Continuous Delivery in agile Software Development

Exercise 04 (accompanying Chapter „Continuous Deployment“)

FH-Prof. DI Dr. Marc Kurz

Information & Prerequisites

- In this exercise, you will work with Docker and you will build an image from a Dockerfile.
  - For information about Dockerfiles: https://docs.docker.com/engine/reference/builder/

- Requirements:
  - Docker installed
    - Windows: https://docs.docker.com/docker-for-windows/install/
    - Mac: https://docs.docker.com/docker-for-mac/install
  - DockerHub Account
    - https://hub.docker.com/signup

Instructions (Part 1)

In this part, you will write a Dockerfile and you will build an image from this Dockerfile

- Clone the Git repo to your local computer:
  - > https://github.com/mrckurz/cd2020-ex04
- check if the go program runs locally
  - > `go run main.go`
  - > you should be able to access http://localhost:8888
  - > additionally, the test should run successfully: `go test -v`
- Modify the Dockerfile in the repo
- Build a Docker image based on your Dockerfile
  - > Image tag: [YOUR-DOCKERHUB-ACCOUNT]/my-first-image:0.0.1
  - >

```
docker image build -f Dockerfile -t [YOUR-DOCKERHUB-ACCOUNT]/my-first-image:0.0.1 ./
```

UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA

Instructions (Part 1)

- List all images that are stored in your local registry
  - `docker images`
  - (alternatively, you should also be able to list the images via Docker Desktop)
- Authenticate to the container registry
  - `docker login`

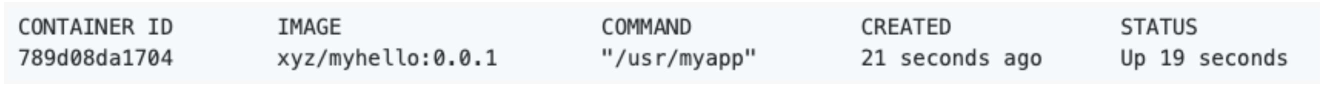
```
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head
Username: YOUR-DOCKERHUB-ACCOUNT
Password: YOUR-DOCKERHUB-PASSWORD
Login Succeeded
```

- Push the created image to your DockerHub account
  - `docker image push [YOUR-DOCKERHUB-ACCOUNT]/my-first-image:0.0.1`
- Verify the push on your account: https://hub.docker.com

Instructions (Part 2)

In this part, you will build a Docker image and run a container from this image

- Create image from the provided Dockerfile:
  - > `docker image build -t [your-dockerhub-account]/myhello:0.0.1 ./`
- Run the container from the image and expose the container port: **8888** to the host port: **9090**
  - > `docker container run -p 9090:8888 [your-dockerhub-account]/myhello:0.0.1`
- Open a browser and go to: http://localhost:9090
- See your container running on your local Docker daemon:
  - > `docker ps`



```
CONTAINER ID        IMAGE                COMMAND             CREATED             STATUS
789d08da1704        xyz/myhello:0.0.1    "/usr/myapp"        21 seconds ago      Up 19 seconds
```

- Stop your container
  - > `docker stop 789d08da1704`

Instructions (Part 3)

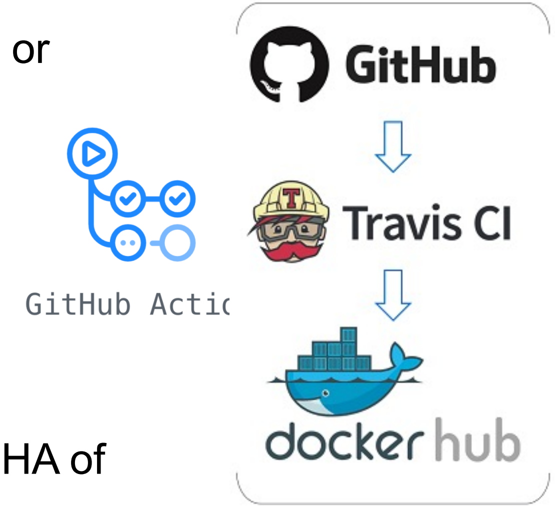
In this part you will let Travis/GitHub Actions build your Docker image and upload to DockerHub

- Either use the project from the previous In-Class Exercise or use this example
- Let Travis / GitHub Actions create the Docker image and upload this to DockerHub
- The following snippets might be helpful:

```
- echo "$REGISTRY_PASSWORD" | docker login --username $REGISTRY_USER --password-stdin
- docker build -f Dockerfile -t YOUR-DOCKERHUB-ACCOUNT/demo:latest ./
```

- Extend the Travis / Github Actions configuration file with a Docker tag command - the tag has to be the Git commit SHA of this build:

```
.
GIT_SHA="$(git rev-parse --short HEAD)"
docker tag YOUR-DOCKERHUB-ACCOUNT/demo:latest YOUR-DOCKERHUB-ACCOUNT/demo:$GIT_SHA
```



Instructions (Part 3)

- Extend the config file with a Docker push command:

```
- docker push YOUR-DOCKERHUB-ACCOUNT/demo:latest
- docker push YOUR-DOCKERHUB-ACCOUNT/demo:$GIT_SHA
```

- Finally, trigger a build by a code change
- Watch as your tests are being executed, the artoifact is being built and pushed to Dockerhub -- can you find it?

- pull it and run your image
  > `docker image pull ...`

Instructions (Part 4)

- Integrate the vulnerability scanner trivy into your pipeline
  - see https://github.com/aquasecurity/trivy-action
  - make sure the scanner is being executed upon every build
  - it should scan the image and also Code and IaC fragments (in our case the Dockerfile)
- configure a quality gate that acts upon the severity levels (i.e. CRITICAL, HIGH)

- submit a protocol (including screenshots and general documentation) and the link to your Git-repo via E-Learning no later than
  - **Tuesday, May 9th, 2022**

UNIVERSITY
OF APPLIED SCIENCES
UPPER AUSTRIA

Continuous Delivery in agile Software Development

Exercise 04 (accompanying Chapter „Continuous Deployment“)

FH-Prof. DI Dr. Marc Kurz

HAGENBERG | LINZ | STEYR | WELS