022133cfb size: 856
④ Zainalikhkhan@Ubuntu:~/Desktop/SCD Projects# docker push zainalik157/nodevault:latest
The push refers to repository [docker.io/zainalik157/nodevault]
16594c89ce65: Layer already exists
d971d0e834b0: Layer already exists
25f2da83641: Layer already exists
ec9dbd37c446: Layer already exists
370f20d105cf: Layer already exists
16594c89ce65: Layer already exists
495de74ab082: Layer already exists
41c374341180: Layer already exists
44fb7006f54: Layer already exists
latest: digest: sha256:aac793a9b817c43f24d2d1ef8c0adc6f8413ec127efcd5850be1512022133cfb size: 856
④ Zainalikhkhan@Ubuntu:~/Desktop/SCD Projects

```

Figure 14: Docker Push

Docker Hub URL: <https://hub.docker.com/r/zainalik157/nodevault>

### 3.8 Step 11: Commit and Merge

```

1 # Commit any remaining changes
2 git add .
3 git commit -m "Finalized Docker containerization"
4
5 # Switch to master and merge
6 git checkout master
7 git merge feature/containerization
8
9 # Push tags
10 git push --tags

```

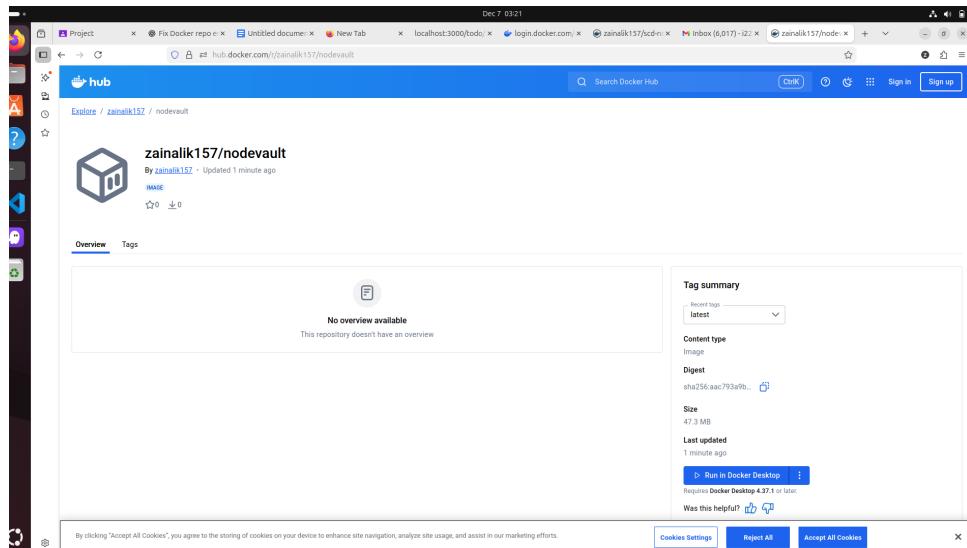


Figure 15: Docker Hub Repository

```

image-1.png 695674a095: Layer already exists
image-2.png 41c3743413bd: Layer already exists
image-3.png 55d0fbcb15f4: Layer already exists
image-4.png 4f4fb708ef54: Layer already exists
latest: digest: sha256:aec793a9fb7c3f24d261ef8d8adc6f8413ec127efcd5850be1512022133cfb size: 856
fatal: not a git repository (or any of the parent directories): .git
@ ZainAlKhan@Ubuntu:~/Desktop/SCD Project$ git commit -m "Finalized Docker containerization"
fatal: not a git repository (or any of the parent directories): .git
@ ZainAlKhan@Ubuntu:~/Desktop/SCD Project$ git add .
@ ZainAlKhan@Ubuntu:~/Desktop/SCD Project$ git add .
@ ZainAlKhan@Ubuntu:~/Desktop/SCD Project$ git commit -m "Finalized Docker containerization"
On branch feature/containerization
Your branch is ahead of 'origin/master' by 3 commits.
 (use "git push" to publish your local commits)
@ ZainAlKhan@Ubuntu:~/Desktop/SCD Project$ git merge feature/containerization
Updating 7cd11d5...1ae2e2c5
Fast-forward
 dockerignore | 9 ++++++-----
 Dockerfile | 26 ++++++++++++++++++
 2 files changed, 35 insertions(+)
 create mode 100644 .dockerignore
 create mode 100644 Dockerfile
@ ZainAlKhan@Ubuntu:~/Desktop/SCD Project$ git push -tags
error: did you mean ``-tags'' (with two dashes)?
@ ZainAlKhan@Ubuntu:~/Desktop/SCD Project$ git push -tags
fatal: Authentication failed for 'https://github.com/LaiBaImran1500/SCDProject25.git/'
@ ZainAlKhan@Ubuntu:~/Desktop/SCD Project$

```

Figure 16: Commit and Merge

### 3.9 Summary - Part 4

## 4 Part 5: Deploy Containers Manually

### 4.1 Overview

In this section, we deploy the containers manually using Docker CLI commands only (no YAML files). We will:

1. Create a private Docker network
2. Attach volumes for persistent MongoDB data
3. Configure ports and environment variables
4. Demonstrate data persistence

### 4.2 Step 1: Clean Up Existing Containers

First, stop and remove any existing containers:

| Task                            | Status |
|---------------------------------|--------|
| Created containerization branch | ✓      |
| Created Dockerfile              | ✓      |
| Created .dockerignore           | ✓      |
| Built Docker image              | ✓      |
| Tested with MongoDB container   | ✓      |
| Documented container logs       | ✓      |
| Documented container processes  | ✓      |
| Published to Docker Hub         | ✓      |

Table 2: Part 4 Summary

```

1 # Stop and remove existing containers
2 docker stop nodevault-app mongodb 2>/dev/null
3 docker rm nodevault-app mongodb 2>/dev/null
4
5 # Remove existing network
6 docker network rm nodevault-network 2>/dev/null
7
8 # Verify cleanup
9 docker ps -a

```

#### SCREENSHOT 22: Cleanup

```

fatal: Authentication failed for https://github.com/ZainAliKhan1500/SCDProject25.git/
• ZainAliKhan@Ubuntu:~/Desktop/SCD Project/SCDProject25$ cd "/home/ZainAliKhan/Desktop/SCD Project" && docker stop nodevault-app
2>/dev/null; docker rm nodevault-app mongodb 2>/dev/null; echo "Cleanup done"
nodevault-app
mongodb
nodevault-app
mongodb
Cleanup done

```

Figure 17: Cleanup

### 4.3 Step 2: Create Private Docker Network

```

1 # Create a private bridge network
2 docker network create --driver bridge --internal
 nodevault-private-network
3
4 # Verify network creation
5 docker network ls
6
7 # Inspect network details
8 docker network inspect nodevault-private-network

```

#### Explanation:

- **-driver bridge:** Creates a bridge network for container communication
- **-internal:** Makes the network private (no external access)
- Containers on this network can communicate with each other but are isolated from the host network

## SCREENSHOT 23: Private Network Created

```
8abd01db7e12 mongo:latest "docker-entrypoint.s..." 9 seconds ago Up 8 seconds 27017/tcp mongodb
● ZainAlikhan@Ubuntu:~/Desktop/SCD Projects$ docker network ls
-private-network
NETWORK ID NAME DRIVER SCOPE
59f027e7fe29 bridge bridge local
8c562470cb29 host host local
9054d867f27b nodevault-private-network bridge local
945a427cb970 none null local
● ZainAlikhan@Ubuntu:~/Desktop/SCD Projects$ docker network inspect nodevault-private-network
[{
 "Name": "nodevault-private-network",
 "Id": "9054d867f27b7ff8be783e6f352658d8fe480db5dc105dfc836e405ecd65453",
 "Created": "2025-12-07T03:28:38.800771539+05:00",
 "Scope": "local",
 "Driver": "bridge",
 "EnableIPv4": true,
 "EnableIPv6": false,
 "IPAM": {
 "Driver": "default",
 "Options": {},
 "Config": [
 {
 "Subnet": "172.18.0.0/16",
 "IPRange": "",
 "Gateway": "172.18.0.1"
 }
]
 },
 "Internal": false,
 "Attachable": false,
 "Ingress": false,
 "ConfigFrom": {
 "Network": ""
 },
 "ConfigOnly": false,
 "Options": {},
 "Labels": {},
 "Containers": {
 "8abd01db7e12e29d1b00bf09625f5317d77b0f49f84f5ce60a24ac201cc6a1df": {
 "Name": "mongodb",
 "EndpointID": "82e5da938b9ea9ddc65998d63f2ffd70c6e91a5ac3b9f8a7611658c4b10c3ee",
 "MacAddress": "9e:31:ee:b4:01:bb",
 "IPv4Address": "172.18.0.2/16",
 "IPv6Address": ""
 }
 },
 "Status": {
 "IPAM": {
 "Subnets": {
 "172.18.0.0/16": {
 "L
```

Figure 18: Private Network Created

## 4.4 Step 3: Create Docker Volume for MongoDB Persistence

```
1 # Create a named volume for MongoDB data
2 docker volume create mongodb-data
3
4 # Verify volume creation
5 docker volume ls
6
7 # Inspect volume
8 docker volume inspect mongodb-data
```

## SCREENSHOT 24: Volume Created

## 4.5 Step 4: Run MongoDB Container with Volume

```
1 # Run MongoDB with volume attached
2 docker run -d \
3 --name mongodb \
4 --network nodevault-private-network \
5 -v mongodb-data:/data/db \
6 -e MONGO_INITDB_DATABASE=nodevault \
7 mongo:latest
```

```
● ZainAliKhan@Ubuntu:~/Desktop/SCD Project$ docker volume ls
a
DRIVER VOLUME NAME
local 7f2bd2cb5ca441c8c029a8da6cae4ff9c724c8d593f923d1fde0aa37c7d3f03a
local 8ddf431ba5c0b8318c02939ad9a4fce59c7e25cab4b6ac2e6e3b0afe9d78f16d
local b04e0b930876fc2c321669bf1f0246c92daab8a313d07dd36ed870869f95
local elda742cb692f3aab896b6d29140acabd3edce728b41b32b149fe8858eb37132
local fcccd70c45f360ebeb3df5bd572122a7460d0ed0258eb6683b6088313524e8e7
local mongodb-data
● ZainAliKhan@Ubuntu:~/Desktop/SCD Project$ docker volume inspect mongodb-data
[
 {
 "CreatedAt": "2025-12-07T03:28:51+05:00",
 "Driver": "local",
 "Labels": null,
 "Mountpoint": "/var/lib/docker/volumes/mongodb-data/_data",
 "Name": "mongodb-data",
 "Options": null,
 "Scope": "local"
 }
]
● ZainAliKhan@Ubuntu:~/Desktop/SCD Project$
```

Figure 19: Volume Created

```
8
9 # Verify MongoDB is running
10 docker ps
11
12 # Check MongoDB logs
13 docker logs mongodb
```

## Command Breakdown:

- `-d`: Run in detached mode (background)
  - `-name mongodb`: Container name
  - `-network nodevault-private-network`: Connect to private network
  - `-v mongodb-data:/data/db`: Mount volume for data persistence
  - `-e MONGO_INITDB_DATABASE=nodevault`: Set initial database name

SCREENSHOT 25: MongoDB Container Running

```
}

● ZainAliKhan@Ubuntu:~/Desktop/SCD Project$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
8abd01db7e12 mongo:latest "docker-entrypoint.s..." 4 minutes ago Up 4 minutes 27017/tcp mongodb

● ZainAliKhan@Ubuntu:~/Desktop/SCD Project$ docker logs mongodb
{"t": {"$date": "2025-12-06T22:29:32.191+00:00"}, "s": "I", "c": "-", "id": 8991200, "ctx": "main", "msg": "Shutting down", "r": {"seed": 3758958863}}
{"t": {"$date": "2025-12-06T22:29:32.200+00:00"}, "s": "I", "c": "CONTROL", "id": 97374, "ctx": "main", "msg": "Automating 1.0 and TLS 1.1, to force-enable TLS 1.1 specify --sslDisabledProtocols 'TLS1_0'; to force-enable TLS 1.0 specify 'none'"}
{"t": {"$date": "2025-12-06T22:29:32.207+00:00"}, "s": "I", "c": "NETWORK", "id": 4915701, "ctx": "main", "msg": "Initiation", "attr": {"spec": {"incomingExternalClient": {"minWireVersion": 0, "maxWireVersion": 27}, "incomingInternalClient": {"WireVersion": 27}, "outgoing": {"minWireVersion": 6, "maxWireVersion": 27}, "isInternalClient": true}}}
{"t": {"$date": "2025-12-06T22:29:32.207+00:00"}, "s": "I", "c": "CONTROL", "id": 5945603, "ctx": "main", "msg": "Multiplexing"}
{"t": {"$date": "2025-12-06T22:29:32.207+00:00"}, "s": "I", "c": "CONTROL", "id": 4615611, "ctx": "initandlisten", "attr": {"pid": 1, "port": 27017, "dbPath": "/data/db", "architecture": "64-bit", "host": "8abd01db7e12"}}
{"t": {"$date": "2025-12-06T22:29:32.208+00:00"}, "s": "I", "c": "CONTROL", "id": 23403, "ctx": "initandlisten", "r": {"buildInfo": {"version": "8.2.2", "gitVersion": "594f839cc1f4385be9a690131412d67b249a0", "openSSLVersion": "OpenSSL modules": [], "allocator": "tcmalloc-google", "environment": {"distmod": "ubuntu2404", "distarch": "x86_64", "target_arch": "x86_64", "os": {"name": "Ubuntu", "version": "24.04"}}, "t": {"$date": "2025-12-06T22:29:32.208+00:00"}, "s": "I", "c": "CONTROL", "id": 51765, "ctx": "initandlisten", "attr": {"os": {"name": "Ubuntu", "version": "24.04"}}, "t": {"$date": "2025-12-06T22:29:32.208+00:00"}, "s": "I", "c": "CONTROL", "id": 21951, "ctx": "initandlisten", "r": {"seed": 3758958863}}
```

Figure 20: MongoDB Container Running

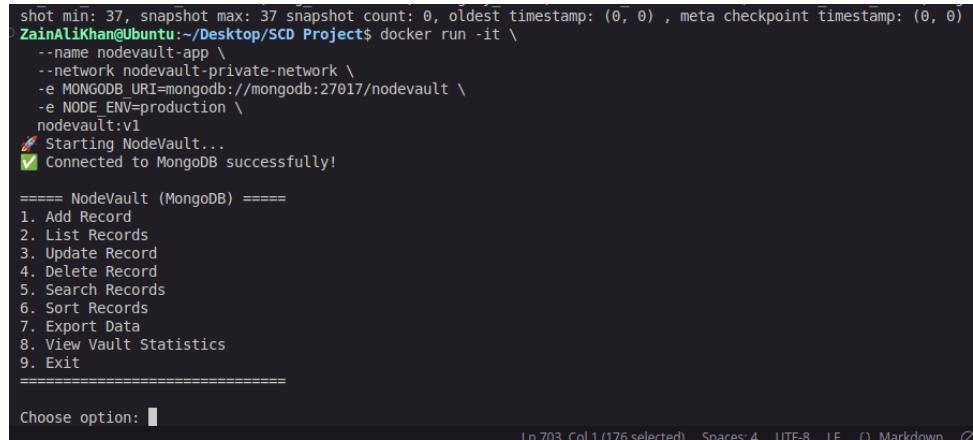
## 4.6 Step 5: Run NodeVault Backend Container

```
1 # Run NodeVault backend container
2 docker run -it \
3 --name nodevault-app \
4 --network nodevault-private-network \
5 -e MONGODB_URI=mongodb://mongodb:27017/nodevault \
6 -e NODE_ENV=production \
7 nodevault:v1
8
9 # For detached mode (background):
10 docker run -d \
11 --name nodevault-app \
12 --network nodevault-private-network \
13 -e MONGODB_URI=mongodb://mongodb:27017/nodevault \
14 -e NODE_ENV=production \
15 nodevault:v1
```

### Command Breakdown:

- `-it`: Interactive mode with terminal
- `--network nodevault-private-network`: Same network as MongoDB
- `-e MONGODB_URI=...`: Environment variable for database connection
- `-e NODE_ENV=production`: Set production environment

### SCREENSHOT 26: NodeVault Container Running



```
shot min: 37, snapshot max: 37 snapshot count: 0, oldest timestamp: (0, 0) , meta checkpoint timestamp: (0, 0)
ZainAlikhan@Ubuntu:~/Desktop/SCD Projects$ docker run -it \
 --name nodevault-app \
 --network nodevault-private-network \
 -e MONGODB_URI=mongodb://mongodb:27017/nodevault \
 -e NODE_ENV=production \
 nodevault:v1
Starting NodeVault...
Connected to MongoDB successfully!
===== NodeVault (MongoDB) =====
1. Add Record
2. List Records
3. Update Record
4. Delete Record
5. Search Records
6. Sort Records
7. Export Data
8. View Vault Statistics
9. Exit
=====
Choose option: [
```

Figure 21: NodeVault Container Running

## 4.7 Step 6: Verify Network Isolation (Proof of Private Network)

```
1 # Check that containers are on the private network
2 docker network inspect nodevault-private-network
3
```

```

4 # Try to access MongoDB from host (should fail with internal
 network)
curl http://localhost:27017 2>&1 || echo "Cannot access - Network
 is private!"

6
7 # Verify containers can communicate internally
docker exec nodevault-app ping -c 2 mongodb

```

SCREENSHOT 27: Network Isolation Proof

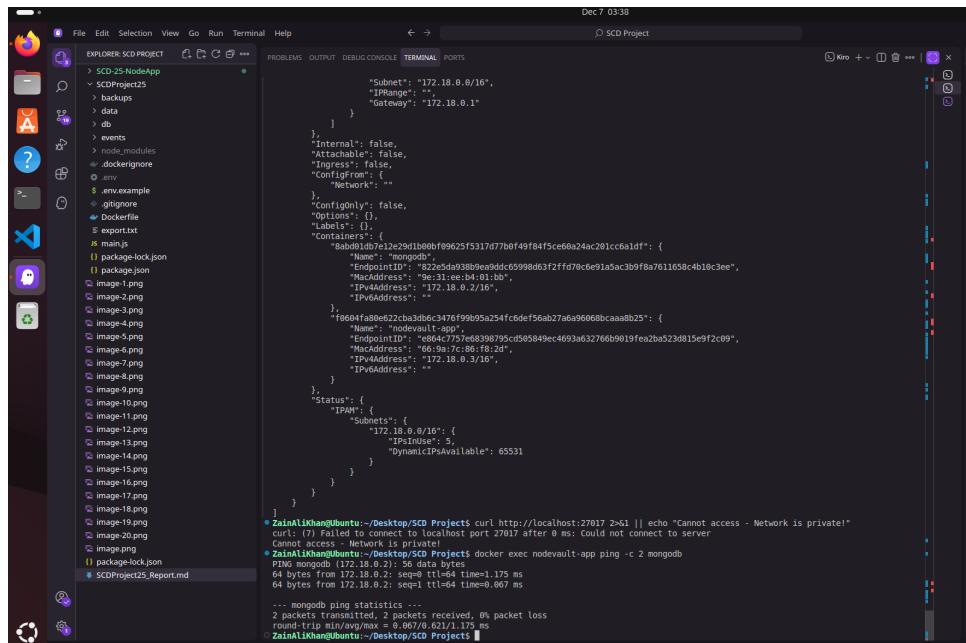


Figure 22: Network Isolation Proof

## 4.8 Step 7.1: Add Data to the Application

```

1 # Run the app and add some records
2 docker run -it \
3 --name nodevault-app \
4 --network nodevault-private-network \
5 -e MONGODB_URI=mongodb://mongodb:27017/nodevault \
6 nodevault:v1

```

Add 2-3 records using the menu (option 1).

## 4.9 Step 7.2: Destroy and Recreate Containers

```

1 # Stop and remove the app container
2 docker stop nodevault-app
3 docker rm nodevault-app
4
5 # Stop and remove MongoDB container
6 docker stop mongodb

```

```

7 docker rm mongodb
8
9 # Verify containers are removed
10 docker ps -a

```

SCREENSHOT 29: Containers Destroyed

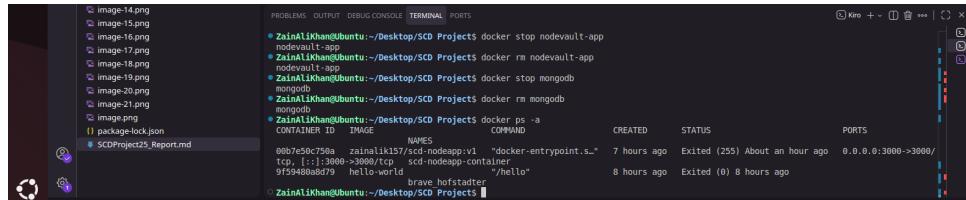


Figure 23: Containers Destroyed

## 4.10 Step 7.3: Relaunch Containers

```

1 # Relaunch MongoDB with same volume
2 docker run -d \
3 --name mongodb \
4 --network nodevault-private-network \
5 -v mongodb-data:/data/db \
6 mongo:latest
7
8 # Wait for MongoDB to start
9 sleep 5
10
11 # Relaunch NodeVault
12 docker run -it \
13 --name nodevault-app \
14 --network nodevault-private-network \
15 -e MONGODB_URI=mongodb://mongodb:27017/nodevault \
16 nodevault:v1

```

List records (option 2) - Data should still be there!

SCREENSHOT 30: Data Persistence Verified

The records still exist after container restart.

## 4.11 Complete List of Docker Commands Used

```

1 # Network Commands
2 docker network create --driver bridge --internal
3 nodevault-private-network
4 docker network ls
5 docker network inspect nodevault-private-network
6 docker network rm nodevault-private-network
7
8 # Volume Commands
9 docker volume create mongodb-data

```

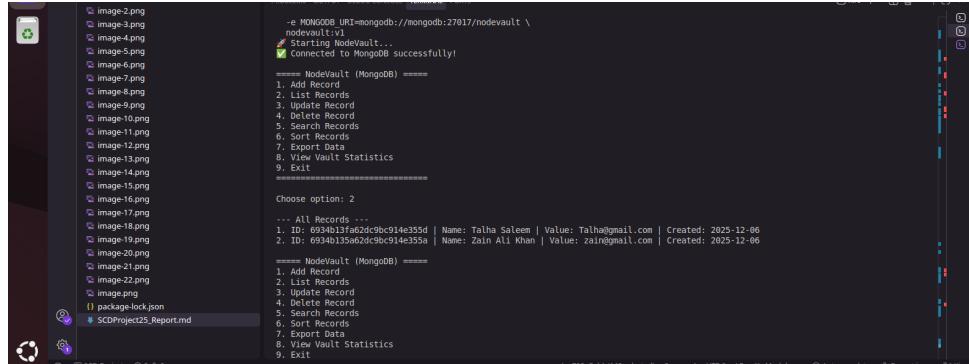


Figure 24: Data Persistence Verified

```

9 docker volume ls
10 docker volume inspect mongodb-data
11
12 # MongoDB Container Commands
13 docker run -d \
14 --name mongodb \
15 --network nodevault-private-network \
16 -v mongodb-data:/data/db \
17 -e MONGO_INITDB_DATABASE=nodevault \
18 mongo:latest
19
20 # NodeVault Container Commands
21 docker run -it \
22 --name nodevault-app \
23 --network nodevault-private-network \
24 -e MONGODB_URI=mongodb://mongodb:27017/nodevault \
25 -e NODE_ENV=production \
26 nodevault:v1
27
28 # Management Commands
29 docker ps
30 docker ps -a
31 docker logs <container_name>
32 docker stop <container_name>
33 docker rm <container_name>
34 docker exec <container_name> <command>

```

## 4.12 Difficulties in Manual Container Setup

## 4.13 Time and Effort Analysis

**Conclusion:** Manual container deployment is time-consuming, error-prone, and difficult to maintain. This highlights the need for Docker Compose (Part 6) to simplify the process.

| Challenge              | Description                                                                                                         |
|------------------------|---------------------------------------------------------------------------------------------------------------------|
| Network Configuration  | Manually creating and managing networks requires understanding of Docker networking concepts. Easy to misconfigure. |
| Volume Management      | Must remember to attach volumes correctly every time. Missing <code>-v</code> flag loses all data.                  |
| Environment Variables  | Must pass all env vars via <code>-e</code> flags. Easy to forget or mistype.                                        |
| Container Dependencies | Must start containers in correct order (MongoDB before app). No automatic dependency management.                    |
| Command Length         | Commands become very long with all options. Error-prone when typing manually.                                       |
| Reproducibility        | Hard to reproduce exact same setup. Must remember all flags and options.                                            |
| Port Conflicts         | Must manually track which ports are in use.                                                                         |
| No Health Checks       | No automatic restart if container fails.                                                                            |

Table 3: Difficulties in Manual Container Setup

| Task                            | Estimated Time   |
|---------------------------------|------------------|
| Understanding Docker networking | 30-60 minutes    |
| Creating and testing network    | 15-20 minutes    |
| Setting up volumes              | 10-15 minutes    |
| Configuring MongoDB container   | 15-20 minutes    |
| Configuring NodeVault container | 15-20 minutes    |
| Testing and debugging           | 30-45 minutes    |
| Documenting commands            | 20-30 minutes    |
| <b>Total</b>                    | <b>2-3 hours</b> |

Table 4: Time and Effort Analysis

## 5 Part 6: Simplifying with Docker Compose

### 5.1 Overview

Docker Compose simplifies multi-container deployment by defining all services, networks, and volumes in a single YAML file. This eliminates the need for multiple long Docker CLI commands.

### 5.2 Step 1: Create docker-compose.yml

```

1 version: '3.8'
2
3 services:
4 # MongoDB Database Service
5 mongodb:
6 image: mongo:latest
7 container_name: nodevault-mongodb
8 restart: unless-stopped

```

```

9 environment:
10 - MONGO_INITDB_DATABASE=nodevault
11 volumes:
12 - mongodb-data:/data/db
13 networks:
14 - nodevault-network
15 healthcheck:
16 test: echo 'db.runCommand("ping").ok' | mongosh
17 localhost:27017/nodevault --quiet
18 interval: 10s
19 timeout: 5s
20 retries: 5
21
22 # NodeVault Backend Service
23 backend:
24 image: nodevault:v1
25 container_name: nodevault-backend
26 restart: unless-stopped
27 depends_on:
28 - mongodb:
29 condition: service_healthy
30 environment:
31 - MONGODB_URI=mongodb://mongodb:27017/nodevault
32 - NODE_ENV=production
33 env_file:
34 - .env
35 networks:
36 - nodevault-network
37 stdin_open: true
38 tty: true
39
40 # Custom Bridge Network
41 networks:
42 nodevault-network:
43 driver: bridge
44 name: nodevault-compose-network
45
46 # Persistent Volume for MongoDB
47 volumes:
48 mongodb-data:
49 driver: local
 name: nodevault-mongodb-data

```

### 5.3 Step 2: Update .env File

```

1 # MongoDB Connection String
2 MONGODB_URI=mongodb://mongodb:27017/nodevault
3
4 # Node Environment
5 NODE_ENV=production

```

## 5.4 Step 3: Stop Existing Containers

```
1 # Stop and remove manually created containers
2 docker stop nodevault-app mongodb 2>/dev/null
3 docker rm nodevault-app mongodb 2>/dev/null
4
5 # Remove old network
6 docker network rm nodevault-private-network 2>/dev/null
```

## 5.5 Step 4: Start Services with Docker Compose

```
1 cd ~/Desktop/SCD\ Project/SCDProject25
2
3 # Start all services
4 docker-compose up -d
5
6 # Or with build flag (if image needs rebuilding)
7 docker-compose up -d --build
```

SCREENSHOT 32: Docker Compose Up

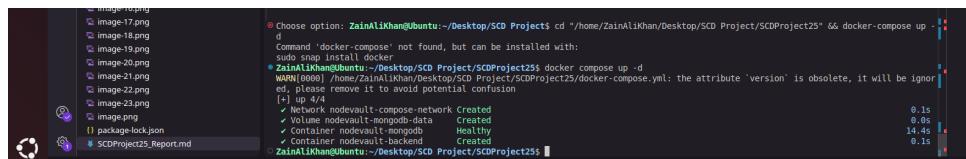


Figure 25: Docker Compose Up

## 5.6 Step 5: Verify Services are Running

```
1 # Check running containers
2 docker-compose ps
3
4 # Or use docker ps
5 docker ps
```

SCREENSHOT 33: Services Running

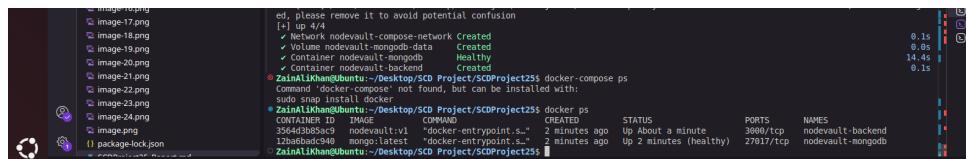


Figure 26: Services Running

## 5.7 Step 7: Test the Application

```
1 # Attach to the backend container
2 docker attach nodevault-backend
3
4 # Or run interactively
5 docker-compose exec backend node main.js
```

SCREENSHOT 35: Application Working

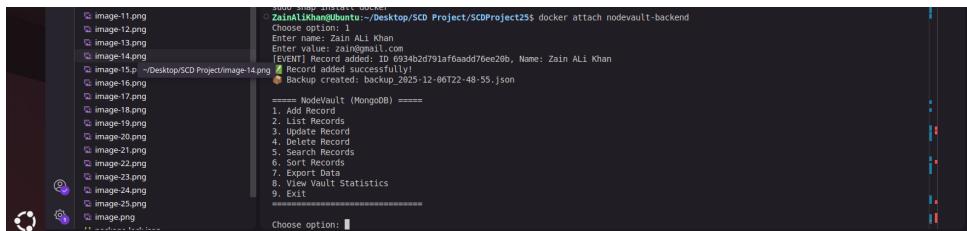


Figure 27: Application Working

## 5.8 Step 8: Verify Network and Volumes

```
1 # Check network
2 docker network ls | grep nodevault
3
4 # Check volumes
5 docker volume ls | grep nodevault
6
7 # Inspect network
8 docker network inspect nodevault-compose-network
```

SCREENSHOT 36: Network and Volumes

## 5.9 Step 9: Stop Services

```
1 # Stop all services
2 docker-compose down
3
4 # Stop and remove volumes (careful - deletes data!)
5 docker-compose down -v
```

## 5.10 Docker Compose vs Manual Deployment Comparison

### 5.11 Benefits of Docker Compose

1. **Single Command Deployment:** `docker-compose up` starts everything
2. **Declarative Configuration:** All settings in one readable YAML file
3. **Automatic Networking:** Services can communicate by name

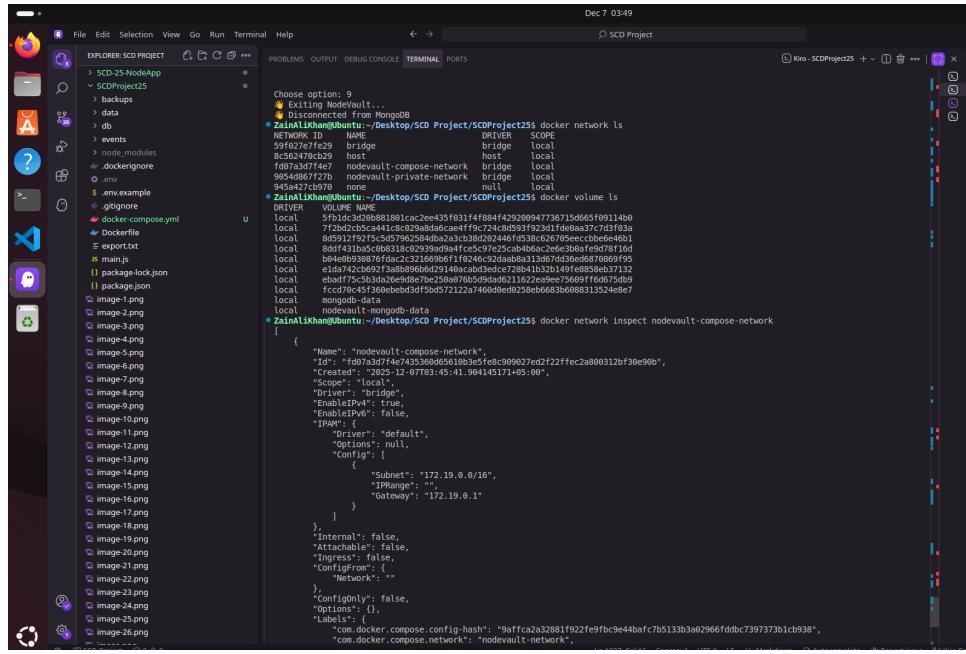


Figure 28: Network and Volumes

| Aspect             | Manual CLI                     | Docker Compose                     |
|--------------------|--------------------------------|------------------------------------|
| Commands needed    | 10+ commands                   | 1 command                          |
| Configuration      | Flags in command line          | YAML file                          |
| Reproducibility    | Hard to reproduce              | Easy - just share YAML             |
| Network setup      | Manual creation                | Automatic                          |
| Volume setup       | Manual creation                | Automatic                          |
| Dependencies       | Manual ordering                | <code>depends_on</code> handles it |
| Environment vars   | Multiple <code>-e</code> flags | <code>.env</code> file             |
| Maintenance        | Edit commands                  | Edit YAML file                     |
| Team collaboration | Share commands                 | Share docker-compose.yml           |

Table 5: Docker Compose vs Manual Deployment Comparison

4. **Volume Management:** Persistent storage defined in config
  5. **Environment Variables:** Loaded from .env file automatically
  6. **Health Checks:** Ensures dependencies are ready before starting
  7. **Easy Scaling:** Can scale services with docker-compose up -scale
  8. **Version Control:** YAML file can be committed to git

| Task                         | Status |
|------------------------------|--------|
| Created docker-compose.yml   | ✓      |
| Defined backend service      | ✓      |
| Defined database service     | ✓      |
| Configured custom network    | ✓      |
| Configured volumes           | ✓      |
| Used .env file               | ✓      |
| Services up with one command | ✓      |

Table 6: Part 6 Summary

## 5.12 Summary - Part 6

# 6 Part 7: Update Project Repo to include Docker Compose

## 6.1 Step 1: Update docker-compose.yml to Build from Dockerfile

The docker-compose.yml is updated to build the image from the Dockerfile instead of using a pre-built image:

```

1 services:
2 # MongoDB Database Service
3 mongodb:
4 image: mongo:latest
5 container_name: nodevault-mongodb
6 restart: unless-stopped
7 environment:
8 - MONGO_INITDB_DATABASE=nodevault
9 volumes:
10 - mongodb-data:/data/db
11 networks:
12 - nodevault-network
13 healthcheck:
14 test: echo 'db.runCommand("ping").ok' | mongosh
15 localhost:27017/nodevault --quiet
16 interval: 10s
17 timeout: 5s
18 retries: 5
19
20 # NodeVault Backend Service
21 backend:
22 build:
23 context: .
24 dockerfile: Dockerfile
25 image: nodevault:latest
26 container_name: nodevault-backend
27 restart: unless-stopped
28 depends_on:
29 - mongodb:

```

```

29 condition: service_healthy
30 environment:
31 - MONGODB_URI=mongodb://mongodb:27017/nodevault
32 - NODE_ENV=production
33 networks:
34 - nodevault-network
35 stdin_open: true
36 tty: true
37
38 # Custom Bridge Network
39 networks:
40 nodevault-network:
41 driver: bridge
42 name: nodevault-compose-network
43
44 # Persistent Volume for MongoDB
45 volumes:
46 mongodb-data:
47 driver: local
48 name: nodevault-mongodb-data

```

**Key Change:** Added build section to backend service:

```

1 build:
2 context: .
3 dockerfile: Dockerfile

```

## 6.2 Step 2: Clean Slate - Remove All Docker Images

```

1 # Stop all running containers
2 docker compose down
3
4 # Remove all unused containers, networks, images, and volumes
5 docker system prune -a
6
7 # Verify images are removed
8 docker images

```

**WARNING:** docker system prune -a removes ALL unused Docker resources. Use with caution!

**SCREENSHOT 37: Clean Slate**

## 6.3 Step 3: Build and Run with Docker Compose

```

1 cd ~/Desktop/SCD\ Project/SCDProject25
2
3 # Build and start all services
4 docker compose up --build

```

**SCREENSHOT 38: Docker Build Process**

**SCREENSHOT 39: Services Running**

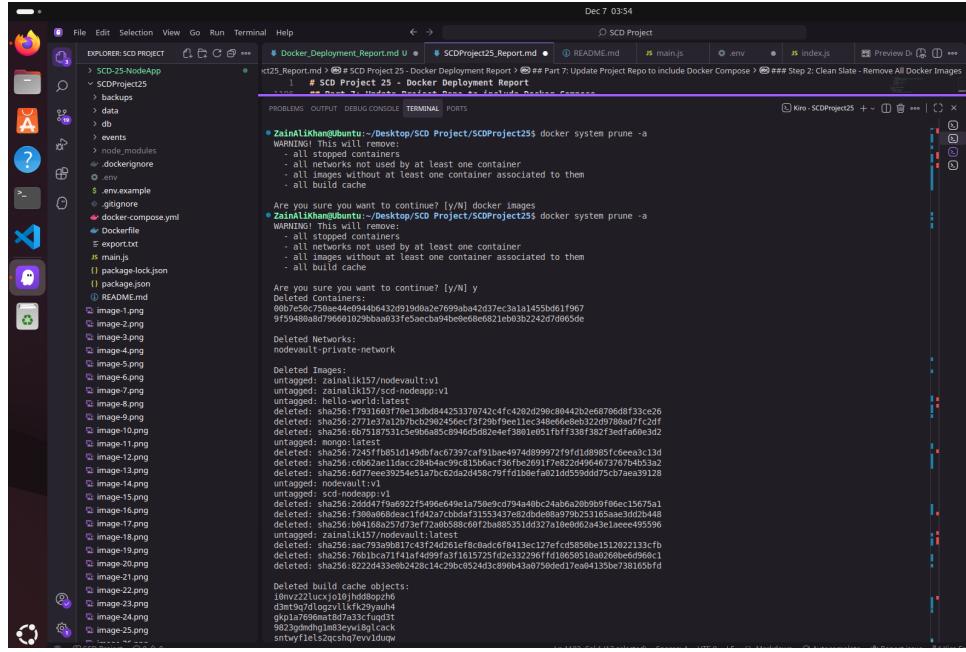


Figure 29: Clean Slate - Part 1



Figure 30: Clean Slate - Part 2

## 6.4 Step 4: Verify Application is Working

```

1 # In another terminal, check running containers
2 docker ps
3
4 # Attach to backend to use the application
5 docker attach nodevault-backend

```

SCREENSHOT 40: Application Functioning

## 6.5 Step 5: Create README.md

A README.md file has been created with:

- Project description
- Features list
- Prerequisites
- Quick start instructions
- Docker commands
- Project structure
- Menu options

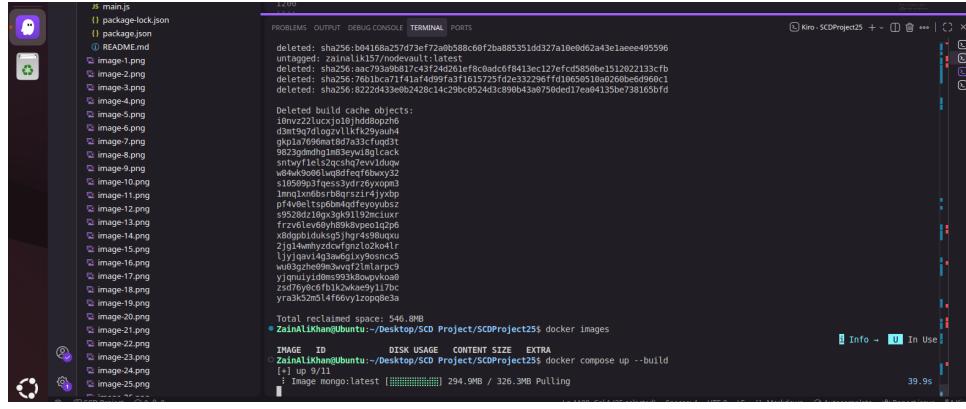


Figure 31: Docker Build Process

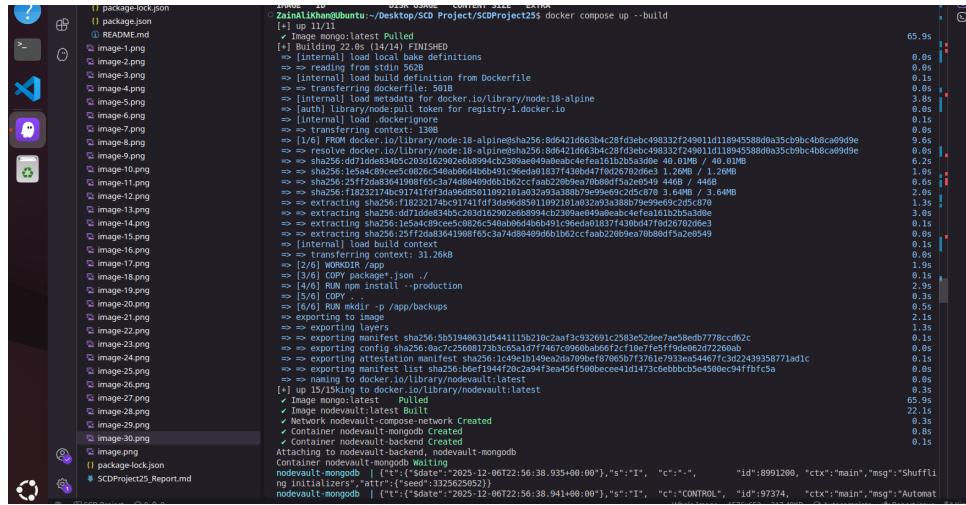


Figure 32: Services Running

## 6.6 Step 6: Commit and Push to GitHub

```

1 cd ~/Desktop/SCD\ Project/SCDProject25
2
3 # Check current branch
4 git branch
5
6 # Switch to master if needed
7 git checkout master
8
9 # Merge feature branch (if not already merged)
10 git merge feature/containerization
11
12 # Stage all changes
13 git add .
14
15 # Commit
16 git commit -m "Added Docker Compose with build configuration and
 README"
17

```

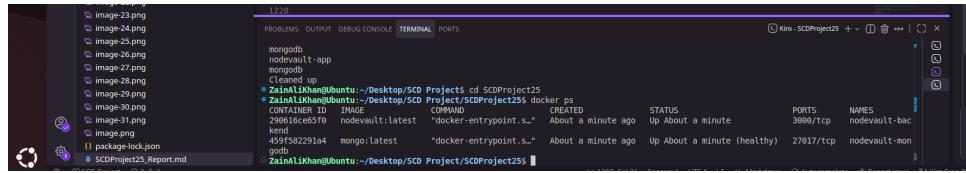


Figure 33: Running Containers

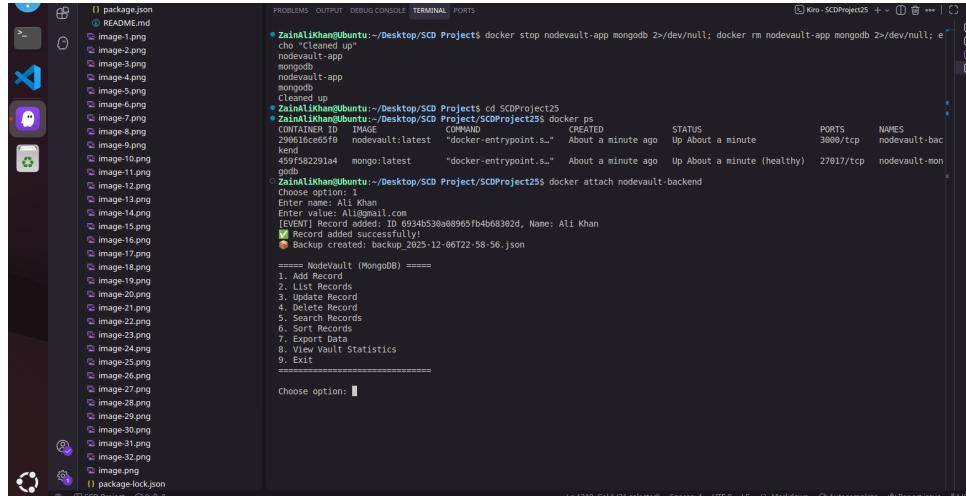


Figure 34: Application Functioning - Part 1

```

18 # Create final version tag
19 git tag -a v4.0 -m "Version 4.0: Complete Docker Compose setup"
20
21 # Push to GitHub
22 git push origin master
23
24 # Push tags
25 git push --tags

```

#### SCREENSHOT 41: Git Commit and Push

- Take a screenshot showing the commit
- Take a screenshot showing the push to GitHub

## 6.7 Step 7: Verify on GitHub

Visit your GitHub repository to verify:

- docker-compose.yml is present
- Dockerfile is present
- README.md is present
- .env.example is present (not .env - it should be in .gitignore)

#### SCREENSHOT 42: GitHub Repository

Figure 35: Application Functioning - Part 2

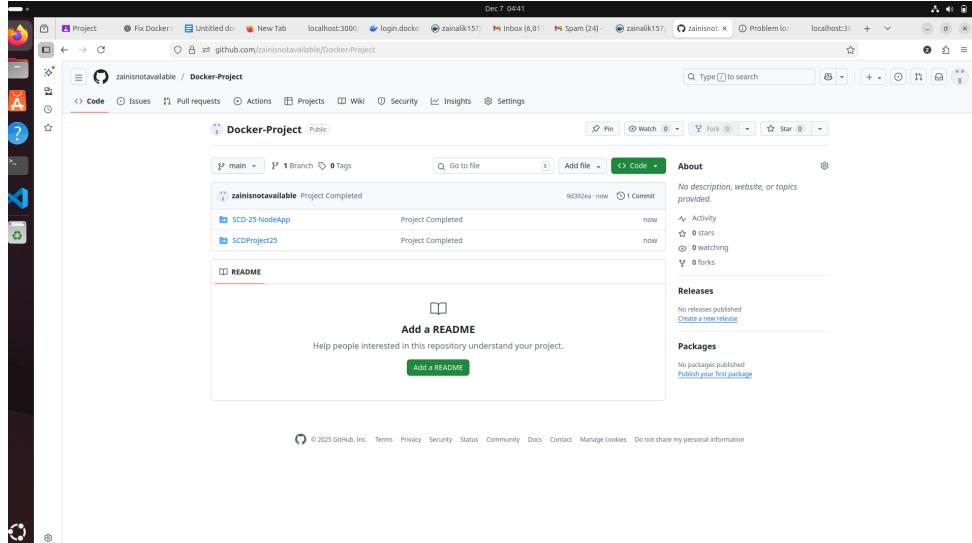


Figure 36: GitHub Repository

## 6.8 Files Committed

## 6.9 Issues Encountered and Solutions

## 6.10 Summary - Part 7

## 7 Final Project Summary

### 7.1 All Parts Completed

## 7.2 Technologies Used

- Node.js 18
  - MongoDB
  - Docker

| File                            | Description                        |
|---------------------------------|------------------------------------|
| <code>docker-compose.yml</code> | Docker Compose configuration       |
| <code>Dockerfile</code>         | Docker image build instructions    |
| <code>.dockerignore</code>      | Files to exclude from Docker build |
| <code>README.md</code>          | Project documentation              |
| <code>.env.example</code>       | Example environment variables      |
| <code>.gitignore</code>         | Git ignore rules                   |
| <code>main.js</code>            | Main application with all features |
| <code>db/</code>                | Database layer with MongoDB        |
| <code>events/</code>            | Event handling                     |
| <code>package.json</code>       | Node.js dependencies               |

Table 7: Files Committed

| Issue                                         | Solution                                                                            |
|-----------------------------------------------|-------------------------------------------------------------------------------------|
| <code>docker-compose</code> command not found | Use <code>docker compose</code> (without hyphen) on newer Docker versions           |
| Version attribute warning                     | Removed <code>version: '3.8'</code> as it's obsolete in newer Docker Compose        |
| MongoDB health check                          | Added proper health check to ensure MongoDB is ready before backend starts          |
| Interactive CLI in container                  | Added <code>stdin_open: true</code> and <code>tty: true</code> for interactive mode |

Table 8: Issues Encountered and Solutions

- Docker Compose
- Git/GitHub

### 7.3 Key Learnings

1. **Environment Consistency:** Docker ensures the same environment across development and production
2. **Container Orchestration:** Docker Compose simplifies multi-container deployments
3. **Data Persistence:** Volumes ensure data survives container restarts
4. **Network Isolation:** Private networks secure container communication
5. **Infrastructure as Code:** `docker-compose.yml` defines entire infrastructure

| Task                                         | Status |
|----------------------------------------------|--------|
| Updated docker-compose.yml with build config | ✓      |
| Cleaned Docker environment                   | ✓      |
| Built images with docker compose up -build   | ✓      |
| Verified application working                 | ✓      |
| Created README.md                            | ✓      |
| Committed all changes                        | ✓      |
| Pushed to GitHub                             | ✓      |

Table 9: Part 7 Summary

| Part   | Description                                                           | Status |
|--------|-----------------------------------------------------------------------|--------|
| Part 3 | Feature Implementation (Search, Sort, Export, Backup, Stats, MongoDB) | ✓      |
| Part 4 | Containerize Application                                              | ✓      |
| Part 5 | Manual Container Deployment                                           | ✓      |
| Part 6 | Docker Compose Setup                                                  | ✓      |
| Part 7 | Update Repo with Docker Compose                                       | ✓      |

Table 10: All Parts Completed