Lab 23: Docker Build Caching and Artifacts

Lab overview

In this lab activity, you will optimize Docker image builds by enabling caching within a GitHub Actions workflow. This approach significantly reduces build times for large applications by reusing unchanged layers. You will also store Docker images as artifacts for testing and debugging.

In this lab, you will:

- Analyze an existing Docker build process to identify inefficiencies
- Implement Docker layer caching using GitHub Actions and Docker Hub
- Trigger a workflow and observe the impact of caching on build performance
- Publish and download the Docker image as a GitHub artifact
- Clean up deployed resources to avoid incurring unnecessary costs

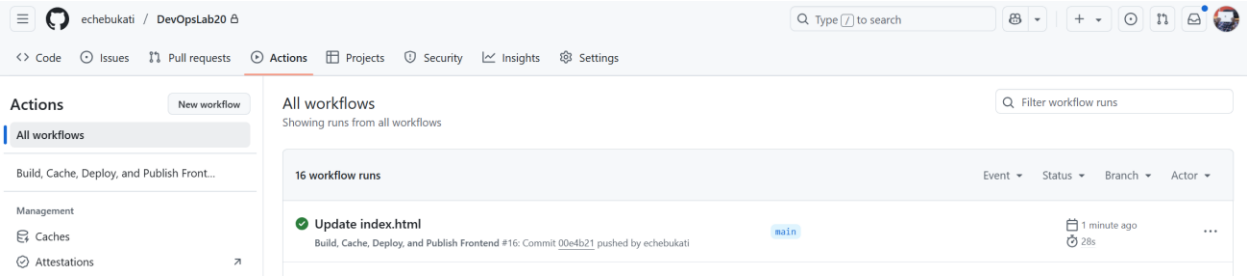
Estimated completion time

30 minutes

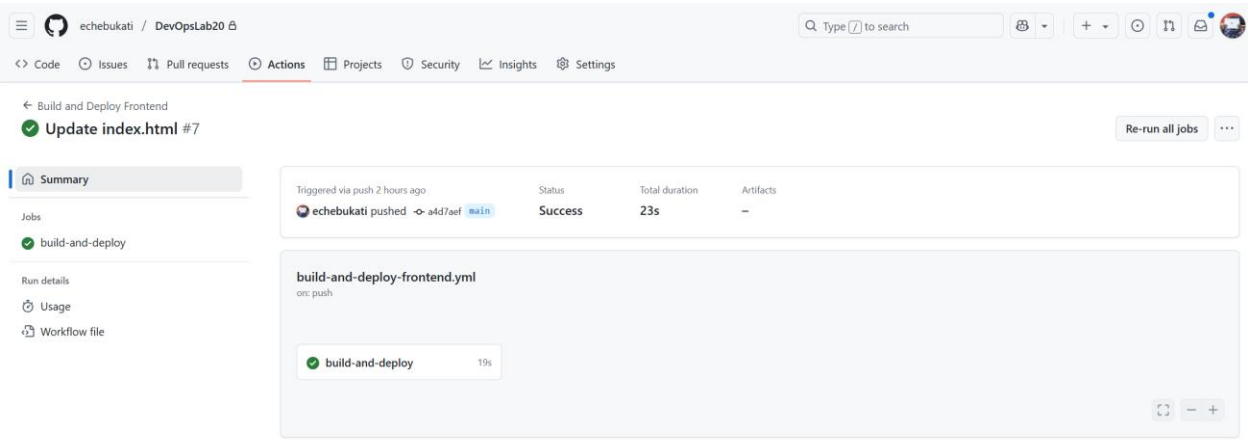
Task 1: Exploring the current build

In this task, you will explore the build step from the previous workflow run, specifically with a view to optimizing it.

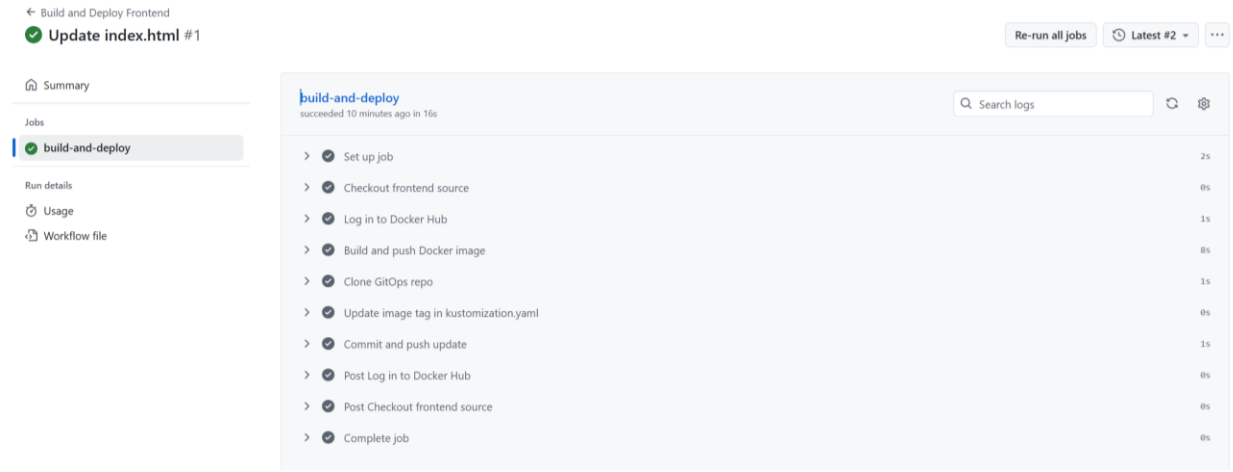
1. Go to **github.com** and open your **DevOpsLab20** repository. Then, navigate to **Actions** and select your latest workflow run from the previous lab (Lab 22).



2. When it loads, click on the **build-and-deploy** job.



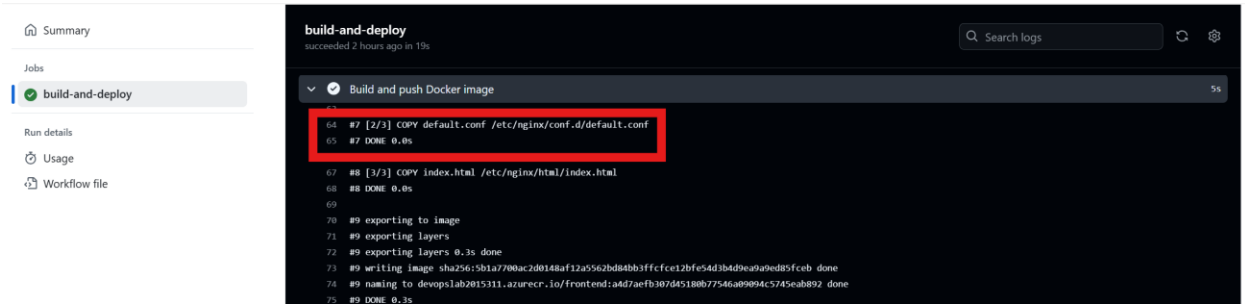
3. Expand the **Build and push Docker image** step.



4. Scroll down to the line that logs the following:

```
[2/3] COPY default.conf /etc/nginx/conf.d/default.conf
```

5. Notice that it says **DONE**.

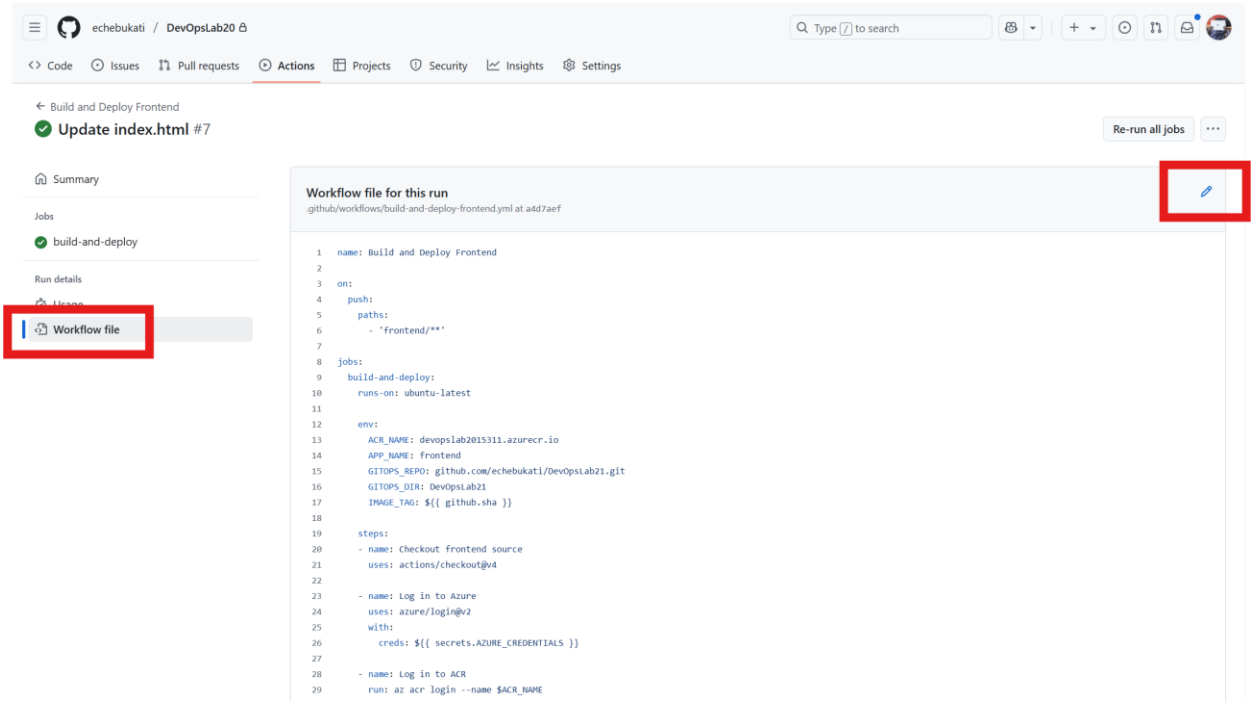


> **Note**
>
> In Lab 22, you only made modifications to **index.html**, yet when you run the build, Docker copied the **default.conf** file as well. This is inefficient, and can slow down your builds when application code becomes large.

Task 2: Adding caching to GitHub Actions workflow

In this task, you will update the GitHub Actions workflow from Lab 20 to introduce Docker caching.

1. Click on **Workflow file** to view the workflow file then click on the pencil icon to edit it.



2. Edit the workflow file and replace all the contents with this code. Replace ***<docker_username>*** with your Docker Hub username, and ***<your-github-username>*** with your GitHub username.

> **Note**
>
> A sample file is in the desktop folder Sample Lab Files/Lab23.

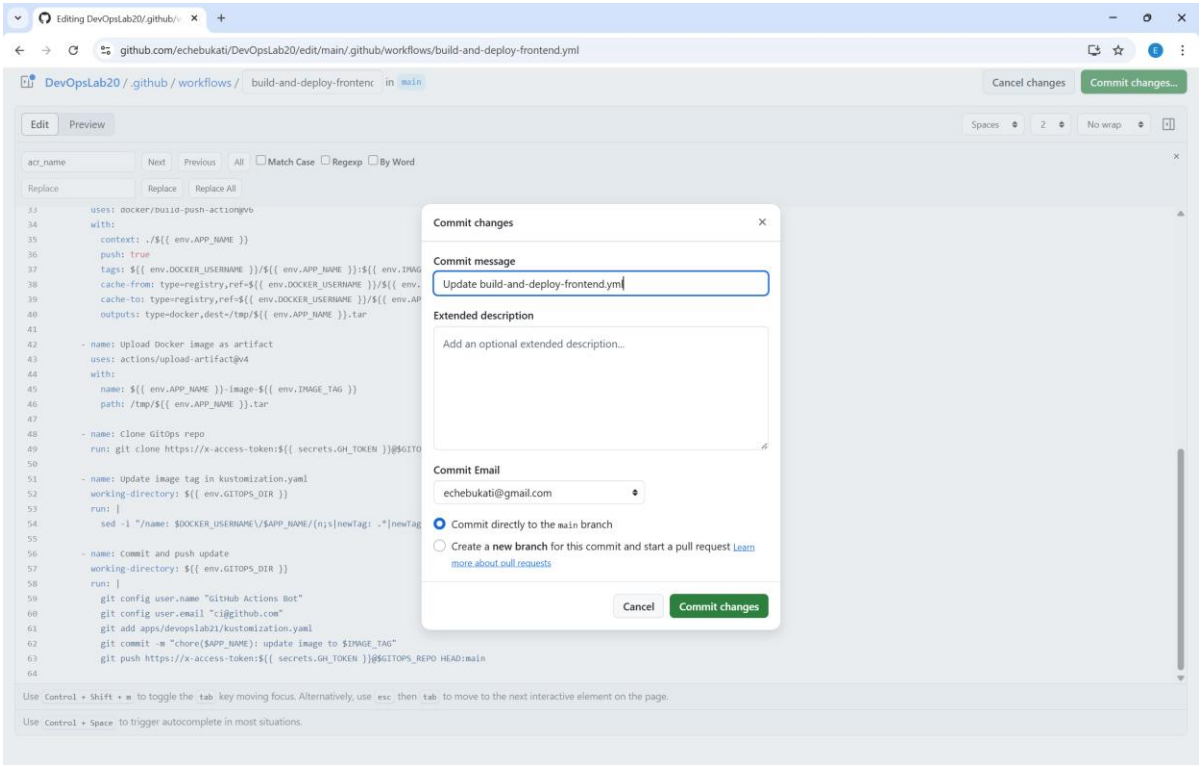
```
name: Build, Cache, Deploy, and Publish Frontend


on:

  push:

    paths:

      - 'frontend/**'
```

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

    env:
      APP_NAME: frontend
      DOCKER_USERNAME: <docker_username>
      GITOPS_REPO: github.com/<your-github-username>/DevOpsLab21.git
      GITOPS_DIR: DevOpsLab21
      IMAGE_TAG: ${{ github.sha }}

    steps:
    - name: Checkout frontend source
      uses: actions/checkout@v4

    - name: Log in to Docker Hub
      uses: docker/login-action@v3
      with:
        username: ${{ env.DOCKER_USERNAME }}
        password: ${{ secrets.DOCKER_PASSWORD }}

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v3

    - name: Build and push Docker image with caching
```

```
      uses: docker/build-push-action@v6

      with:

        context: ./${{ env.APP_NAME }}

        push: true

        tags: ${{ env.DOCKER_USERNAME }}/${{ env.APP_NAME }}:${{
env.IMAGE_TAG }}

        cache-from: type=registry,ref=${{ env.DOCKER_USERNAME }}/${{
env.APP_NAME }}:buildcache

        cache-to: type=registry,ref=${{ env.DOCKER_USERNAME }}/${{
env.APP_NAME }}:buildcache,mode=max

        outputs: type=docker,dest=/tmp/${{ env.APP_NAME }}.tar


    - name: Upload Docker image as artifact

      uses: actions/upload-artifact@v4

      with:

        name: ${{ env.APP_NAME }}-image-${{ env.IMAGE_TAG }}

        path: /tmp/${{ env.APP_NAME }}.tar


    - name: Clone GitOps repo

      run: git clone https://x-access-token:${{ secrets.GH_TOKEN
}}@$GITOPS_REPO


    - name: Update image tag in kustomization.yaml

      working-directory: ${{ env.GITOPS_DIR }}

      run: |

        sed -i "/name: $DOCKER_USERNAME\/$APP_NAME/{n;s|newTag:
.*|newTag: $IMAGE_TAG|}" apps/devopslab21/kustomization.yaml
```

```
    - name: Commit and push update
      working-directory: ${{ env.GITOPS_DIR }}
      run: |
        git config user.name "GitHub Actions Bot"
        git config user.email "ci@github.com"
        git add apps/devopslab21/kustomization.yaml
        git commit -m "chore($APP_NAME): update image to $IMAGE_TAG"
        git push https://x-access-token:${{ secrets.GH_TOKEN
}}@$GITOPS_REPO HEAD:main
```

3. Commit the changes directly to the main branch.

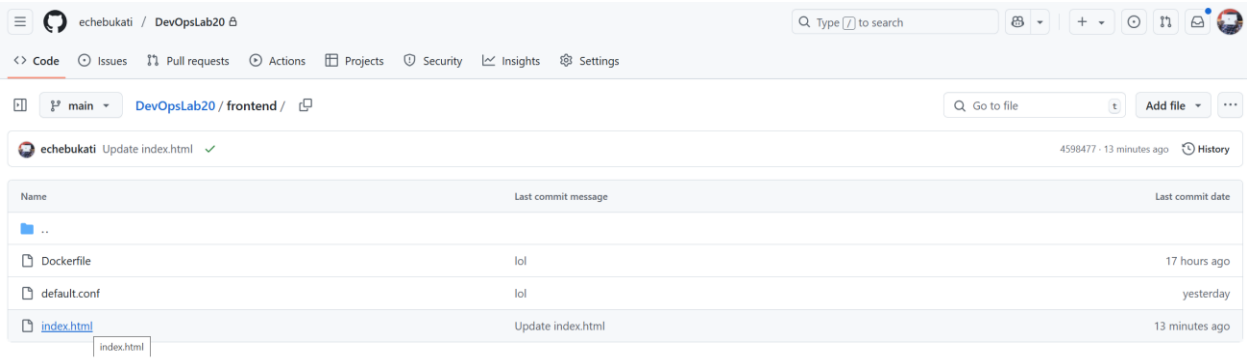


4. This code makes the following changes to the **build-and-deploy** workflow:
   - Sets up Docker Buildx that will allow you to build with caching enabled.
   - Sets up a pre-built action for building and pushing images with caching.
   - Sets up **cache-from** to reference an existing cache during the build.
   - Sets up **cache-to** to update the Docker Hub cache after building.
   - Creates an artifact of the build which is uploaded to GitHub Actions Artifacts.

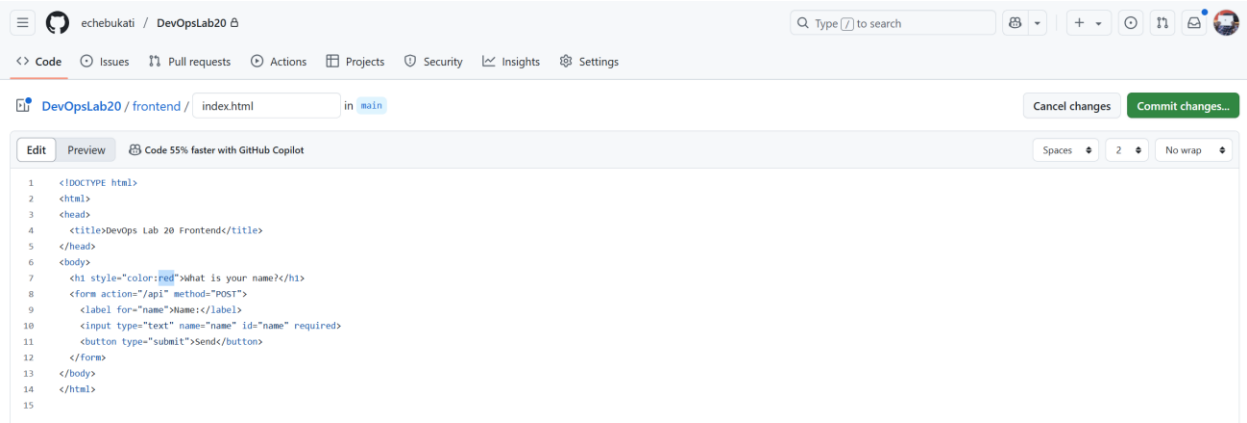
Task 3: Triggering deployment and notice caching

In this task, you will trigger a frontend build and then review the workflow run to see if caching was used.

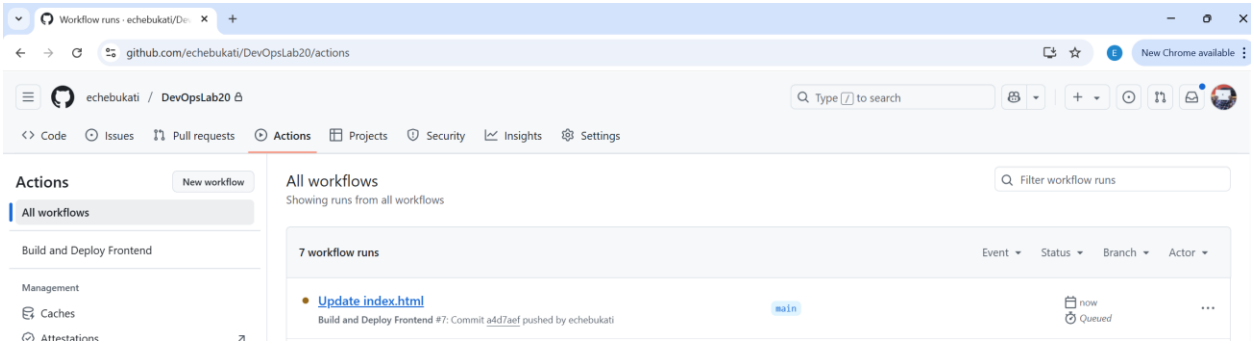
1. Go back to your **DevOpsLab20** repository root and open up **frontend/index.html**. You are going to make a modification to the application code to trigger a fresh deployment.



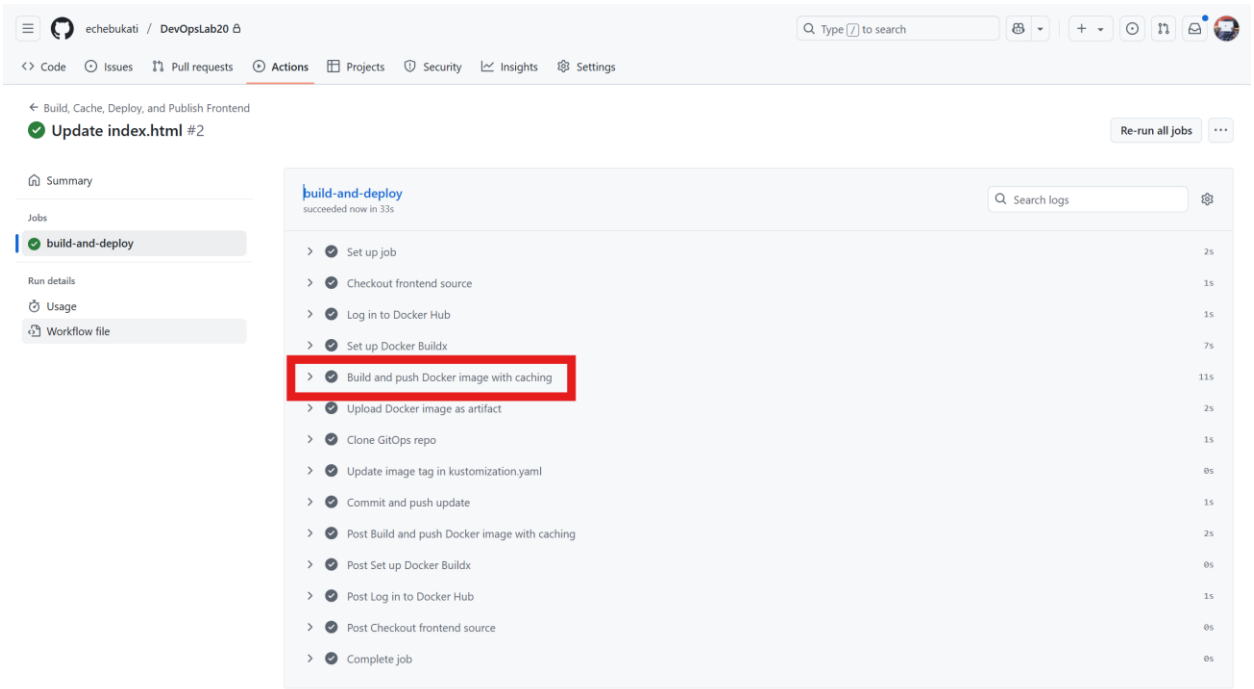
2. Modify the **<h1>** style color from **blue** to **red**. Commit the changes directly to the **main** branch.



3. Go to **Actions** tab and observe that a new workflow has been created. Click on it, then click on the **build-and-deploy** job.



4. Wait for the workflow run complete successfully, then expand the **Build and push Docker image** section.



5. Scroll down to the line that logs action.

```
[2/3] COPY default.conf /etc/nginx/conf.d/default.conf
```

6. Notice that it says CACHED. If you don't notice any change, you can re-run the workflow job from the top-right corner.

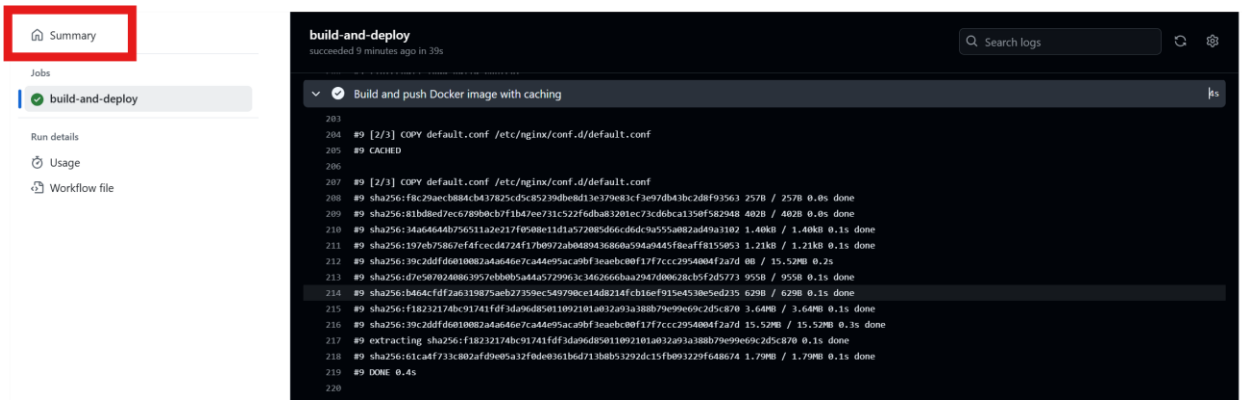
Note

Because **default.conf** was not modified, and because caching is enabled, this file was not copied afresh. The existing build layer was reused. When files get very large, this will help speed up builds.

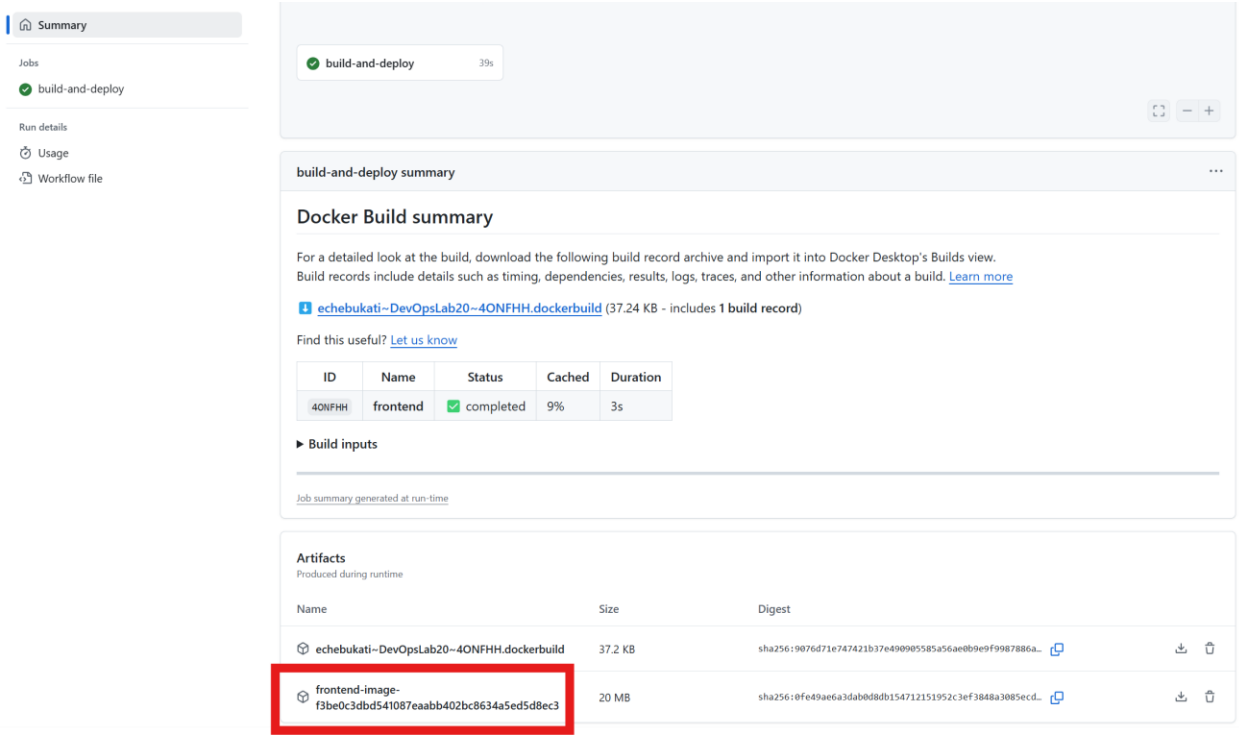
Task 4: Exploring a build artifact

In this task, you will download and run your Docker build artifact locally. Creating an artifact of a Docker image can be useful for debugging, historical reference, or offline testing. Because developers may not always have access to the container registry, this is an alternative way for them to access the builds.

1. Click on Summary to go back to the workflow run page.



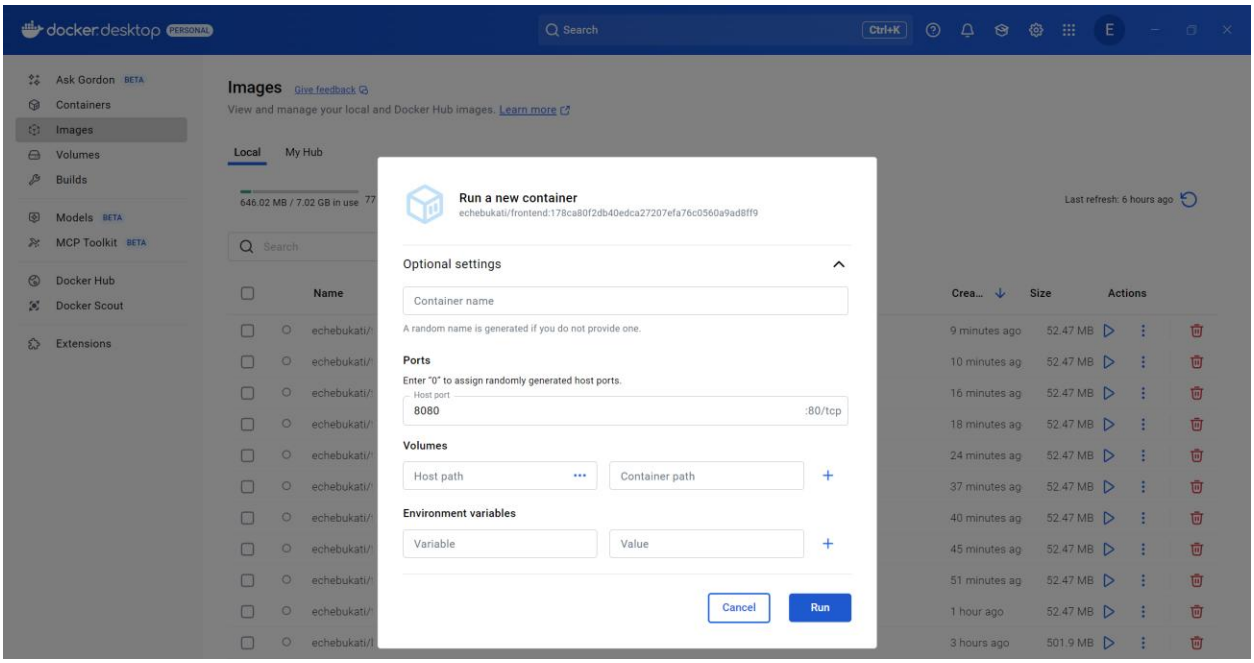
2. Here, you are presented with information about your Docker build (generated by the Docker action). Under artifacts, you can also see your **frontend-image-<sha>** image. Click on it to download it.



3. The image can be loaded locally by running the following code.

```
docker load -i frontend-image-<sha>.tar
```

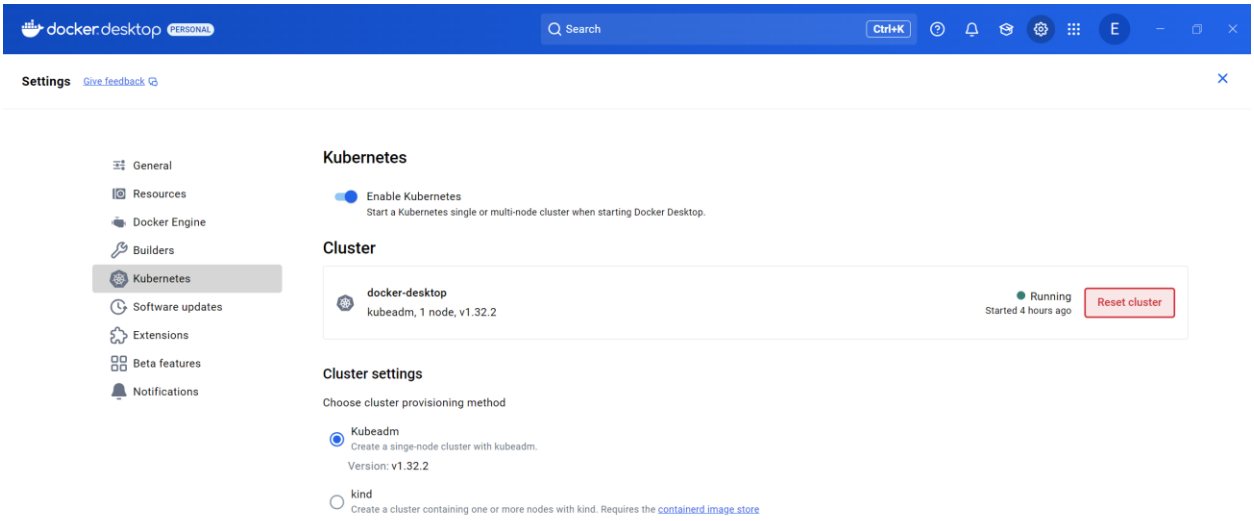
4. Then, you can proceed to run the image locally using Docker Desktop on port 8080.



5. The application run will fail due to the lack of a backend. Can you fix it?

Task 5: Cleaning up resources

1. Reset the Kubernetes cluster on Docker Desktop to delete all local resources.



Lab review

1. What does enabling Docker caching in GitHub Actions help to achieve?
   A. Faster build times by reusing unchanged layers
   B. Faster container runtime performance
   C. Reduced Docker image size
   D. Elimination of the need for Dockerfiles
2. Why is it useful to save a Docker image as a GitHub Actions artifact despite pushing it to the container registry during the build process?

STOP

You have successfully completed this lab.