Walkthrough a Merge

1. Workflow:
   - Use Git to manage your Dockerfile and code changes
   - Build the image locally to test
   - Push changes and let CI/CD pipelines handle deployment
2. CI/CD:
   - The pipeline will build the image and deploy it
   - New deployments will automatically pull the latest image
3. Troubleshooting:
   - Check build logs for errors
   - Ensure you're on the correct Git branch with the latest changes
   - Verify the Dockerfile syntax is correct
4. Best practices:
   - Test builds locally before pushing changes
   - Use version control for Dockerfiles and related code
   - Monitor deployments to catch any issues early

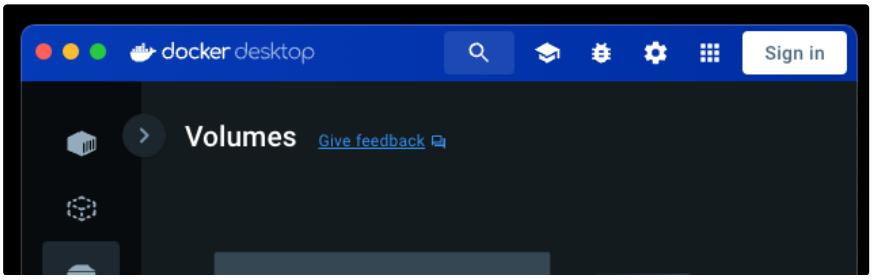
Remember to always follow security best practices when working with Docker, especially regarding secrets and sensitive data. The exact implementation may vary based on your specific project setup and requirements.

This walkthrough
Video

This merge discussed in the walkthrough was in relation to a docker image rebuild
Firstly there was a conversation regarding docker desktop set up and testing of the iamge

Some key points about using Docker:

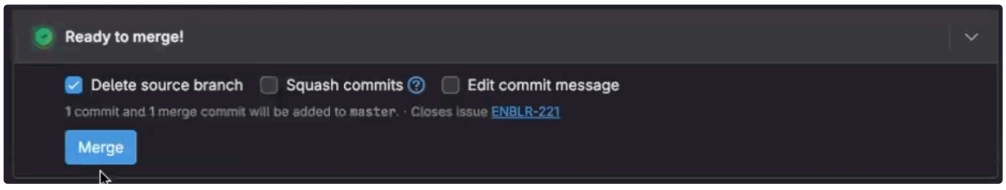
1. Docker Desktop settings advised:
   - Go to Settings (cog icon on the right)



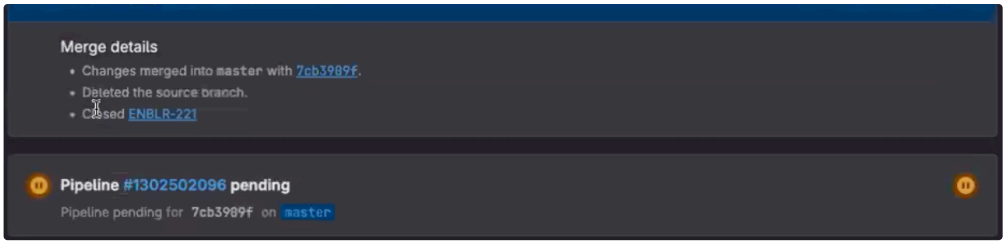
   - Under Resources, CPU limit to Max, increase memory limit to around 12GB, swap to 3GB, and virtual disk to half your available space

- This helps avoid resource-related issues

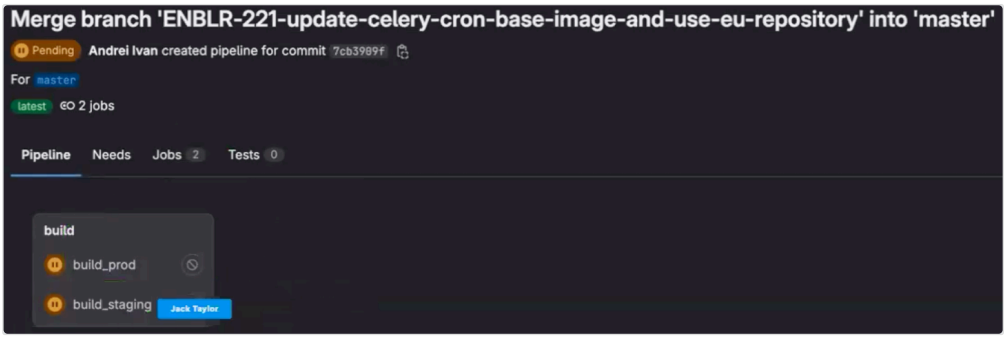
2. Building Docker images:
   In this instance we were checking the workings of the celery docker image to allow checks before merge.
   This required the cloning of the repository `git clone git@gitlab.com:vetor.ai/celery-image.git`. We then ensured we were inside the repository `cd celery-image`.
3. Ensure you have a Dockerfile in the directory `ls -lha`. You should see a text file named Dockerfile
   - Use the command: `docker build -t <image-name> .` (the end `.` is important)
     The `-t` specifies a tag for the image.
     The `.` specifies to use the current directory as the build context
   - This should build the docker image - `docker images -a` will show the image if built
4. Managing secrets:
   - Avoid putting passwords directly in Dockerfiles or environment variables
   - Use Docker secrets or external secret management solutions for sensitive data
5. In this instance there was need to adapt the and check the Dockerfile for the build but it would be expected that the the image had been built and tested before this point.
6. To test the image `docker run -it <image name> bash` will run the image in an interactive manner, dropping you into the running container at a command prompt.
7. If happy with the resulting build and all appears correct, within gitlab on the pull request you will be able to approve and then click merge.



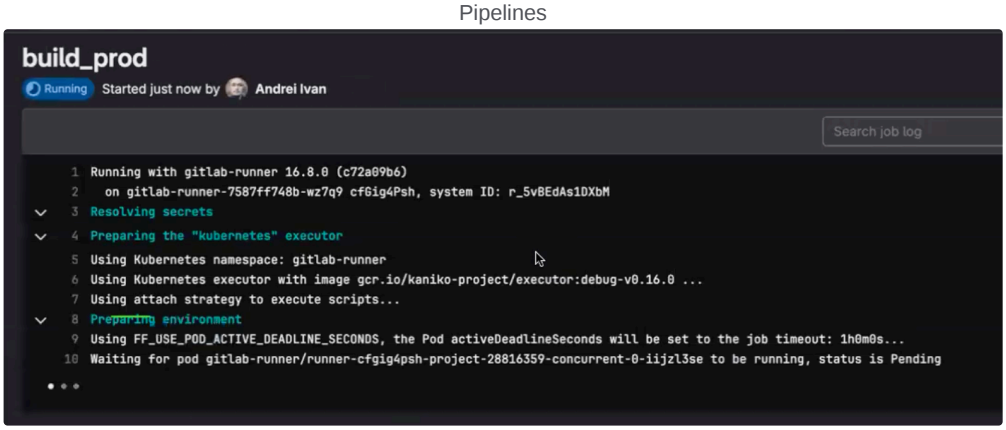
8. This will begin the pipeline built to push the image to google container repository (this may have changed to Artifact Registry)
9. Clicking the link indicated by the pipeline (`#1302502096`) will take you to the pipeline where you can monitor.



10. Here we can see the pipelines running and by clicking on the individual builds you will see the progress and any issues

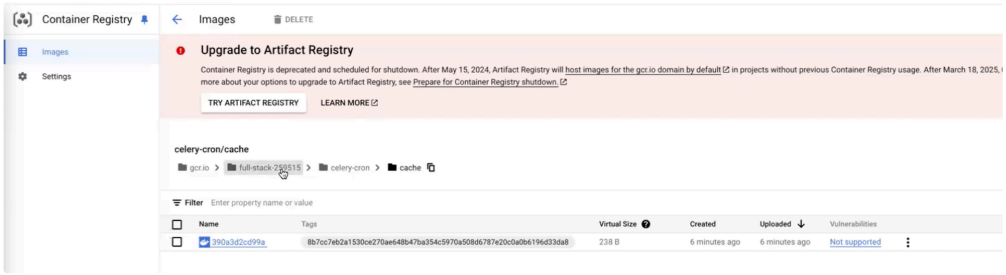


Pipelines



Pipeline details

It is possible to check the output image in gcr.io within the gcp console



> Container Registry is deprecated and scheduled for shutdown. After May 15, 2024, Artifact Registry will host images for the gcr.io domain by default in projects without previous Container Registry usage. After March 18, 2025, Container Registry will be shut down. To learn more about your options to upgrade to Artifact Registry, see Prepare for Container Registry shutdown.)