



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Kapitel 02

AUTOMATISIERUNG UND CI/CD

Wo wir jetzt stehen

Lokal können Sie das bereits:

```
uv run ruff check .  
uv run ruff format .  
uv run pyright  
git push    # PR öffnen
```

Aber was passiert, wenn ...

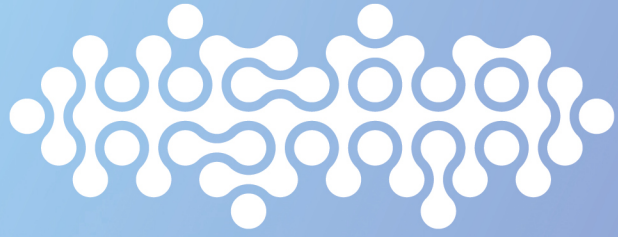
- ein Teammitglied vor dem Push die Prüfungen vergisst?
- ein Mitwirkender Ihre Qualitätsstandards gar nicht kennt?
- Sie wollen, dass **jeder PR** Ihren Standard erfüllt — automatisch?

Antwort: Continuous Integration / Continuous Deployment.

Lernziele

Am Ende dieser Vorlesung können Sie:

1. **CI/CD erklären** und begründen, warum es wichtig ist
2. Einen **GitHub-Actions-Workflow** lesen und schreiben
3. **Automatisierte Qualitätsgates** für jeden PR aufsetzen
4. **Pre-commit-Hooks** für sofortiges Feedback einrichten
5. **Copilot** zur Beschleunigung der Workflow-Erstellung nutzen
6. **Branch Protection** so konfigurieren, dass Qualität erzwungen wird



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Was ist CI/CD?

1

"Bei mir läuft es."

Dev A — Python 3.12

```
def strip_prefix(items):  
    return [  
        x.removeprefix("data_")  
        for x in items  
    ]
```

✅ Lokal alles grün.

Dev B — Python 3.8

```
AttributeError:  
'str' object has no  
attribute 'removeprefix'
```

❌ Beim Mergen kracht es.

Beide hielten sich für fertig. Erst der Merge zeigt die Wahrheit.

Continuous Integration (CI)

Fünf Prinzipien:

1. **Häufige Integration** — täglich in den gemeinsamen Branch
2. **Automatisierter Build** — jeder Push triggert einen Build
3. **Automatisierte Tests** — der Build führt die Tests aus
4. **Schnelles Feedback** — Minuten, nicht Tage
5. **Geteilte Verantwortung** — Build kaputt = Team-Problem

push → install → lint → test →  Merge erlaubt /  blockiert

Continuous Delivery vs. Deployment

Continuous Delivery

- Pipeline erzeugt ein release-fähiges Artefakt
- **Manuelle Freigabe** vor Produktion
- Üblich in regulierten Umgebungen

Continuous Deployment

- Jeder grüne Build → **Produktion**
- Keine manuellen Schritte
- Setzt starke Tests + Monitoring voraus

Gleiche Pipeline. Unterschiedliches Vertrauen in die Tests.

Warum überhaupt? Die Zahlen

Kennzahl	Ohne CI/CD	Mit CI/CD
Zeit bis ein Bug entdeckt wird	Tage–Wochen	Minuten
Deployment-Frequenz	wöchentlich / monatlich	mehrmals täglich
Fehlgeschlagene Deployments	20–30 %	< 5 %
Zeit bis Produktion repariert	Stunden–Tage	Minuten

Bug-Kosten: 1× in Dev → 10× in QA → 100× in Staging → **1000× in Produktion.**
CI fängt Probleme, solange sie noch billig sind.

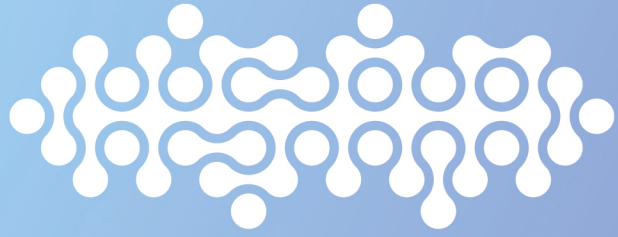


Diskussion (2 Min)

Hand hoch: Wer hat schon einmal Zeit an einen *"Bei mir läuft es"*-Bug verloren?

Tauschen Sie sich zu zweit aus:

- Was war kaputt?
- Wer hat debuggt?
- Wie lange hat es bis zur Lösung gedauert?
- Hätte CI das gefangen?



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

GitHub Actions

2

Das Vokabular

- **Workflow** — YAML-Datei in `.github/workflows/`
- **Event** — was den Workflow triggert (`push`, `pull_request`, ...)
- **Job** — Schrittfolge auf einem Runner
- **Step** — eine einzelne Aufgabe (Befehl oder Action)
- **Action** — wiederverwendbare Einheit (wie eine Funktion)
- **Runner** — die VM, die alles ausführt

Mentales Modell: *Bei Event X laufen Jobs, die Steps ausführen, die Actions auf Runnern aufrufen.*

Ein minimaler Workflow

```
name: Mein erster Workflow
on: [push, pull_request]
jobs:
  hello:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: echo "Hallo Welt!"
```

Mehr nicht — als `.github/workflows/hello.yml` committen, und GitHub führt das bei jedem Push und PR aus.

Marketplace: Wem vertrauen Sie?

✓ Grüne Flaggen

- `actions/*` — offiziell von GitHub
- ✓ Verifizierter Anbieter (Docker, AWS, Astral, ...)
- 1 000+ Sterne, breit eingesetzt
- Update in den letzten ~6 Monaten

✗ Rote Flaggen

- Mehr als ein Jahr nicht aktualisiert
- < 100 Sterne, kaum genutzt
- Keine README / keine Beispiele
- Stapel ungelöster Issues

Faustregel: offiziell → verifiziert → Community (und vorher in den Quellcode schauen).

Versionen pinnen

Form	Beispiel	Verhalten	Wann nutzen?
Major	@v4	folgt der neuesten 4.x.x	Standard — Patches automatisch
Exakt	@v4.1.1	eingefroren	Reproduzierbarkeit, Compliance
Branch	@main	jeder Commit	nur zum Testen
SHA	@8ade135...	eingefroren	sicherheitskritisch

@v4 liefert kostenlose Sicherheitspatches. @main liefert kostenlose Überraschungen.

Actions, die Sie dieses Semester sehen

- uses: actions/checkout@v4 # Repo klonen
- uses: astral-sh/setup-uv@v4 # uv installieren (Python + Deps)
- uses: actions/cache@v4 # Dependencies cachen
- uses: actions/upload-artifact@v4 # Build-Outputs sichern

Warum `astral-sh/setup-uv` statt `actions/setup-python` ?

uv installiert Python *und* löst Abhängigkeiten auf — eine Action, schneller, identisch zum lokalen Workflow.

Häufige Trigger

```
on:  
  push:  
    branches: [main, develop]  
  pull_request:  
    branches: [main]
```

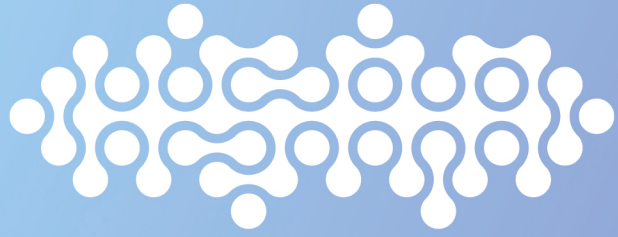
```
on:  
  schedule:  
    - cron: '0 0 * * *'  
  workflow_dispatch: # "Run workflow"
```

`pull_request` ist das Arbeitspferd — genau hier gehören Qualitätsgates hin.

Preise auf einer Folie

- **Öffentliche Repos** →  unbegrenzt kostenlos (Linux / Win / macOS)
- **Private Repos** → 2 000 Freiminuten/Monat, danach kostenpflichtig
- Linux **1×**, Windows **2×**, macOS **10×** Multiplikator

Für diesen Kurs: halten Sie Ihr Projekt-Repo **öffentlich**. CI ist gratis, und Ihr Portfolio profitiert.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Ihr erster Workflow

3

.github/workflows/quality.yml

```
name: Code Quality
on:
  pull_request:
    branches: [main]
jobs:
  quality:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: astral-sh/setup-uv@v4
      - run: uv sync --dev
      - run: uv run ruff check .
      - run: uv run ruff format --check .
      - run: uv run pyright
```

Drei Qualitätsgates bei jedem PR. Grün = mergebar. Rot = blockiert.

Warum Caching wichtig ist

Erster Lauf

Deps installieren ... 30-45 s

Mit Cache (Lock unverändert)

Cache wiederherstellen ... 3-5 s

Der Trick: der Cache-Key ist ein Hash über `uv.lock`.

- Lock ändert sich → Key ändert sich → frische Installation
- Lock unverändert → Cache-Hit → Download entfällt

`uv.lock` ist kein Müll — es macht CI schnell und reproduzierbar.



Praktische Übung (5 Min)

Öffnen Sie Ihr Road-Profile-Viewer-Repo und:

1. Legen Sie `.github/workflows/quality.yml` an
2. Fügen Sie den Workflow von der vorigen Folie ein
3. Committen Sie auf einem Branch, pushen Sie, öffnen Sie einen PR
4. Beobachten Sie den **Actions**-Tab im Browser

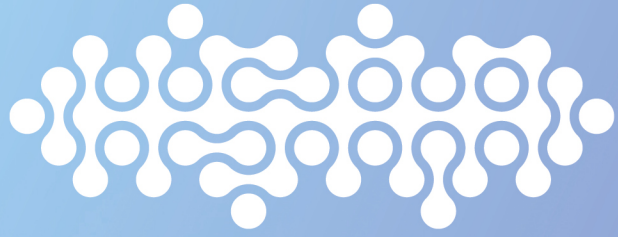
Vergleichen Sie mit Ihrer Sitznachbarin: ist es grün? Wenn nicht — was steht im Log?

Wenn eine Prüfung fehlschlägt

```
Run uv run ruff check .  
src/main.py:16:1: E401 Multiple imports on one line  
src/main.py:29:5: E225 Missing whitespace around operator  
Found 2 errors.  
Error: Process completed with exit code 1.
```

Schleife: ✗ → lokal fixen → push → ✓ → Merge.

GitHub blockiert den Merge, bis die Schleife sich schließt.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Fortgeschrittene Muster

4

Parallele Jobs

```
jobs:  
  lint:    { ... }  
  format:  { ... }  
  typecheck:{ ... }
```

- ✓ Alle drei laufen **gleichzeitig**
- ✓ Fehler zeigt das genaue Gate

⚠ Verdreifacht die Runner-Minuten

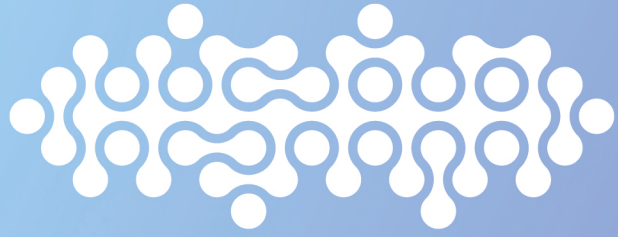
⚠ Setup-Schritte wiederholen sich

Trade-off: Geschwindigkeit & Klarheit gegen Minuten. Bei öffentlichen Repos: Geschwindigkeit gewinnt.

Matrix-Tests

```
strategy:
  matrix:
    python-version: ['3.12', '3.13']
steps:
  - uses: actions/checkout@v4
  - uses: astral-sh/setup-uv@v4
  - run: uv python pin ${ matrix.python-version }
  - run: uv sync --dev
  - run: uv run pytest
```

Zwei parallele Jobs aus einer Konfiguration. Fängt den `removeprefix` -Bug von Folie 1, bevor er die Nutzer trifft.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Copilot für Workflows

5

Erst manuell, dann KI

Sie haben gelernt, YAML zu **lesen** — damit können Sie:

-  verifizieren, was Copilot generiert
-  **präzise** Prompts schreiben (mit dem richtigen Vokabular)
-  Fehler debuggen, wenn sie auftreten
-  gute Vorschläge von selbstbewusst klingenden falschen unterscheiden

KI verstärkt Expertise; sie ersetzt sie nicht.

Vage vs. präzise Prompts

✗ Vage

"Erstelle einen GitHub-Actions-Workflow für mein Python-Projekt."

Sie bekommen *irgendetwas*. Vielleicht `setup-python`. Vielleicht `pip`. Vielleicht `@v2`.

✓ Präzise

Trigger bei PRs auf `main`. Nutze `astral-sh/setup-uv@v4`. Führe `ruff check`, `ruff format --check`, `pyright` aus. Cache `~/.cache/uv` mit `uv.lock` als Schlüssel.

Sie bekommen genau das.

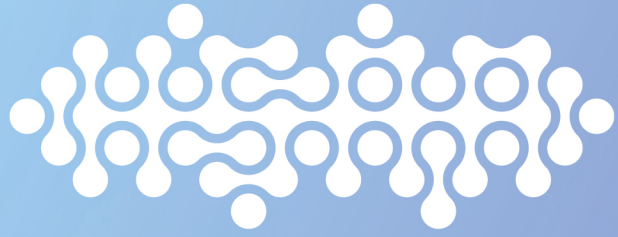
Mit Copilot arbeiten — Dos und Don'ts

✓ So

- Präzise Prompts mit korrekten Begriffen
- **Jede** generierte Zeile prüfen
- Action-Versionen im Marketplace verifizieren
- Iterieren — Prompt verfeinern
- Copilot bitten, Unklares zu **erklären**

✗ Nicht so

- Ungelesenes YAML übernehmen
- Grundlagen überspringen
- Veraltete @v2 -Pins blind akzeptieren
- Sicherheitsfragen ignorieren
- Beim ersten Vorschlag stehen bleiben



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Pre-commit-Hooks

6

Warum Pre-commit?

Ohne Hooks

```
commit → push → CI (2 Min) → ✗  
fix    → push → CI (2 Min) → ✓
```

4+ Minuten pro Tippfehler.

Mit Hooks

```
commit → Hook (3 s) → ✗  
fix    → commit → Hook (3 s) → ✓
```

Sekunden. Schlechte Commits verlassen den Laptop nie.

.pre-commit-config.yaml

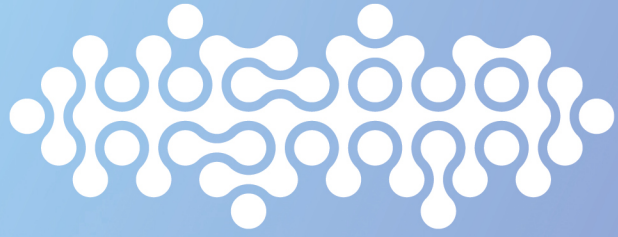
```
repos:
  - repo: https://github.com/astral-sh/ruff-pre-commit
    rev: v0.8.0
    hooks:
      - id: ruff
        args: [--fix]
      - id: ruff-format
  - repo: https://github.com/RobertCraigie/pyright-python
    rev: v1.1.388
    hooks:
      - id: pyright
```

```
uv tool install pre-commit && pre-commit install
```

Ein Trichter aus Qualitätsgates

Ebene	Werkzeug	Wann	Tempo	Fängt
IDE	Pylance, Ruff-Ext.	beim Tippen	sofort	Tippfehler, Typen
Pre-commit	Hooks	vor dem Commit	< 10 s	Stil, Lint
CI	GitHub Actions	nach dem Push	1–3 Min	Umgebungs-Bugs
Review	Menschen	vor dem Merge	Stunden	Logik, Design

99 % werden in der IDE gefangen. Jede Ebene fängt, was die darüber verpasst hat.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Branch Protection

7

`main` un-pushbar machen

Repository → Settings → Branches → Regel für `main`:

- ☒ Pull Request vor Merge erforderlich
- ☒ **Mindestens eine Genehmigung** erforderlich
- ☒ Veraltete Genehmigungen bei neuen Commits verwerfen
- ☒ Status-Checks erforderlich: `quality` muss grün sein
- ☒ Branch muss aktuell sein
- ☒ Conversations müssen aufgelöst sein

Nach dieser Regel liefert `git push origin main`:

`error: GH006: Protected branch update failed`. Gut so.



Diskussion (3 Min)

Sollten Administratoren von Branch Protection ausgenommen sein?

- Pro: Hotfix ohne Zeremonie, "ich weiß, was ich tue"
- Kontra: der Schutz schützt nur das, was er auch schützen darf

Argumentieren Sie beide Seiten. Und im Repo Ihres Teams: sind Admins ausgenommen oder nicht? Warum?

Mitnehmen

1. **CI/CD ist nicht optional** — moderne Teams nutzen es
2. **Schichten Sie die Gates** — IDE → Pre-commit → CI → Review
3. **Automatisieren Sie alles** — *"bitte daran denken"* skaliert nicht
4. **GitHub Actions ist gratis** für öffentliche Repos — keine Ausrede
5. **KI verstärkt Expertise** — erst Grundlagen, dann Copilot

Die Vorlesung deckt mehr ab: Custom Actions, Secrets, Deployment-Pipelines, Notifications, Status-Badges. Folgen Sie den Section-Links oben.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Fragen?

VIELEN DANK