docker-compose sample

We are going to build a smaple application using docker compose with two resources web server and a postgress database.

We are going to install **Node.js** and **PostgreSQL** setup using a `.env` file to manage sensitive configuration values like usernames, passwords, and database names.

✅ Instructions for the Project

This guide walks you through setting up a **Node.js web server with PostgreSQL** using **Docker Compose**.
You'll also implement **a Dockerfile for production deployments**.

📌 Step 1: Project Structure

After completing all steps, your project should look like this:

```
.
├── app/                        # Node.js Application Code
│   ├── package.json
│   ├── package-lock.json
│   ├── index.js
│   ├── Dockerfile               # Dockerfile for Node.js app
├── postgres-data/               # PostgreSQL Volume Data (Auto-created)
├── docker-compose.yml           # Docker Compose Configuration
├── .env                         # Environment Variables
├── .env.example                 # Example .env File (For Documentation)

```

📌 Step 2: Environment Variables

Create a `.env` file **in the root directory** and define the database credentials:

```
POSTGRES_USER=user
POSTGRES_PASSWORD=password
POSTGRES_DB=DB1
DB_HOST=db
```

> 🔹 Also, create a `.env.example` file (without actual values) to document required environment variables.

📌 Step 3: Final `docker-compose.yml`

This version **binds volumes**, adds **restart policies**, and properly configures environment variables.

```

services:
  node:
```

```yaml
    build: ./app  # ✅ Build using a Dockerfile in the ./app directory
    container_name: node-app
    working_dir: /app
    volumes:
      - ./app:/app  # ✅ Sync application source code
    ports:
      - "3000:3000"
    command: ["sh", "-c", "sleep 5 && npm start"]  # ✅ Delay startup to ensure DB readiness
    env_file:
      - .env
    depends_on:
      - db
    restart: always  # ✅ Restart if the container crashes

  db:
    image: postgres:latest
    container_name: postgres-db
    restart: always
    volumes:
      - postgres-data:/var/lib/postgresql/data  # ✅ Persistent data storage
      - ./postgres-init:/docker-entrypoint-initdb.d
    ports:
      - "5432:5432"
    env_file:
      - .env
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U $POSTGRES_USER"]
      interval: 10s
      timeout: 5s
      retries: 5

volumes:
  postgres-data:  # ✅ Persistent PostgreSQL data

```

📌 Step 4: Final `Dockerfile` for Node.js

Create a **Dockerfile** inside the `app/` folder.

```dockerfile
# Use Node.js LTS image
FROM node:lts

# Set working directory
WORKDIR /app

# Copy package.json and package-lock.json for dependency installation
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy application source code
COPY . .

# Expose application port
EXPOSE 3000

```

```
# Start the application
CMD ["npm", "start"]

```

📌 Step 5: Setting Up Your Node.js Application

Navigate to your project folder and **create the** `app/` **directory**:

```
mkdir app && cd app

```

✅ Initialize a Node.js Project

Run:

```
npm init -y

```

✅ Install Dependencies

```
npm install pg cors express body-parser dotenv

```

✅ Create `index.js`

Inside `app/`, create `index.js` and add the following:

```
const express = require("express");
const { Pool } = require("pg");
const bodyParser = require("body-parser");
const cors = require("cors");

const app = express();
const PORT = 3000;

// ✅ Enable CORS for API access
app.use(cors());
app.use(bodyParser.urlencoded({ extended: true }));

// ✅ Use Connection Pooling
const pool = new Pool({
  host: process.env.DB_HOST || "localhost",
  port: 5432,
  user: process.env.POSTGRES_USER || "postgres",
  password: process.env.POSTGRES_PASSWORD || "password",
  database: process.env.POSTGRES_DB || "postgres",
  max: 10,
  idleTimeoutMillis: 30000,
});

// ✅ Initialize Database
const initializeDatabase = async () => {
  try {
```

```
    const client = await pool.connect();
    console.log("✅ Connected to PostgreSQL");

    await client.query(`
      CREATE TABLE IF NOT EXISTS users (
        id SERIAL PRIMARY KEY,
        name VARCHAR(100),
        email VARCHAR(100) UNIQUE NOT NULL
      );
    `);
    console.log("✅ Table 'users' is ready");

    const existingUsers = await client.query("SELECT COUNT(*) FROM users;");
    if (parseInt(existingUsers.rows[0].count, 10) === 0) {
      await client.query(`
        INSERT INTO users (name, email) VALUES
        ('Alice Johnson', 'alice@example.com'),
        ('Bob Smith', 'bob@example.com')
      `);
      console.log("✅ Sample data inserted");
    } else {
      console.log("✅ Sample data already exists, skipping insert");
    }

    client.release();
  } catch (err) {
    console.error("❌ Database initialization error:", err);
  }
};

// ✅ Serve Web Page with User List & Form
app.get("/", async (req, res) => {
  try {
    const client = await pool.connect();
    const serverInfo = await client.query("SELECT version();");
    const users = await client.query("SELECT * FROM users;");
    client.release();

    res.send(`
      <html>
      <head><title>Database Info</title></head>
      <body style="font-family: Arial, sans-serif; text-align: center;">
        <h1>🚀 PostgreSQL Database Information</h1>
        <p><strong>Connected Database:</strong> ${process.env.DB_NAME}</p>
        <p><strong>Server:</strong> ${process.env.DB_HOST}</p>
        <p><strong>User:</strong> ${process.env.DB_USER}</p>
        <p><strong>PostgreSQL Version:</strong> ${serverInfo.rows[0].version}</p>

        <h2>📜 Users in Database</h2>
        <ul>
          ${users.rows.map(user => `<li>${user.id}: ${user.name} (${user.email})</li>`).join("")}
        </ul>

        <h2>➕ Add a New User</h2>
        <form action="/add-user" method="POST">
          <input type="text" name="name" placeholder="Full Name" required>
          <input type="email" name="email" placeholder="Email" required>
          <button type="submit">Add User</button>
```

```
        </form>
      </body>
      </html>
    `);
  } catch (err) {
    console.error("❌ Error fetching data:", err);
    res.status(500).send("Error fetching database details.");
  }
});

// ✅ POST Route to Handle Form Submission and Insert User
app.post("/add-user", async (req, res) => {
  const { name, email } = req.body;

  if (!name || !email) {
    return res.status(400).send("Name and email are required.");
  }

  try {
    const client = await pool.connect();
    const result = await client.query("SELECT * FROM users WHERE email = $1;", [email]);

    if (result.rows.length > 0) {
      client.release();
      return res.send("<p style='color:red;'>❌ This email is already registered.</p> <a href='/'>Go Back</a>");
    }

    await client.query("INSERT INTO users (name, email) VALUES ($1, $2);", [name, email]);
    client.release();
    res.redirect("/");
  } catch (err) {
    console.error("❌ Error inserting user:", err);
    res.status(500).send("Error inserting user.");
  }
});

// ✅ Start Express Server
app.listen(PORT, () => {
  console.log(`✅ Server running at http://localhost:${PORT}`);
});

// ✅ Initialize the database
initializeDatabase();

```

📌 Step 6: Running the Project

Before starting the application for first time package.json file need to be edited. Edit the scrip section adding “start” scrip as bellow:

```
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node index.js"

```

```
  },
```

Run the following command to start the application:

```
cd ..
docker-compose up -d

```

Verify services:

```
docker ps

```

Visit:

- **Web app:** `http://localhost:3000`
- **PostgreSQL (DBeaver):** `localhost:5432` (use `.env` credentials)

Benefits of Using `.env` Files

- Centralized management of sensitive configuration values.
- Prevents hardcoding of credentials in the `docker-compose.yml` file.
- Makes it easier to switch environments by swapping `.env` files.