GitHub Actions with Docker

Integrating GitHub Actions with Docker allows you to automate Docker-related tasks within your CI/CD pipelines directly from your GitHub repository. This integration facilitates the seamless building, testing, and deployment of Dockerized applications. Here's a brief explanation of how it works:

- **Workflow Definition:** GitHub Actions workflows are defined in YAML files within the .github/workflows/ directory of your repository. These workflows define a series of steps to be executed whenever certain events occur, such as pushing code changes or manually triggering the workflow.
- **Docker Setup:** Within your GitHub Actions workflow, you can set up Docker by using available actions or running Docker commands directly. Actions such as docker/setup-buildx-action can be used to set up Docker Buildx, which provides extended capabilities for building Docker images.
- **Building Docker Images:** Once Docker is set up, you can use Docker commands to build Docker images from your application's source code. Typically, this involves running a docker build command with appropriate options and arguments to generate a Docker image based on a Dockerfile.
- **Pushing Images to Registry:** After building the Docker image, you may want to push it to a Docker registry such as Docker Hub or a private registry. GitHub Actions allows you to use Docker login commands to authenticate with the registry using credentials stored as secrets, and then push the built image using docker push.
- **Handling Secrets:** Secrets such as Docker Hub credentials or other sensitive information required for the Docker-related tasks can be securely stored in GitHub repository secrets. These secrets can then be accessed within your GitHub Actions workflow, allowing you to authenticate with Docker registries without exposing sensitive information in your code.

> **NOTE:** Adding Environment Variables
>
> To add environment variables to this workflow, you can use GitHub Secrets, which allow you to securely store sensitive information.
>
> - Navigate to your GitHub repository.
> - Go to the "Settings" tab.
> - In the left sidebar, click on "Secrets".
> - Click on the "New repository secret" button.
> - Enter the name and value of your environment variable.
> - Click on "Add secret" to save it.
>
> In the provided workflow, environment variables are used for Docker Hub authentication. The `DOCKER_USERNAME` and `DOCKER_PASSWORD` secrets are used to log in to Docker Hub.

Testing Docker in Workflow

1. Create Workflow File:

- Create a new file named `26-01-Docker.yml` in the `.github/workflows` directory of your repository.

2. Define Workflow:

- Copy and paste the following YAML content into the `26-01-Docker.yml` file:

```yaml
name: 26-01-Docker

on:
  workflow_dispatch:

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

    steps:
      - name: Checkout repository
        uses: actions/checkout@v2

      - name: Set up JDK
        uses: actions/setup-java@v2
        with:
          java-version: '11'
          distribution: 'adopt'
          cache: maven

      - name: Build with Maven
        run: mvn clean install

      - name: Build Docker image
        run: docker build -t username/app .

      - name: Log in to Docker Hub
        run: echo "${{ secrets.DOCKER_PASSWORD }}" | docker login -u "${{
secrets.DOCKER_USERNAME }}" --password-stdin

      - name: Push image to Docker Hub
        run: docker push username/app
```

3. Testing the Workflow:

- **Commit Changes:** Commit the `26-01-Docker.yml` file to your repository.
- **Set Secrets:** Make sure you have set up the required secrets `DOCKER_USERNAME` and `DOCKER_PASSWORD` in your repository's secrets settings with appropriate Docker Hub credentials.
- **Trigger the Workflow:**
  - Go to the Actions tab in your GitHub repository.
  - Click on the "Run workflow" button for the `26-01-Docker` workflow.
  - Optionally, you can specify inputs if your workflow expects them.
- **Review Workflow Execution:**

- Check the Actions tab for the workflow run triggered by your manual action.
- Inspect the logs of each step to ensure that the build and push processes complete without errors.
- Verify on Docker Hub that the image has been pushed successfully.