CI CD Docker

1) Go to your project directory and open git bash in the directory.
2) Initialise an empty git repository in the terminal using the command:
   git init
   //git init is a Git command that initializes a new Git repository in your current working directory. When you run git init, Git creates a new repository, adding the necessary files and subdirectories required for version control.
3) Check the status of your current files using the command:
   git status
   // Git will respond with a summary of the current state of your repository. It will show information such as:

   - **Changes that are not staged for commit (modified files).**
   - **Changes that are staged for commit (files added using git add).**
   - **Untracked files (files that are not yet part of the repository).**
   - **Branch information (current branch and whether it's up to date).**

   The git status command is valuable for understanding which files have been modified, which changes are staged and ready to be committed, and which files are untracked.
4) Now before committing the changes, add the files to the staging area using the command:
   git add .
   // The git add . command is used in Git to stage all changes in your working directory for the next commit. When you make modifications to files in your Git repository, Git recognizes these changes as either modified, new, or deleted. However, these changes are not automatically included in the next commit; you need to stage them first. Running git add . stages all changes, including modified, new, and deleted files, so that they are ready to be committed. After staging these changes, you can commit them using git commit.
5) Now commit the changes:
   git commit -m “initial commit”
   //The git commit -m "initial commit" command is used to create a new commit in Git with a commit message indicating that it is the initial commit of the project. This command is commonly used after staging changes using git add . or git add -A to add all changes to the staging area.
6) Go to your github account and create a repository for the project.
7) After creating the repository, copy the repository url from the “**code**” section (highlighted in blue) and go back to the git bash terminal.
8) Run the command:
   git remote add origin “paste the url that you copied”
   //The git remote add origin <URL> command is used in Git to associate a remote repository URL with your local Git repository. This allows you to push and pull changes between your local repository and the remote repository.

   Here's how you can use git remote add origin <URL>:
9) Push the code files to the github

```
git push -u origin master
```

// The git push -u origin master command is used in Git to push the changes from your local master branch to the remote repository (origin). Here's what each part of the command does:

git push: This Git command is used to upload local repository content to a remote repository.
-u origin master: This part of the command specifies the remote (origin) and the branch (master) to push the changes to.

10) Go to your repository, refresh the page and check whether the files are uploaded or not.
11) Now go to "**Add file**" and then "**Create new file**" option
12) Create a dockerfile and include the following instructions:

```
FROM node:latest
WORKDIR /usr/src/app
COPY package.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["node", "index.js"]
```

    This dockerfile contains the code for node.js application

13) After making the dockerfile commit changes in it and save.
14) Go to "**Settings**" and go to "**Secret and Variables**" in the left panel.
15) Choose the "**Actions**" and then click on "**New Repository Secret**".
16) Add your docker hub username and passwords in the repository secret with the name:
    DOCKER_USERNAME
    DOCKER_PASSWORD
17) Go to actions and click on "**set up a workflow yourself**" highlighted in blue at the top.
18) A **main.yaml** file will be opened. Inside the **main.yaml** file, copy the following code:

```
name: Publish Docker image

on:
  push:
    branches: ['master']

jobs:
  push_to_registry:
    name: Push Docker image to Docker Hub
    runs-on: ubuntu-latest
    steps:
      - name: Check out the repo
        uses: actions/checkout@v3

      - name: Log in to Docker Hub
```

```
        uses: docker/login-action@f054a8b539a109f9f41c372932f1ae047eff08c9
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Extract metadata (tags, labels) for Docker
        id: meta
        uses: docker/metadata-action@98669ae865ea3cffbcbaa878cf57c20bbf1c6c38
        with:
          images: avita207/testjodejsapp

      - name: Build and push Docker image
        uses: docker/build-push-action@ad44023a93711e3deb337508980b4b5e9bcdc5dc
        with:
          context: .
          push: true
          file: ./Dockerfile
          tags: ${{ steps.meta.outputs.tags }}
          labels: ${{ steps.meta.outputs.labels }}
```

Note: Make sure you change your docker hub username, image name and your dockerfile location.

19) After adding the code, commit the changes in the main.yaml file.
20) Go to “**Actions**” and see whether the file is being created or not. If yes, then the instructions that are included in the yaml file will be executed and the image will be pushed to the docker hub.
21) Login to your docker hub and check whether the image is pushed or not.
22) Now if that works fine, make some changes in the dockerfile.
23) Go to dockerfile, click on edit and then change it. Eg: change the port from 3000 to 4000.
24) Go to “**Actions**”. You can see that the dockerfile will be updating automatically and the updated image will be pushed to the docker hub.