Project title

Building & Deploying a Containerized To-Do App on AWS

Course name & code

Cloud Application Development [CSY401]

Student name(s) : Bharath Kumar J

Student ID(s) : CU22BSC003A

Instructor's name: Uday Josia

Building & Deploying a Containerized To-Do App on AWS

This guide follows the structural template and rigorous step-by-step format. This guide is divided into distinct **Tasks**, each with an **Overview**, **Implementation** steps, and **Verification** to ensure a smooth, error-free experience.

Here is Git Hub repository link: Todo List

Project: Containerized To-Do App on AWS

📋 **Table of Contents**

What you will accomplish

In this tutorial, we will:

- Create a managed NoSQL database using **MongoDB Atlas**.
- Build a **Node.js** application and push it to **GitHub**.
- Containerize the application using **Docker**.
- Provision an **AWS EC2** instance to host your application.
- Deploy the application using **Docker Compose**.

Prerequisites

Before starting this tutorial, you will need:

- An **AWS Account** with administrator-level access.
- A **GitHub Account**.
- **Node.js** and **npm** installed on your local environment.
- **Docker Desktop** installed locally.

Building & Deploying a Containerized To-Do App on AWS

Task 1: Set up Cloud Database

Time to complete	10 minutes
Requires	MongoDB Atlas Account
Get help	MongoDB Atlas Docs

Overview

We will create a hosted MongoDB database. This ensures your data persists even if your Docker containers are restarted.

Implementation

Step 1: Create a Cluster

1. **Sign Up/Log In:** Navigate to MongoDB Atlas and sign in.
2. **Build Database:**
    - Click the **+ Create** button.
    - Select the **M0 (Free)** tier.
    - **Provider:** Select **AWS**.
    - **Region:** Select a region close to you (e.g., **us-east-1**).
    - Click **Create Deployment**.

Step 2: Configure Security

1. **Create Database User:**
    - Navigate to **Database Access** in the sidebar.
    - Click **+ Add New Database User**.
    - **Authentication Method:** Password.
    - **Username:** todo_user
    - **Password:** securePassword123 (or generate a secure one).
    - Click **Add User**.
2. **Network Access:**
    - Navigate to **Network Access** in the sidebar.
    - Click **+ Add IP Address**.
    - Select **Allow Access from Anywhere** (0.0.0.0/0).
    - Click **Confirm**.
    - *Note: This allows your AWS EC2 instance to connect without static IP configuration.*

Building & Deploying a Containerized To-Do App on AWS

Step 3: Get Connection String

1. Navigate to **Database** (sidebar).
2. Click **Connect** on your cluster.
3. Select **Drivers**.
4. Set **Driver** to **Node.js**.
5. Copy the connection string shown. It will look like this:

mongodb+srv://**todo_user:<password>**@cluster0.mongodb.net/?retryWrites=true&w=majority

Conclusion

We have successfully provisioned a managed MongoDB cluster and generated the credentials required for your application to connect.

Task 2: Build the Node.js Application

Time to complete	15 minutes
Requires	Terminal, VS Code

Overview

In this task, you will initialize a Node.js project and create a basic Express server that connects to our MongoDB database.

Implementation

Step 1: Initialize Project

1. Open your terminal and run the following commands to create the project directory:

Bash

```
mkdir todo-app
cd todo-app
npm init -y
```

2. Install the required dependencies:

Bash

```
npm install express mongoose dotenv
```

Building & Deploying a Containerized To-Do App on AWS

Step 2: Create Application Code

1. Create a file named server.js in the root folder and paste the following code:

   Check out code git hub repo

……..

Step 3: Initialize Git

1. Initialize a git repository to prepare for deployment:

Bash

- git init
- git add .
- git commit -m "Initial commit of Todo App"

2. Create a new repository on **GitHub** named node-todo-docker.

3. Push your code (replace <your-username> with your GitHub username):

Bash

- git remote add origin https://github.com**/<your-username>/**node-todo-docker.git
- git branch -M main
- git push -u origin main

Conclusion

You have built a functional Node.js API and pushed the source code to a remote GitHub repository.

Task 3: Containerize with Docker

Time to complete	5 minutes
Requires	Docker Desktop

Overview

Now we will create a Dockerfile to define your application's environment, ensuring it runs consistently on your local machine and the AWS cloud.

Implementation

Step 1: Create Dockerfile

1. In the root of your project, create a file named Dockerfile (no extension).

2. Paste the following content:

……

Building & Deploying a Containerized To-Do App on AWS

Step 2: Create Docker Ignore file

1. Create a file named .dockerignore to prevent unnecessary files from being copied:

Plaintext

- node_modules
- npm-debug.log
- .git

Step 3: Push Changes

1. Commit these new files to GitHub so they are available for your EC2 instance:

Bash

- git add .
- git commit -m "Add Docker configuration"
- git push

Conclusion

Your application is now "Dockerized." The instructions to build the environment are stored in code within your repository.

Task 4: Launch AWS EC2 Instance

Time to complete	5 minutes
Requires	AWS Console

Overview

We will launch a virtual server (EC2) to host your Docker containers.

Implementation

Step 1: Launch Instance

1. Log in to the **AWS Management Console**.
2. Navigate to **EC2** > **Instances** > **Launch Instances**.
3. **Name:** Todo-App-Server.
4. **Application and OS Images (AMI):** Select **Amazon Linux 2023**.
5. **Instance Type:** Select **t2.micro** (Free Tier eligible).

Step 2: Configure Key Pair

1. Under **Key pair (login)**, select **Create new key pair**.
2. **Key pair name:** todo-key.
3. **Key pair type:** RSA.

4. **Private key file format:** .pem.
5. Click **Create key pair**. The file will download automatically. **Keep this file safe.**

Step 3: Network Settings

1. Under **Network settings**, check the boxes for:
    - **Allow SSH traffic from Anywhere** (0.0.0.0/0).
    - **Allow HTTP traffic from the internet**.
2. Click **Edit** (top right of the Network settings box).
3. Click **Add security group rule** to allow traffic to your Node app:
    - **Type:** Custom TCP
    - **Port range:** 3000
    - **Source Type:** Anywhere (0.0.0.0/0)

Step 4: Launch

1. Click **Launch instance**.
2. Click the **Instance ID** to view your running instance. Copy the **Public IPv4 address**.

Conclusion

We have a running Linux server in the cloud, configured to accept traffic on port 3000.

Task 5: Deploy the Application

Time to complete	10 minutes
Requires	SSH Terminal

Overview

We will now connect to EC2 instance, install Docker, pull your code from GitHub, and run the app using Docker Compose.

Implementation

Step 1: Connect to EC2

1. Open your terminal where todo-key.pem is located.
2. Set permissions for the key:

AWS requires the key file to be "read-only" by you. If you try to use it right now, you will likely get an error saying WARNING: UNPROTECTED PRIVATE KEY FILE!.

Run these exact commands in your **PowerShell** window (inside the folder where your key is):

1. **Reset permissions:**

Building & Deploying a Containerized To-Do App on AWS

PowerShell

```
icacls.exe todo-key.pem /reset
```

2. **Grant Read-Only access to your user:**

PowerShell

```
icacls.exe todo-key.pem /grant:r "$($env:USERNAME):(R)"
```

eg: icacls.exe todo-key.pem /grant:r "Bharath kumar":(R)

3. **Remove inherited permissions (This locks it down):**

PowerShell

```
icacls.exe todo-key.pem /inheritance:r
```

Step 3: Connect via SSH

Now that the file exists and is secure, run the connection command. **Do not paste the key content.** Just reference the filename.

1. Make sure you are in the folder with the .pem file.
2. Run this command (replace the 1.2.3.4 with your AWS Public IP):

Get Your Real IP Address

1. Go back to the **AWS Console** in your browser.
2. Navigate to **EC2** > **Instances**.
3. Click on the checkbox next to your instance (Todo-App-Server).
4. Look at the bottom panel (Details tab) or the column named **Public IPv4 address**.
5. **Copy that number** (it will look something like 54.123.45.67 or 3.90.xx.xx).

Step 2: Run the Command with the Real IP

Go back to your Command Prompt and run the ssh command again, but paste your **real IP** at the end.

Example: If your IP is 54.200.100.50, you would type:

DOS

PowerShell

```
ssh -i "todo-key.pem" ec2-user@54.100.200.50
```

Building & Deploying a Containerized To-Do App on AWS

The "Amazon Linux 2023" graphic confirms we have successfully connected to the cloud computer.

Now you need to **install Docker** on this server so it can run our app. Run these commands one by one in your terminal:

Step 1: Install & Start Docker

Copy and paste each line into your terminal and hit **Enter**:

1. Update the installed packages:

Bash

```
sudo yum update -y
```

2. Install Docker:

Bash

```
sudo yum install docker -y
```

3. Start the Docker service:

Bash

```
sudo service docker start
```

4. Give your user permission to use Docker (so you don't have to type sudo every time):

Bash

```
sudo usermod -a -G docker ec2-user
```

Step 2: Refresh Permissions

For the permission change (Step 4) to take effect, you must log out and log back in.

1. Type this to disconnect:

Bash

```
exit
```

2. Now, press the **Up Arrow** key on your keyboard to bring back the last SSH command (or type it again) and hit **Enter** to reconnect:

Bash

```
ssh -i "todo-key.pem" ec2-user@13.203.86.255
```

Building & Deploying a Containerized To-Do App on AWS

First, **verify that Docker is working** by running this command:

Bash

```
docker info
```

If you see a lot of output about "Client" and "Server", you are good to go! If it says "permission denied", let me know.

Next: Create Your App Files

We will create the application files directly on your server.

1. Create a folder for your project:

Bash

```
mkdir todo-app
cd todo-app
```

2. Create the Dockerfile:

- Type nano Dockerfile and hit Enter.
- **Copy** the text below and **Paste** it into the terminal (Right-click to paste in Windows PowerShell).

Dockerfile

```
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3000

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

- **To Save & Exit:** Press Ctrl+O, hit Enter, then press Ctrl+X.

Building & Deploying a Containerized To-Do App on AWS

3. Create package.json:

- Type nano package.json and hit Enter.
- Paste this code:

JSON

```
{
  "name": "todo-app",
  "version": "1.0.0",
  "main": "server.js",
  "dependencies": {
    "express": "^4.18.2",
    "mongoose": "^7.0.3"
  }
}
```

- **To Save & Exit:** Press Ctrl+O, hit Enter, then press Ctrl+X.

4. Create server.js:

- Type nano server.js and hit Enter.
- Paste this code:

……

- **Save & Exit** (Ctrl+O, Enter, Ctrl+X).

5. Create the UI:

- Type mkdir public and hit Enter.
- Type nano public/index.html and hit Enter.
- Paste this code:

……

Save & Exit (Ctrl+O, Enter, Ctrl+X).

Final Step: Run the App!

1. **Build the Docker Image:**

Bash

```
docker build -t todo-app .
```

(Wait for it to finish downloading and installing everything).

Building & Deploying a Containerized To-Do App on AWS

2. **Run the Container:**
    - We need our **MongoDB Connection String** from Task 1.
    - Run this command (replace the part in quotes with your real connection string):

Bash

```
docker run -d -p 3000:3000 -e MONGO_URI="YOUR_MONGODB_CONNECTION_STRING" todo-app
```

3. **Check the website:** Open your browser and go to: Webpage Link