Configure CI/CD for your application

Aim

Automate the process of building your images when you make changes to their build instructions.

Motivation

As a cloud developer you will need to update cloud applications regularly, by changing their base image definitions. This could be to add functionality, or simply to update a version of a library used (e.g. Flask) because a new version is available.

Your deployed cloud applications, based on your image(s), need to be able to pull changes made to the images. But you don't want to have to build and push to the Docker Hub for every line of code you change in the application.

Therefore, we will implement an automated build pipeline that publishes the `latest` version of our application image(s) to Docker Hub automatically, whenever we commit and push changes to our code repository.

> This pipeline can later be extended to also deploy our image(s) to our Cloud services/swarms once the new image is published.

Requirements

- An account on Docker Hub, such as the one you set up in the previous worksheet.
- You'll need a free GitHub account if you don't already have one. Sign up here.

Get started with GitHub Actions

This tutorial walks you through the process of setting up and using Docker GitHub Actions for building Docker images, and pushing images to Docker Hub. You will complete the following steps:

1. Create a new repository on GitHub.
2. Define the GitHub Actions workflow.
3. Run the workflow.

To follow this tutorial, you need a Docker ID and a GitHub account.

Step one: Create the repository

Create a GitHub repository and configure the Docker Hub secrets.

1. Create a new GitHub repository using this template repository.

The repository contains a simple Dockerfile, and nothing else. Feel free to use another repository containing a working Dockerfile if you prefer.

2. Open the repository **Settings**, and go to **Secrets** > **Actions**.
3. Create a new secret named `DOCKERHUB_USERNAME` and your Docker ID as value.
4. Create a new Personal Access Token (PAT) for Docker Hub. You can name this token `clockboxci`.
5. Add the PAT as a second secret in your GitHub repository, with the name `DOCKERHUB_TOKEN`.

With your repository created, and secrets configured, you're now ready for action!

Step two: Set up the workflow

Set up your GitHub Actions workflow for building and pushing the image to Docker Hub.

1. Go to your repository on GitHub and then select the **Actions** tab.
2. Select **set up a workflow yourself**.

   This takes you to a page for creating a new GitHub actions workflow file in your repository, under `.github/workflows/main.yml` by default.
3. In the editor window, copy and paste the following YAML configuration.
4.
   ```
   name: ci

   on:
     push:
       branches:
         - "main"

   jobs:
     build:
       runs-on: ubuntu-latest
   ```

   - `name`: the name of this workflow.
   - `on.push.branches`: specifies that this workflow should run on every push event for the branches in the list.
   - `jobs`: creates a job ID (`build`) and declares the type of machine that the job should run on.

For more information about the YAML syntax used here, see Workflow syntax for GitHub Actions.

Step three: Define the workflow steps

Now the essentials: what steps to run, and in what order to run them.

```
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      -
        name: Checkout
        uses: actions/checkout@v3
      -
        name: Login to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}
      -
        name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2
      -
        name: Build and push
        uses: docker/build-push-action@v4
        with:
          context: .
          file: ./Dockerfile
          push: true
          tags: ${{ secrets.DOCKERHUB_USERNAME }}/clockbox:latest
```

The previous YAML snippet contains a sequence of steps that:

1. Checks out the repository on the build machine.
2. Signs in to Docker Hub, using the Docker Login action and your Docker Hub credentials.
3. Creates a BuildKit builder instance using the Docker Setup Buildx action.
4. Builds the container image and pushes it to the Docker Hub repository, using Build and push Docker images.

   The `with` key lists a number of input parameters that configures the step:

   - `context`: the build context.
   - `file`: filepath to the Dockerfile.
   - `push`: tells the action to upload the image to a registry after building it.
   - `tags`: tags that specify where to push the image.

Add these steps to your workflow file. The full workflow configuration should look as follows:

```
name: ci

on:
  push:
    branches:
      - "main"

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      -
        name: Checkout
        uses: actions/checkout@v3
      -
        name: Login to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}
      -
        name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2
      -
        name: Build and push
        uses: docker/build-push-action@v4
        with:
          context: .
          file: ./Dockerfile
          push: true
          tags: ${{ secrets.DOCKERHUB_USERNAME }}/clockbox:latest
```

Run the workflow

Save the workflow file and run the job.

1. Select **Start commit** and push the changes to the `main` branch.

   After pushing the commit, the workflow starts automatically.

2. Go to the **Actions** tab. It displays the workflow.

   Selecting the workflow shows you the breakdown of all the steps.

3. When the workflow is complete, go to your repositories on Docker Hub.

   If you see the new repository in that list, it means the GitHub Actions successfully pushed the image to Docker Hub!

Bootcamp Challenge

Automate the Docker build workflow for the app you were working with in the previous practical.

1. Create a GitHub repository for your `python-docker` application, on GitHub.com. Copy the clone URL (https version).
2. Go into your application directory on the terminal (containing the Dockerfile) and run `git init` to initialise the directory for Git version control.
3. Next run `git remote add origin YOUR-HTTPS-GIT-REPOSITORY-URL` with the URL you copied above.
4. Add a new `main.yml` file in the correct location (it will be in a subdirectory of your repository as outlined in the instructions above).
5. Add the correct build and push instructions to the `yml` file, so that your Dockerfile will be used to build an image and push it to the Docker Hub.
6. Add and commit your changes: `git commit -am "added auto build script"`.
7. Push the changes to GitHub: `git push -u origin main`.
8. Check that the GitHub action you defined was triggered (in GitHub) and that the image was pushed (in Docker Hub).