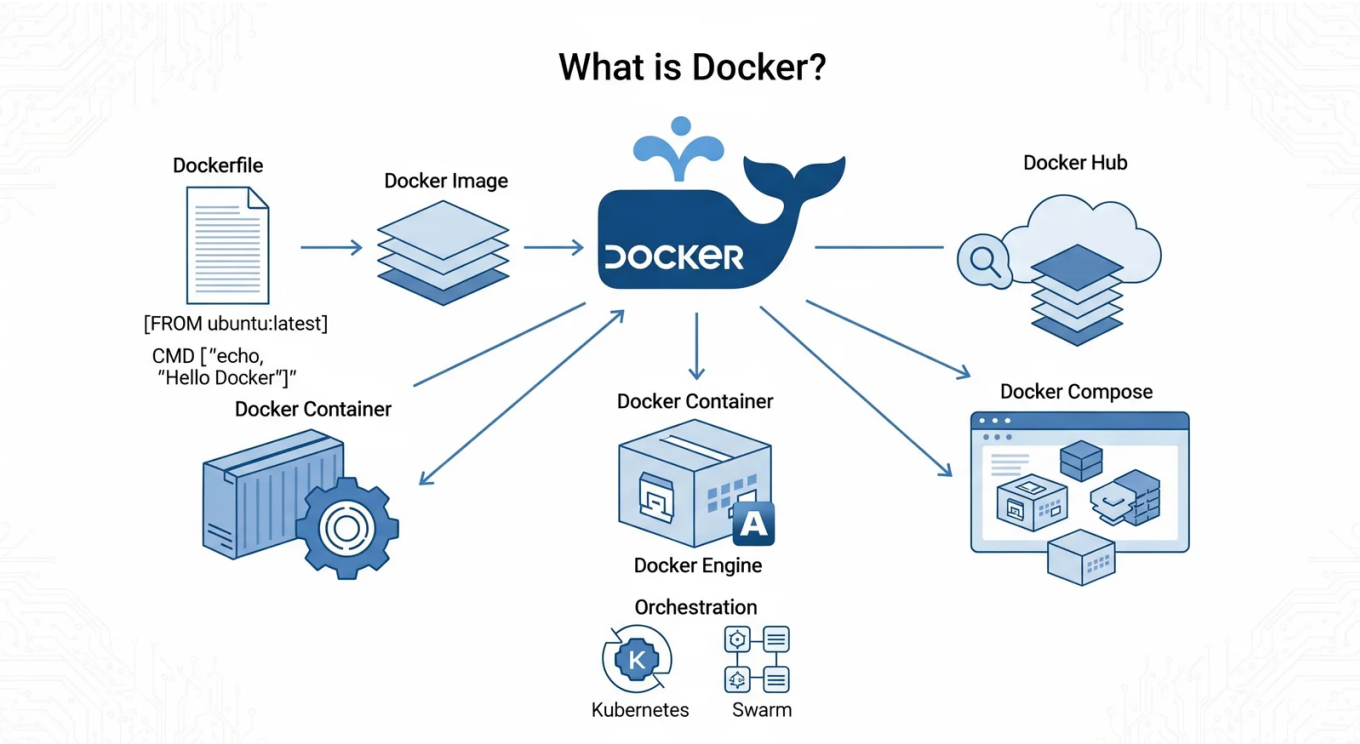
JavaScript and Node.js Application with Docker

Author: Oluwaseun Osunsola
Environment & Tools: AWS ECR, Windows, Docker Desktop, Docker Compose, VSCode
Project Link: https://github.com/Oluwaseunoa/DevOps-Projects/tree/main/Docker-Projects

📌 Project Overview



This project demonstrates how to **containerize a JavaScript (Node.js) application using Docker**, run it alongside **MongoDB** using **Docker Compose**, persist data using **Docker volumes**, and deploy the built image to a **private Docker registry on AWS (Amazon ECR)**.

The application is a simple Node.js + Express web app that stores and retrieves user profile data from MongoDB and serves static content.

🎯 Learning Objectives

This project covers the following Docker concepts:

- Introduction to Docker and containers
- Containers vs Virtual Machines
- Installing Docker
- Core Docker commands
- Debugging containers
- Developing applications with containers
- Building Docker images using Dockerfile
- Running multi-container applications with Docker Compose
- Persisting data with Docker volumes

- Pushing Docker images to a private registry (AWS ECR)
- Deploying a containerized application

🛠️ Technologies Used

- **Node.js**
- **Express.js**
- **MongoDB**
- **Docker**
- **Docker Compose**
- **AWS Elastic Container Registry (ECR)**
- **Mongo Express**

📂 Project Structure

```
.
├── app/
│   ├── images/
│   ├── index.html
│   ├── server.js
│   ├── package.json
│   └── package-lock.json
├── Dockerfile
├── docker-compose.yaml
└── README.md
```

🚀 Application Features

- Serves a static HTML page
- Uploads and retrieves user profile information
- Stores data in MongoDB
- Exposes REST endpoints:
  - `GET /`
  - `GET /get-profile`
  - `POST /update-profile`
  - `GET /profile-picture`

🐳 Dockerfile

The Dockerfile builds a lightweight Node.js image using Alpine Linux.

```
FROM node:13-alpine

ENV MONGO_DB_USERNAME=admin \
    MONGO_DB_PWD=password

WORKDIR /home/app

COPY app/package*.json ./
RUN npm install

COPY app/ .

EXPOSE 3000

CMD ["node", "server.js"]
```

Docker Compose Configuration

Docker Compose is used to orchestrate three services:

- **my-app** – Node.js application
- **mongodb** – MongoDB database
- **mongo-express** – MongoDB web UI

```
version: '3'

services:
  my-app:
    image: 832959958705.dkr.ecr.us-east-1.amazonaws.com/my-app:1.0
    ports:
      - 3000:3000

  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

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

How to Run the Project

Prerequisites

- Docker installed
- Docker Compose installed
- Internet connection (to pull images)

Start the Application

```
docker compose up
```

Or (older Docker versions):

```
docker-compose up
```

Access the Application

Service	URL
Node.js App	http://localhost:3000
Mongo Express	http://localhost:8080

Data Persistence

MongoDB data is persisted using a **Docker volume**:

```
mongo-data
```

This ensures database data remains intact even if containers are stopped or removed.

AWS ECR Integration

The Node.js image is built and pushed to **AWS Elastic Container Registry (ECR)**.

Authenticate Docker to ECR

```
aws ecr get-login-password --region us-east-1 \
| docker login --username AWS --password-stdin \
832959958705.dkr.ecr.us-east-1.amazonaws.com
```

Push Image to ECR

```
docker tag my-app:1.0 832959958705.dkr.ecr.us-east-1.amazonaws.com/my-app:1.0
docker push 832959958705.dkr.ecr.us-east-1.amazonaws.com/my-app:1.0
```

Key Concepts Demonstrated

- Container-based development
- Service-to-service communication via Docker networking
- Using service names as hostnames
- Environment-based configuration
- Persistent storage with volumes
- Private Docker registry usage
- Multi-container orchestration

Wrap Up

This project provides a **complete, real-world example** of building, containerizing, deploying, and running a modern JavaScript application using Docker and Docker Compose. It bridges development and deployment while introducing essential DevOps practices.