



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Chapter 02

AUTOMATION AND CI/CD

Where We Are Now

You can already do this locally:

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

But what happens when...

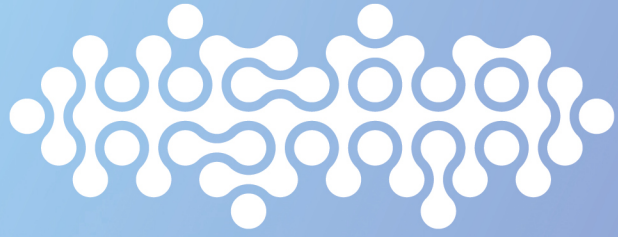
- a teammate forgets to run the checks before pushing?
- a contributor doesn't know your quality bar exists?
- you want **every PR** to meet that bar — automatically?

Answer: Continuous Integration / Continuous Deployment.

Learning Objectives

By the end of today you will be able to:

1. Explain **what CI/CD is** and why it matters
2. Read and write a **GitHub Actions workflow**
3. Build automated **quality gates** for every PR
4. Add **pre-commit hooks** for instant local feedback
5. Use **Copilot** to accelerate workflow authoring
6. Wire up **branch protection** so quality is enforced, not requested



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

What is CI/CD?

1

"It works on my machine."

Dev A — Python 3.12

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

✓ Works locally.

Dev B — Python 3.8

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

✗ Breaks on integration.

Both devs were "done." Only the merge surfaces the truth.

Continuous Integration (CI)

Five principles:

1. **Frequent integration** — merge to shared branch daily
2. **Automated build** — every push triggers a build
3. **Automated testing** — build runs the test suite
4. **Fast feedback** — minutes, not days
5. **Shared responsibility** — broken build = team's problem

```
push → install → lint → test →  allow merge /  block
```

Continuous Delivery vs. Deployment

Continuous Delivery

- Pipeline produces a release-ready artifact
- **Human approval** before production
- Common in regulated / enterprise

Continuous Deployment

- Every passing build → **production**
- Zero manual steps
- Requires strong tests + monitoring

Same plumbing. Different trust level in the tests.

Why bother? The numbers

Metric	Without CI/CD	With CI/CD
Time to detect a bug	days–weeks	minutes
Deployment frequency	weekly / monthly	multiple per day
Failed deployments	20–30%	< 5%
Time to fix production	hours–days	minutes

Bug-fix cost: 1× in dev → 10× in QA → 100× in staging → **1000× in production.**
CI catches things while they're still cheap.

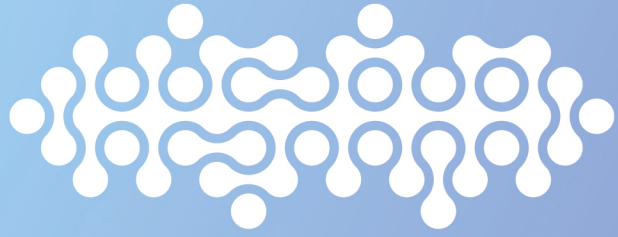


Discussion (2 min)

Show of hands: who has lost time to a *"works on my machine"* bug?

Pair up and trade stories:

- What broke?
- Who debugged it?
- How long did it take to find?
- Would CI have caught it?



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

GitHub Actions

2

The vocabulary

- **Workflow** — YAML file in `.github/workflows/`
- **Event** — what triggers it (`push`, `pull_request`, ...)
- **Job** — set of steps on one runner
- **Step** — a single task (a command or an action)
- **Action** — a reusable unit (like a function)
- **Runner** — the VM that executes everything

Mental model: *On event X, run jobs that run steps that call actions on runners.*

A minimal workflow

```
name: My First Workflow
on: [push, pull_request]
jobs:
  hello:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: echo "Hello, World!"
```

That's it — commit this as `.github/workflows/hello.yml` and GitHub runs it on every push and PR.

Marketplace: who do you trust?

✓ Green flags

- `actions/*` — GitHub official
- ✓ Verified creator (Docker, AWS, Astral, ...)
- 1 000+ stars, used by many repos
- Updated within ~6 months

✗ Red flags

- No update in over a year
- < 100 stars, no usage
- No README / examples
- Pile of unanswered issues

Rule of thumb: official → verified → community (and only after you've read the source).

Pinning a version

Form	Example	Behaviour	Use when...
Major	@v4	tracks latest 4.x.x	default — patches & features auto
Exact	@v4.1.1	frozen	reproducibility, compliance
Branch	@main	every commit	only for testing
SHA	@8ade135...	frozen	security-critical

@v4 gives you free security patches. @main gives you free surprises.

Actions you'll meet this term

- uses: actions/checkout@v4 # clone the repo
- uses: astral-sh/setup-uv@v4 # install uv (Python + deps)
- uses: actions/cache@v4 # cache deps for speed
- uses: actions/upload-artifact@v4 # save build outputs

Why `astral-sh/setup-uv` **instead of** `actions/setup-python` ?

uv installs Python *and* resolves deps — one action, faster, matches what we run locally.

Common triggers

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

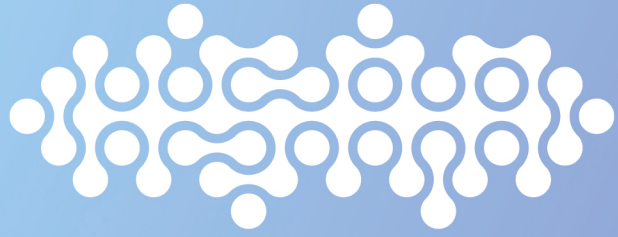
```
on:  
  schedule:  
    - cron: '0 0 * * *'  
  workflow_dispatch: # "Run workflow"
```

`pull_request` is the workhorse: it's where quality gates belong.

Pricing in one slide

- **Public repos** →  unlimited free minutes (Linux / Win / macOS)
- **Private repos** → 2 000 free minutes/month, then paid
- Linux **1×**, Windows **2×**, macOS **10×** multiplier

For this course: keep your project repo **public**. CI is free, and your portfolio benefits.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Your First 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
```

Three quality gates run on every PR. Green = mergeable. Red = blocked.

Why caching matters

First run

```
install deps ... 30-45 s
```

Cached run (lock unchanged)

```
restore cache ... 3-5 s
```

The trick: the cache key is a hash of `uv.lock`.

- Lock changes → key changes → fresh install
- Lock unchanged → cache hit → skip download

`uv.lock` isn't clutter; it's how CI stays fast and reproducible.



Hands-on (5 min)

Open your Road Profile Viewer repo and:

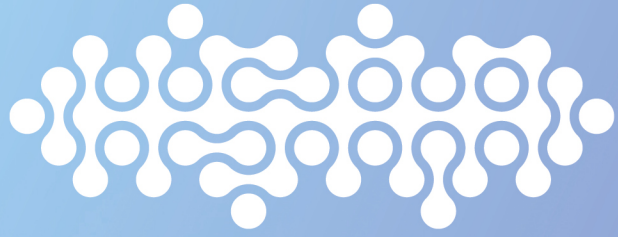
1. Create `.github/workflows/quality.yml`
2. Paste the workflow from the previous slide
3. Commit on a branch, push, open a PR
4. Watch the **Actions** tab in your browser

Compare with your neighbour: did it pass? If it failed — what did the log show?

When a check fails

```
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.
```

Loop: ❌ → fix locally → push → ✅ → merge.
GitHub blocks the merge until the loop closes.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Advanced Patterns

4

Parallel jobs

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

- ✓ All three run **simultaneously**
- ✓ Failure points to the exact gate

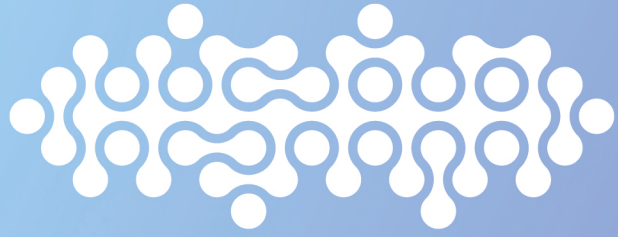
- ⚠ Triples the runner minutes
- ⚠ Setup steps repeat per job

Trade-off: speed and clarity vs. minutes.
For public repos, take the speed.

Matrix testing

```
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
```

Two parallel jobs, one config. Catches the `removeprefix` bug from slide 1 before users do.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Copilot for Workflows

5

Manual first, AI second

You learned to **read** YAML so you can:

-  Verify what Copilot generates
-  Write **specific** prompts (using the right vocabulary)
-  Debug failures when they happen
-  Tell good suggestions from confident-sounding bad ones

AI amplifies expertise; it doesn't replace it.

Vague vs. specific prompts

✗ Vague

"Create a GitHub Actions workflow for my Python project."

You'll get *something*. Maybe `setup-python`. Maybe `pip`. Maybe pinned to `@v2`.

✓ Specific

Trigger on PRs to `main`. Use `astral-sh/setup-uv@v4`. Run `ruff check`, `ruff format --check`, `pyright`. Cache `~/.cache/uv` keyed on `uv.lock`.

You'll get exactly what you wanted.

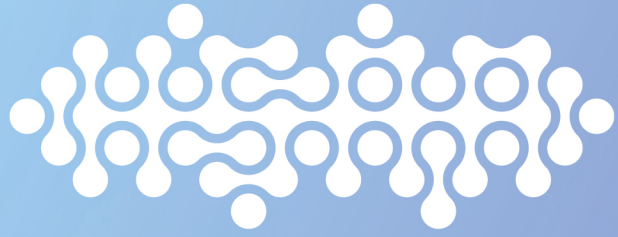
Working with Copilot — DOs and DON'Ts

✓ Do

- Write specific prompts using correct terms
- Review **every** generated line
- Verify action versions on the marketplace
- Iterate — refine the prompt
- Ask Copilot to **explain** unfamiliar bits

✗ Don't

- Trust unread YAML
- Skip the fundamentals
- Accept outdated `@v2` pins blindly
- Ignore security implications
- Stop at the first suggestion



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Pre-commit Hooks

6

Why pre-commit?

Without hooks

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

4+ minutes per typo.

With hooks

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

Seconds. Bad commits never leave your laptop.

.pre-commit-config.yaml

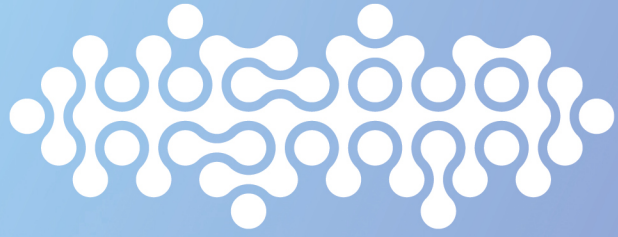
```
repos:
  - repo: https://github.com/astral-sh/ruff-pre-commit
    rev: v0.8.0
    hooks:
      - id: ruff
        args: [--fix]
      - id: ruff-format
  - repo: https://github.com/RobertCraigie/pyright-python
    rev: v1.1.388
    hooks:
      - id: pyright
```

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

A funnel of quality gates

Level	Tool	When	Speed	Catches
IDE	Pylance, Ruff ext.	as you type	instant	typos, types
Pre-commit	hooks	before commit	< 10 s	style, lint
CI	GitHub Actions	after push	1–3 min	env-specific
Review	humans	before merge	hours	logic, design

99 % caught in IDE. Each layer below catches what the layer above missed.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Branch Protection

7

Make `main` un-pushable

Repository → Settings → Branches → Add rule for `main`:

- ☒ Require a pull request before merging
- ☒ Require **1+ approval**
- ☒ Dismiss stale approvals on new commits
- ☒ Require status checks: `quality` must pass
- ☒ Require branches to be up-to-date
- ☒ Require conversation resolution

After this rule, `git push origin main` returns:

```
error: GH006: Protected branch update failed . Good.
```



Discussion (3 min)

Should administrators be exempt from branch protection?

- Pros: hotfix without ceremony, "I know what I'm doing"
- Cons: the protection only protects what it's allowed to

Argue both sides. Then: in your team's repo, are admins included or excluded? Why?

Take-aways

1. **CI/CD is non-optional** — every modern team uses it
2. **Layer your gates** — IDE → pre-commit → CI → review
3. **Automate everything** — *"please remember"* never scales
4. **GitHub Actions is free** for public repos — no excuse
5. **AI amplifies expertise** — fundamentals first, Copilot second

Lecture covers more: custom actions, secrets, deployment pipelines, notifications, status badges. Follow the section links above to dive in.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Questions?

THANK YOU