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

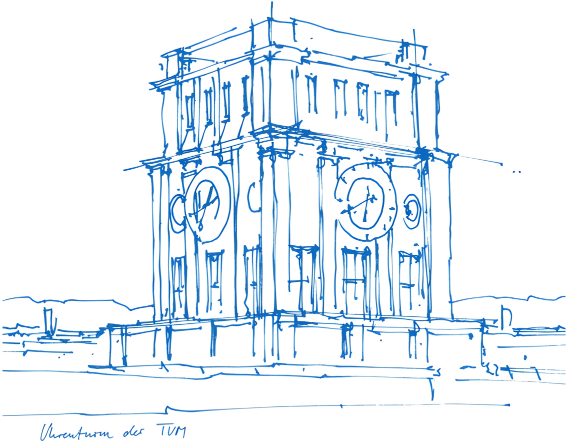
Open Source Lab

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 6, 2024



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, ...)

Terminology

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)

Docker

- 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

Dockerfile

Most important commands:

FROM Specifies a base image (use `scratch` if no base image is required).

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.

CMD Specifies a default command, which is invoked during container start. 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/

Dockerfile - Example

```
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 - Open Source Lab Dice

1. Clone https://gitlab.lrz.de/open-source-lab/dice
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 `golang:1.23-alpine3.20` as base image
- Expose container port using `-p 8080:8080` with `docker run`

5 Minutes

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/)

Besides Docker, there are many other projects to create, manage and run containers. E.g. podman (https://podman.io/)

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

- 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.

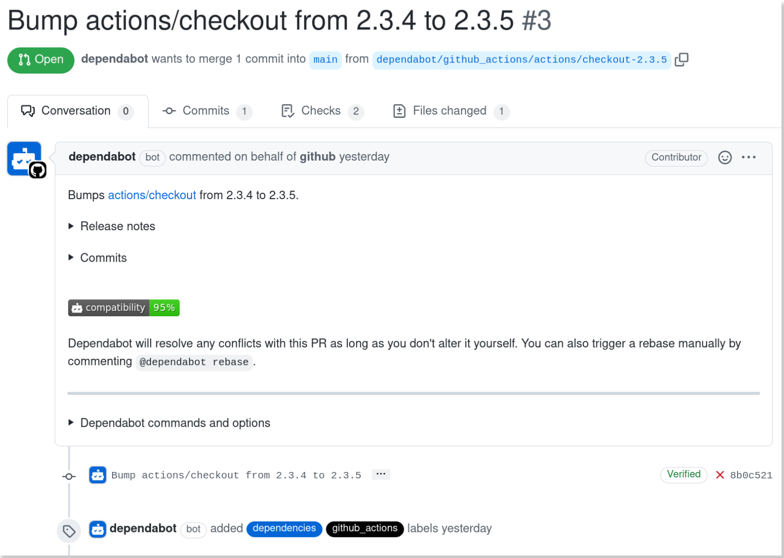
Dependabot

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.

Alternative: Renovate https://github.com/renovatebot/renovate



Bump actions/checkout from 2.3.4 to 2.3.5 #3

Open **dependabot** wants to merge 1 commit into `main` from `dependabot/github_actions/actions/checkout-2.3.5`

dependabot bot commented on behalf of **github** yesterday

Bumps actions/checkout from 2.3.4 to 2.3.5.

▸ Release notes

▸ Commits

Dependabot will resolve any conflicts with this PR as long as you don't alter it yourself. You can also trigger a rebase manually by commenting `@dependabot rebase`.

▸ Dependabot commands and options

```
Bump actions/checkout from 2.3.4 to 2.3.5
```

dependabot bot added dependencies github_actions labels yesterday

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

Github CI

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

Can be executed locally using
https://github.com/nektos/act

```yaml
name: CI

on: [push, pull_request]

jobs:
  build:
    runs-on: ubuntu-latest
    container: texlive/texlive:latest

    steps:
    - uses: actions/checkout@v4

    - name: Build pdf
      run: make pdf

    - name: Upload pdf
      uses: actions/upload-artifact@v3
      with:
        name: thesis
        path: build/main.pdf
        if-no-files-found: error
        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
default:
   image:
      name: texlive/texlive:latest
   tags:
   - Docker

stages:
- build

pdf:
   stage: build
   script:
   - make all
   artifacts:
      paths:
      - main.pdf
      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