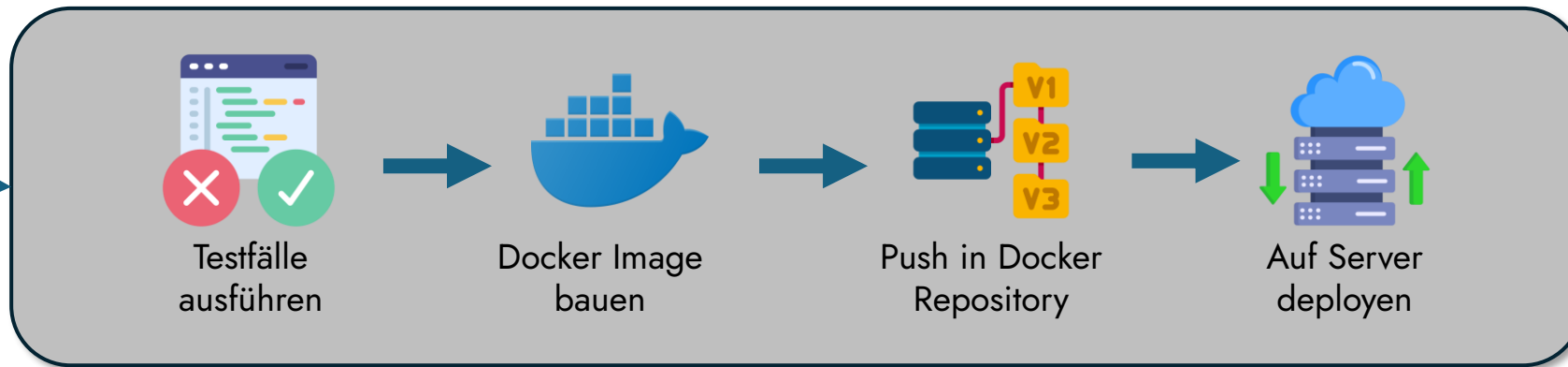


# Continuous Integration und Continuous Deployment

Fortgeschrittene Softwaretechnik WS 2024/25

# Ziel



- Bauen einer grundlegenden CI/CD Pipeline
- Grundlegende Konzepte von GitLab CI/CD

# Warum GitLab

GitLab hat sich in den letzten Jahren immer mehr zu einer kompletten DevOps Plattform entwickelt



Alles an einem Ort

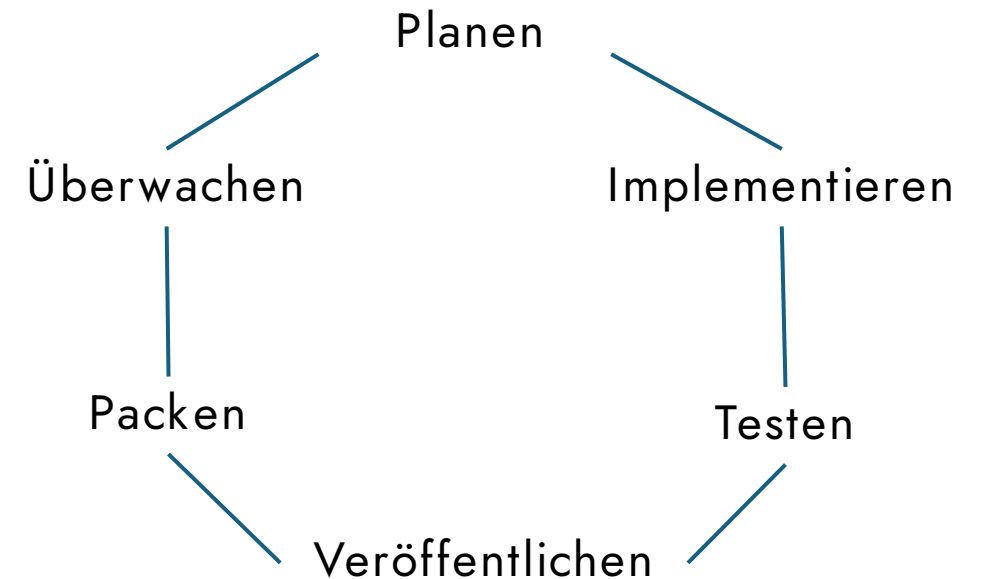
Gehostet von der Uni-Trier



<https://gitlab.uni-trier.de/>

# DevOps Komponenten in GitLab

Planning	Source Code Management	Continuous Integration	Security	Compliance	Artifact Registry	Continuous Delivery	Observability
DevOps Reports DORA Metrics Value Stream Management Value Stream Forecasting Service Desk Wiki Portfolio Management Team Planning Generate issue description Discussion Summary Design Management	Remote Development Source Code Management Web IDE GitLab CLI Code Review Workflow Code Suggestions Code Explanation Code Review Summary Test Generation Code Refactorization GitLab Duo for the CLI	Secrets Management <a href="#">Review Apps</a> Code Testing and Coverage Merge Trains Suggested Reviewers Merge Request Summary Root Cause Analysis Discussion Summary Merge Commit Message Generation Pipeline Composition and Component Catalog	Container Scanning Software Composition Analysis API Security Coverage-guided Fuzz Testing DAST Code Quality Secret Detection SAST Vulnerability Explanation Vulnerability Resolution GitLab Advisory Database	Release Evidence Compliance Management Audit Events Software Bill of Materials Dependency Management Vulnerability Management Security Policy Management	Virtual Registry Container Registry Helm Chart Registry Package Registry Model Registry (Beta) Dependency Proxy	Release Orchestration Infrastructure as Code Pages Feature Flags Environment Management Deployment Management Auto DevOps	On-call Schedule Management Incident Management Error Tracking Product Analytics Visualization AI Product Analytics AI Impact Dashboard Metrics Distributed Tracing Logs
Replacement for Jira	Replacement for GitHub	Replacement for Jenkins	Replacement for Snyk		Replacement for JFrog	Replacement for Harness	Replacement for Sentry



# Erinnerung CI/CD

CI → Continuous Integration

CD → Continuous Deployment oder Continuous Delivery

Automatisches und stetiges...



Ändern



Testen

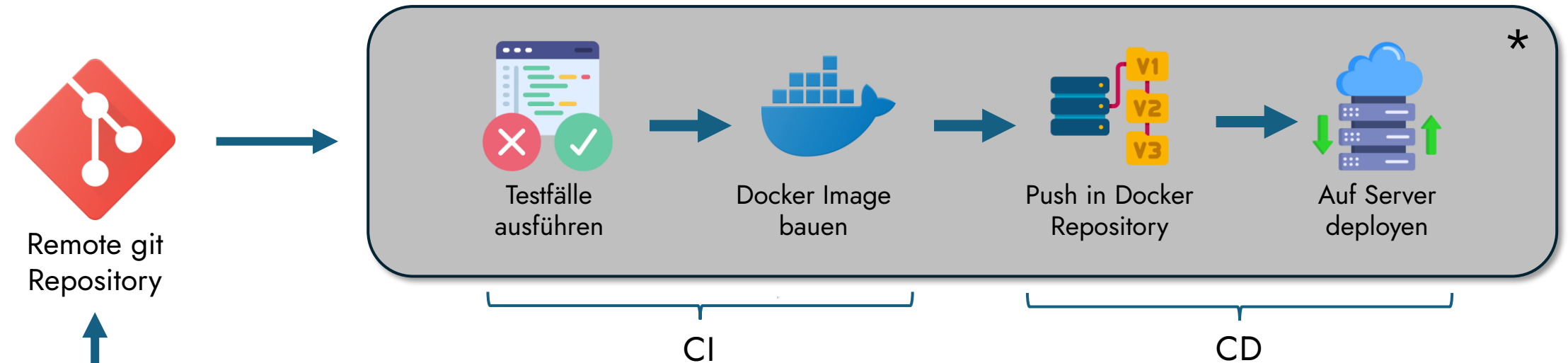


Bauen



Deployen

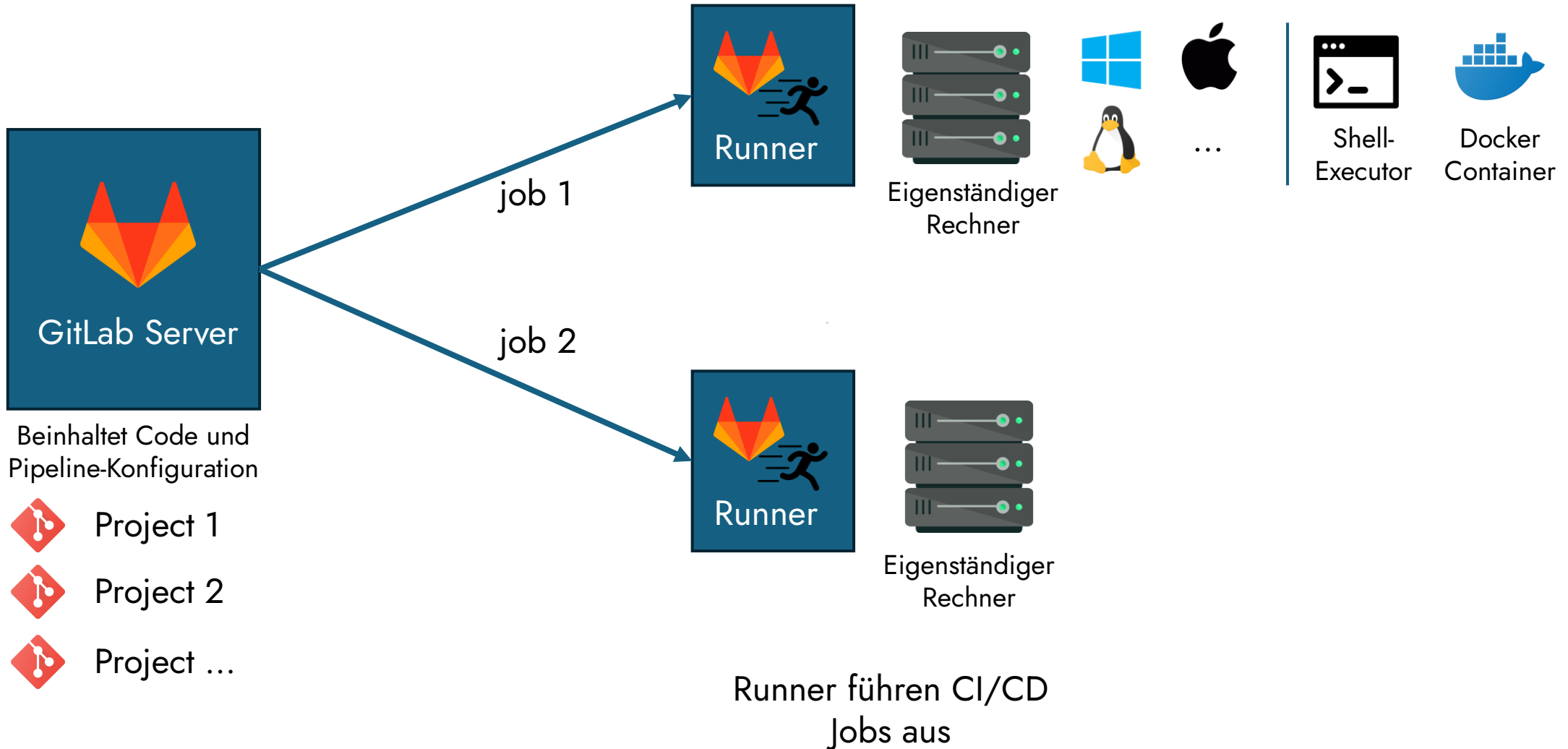
# Ablauf CI/CD



Jede neue Code-Version triggert die definierte CI/CD-Pipeline in GitLab

\* Abhängig von der definierten CI/CD Konfiguration

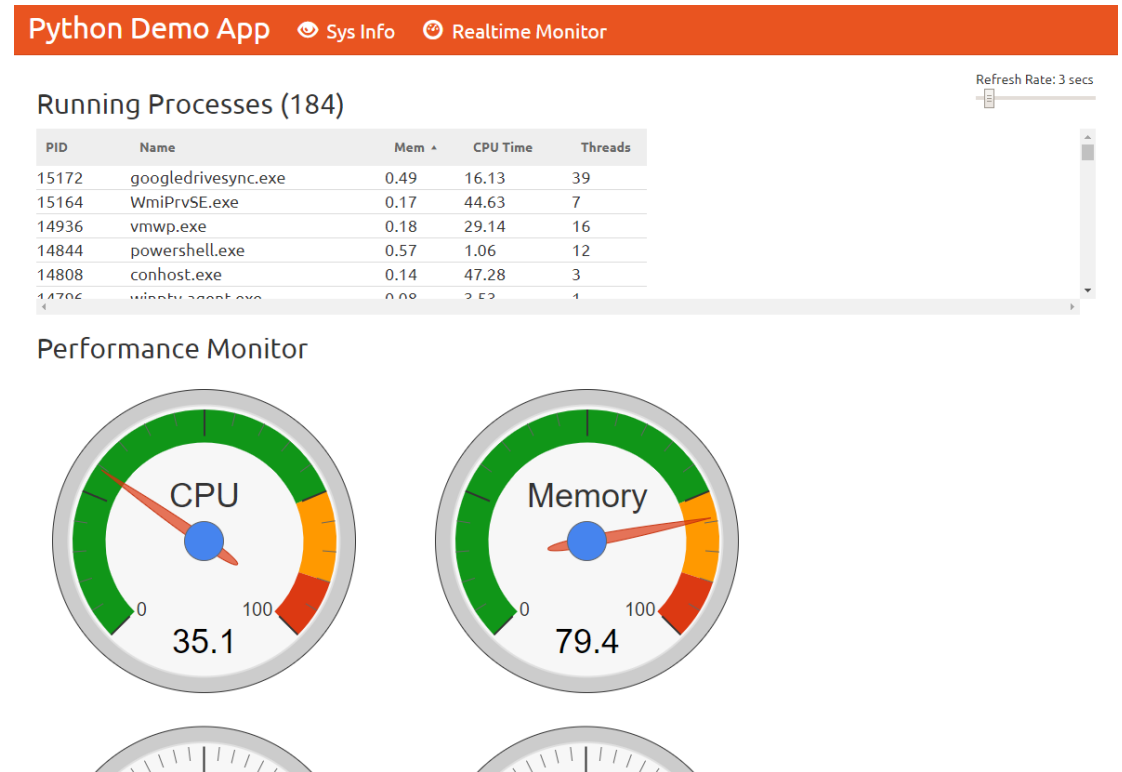
# GitLab Architektur Part 1



# Demo App

Link zum Repo: <https://gitlab.uni-trier.de/kreberl/fst-demo-app>

Kein Verständnis des Python-Codes benötigt,  
aber Grundlagen über die Funktionsweise  
der Verwendeten Tools



Original Repo: <https://github.com/benc-uk/python-demoapp>



# Demo App

Name	Last commit	Last update
📁 .devcontainer	Initial commit	20 minutes ago
📁 .github	Initial commit	20 minutes ago
📁 .vscode	Initial commit	20 minutes ago
📁 build	Initial commit	20 minutes ago
📁 deploy	Initial commit	20 minutes ago
📁 src	Initial commit	20 minutes ago
📁 tests	Initial commit	20 minutes ago
🐋 .dockerignore	Initial commit	20 minutes ago
📄 .flake8	Initial commit	20 minutes ago
🔴 .gitignore	Initial commit	20 minutes ago
🎨 .prettierrc.yaml	Initial commit	20 minutes ago
📖 CONTRIBUTING.md	Initial commit	20 minutes ago
📄 LICENSE	Initial commit	20 minutes ago
📖 README.md	Initial commit	20 minutes ago
⚙️ makefile	Initial commit	20 minutes ago

..

🐋 Dockerfile

Name

..

📁 static

📁 templates

📁 tests

🐍 \_\_init\_\_.py

🐍 apis.py

🐍 conftest.py

🐍 views.py

# Demo App – Lokal ausführen

## 1. Clone dem App von Gitlab

```
lucas@ST-ESPRIMO-P957:~/Dokumente/FST/app/fst-demo-app$ ls
build  CONTRIBUTING.md  deploy  LICENSE  makefile  README.md  src  tests
```

## 2. Tests ausführen

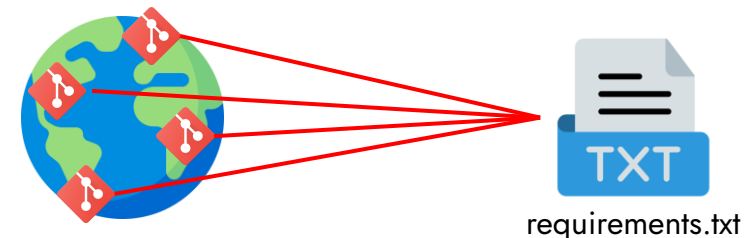
```
$ make test
```

```
lucas@ST-ESPRIMO-P957:~/Dokumente/FST/app/fst-demo-app$ make test
. src/.venv/bin/activate \
&& pytest -v
=====
platform linux -- Python 3.12.3, pytest-6.2.5, py-1.11.0, pluggy-1.5.0
cachedir: .pytest_cache
rootdir: /home/lucas/Dokumente/FST/app/fst-demo-app
collected 5 items

src/app/tests/test_api.py::test_api_process PASSED
src/app/tests/test_api.py::test_api_monitor PASSED
src/app/tests/test_views.py::test_home PASSED
src/app/tests/test_views.py::test_page_content PASSED
src/app/tests/test_views.py::test_info PASSED
```

Verwendet make als Build-System

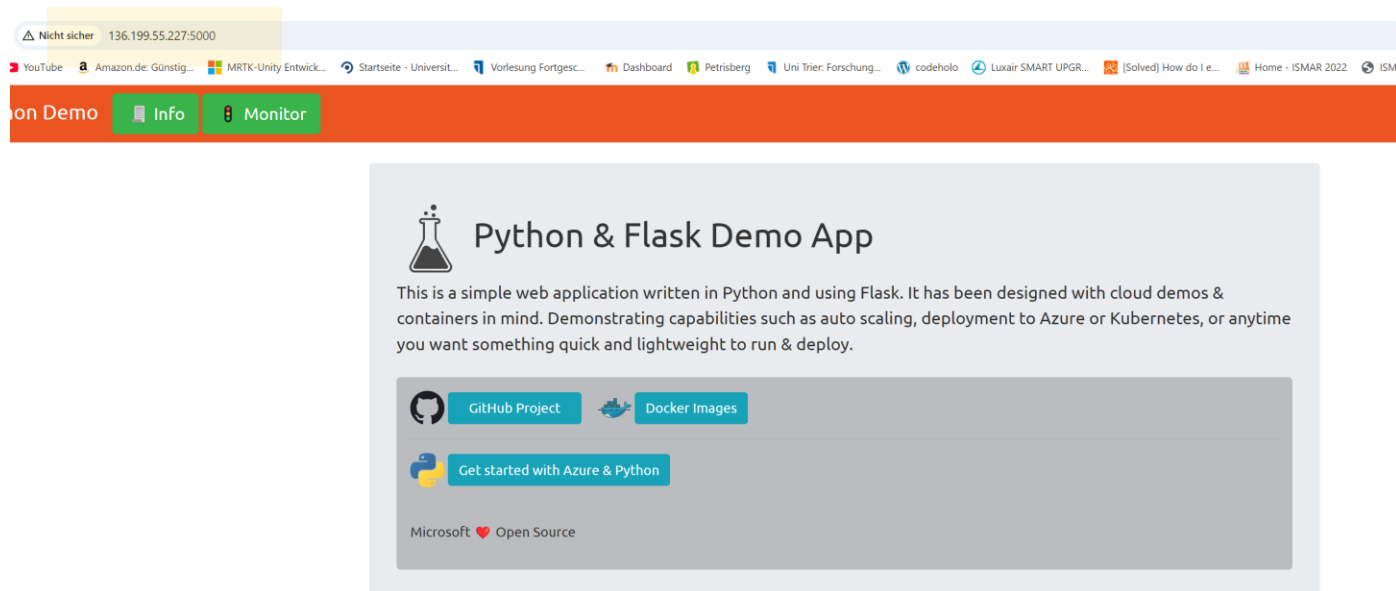
Bevor die Tests ausgeführt werden können, müssen alle Dependencies bezogen werden (Python => pip)



# Demo App – Lokal ausführen

## 3. App ausführen

```
$ make run
```



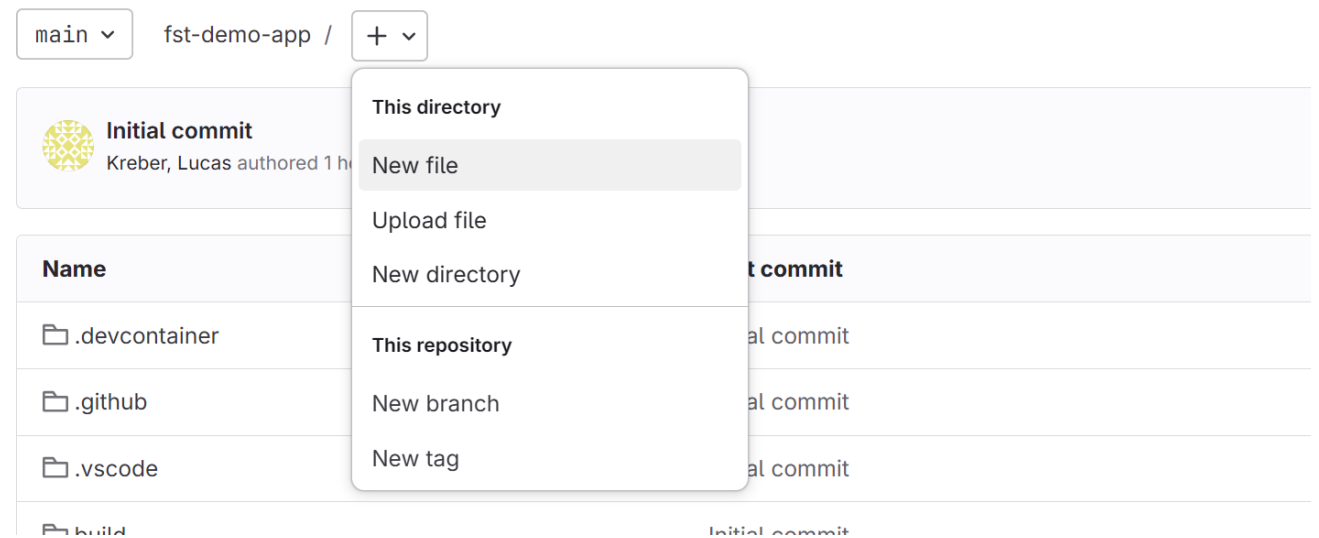
- ✓ Funktionsweise der App
- ✓ Testfälle ausführen
- ✓ App starten
- ✗ CI/CD Pipeline

# Pipeline Configuration File

Die Pipeline ist gescrpted und liegt direkt im Repository als YAML-File

Die Konfigurationsdatei muss folgenden Dateinamen haben:  
.gitlab-ci.yml

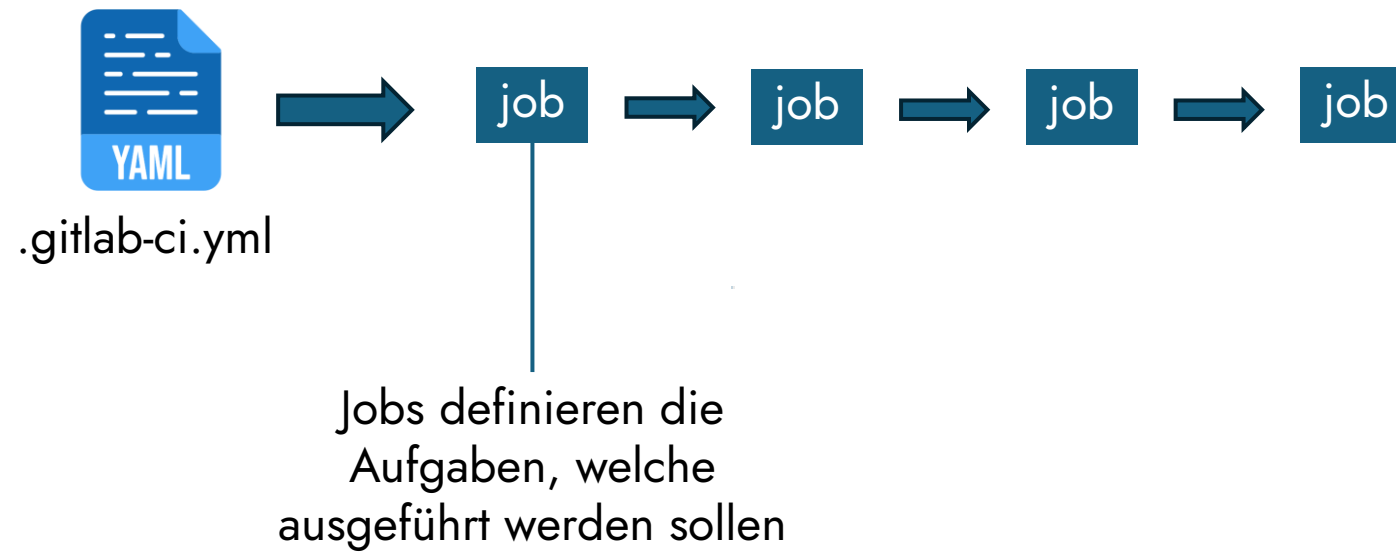
Dort können auch andere Konfigurationen eingebunden werden



Neue Dateien können direkt über die GitLab-UI erstellt und bearbeitet werden

✅ Konfigurationsdatei erstellen ❌ Tests ausführen ❌ Variablen anlegen ❌ Docker build/push ❌ Stages anlegen ❌ Auf Server deployen

# Jobs



# Jobs

Ein Job wird durch einen Namen definiert, gefolgt von Anweisungen (z.B. script, tags, only und anderen)

```
job_name:
  script:
    - echo Building the application # Konkrete Befehle, welche ausgeführt werden sollen
    - npm install
    - npm run build
    - echo Running build process
  tags:
    - docker # Beschränkt Ausführung von Jobs auf Runner, mit entsprechenden tags
  only:
    - main # Startet den Job nur dann, wenn auf dem main-branch Änderungen gemacht werden
```

# Jobs

before\_script und after\_script führen Befehle jeweils vor oder nach der Pipeline bzw. des Jobs aus

```
before_script: # Befehle vor Beginn der Pipeline (global)
- echo Global: Setting up environment

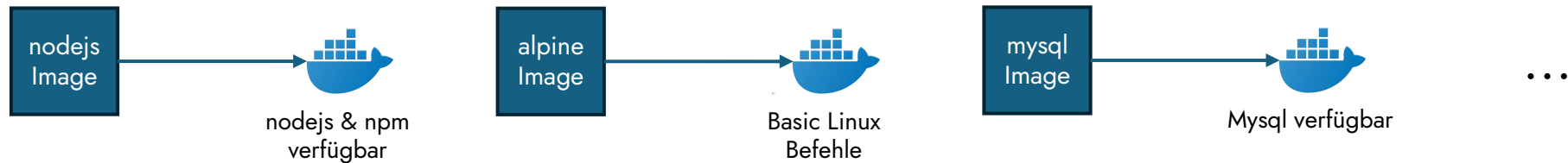
build_job:
  before_script: # Befehle vor Beginn eines Jobs
  - npm install
  script:
  - npm run build
  after_script: # Befehle nach Beendigung des Jobs
  - clear

after_script: # Befehle nach Beendigung der Pipeline (global)
- echo Cleaning up after pipeline
```

# GitLab Architektur Part 2

Wir verwenden Docker-Container, um unsere Jobs auszuführen. Aber wie?

Docker-Container basieren auf verschiedenen Docker-Images



Default GitLab Docker Runner basiert auf einem Ruby Image



→ Kein Python verfügbar

→ Anderes Image verwenden



# GitLab Architektur Part 2



<https://hub.docker.com/>

```
build_job:  
  image: docker_image_name:docker_image_version  
  script:  
    - echo run
```

Nicht immer existiert der ausreichend konfigurierte Container!

Lösung: selber Konfigurieren und fehlende Programme nachinstallieren

Wird in Docker-  
Container  
ausgeführt

```
build_job:  
  image: docker_image_name:docker_image_version  
  before_script:  
    - apt-get update && apt-get install php  
  script:  
    - echo run
```

# Praxis I – Runner anlegen

The image shows the GitLab interface for creating and managing runners. On the left, a sidebar menu has 'Settings' selected, which opens a sub-menu where 'CI/CD' is highlighted. An arrow points from this menu to the 'Runners' page. The 'Runners' page has a section for 'Project runners' with a 'New project runner' button circled in red. Below this, 'Assigned project runners' are listed, including one with ID #128. To the right, the 'Instance runners' section shows a toggle switch for 'Enable instance runners for this project' also circled in red. Below the toggle, three instance runners are listed: #111 (Autoscale Runner), #112 (Docker in Docker Runner), and #104 (General Runner). Annotations with arrows point from the text 'Eigene Runner registrieren' to the 'New project runner' button and from 'Instance runners deaktivieren' to the toggle switch.

Compare revisions  
Snippets  
Locked files  
Build  
Secure  
Deploy  
Operate  
Monitor  
Analyze  
Settings

General  
Integrations  
Webhooks  
Access tokens  
Repository  
Merge requests  
CI/CD  
Packages and registries  
Monitor  
Usage Quotas

## Runners

Runners are processes that pick up and execute CI/CD jobs for GitLab. [What is GitLab Runner?](#)

Register as many runners as you want. You can register runners as separate users, on separate servers, and on your local machine.

### How do runners pick up jobs?

Runners are either:

- **active** - Available to run jobs.
- **paused** - Not available to run jobs.

Tags control which type of jobs a runner can handle. By tagging a runner, you make sure runners only handle the jobs they are equipped to run. [Learn more.](#)

### Project runners

These runners are assigned to this project.

[New project runner](#)

### Assigned project runners

- #128 (5id3bGAR)  
dind docker

[Remove runner](#)

### Instance runners

These runners are available to all groups and projects.

**Enable instance runners for this project**

Available instance runners: 3

- #111 (9c6dfba2)  
Autoscale Runner (executor: docker+machine, driver: virtualbox)  
autoscale docker linux ubuntu
- #112 (nafJsa1)  
Docker in Docker Runner (executor: docker)  
dind docker dockerindocker
- #104 (59fb68b1)  
General Runner (docker executor)  
docker multi ubuntu

Eigene Runner registrieren

Instance runners deaktivieren

# Praxis I – Runner anlegen

## New project runner

Create a project runner to generate a command that registers the runner with all its configurations.

### Tags

#### Tags

Add tags to specify jobs that the runner can run. [Learn more.](#)

dind

Separate multiple tags with a comma. For example, `macos, shared`.

☐ Run untagged jobs

Use the runner for jobs without tags in addition to tagged jobs.

### Configuration (optional)

#### Runner description

☐ Paused

Stop the runner from accepting new jobs.

☐ Protected

Use the runner on pipelines for protected branches only.

☐ Lock to current projects

Use the runner for the currently assigned projects only. Only administrators can change the assigned projects.

#### Maximum job timeout

Maximum amount of time the runner can run before it terminates. If a project has a shorter job timeout period, the job timeout period of the instance runner is used instead.

Enter the job timeout in seconds. Must be a minimum of 600 seconds.

Create runner

dind als tag eintragen  
und run untagged jobs  
auswählen.

Dann Create runner  
klicken

Token wird unter Step 1  
angezeigt

Token kopieren und in  
folgender Liste  
zeilenweise eintragen:

<https://docs.google.com/spreadsheets/d/1ZYglfZxcXRrukoGW4BXPfi4xo64Rjn8dRSSuZNy2ZSU/edit?usp=sharing>

Runner sollte nach  
Registrierung verfügbar  
sein

### Other available runners

#128 (5id3bGAR)

dind docker

# Praxis II – Test ausführen

## Aufgabe:

Baue eine CI-Pipeline. Erzeuge einen Job, welcher die Testfälle der Beispiel-App in einem Docker-Container ausführt.

## Gegeben:

- Verwende als Docker-Image **python:3.9-slim-bookworm**  
(Dieses Image enthält python und pip)
- Zum kompilieren mancher pip-Module wird build-essential und python3-dev benötigt:  
`apt-get update && apt-get install -y build-essential python3-dev`
- Zum Testen wird make benötigt  
`apt-get install make`



Konfigurationsdatei erstellen



Tests ausführen



Variablen anlegen



Docker build/push



Stages anlegen

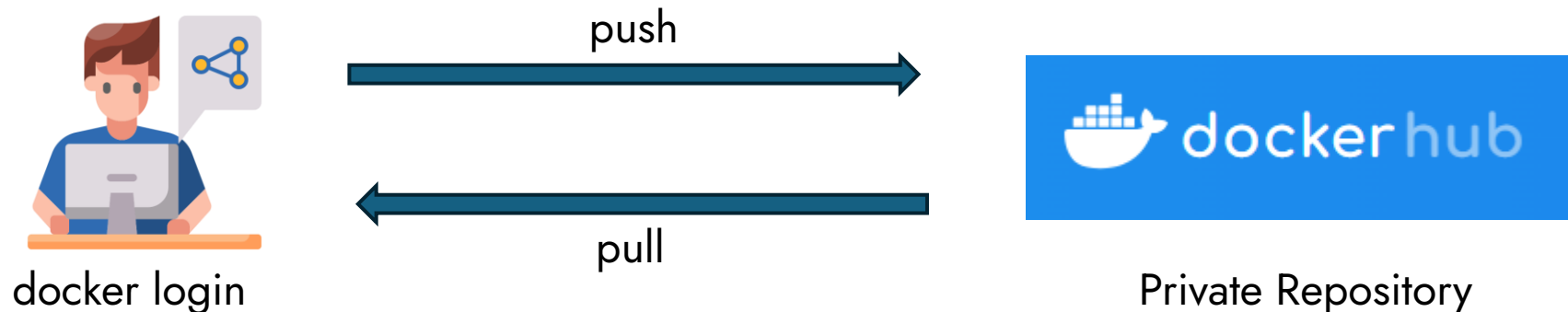


Auf Server deployen

# Anlegen von (zensierten) Variablen

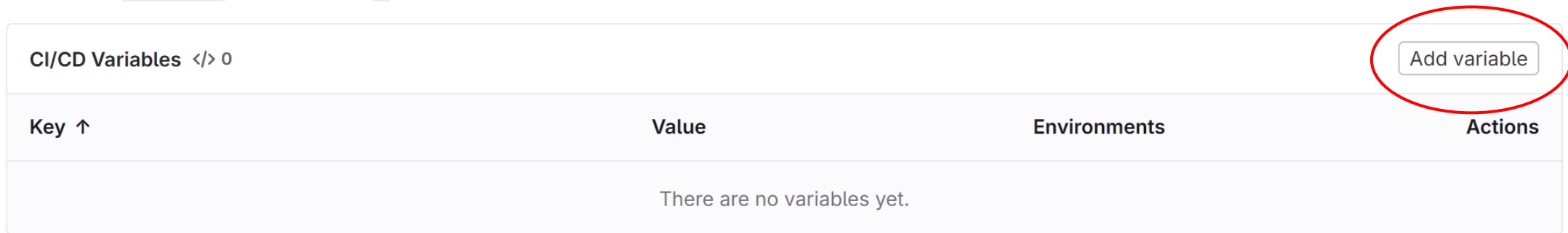
**Problem:** Viele Aktionen, wie zum Beispiel das Pushen von Docker-Containern oder ssh-Verbindungen benötigen **Keys** zum **authentifizieren**.

Diese sollen **nicht sichtbar** im Log der Pipeline sein und auch **nicht** im **Repository gespeichert** sein



# Anlegen von (zensierten) Variablen

## Settings -> CI/CD -> Variables



CI/CD Variables </> 0

Key ↑	Value	Environments	Actions
There are no variables yet.			

### Visibility:

- **Visible:** Werte können in Log zu sehen sein
- **Masked:** Werte sind in Log zensiert, aber in den CI/CD Einstellungen einsehbar
- **Masked and hidden:** Wie Masked, nur das nach Eintragung die Werte auch in den Einstellungen nicht mehr sichtbar sind

**Key:** Name der Variable

**Value:** Wert der Variable

**Zugriff im Script:** \$Name\_of\_var

# Anlegen von Script Variablen

Variablen können auch direkt im Pipeline-Script angelegt werden. Entweder Global oder innerhalb eines Jobs

Achtung: Diese Variablen können im Klartext sichtbar sein!

```
variables:  
  Name: HelloWorld  
  version: 1.0
```

```
test:  
  variables:  
    version: 1.0
```

# Praxis III – DockerHub Login anlegen

## Aufgabe:

Lege eine Variable DOCKER\_USER und eine Variable DOCKER\_PASS an.

## Gegeben:

- **Username:** lucaskreber
- Password\_token:  
dckr\_pat\_GvXQYZReSqP9lxUvhO9lvVpAyTY


✓ Konfigurationsdatei erstellen   ✓ Tests ausführen   ✓ Variablen anlegen   ✗ Docker build/push   ✗ Stages anlegen   ✗ Auf Server deployen




# Docker Image bauen und pushen

**Bauen:** Zum Bauen des Docker-Containers wird ein Dockerfile benötigt. Dieses liegt in unserem Projekt in build/Dockerfile

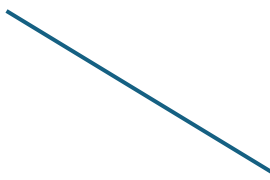
```
docker build -f build/Dockerfile -t RepoName/ImageName:ImageTag .
```



Pfad zum Dockerfile  
(liegt dieses im Root, wird  
dieses Argument nicht  
benötigt)



Gibt den Namen und Tag  
des Containers an, getrennt  
durch :



Der Punkt am Ende gibt  
den Build-Kontext an, in  
diesem Fall das aktuelle  
Verzeichnis. Deshalb nur  
ein Punkt

**Hinweis:** In Docker enthält der Name des Images die Repository Location.  
`hub.docker.com/RepoName/ImageName:ImageTag`

Da `hub.docker.com` der default Wert ist, kann dieser für das Pushen zu  
`dockerHub` weggelassen werden

# Docker Image bauen und pushen

**Pushen:** Zum Pushen wird ein Login sowie das entsprechende Repo benötigt

```
docker login -u username -p password  
docker push RepoName/ImageName:ImageTag
```

# Docker Image bauen und pushen

**Erinnerung:** Unser Runner läuft in einem Docker-Container.

Nun soll ein Docker-Container gebaut werden von einem anderen Docker-Container.

Wir benötigen also **Docker in Docker (dind)**

➡ Offizielles dind Image: docker:27-dind

Das Base Image sollte die gleiche Version wie das dind Image haben

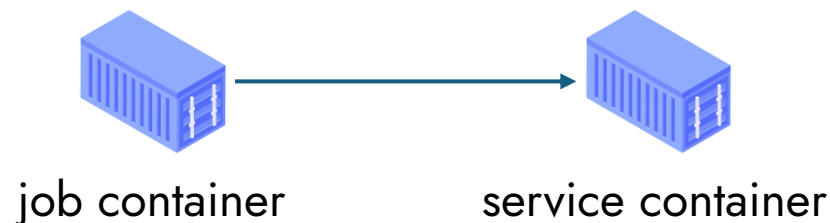
➡ Passendes Base Image: docker:27.3.1

# Gitlab services

```
test_job:
  image: ruby:latest # Haupt-Image des Jobs
  services:
    - postgres:latest
    - docker:27-dind
  variables:
    POSTGRES_DB: test_db
    POSTGRES_USER: user
    POSTGRES_PASSWORD: password
  script:
    - bundle install
    - rails db:create db:migrate
    - rails test
```

Client

Deamon



- services sind temporäre Docker-Container zur Laufzeit des Jobs
- Benötigt ein Job externe Abhängigkeiten z.B. eine Datenbank oder dind, kann dazu ein service genutzt werden
- Der Haupt-Container und der Service-Container teilen sich dasselbe Netzwerk, sodass sie über Hostnamen miteinander kommunizieren können
- Vorteil: Leichter und schneller, ein existierendes Images zu verwenden, anstatt mit jedem Durchlauf alles neu installieren zu müssen

# Praxis IV – Anwendung bauen und pushen

## Aufgabe:

Erstelle einen neuen Job, welcher sich zu Beginn auf Docker Hub einloggt, anschließend den Container baut und in das Repository pusht.

Es soll Docker in Docker mit entsprechendem service verwendet werden.

## Gegeben:

- **Login:** Username und Passwort, zuvor angelegt
- **Base Image:** docker:20.10.16
- **Dind Image:** docker:20.10.16-dind
- **ImageName:** lucaskreber/fst
- **ImageTag:** python-app-1.0-DeinName



Konfigurationsdatei erstellen



Tests ausführen



Variablen anlegen



Docker build/push



Stages anlegen

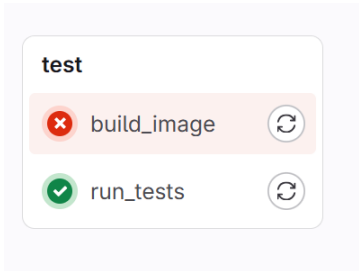


Auf Server deployen

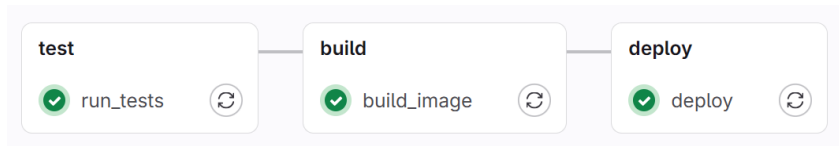
# Stages

Bisher:

Alle Jobs laufen durch, auch wenn zuvor einer fehlgeschlagen ist



Lösung: Stages



```
stages:
  - build
  - test

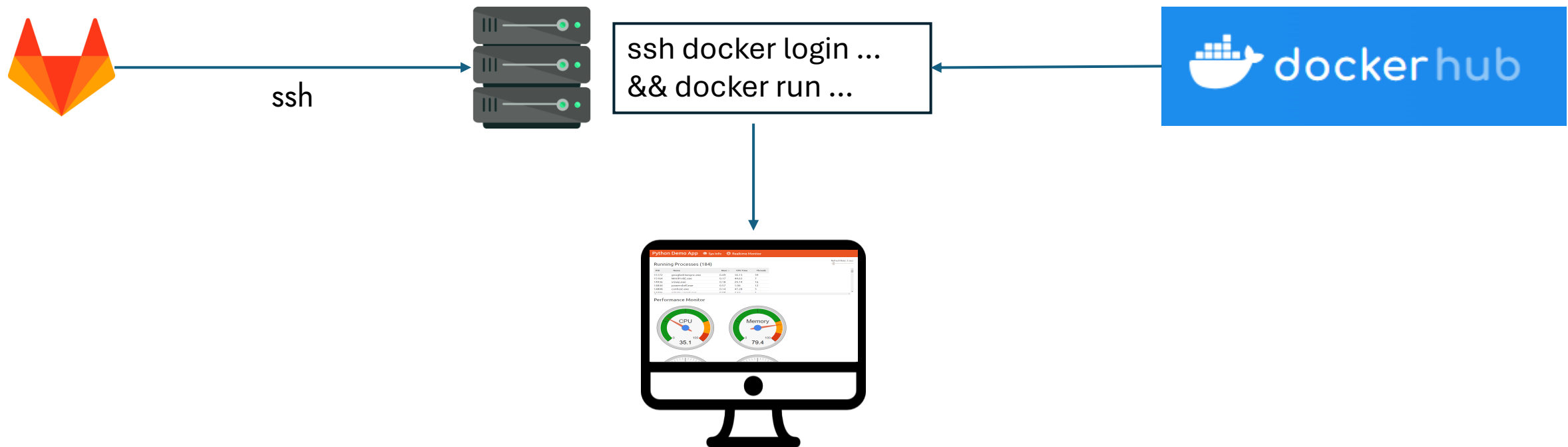
build_job:
  stage: build
  before_script:
    - npm install
  script:
    - npm run build

test_job:
  stage: test
  script:
    - npm test
```

- ✓ Konfigurationsdatei erstellen
- ✓ Tests ausführen
- ✓ Variablen anlegen
- ✓ Docker build/push
- ✓ Stages anlegen
- ✗ Auf Server deployen

# Deployen

Benötigt: Deployment-Server, Private-Key



# Deployen

Einloggen über ssh:

```
ssh -o StrictHostKeyChecking=no -i $SSH_KEY lucas@136.199.55.227 ""
```

```
$ ssh lucas@136.199.55.227
The authenticity of host '136.199.55.227 (136.199.55.227)' can't be established.
ED25519 key fingerprint is SHA256:Rq1sDsixKblyBZFCza9KIVmbZZh8gf0neJnWMVXT8Dg.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? |
```

Keyfile mit  
private key

Nutzername  
und IP von  
Server

Befehle zum  
Ausführen in



# SSH-Key anlegen

Settings -> CI/CD -> Variables:

Key: SSH\_KEY

Value: Inhalt des Keys (Leerzeile am Ende einfügen!)

Type: File

Masked variable: unchecked

Später in Script:

before\_script:

- chmod 400 \$SSH\_KEY

Key herunterladen

<https://seafile.rlp.net/f/78097f1393334db0b185/>

# Praxis V - deployen

## Aufgabe:

Deploye die App auf den Server nach dem Konzept von Folie 33. Verwende als Port den zugewiesenen Port!

**`docker run -d -p port:5000 ImageName`**

-d: detached mode -> Container läuft im Hintergrund

-p: <host\_port>:<container\_port> mapped host port auf container port

## Gegeben:

Damit die App mehrfach deployed werden kann, muss die alte Instanz vorher gestoppt werden. Verwende dazu folgenden Befehl:

```
docker ps -a -q --filter 'publish=<port>' | xargs -r docker stop &&  
docker ps -a -q --filter 'publish=<port>' | xargs -r docker rm
```

ps: Listet alle laufenden Docker-Container auf

-a: Zeigt auch gestoppte Container an

-q: Gibt nur die Container-IDs zurück

--filter: Filtert die Liste und gibt nur Container zurück, die Port 5000 haben

xargs: Nimmt die IDs aus der Ausgabe und übergibt sie als Argumente an den nachfolgenden Befehl



Konfigurationsdatei erstellen



Tests ausführen



Variablen anlegen



Docker build/push



Stages anlegen



Auf Server deployen

# Bildreferenzen

<https://www.flaticon.com/free-icons/python>  
<https://www.flaticon.com/free-icons/testing>  
<https://www.flaticon.com/free-icons/docker>  
<https://www.flaticon.com/free-icons/repository>  
<https://www.flaticon.com/free-icons/deployment>  
<https://www.flaticon.com/free-icons/gitlab>  
<https://www.flaticon.com/free-icons/sport-team>  
<https://www.flaticon.com/free-icons/code>  
<https://www.flaticon.com/free-icons/code>  
<https://www.flaticon.com/free-icons/new-release>  
<https://www.flaticon.com/free-icons/worker>  
<https://www.flaticon.com/free-icons/git>  
<https://www.flaticon.com/free-icons/fast>  
<https://www.flaticon.com/free-icons/server>  
<https://www.flaticon.com/free-icons/computer>  
<https://www.flaticon.com/free-icons/linux>  
<https://www.flaticon.com/free-icons/mac>  
<https://www.flaticon.com/free-icons/globe>  
<https://www.flaticon.com/free-icons/txt-file>  
<https://www.flaticon.com/free-icons/foursquare-check-in>  
<https://www.flaticon.com/free-icons/yaml>  
<https://www.flaticon.com/free-icons/terminal>  
<https://www.flaticon.com/free-icons/lightning>  
<https://www.flaticon.com/free-icons/digital>  
<https://www.flaticon.com/free-icons/container>  
<https://www.flaticon.com/free-icons/imac>