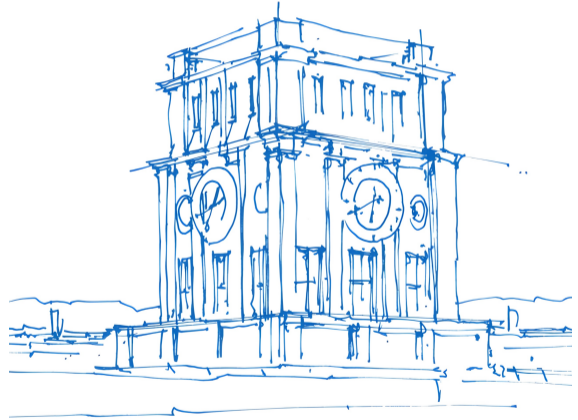TUM

Open Source Lab

CI/CD

Fabian Sauter, Christian Menges, Alexander Stephan

Chair of Connected Mobility
TUM School of Computation, Information and Technology
Technical University of Munich

Garching, November 23, 2022

Uhrenturm der TUM

CI / CD

Continuous integration is a DevOps software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. Continuous integration [. . .] entails both an automation component (e.g. a CI or build service) and a cultural component (e.g. learning to integrate frequently). The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

https://aws.amazon.com/devops/continuous-integration/

Prerequisite: being able to define a consistent environment to ensure that programs not only work on the machine of the developer, but also on the testing server and at the customer.

Solution: Virtualization

Virtualization

■ Virtual machine

Pro:
  ☐ Works on every machine with a hypervisor
  ☐ Very secure

Con:
  ☐ Huge Overhead: Size and performance

For most applications, it is irrelevant what the OS kernel does internally, it is only relevant what the OS lets the application see and do.

Virtualization

- Virtual machine

    Pro:

    - ☐ Works on every machine with a hypervisor
    - ☐ Very secure

    Con:

    - ☐ Huge Overhead: Size and performance

For most applications, it is irrelevant what the OS kernel does internally, it is only relevant what the OS lets the application see and do.

Idea: Use the same kernel, but provide different resources (file system tree, networks, ...) to different processes.

- Containers

    Pro:

    - ☐ Lightweight: small + fast

    Con:

    - ☐ A bug in the kernel can compromise the whole system
    - ☐ Requires different image variants for different architectures (x86, arm, ...)

Image Definition of environment and data (binaries, etc.)
- Consists of layers. This allows to avoid rebuilding the whole image by caching layers. Thus, order the layers from stable to frequently changing.

Container Running image

Registry Image storage (e.g. dockerhub)

- Most famous container platform

- Docker daemon, *dockerd*, manages Containers

- Website: https://www.docker.com/

Logo: https://www.docker.com/sites/default/files/d8/2019-07/Moby-logo.png

Most important commands:

FROM	Specifies a base image.
RUN	Runs a command.
COPY	Copies data from the host or another container into the image.
ADD	Similar to copy, but allows extraction of archives and URLs as source.
ENV	Sets one or more environment variables (format: `key=value`).
USER	Changes the user (Does not create a new user).
ENTRYPOINT	Specifies a command which is always invoked during container start (cannot get overridden via CLI).
CMD	Specifies a default command, which is invoked during container start (can get overridden via CLI). In conjunction with ENTRYPOINT, the specified commands are interpreted as arguments of ENTRYPOINT.
...	

Important: The default user of a Docker image is root. Use the USER command to use a less privileged user.

RUN, ENTRYPOINT and CMD allow to specify a command as string, which is interpreted in a shell, or as string array `["command", "param_1", ...]`, which will invoke the command directly.

Dockerfile reference: https://docs.docker.com/engine/reference/builder/

```
FROM alpine:latest

RUN apk add --no-cache htop

ENTRYPOINT ["htop"]
```

Selection of base image is crucial for the image size (alpine ⇒ small, Ubuntu/Debian ⇒ big)

Cleanup installation artifacts to reduce image size

Dockerfile linter: Hadolint

Build & Run Image

Build image:

```
docker build -t <tag> <path to dir with Dockerfile>
```

Start a new container based on an image specified by tag:

```
docker run <tag>
```

Run image in interactive mode (-it) and mount <host dir> to <container dir> (---rm removes the container after execution (not required)):

```
docker run --rm -it -v <host dir>:<container dir> <tagname>
```

Example: Run image libfuse and mount current directory ($(pwd)) to /libfuse:

```
docker run --rm -it -v $(pwd):/libfuse libfuse
```

Docker Cheat Sheet

Show logs (stdout/stderr) of container:

```
docker logs <container id>
```

Execute command in container

```
docker exec <container id> <command>
docker exec -ti <container id> <command> // interactive, useful for sh, bash, etc.
```

Remove terminated container

```
docker rm <container id>
```

Remove image

```
docker rmi <image id>
```

Copy files between host and container FS:

```
docker cp <path> <container id>:<path> // host -> container
docker cp <container id>:<path> <path> // container -> host
```

Practice

1. Clone https://github.com/libcpr/example-cmake-conan

2. Write a Dockerfile, which compiles the executable and runs the program on container start.

Hints:

- Build instructions can be found in README.md

- Use `gcc:11.3.0-bullseye` as base image

- To install the necessary dependencies, the following commands are required:

```
RUN echo "deb http://deb.debian.org/debian bullseye-backports main" >> /etc/apt/sources.list
RUN apt-get update && apt-get install -y cmake/bullseye-backports python3 python3-pip && pip3
    install conan
```

5 Minutes

Fabian Sauter, Christian Menges, Alexander Stephan | Open Source Lab | opensourcelab@cm.in.tum.de | https://zulip.in.tum.de/ (#Open Source Lab SS23)　　10

Remarks

In case multiple containers should be run together, use docker-compose. E.g. Webserver + Database

For production use, Docker is mostly not sufficient. Consider using Kubernetes https://kubernetes.io/.

If you want to learn more about containerized environments, check "Skalierbare Container-Infrastrukturen" by Oliver Liebel (in German, 3., aktualisierte Auflage 2020, gebunden Rheinwerk Computing, ISBN 978-3-8362-7772-3, available in the library). 1260 pages.

Future of Containers unknown. Although it is in widespread use, it has significant drawbacks such as potentially huge image sizes, a universal kernel and a big attack surface. Maybe Library OSs will be the future of containerized execution.

CI/CD

ПШП

- ALWAYS perform changes to CI/CD scripts in a separate branch, because very often multiple attempts are needed to get it running.

- NEVER hardcode secrets or passwords into your scripts. Inject them using the surrounding system.

- For any project which should last longer, set it up properly. This will safe you a lot of time later.

- Carefully choose a system, since they are barely interchangeable.

Dependabot

Example configuration file for Rust, `/.github/dependabot.yml`:

```yaml
version: 2
updates:
    - package-ecosystem: "cargo"  # Enable crate version updates for the main crate
      directory: "/"   # Look 'Cargo.toml' in the repository root
      schedule:    # Check for updates every day (weekdays)
          interval: "daily"
    - package-ecosystem: "github-actions"   # Enable version updates for Github Actions
      directory: "/"   # Set to '/' to check the Actions used in '.github/workflows'
      schedule:    # Check for updates every day (weekdays)
          interval: "daily"
```

See
https://docs.github.com/en/free-pro-team@latest/github/administering-a-repository/enabling-and-disabling-version-updates
for details.

Source:
https://github.com/
Garfield96/pack/pull/3

Workflow file *.yml:
always located in:
/.github/workflows/

```
1   name: CI
2
3   on: [push, pull_request]
4
5   jobs:
6     build:
7       runs-on: ubuntu-latest
8       container: texlive/texlive:latest
9
10      steps:
11      - uses: actions/checkout@v2
12
13      - name: Build pdf
14        run: make pdf
15
16      - name: Upload pdf
17        uses: actions/upload-artifact@v2
18        with:
19          name: thesis
20          path: build/main.pdf
21          retention-days: 14
```

Source: https://github.com/TUM-Dev/tum-thesis-latex/blob/master/.github/workflows/github-actions-demo.yml

GitLab CI

.gitlab-ci.yml:
always located in:
/ (project root)

```yaml
1   default:
2       image:
3           name: texlive/texlive:latest
4       tags:
5       - Docker
6
7   stages:
8   - build
9
10  pdf:
11      stage: build
12      script:
13      - make all
14      artifacts:
15          paths:
16          - main.pdf
17          expire_in: 1 week
```

Source: https://gitlab.lrz.de/gbs-cm/skript/-/blob/master/.gitlab-ci.yml

Jenkins

- Automation server
- Jobs are defined using a DSL inside a `Jenkinsfile`
- MIT License
- www.jenkins.io
- Repository: https://github.com/jenkinsci/jenkins

Logo: https://www.jenkins.io/images/logo.png