

Softwaretechnik

DevOps, Continuous Integration & Delivery



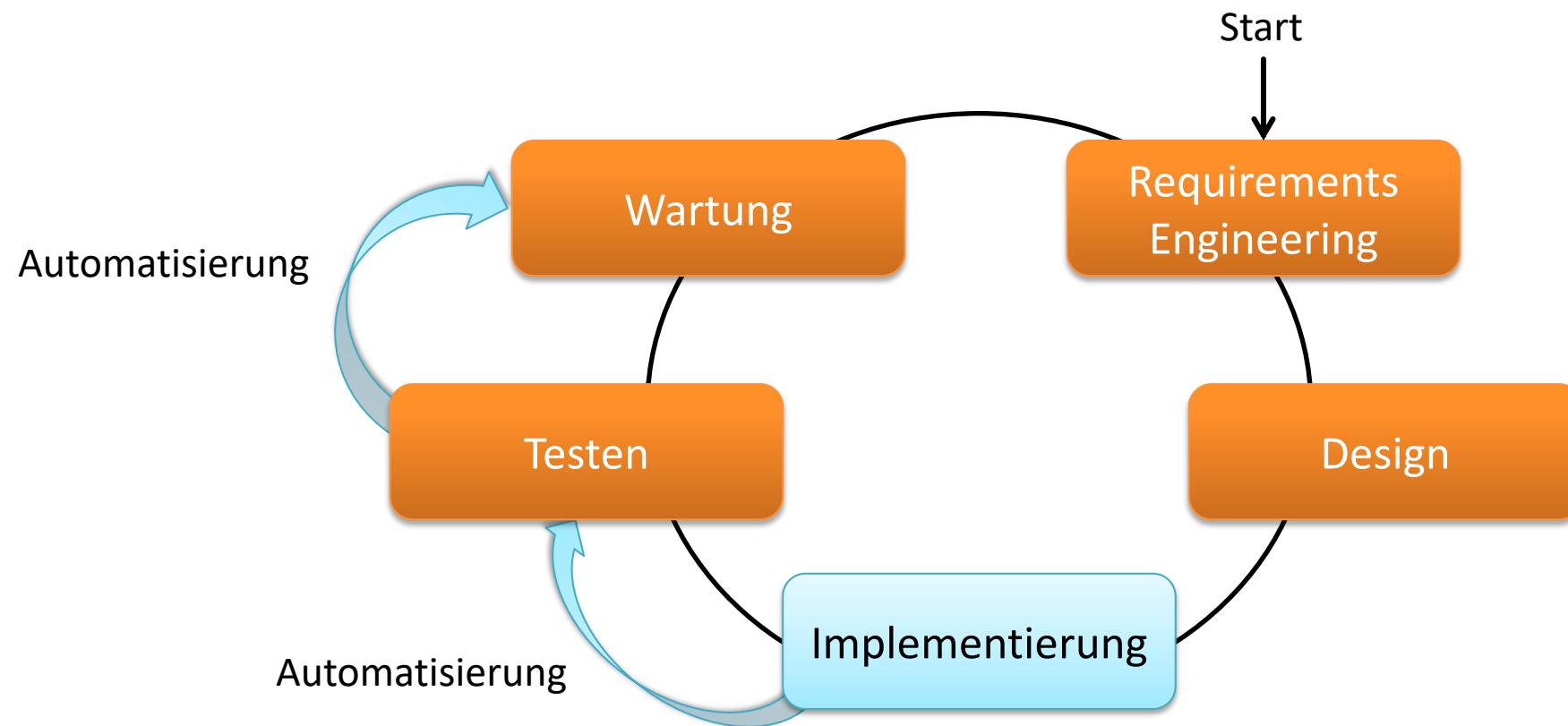
SOFTWARE
SYSTEME

Prof. Dr.-Ing. Norbert Siegmund (Heute: Alina Mailach)
Software Systems



UNIVERSITÄT
LEIPZIG

Softwarelebenszyklus - Einordnung



Lernziele

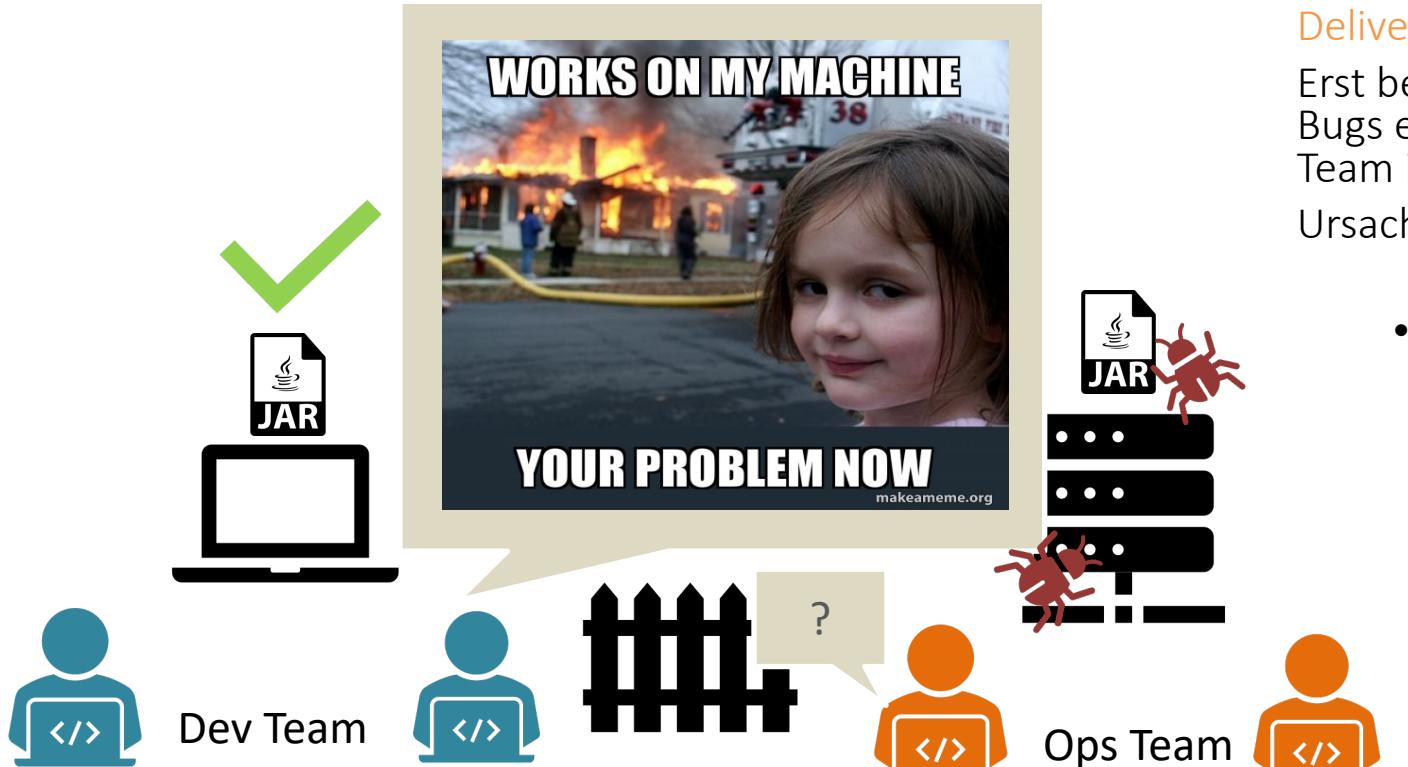
- Einordnung der Begriffe DevOps, Continuous Integration/Delivery/Deployment
- Grundlegende Prinzipien und Praktiken von CI/CD
- Technologien & Tools die bei der Umsetzung der Praktiken helfen

Ablauf

- Was versteht man unter DevOps?
- Continuous Integration
- Continuous Delivery & Deployment
- Technologien & Tools

DevOps

Vor DevOps...



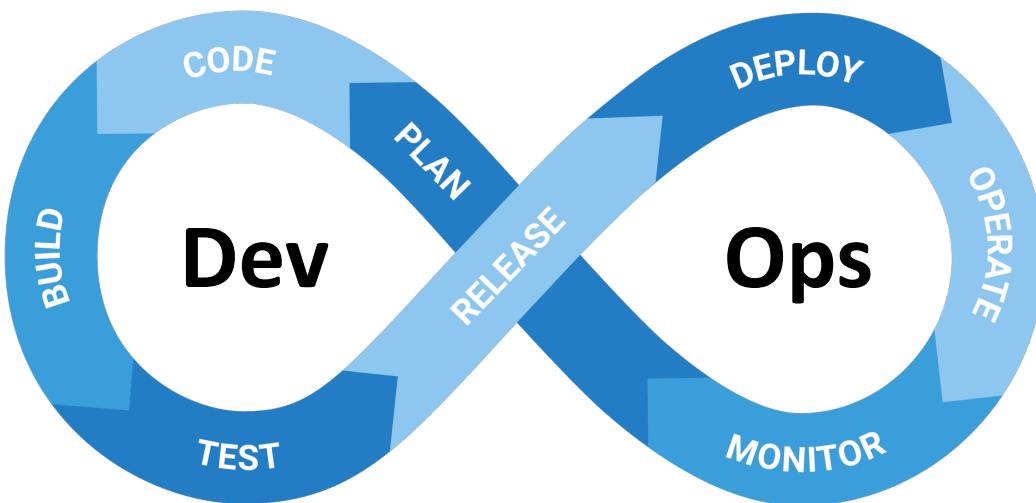
Delivery/Deployment

Erst beim Ausliefern und Bereitstellen der Software werden Bugs entdeckt, die Zusammenarbeit zwischen Dev und Ops Team ist oft schwierig

Ursachen:

- Entwicklungsumgebung und Produktionsumgebung unterscheiden sich teils stark, Gründe dafür können:
 - Betriebssysteme und Versionen,
 - Unterschiedliche Hardware (Prozessorentyp, verfügbarer RAM, GPU, ...),
 - Bibliotheken im Programmiersprachenökosystem,
 - Umgebungsvariablen oder
 - Daten und Ressourcen sein.
- Dev und Ops verfolgen unterschiedliche Ziele (Weiterentwicklung vs. Stabilität) und werden an konkurrierenden Metriken gemessen

DevOps



- Eine Sammlung von technischen Methoden und kulturellen Werten, welche die Lücke zwischen **Softwareentwicklung (Dev)** und **Operation (Ops= Systemadministration, Wartung, Monitoring, Patching, ...)** schließt.
- Durch **Automatisierung** sollen Integrations-, Liefer- und Wartungsprozesse beschleunigt und optimiert werden.
- Zum Erreichen der DevOps Ziele werden Praktiken wie **Continuous Integration**, **Continuous Delivery/Deployment**, **Infrastructure-as-Code**, etc. verwendet.

DevOps Ziele

- Verbesserung der Softwarequalität durch
 - Häufiges Testen,
 - Angleichen der Entwicklungs- und Betriebsumgebungen,
 - Beschränkung auf die tatsächlich benötigten Features durch schnelles und regelmäßiges Feedback durch Stakeholder, und
 - Mehr Security, durch standardisierte Prozesse.
- Minimierung der Entwicklungs- und Lieferdauer
- Ermöglichung agiler Entwicklung (schnelle, flexible und iterative Arbeitsweise sind ohne DevOps Praktiken kaum möglich)
- Entwicklung und Betrieb der Software durch das selbe Team -> „You build it, you run it!“
- Überbrückung von local und remote computing
- Sichere und nachvollziehbare Updates und Rollouts

DevOps Prinzipien

- Entwickle von Anfang an mit dem Fokus auf Qualität
- Alle sind verantwortlich
- Strebe nach kontinuierlicher Verbesserung
- arbeite in kleinen Schritten und mit kurzen Feedbackschleifen
- Automatisiere alles was du automatisieren kannst

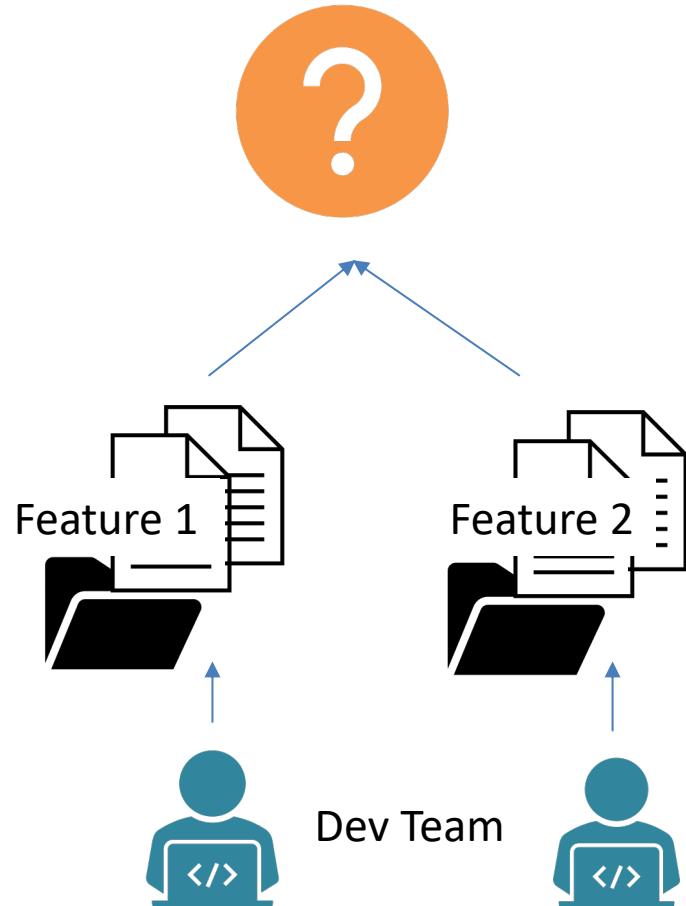
„Computer erledigen wiederkehrende Aufgaben – Menschen lösen Probleme“



Continuous Integration (CI)

Eine Sammlung von Praktiken zur Erleichterung und Automatisierung des Zusammenführens unterschiedlicher Codeänderungen von mehreren Beteiligten in ein einziges Softwareprojekt.

Ohne CI...

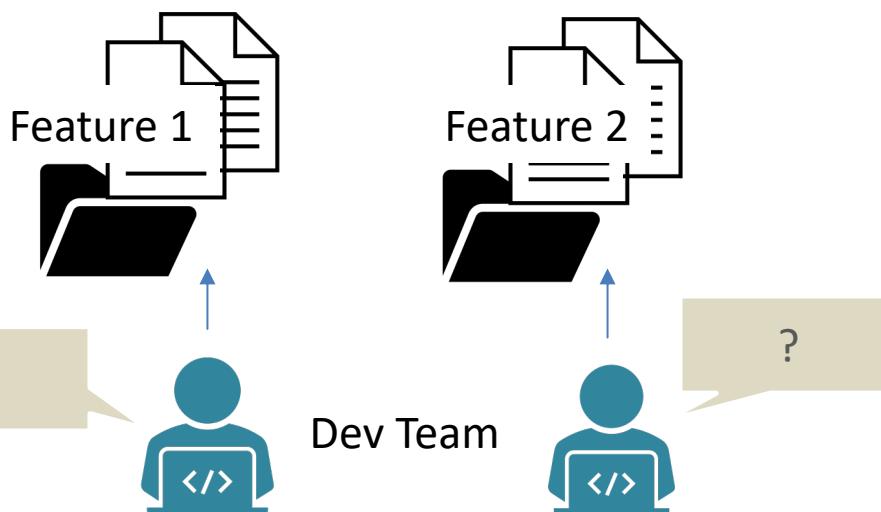


Integration = Verschiedene Entwicklungsstränge, oft von verschiedenen Entwicklerinnen, müssen zusammengeführt werden

A screenshot of a code merge tool interface. The title bar says "Merge Revisions for D:\Projects\Git\Demo\Nancy\src\Nancy\DependencyContextAssemblyCatalog.cs". The main area shows a comparison between "Local Changes (Read-only)" and "Changes from Server (revision 9d4b6795...)". A conflict is highlighted at line 24, column 24, where both sides have changes. The left side has a red box around the line number 24. The right side has a red box around the character position 24. The code editor shows various syntax highlighting and fold markers. Buttons at the bottom include "Accept Left", "Accept Right", "Apply", and "Abort".

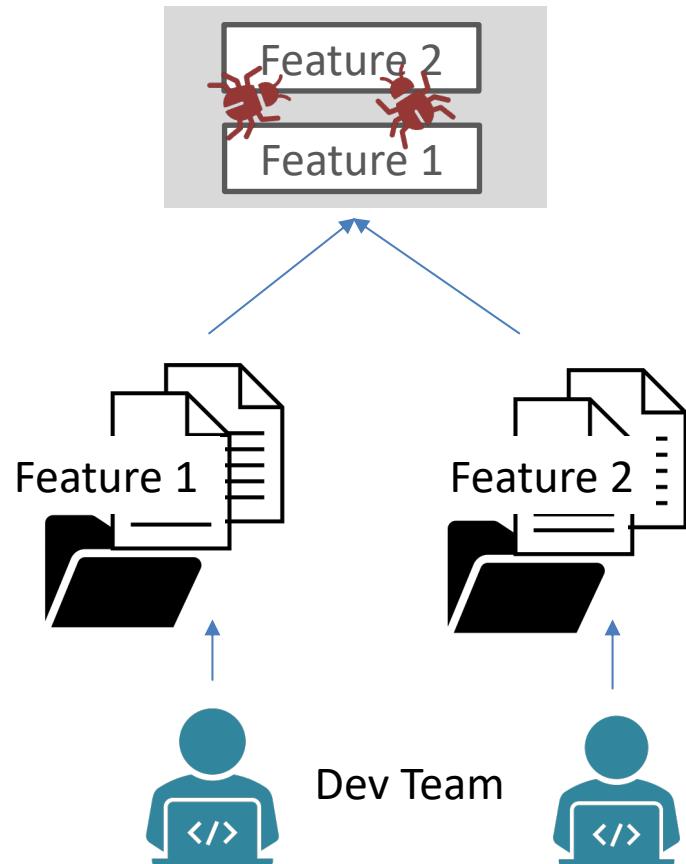
Ohne CI...

„Merge Hell“



- Wie muss der Code angepasst werden damit beide Features korrekt zusammenspielen?
- Oft ist der Code, der integriert werden muss schon mehrere Tage, Wochen oder sogar Monate alt
- Schnittstellen sind ggfs. nicht klar definiert oder müssen angepasst werden
- Konflikte erfordern oft enge Zusammenarbeit, sind schwer zu lösen und können die Entwicklung verlangsamen oder im schlimmsten Fall zum Stillstand bringen

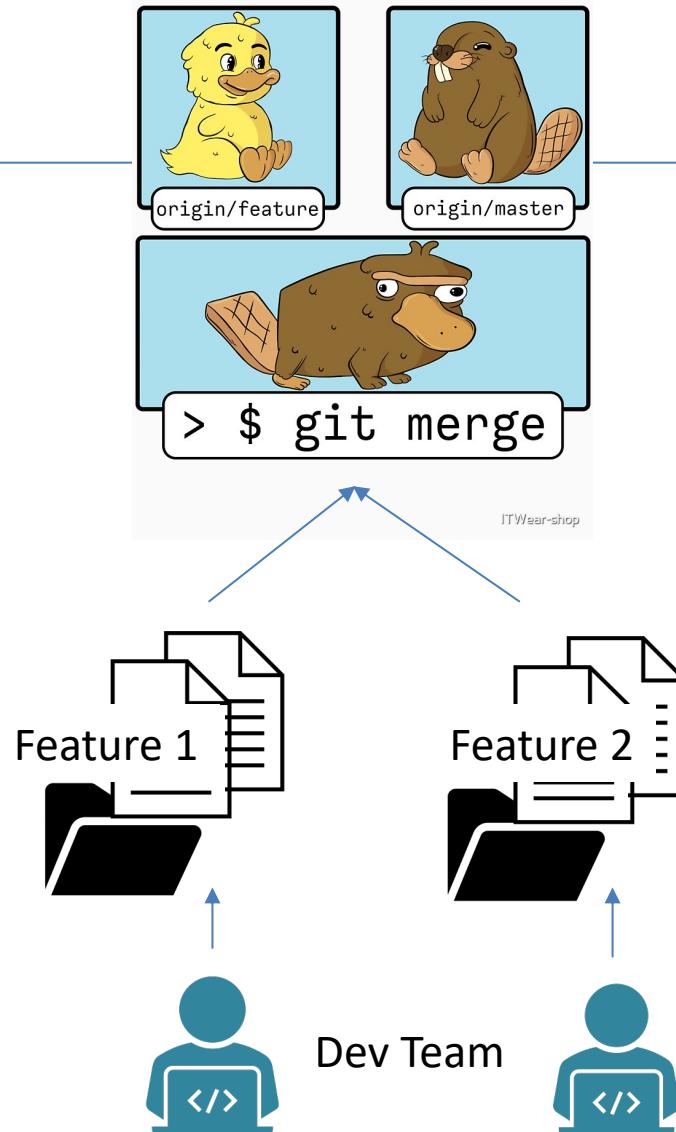
Ohne CI...



Kumulative Bugs (Verification):

- Zeigen sich erst nach der Integration
- Entstehen durch die Interaktion verschiedener Features und Komponenten die vorher nicht zusammen getestet wurden
- Erfordern hohen Aufwand in der Behebung

Ohne CI...



Haben wir das richtige gebaut (Validation)?

- Erst nach der Integration ist es möglich Feedback von verschiedenen Stakeholdern einzuholen
- Frühes Feedback (wie in agilen Szenarien gefordert) ist nicht möglich
- Änderungen sind gegebenenfalls so gravierend, dass die Software nicht ausgeliefert werden kann

Umsetzung von CI

9 Praktiken von CI

1. Ein zentrales Repository für den Quellcode

Ein zentrales Repository

The screenshot shows a GitHub repository page for the project 'pretix'. The repository has 46 watchers, 404 forks, and 1.5k stars. It contains 178 branches and 138 tags. The 'About' section describes it as a ticket shop application for conferences, festivals, concerts, tech events, shows, exhibitions, workshops, barcamps, etc. The 'Code' section lists recent commits from raphaelm, including fixes for RSA encryption crashes and locking mechanisms. The 'Releases' section shows 138 tags. The 'Packages' section indicates no packages have been published. The 'Contributors' section shows 271 contributors, with links to their profiles and a note for '+ 260 contributors'. The 'Languages' section shows Python as the primary language at 77.2%, followed by HTML (13.1%), JavaScript (5.3%), and SCSS (3.8%).

| Commit | Author | Message | Date |
|---|----------|---|--------------|
| Device API: Fix RSA encryption crash (PRETIXEU-8Y2) | raphaelm | New locking mechanism (#2408) | 2 months ago |
| _build | | Fix pyproject.toml wheel build issues (#3313) | 6 months ago |
| deployment/docker | | Docker: Change default number of workers to `2 * \$(nproc)` (#3634) | 3 weeks ago |
| doc | | API: Add endpoints for scheduled exports (#3659) | 5 days ago |
| res | | Update pretix logo to refreshed version (#3114) | 9 months ago |
| src | | Device API: Fix RSA encryption crash (PRETIXEU-8Y2) | yesterday |
| .clabot | | pretix Community Edition moves to AGPLv3-based license (#2023) | 2 years ago |
| .codecov.yml | | Update .codecov.yml | 7 years ago |
| .dockerrcignore | | Extend .dockerrcignore | 3 years ago |
| .gitattributes | | Visualize custom check-in rules (#2053) | 2 years ago |
| .gitignore | | Move build setup to pyproject.toml (#3240) | 7 months ago |
| .gitlab-ci.yml | | Fix pyproject.toml wheel build issues (#3313) | 6 months ago |
| .gitmodules | | Got rid of all submodules | 8 years ago |
| .landscape.yaml | | Add a landscape configuration file | 9 years ago |
| .licenseheader | | pretix Community Edition moves to AGPLv3-based license (#2023) | 2 years ago |
| .readthedocs.yaml | | Fix readthedocs build | last year |
| CODE_OF_CONDUCT.md | | Fix #689 -- Add CODE_OF_CONDUCT.md (#695) | 6 years ago |
| CONTRIBUTING.md | | Update CONTRIBUTING.md | 2 years ago |
| Dockerfile | | Docker: Upgrade to Debian bookworm (#3591) | 2 months ago |
| LICENSE | | pretix Community Edition moves to AGPLv3-based license (#2023) | 2 years ago |
| MANIFEST.in | | Fix pyproject.toml wheel build issues (#3313) | 6 months ago |
| README.rst | | pretix Community Edition moves to AGPLv3-based license (#2023) | 2 years ago |
| SECURITY.md | | Add SECURITY.md and new issue templates | last year |
| pyproject.toml | | New locking mechanism (#2408) | 2 months ago |
| setup.cfg | | Update .gitlab-ci.yml release script | 7 months ago |
| setup.py | | Do not run custom build commands on other packages | 5 months ago |

- Ein zentrales Repository in dem alle Dateien liegen die nötig sind, um die Anwendung zu bauen, z.B.:
 - Sourcecode der Anwendung und Tests
 - Konfigurationsdateien (für die Anwendung, IDE, etc)
 - Datenbankschema
 - Informationen zu Dependencies
- Alles was man zum Bauen der Software braucht, nichts was gebaut wird

9 Praktiken von CI

1. Ein zentrales Repository für den Quellcode
2. Automatisiertes Bauen (Build) & Testen der Anwendung

Automatisiertes Bauen (Build) & Testen

“Anyone should be able to bring in a virgin machine, check the sources out of the repository, issue a single command, and have a running system on their machine.” Martin Fowler

- Ziel: Bauen & Testen der Anwendung mit nur einem Command
- Testen erfordert ausreichend diverse Testsuite, die möglichst große Teile der Codebasis abdeckt (s. Vorlesung zum Testing)
- Buildsysteme
 - Dependency management
 - Code compilation & optimization
 - Konsistente Builds
 - Dokumentation generieren
 - Cross-platform builds
 - Test automation
 - ...



9 Praktiken von CI

1. Ein zentrales Repository für den Quellcode
2. Automatisiertes Bauen (Build) & Testen der Anwendung
3. Jede Entwicklerin commitet auf den Main (Trunk)-Branch jeden Tag

Trunk Based Development (1)

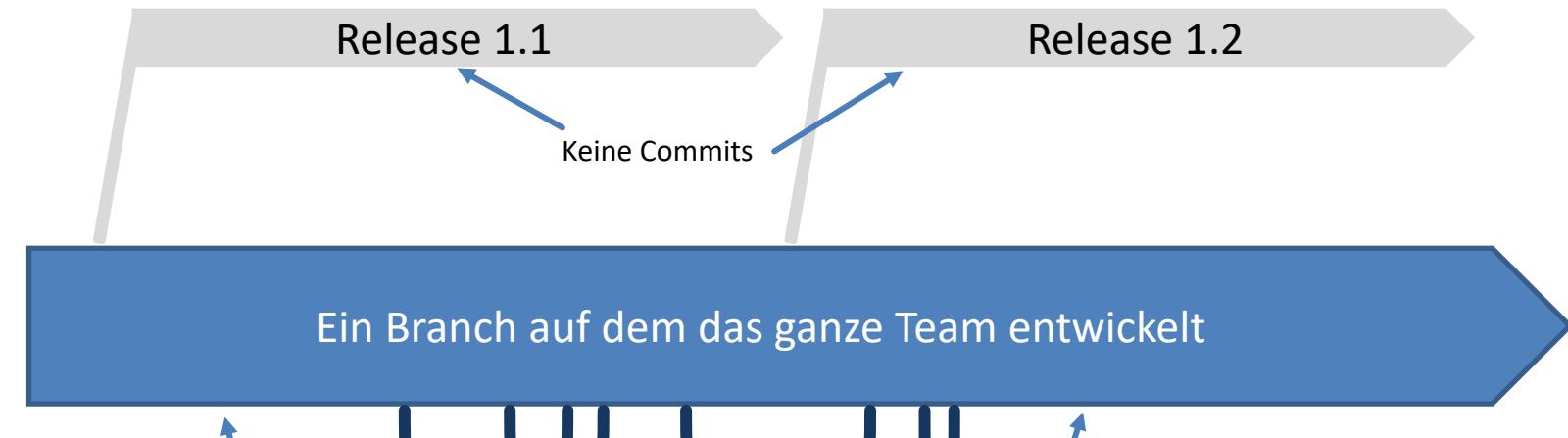
Release-Branches

- werden nach einiger Zeit gelöscht
- Alternativ werden Release-Tags verwendet

Trunk/Main-Branch

Feature-Branches

- Vorallem in größerem Teams
- Kurzlebig



Entwicklerinnen entwickeln in kleinen Commits direkt auf dem Trunk/Main oder auf kurzlebigen Feature-Branches

Trunk Based Development (2)

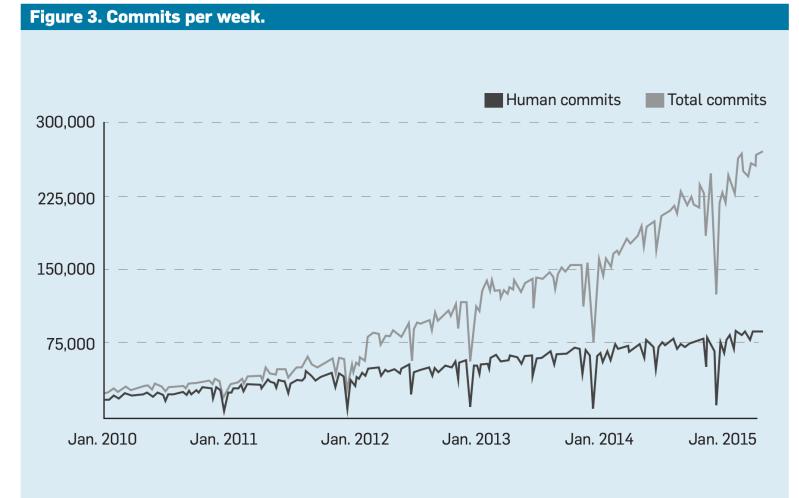
Wie geht man mit unvollständig implementierten Features um?

- Entwicklung hinter Feature-Toggles (oder Feature-Flags)
- Gezieltes an- und ausschalten bestimmter Features

```
1 if(newFeature.IsEnabled){  
2     newFeature();  
3 } else {  
4     oldFeature();  
5 }
```

Kann man mit beliebig großen Teams Trunk Based entwickeln?

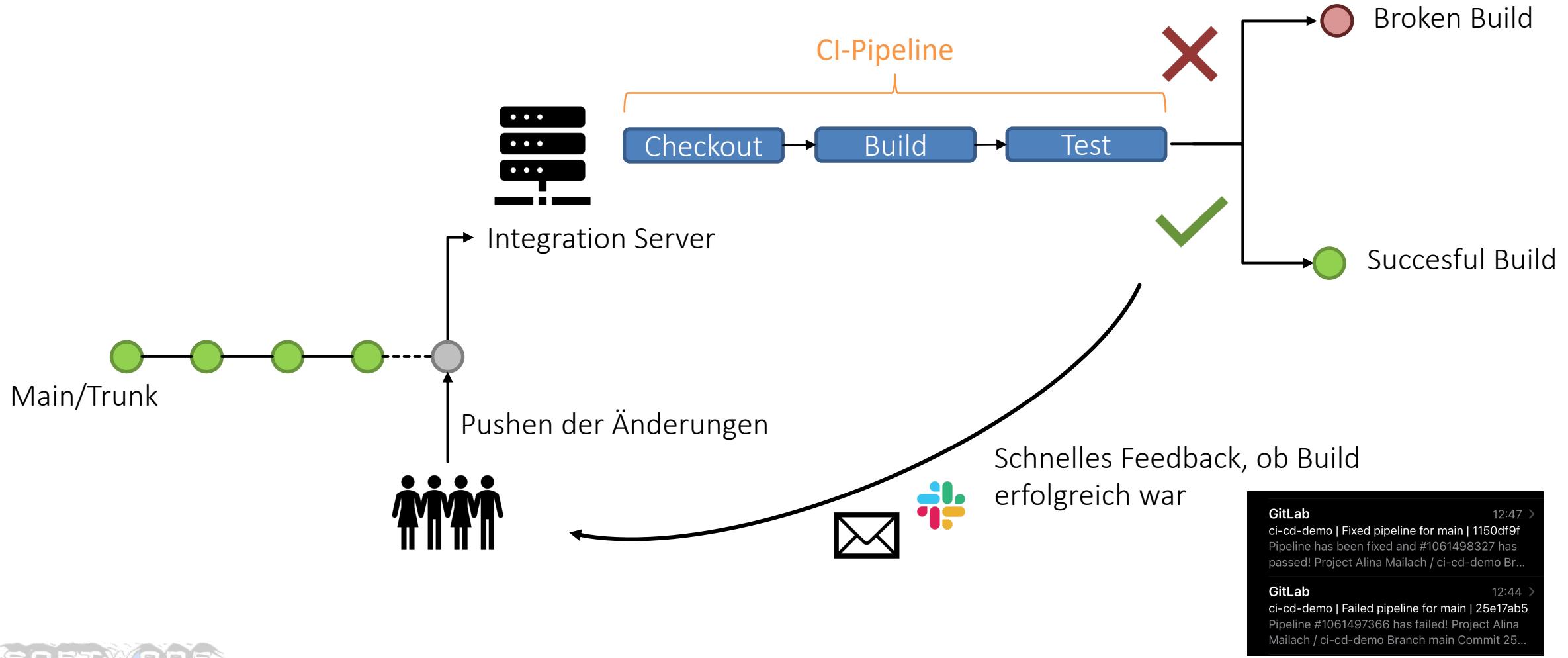
- Google, 2016:
<https://dl.acm.org/doi/pdf/10.1145/2854146>
- 15 Millionen Zeilen Code,
- >25.000 Entwicklerinnen und
- 86TB in einem einzelnen Repository



9 Praktiken von CI

1. Ein zentrales Repository für den Quellcode
2. Automatisiertes Bauen (Build) & Testen der Anwendung
3. Jede Entwicklerin commitet auf den Main (Trunk)-Branch jeden Tag
4. Bei jedem Commit wird der Main-Branch auf einer Integrationsmaschine gebaut und getestet
5. Das Bauen der Anwendung geht schnell und Entwicklerinnen erhalten schnell Feedback

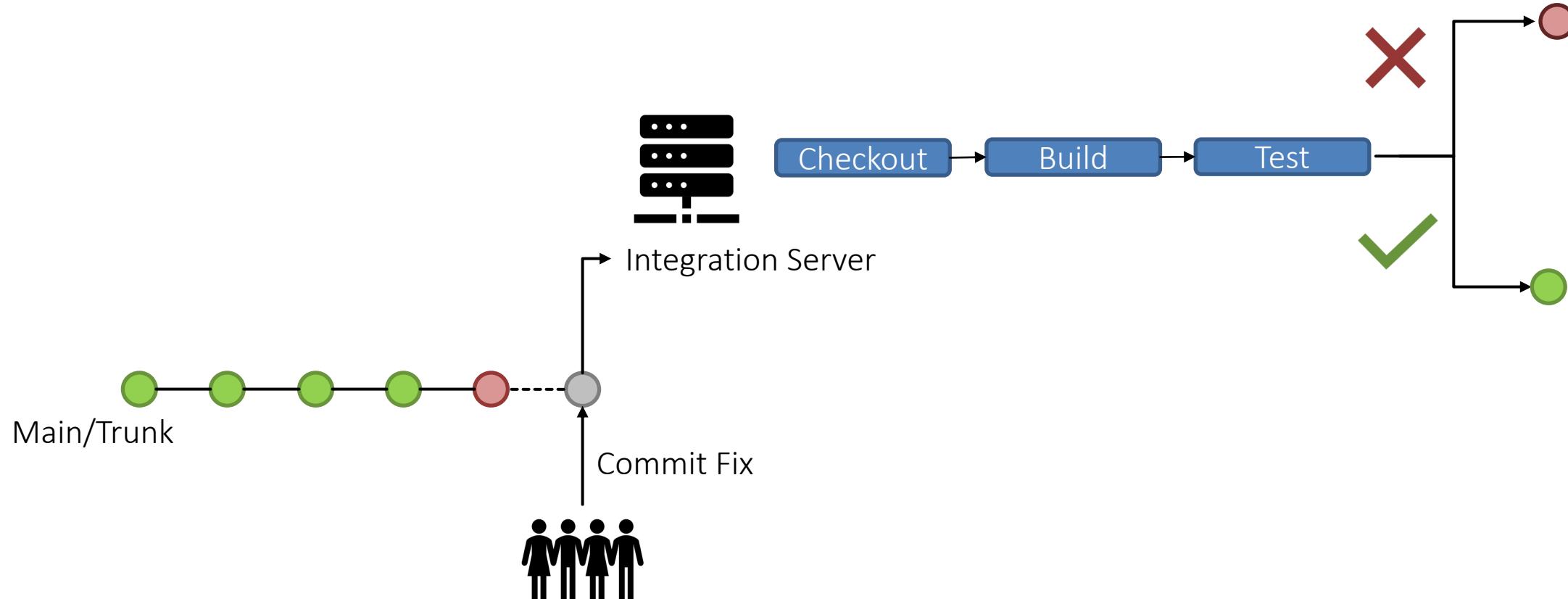
Build bei jedem Commit & schnelles Feedback



9 Praktiken von CI

1. Ein einziges Repository für den Quellcode
2. Automatisiertes Bauen (Build) der Anwendung
3. Automatisiertes Testen der Anwendung
4. Bei jedem Commit wird der Main-Branch auf einer Integrationsmaschine gebaut und getestet
5. Das Bauen der Anwendung geht schnell und Entwicklerinnen erhalten schnell Feedback
6. „Kaputte“ Builds werden sofort gefixt oder rückgängig gemacht (Fix or Revert)

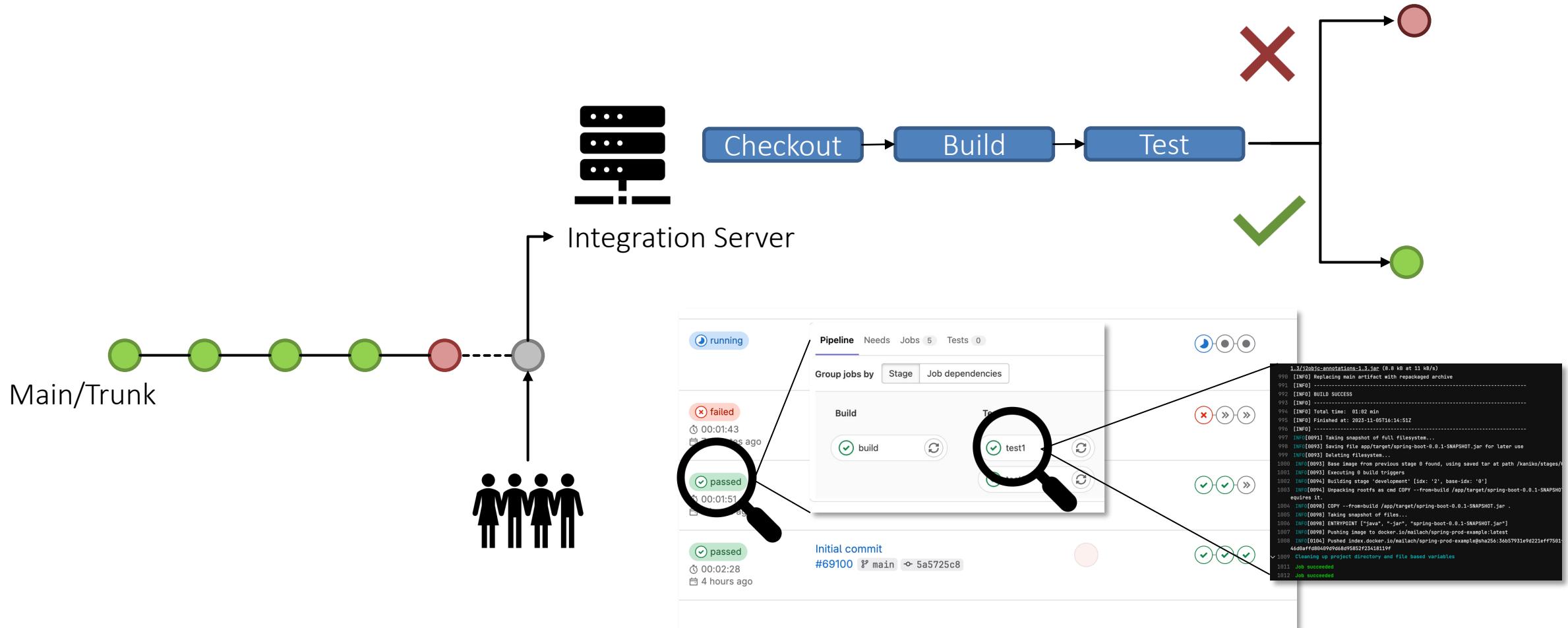
„Kaputte“ Builds werden sofort gefixt



9 Praktiken von CI

1. Ein einziges Repository für den Quellcode
2. Automatisiertes Bauen (Build) der Anwendung
3. Automatisiertes Testen der Anwendung
4. Bei jedem Commit wird der Main-Branch auf einer Integrationsmaschine gebaut und getestet
5. Das Bauen der Anwendung geht schnell und Entwicklerinnen erhalten schnell Feedback
6. „Kaputte“ Builds werden sofort gefixt oder rückgängig gemacht (Fix or Revert)
7. Alle sehen was gerade passiert und welchen Status der Build hat

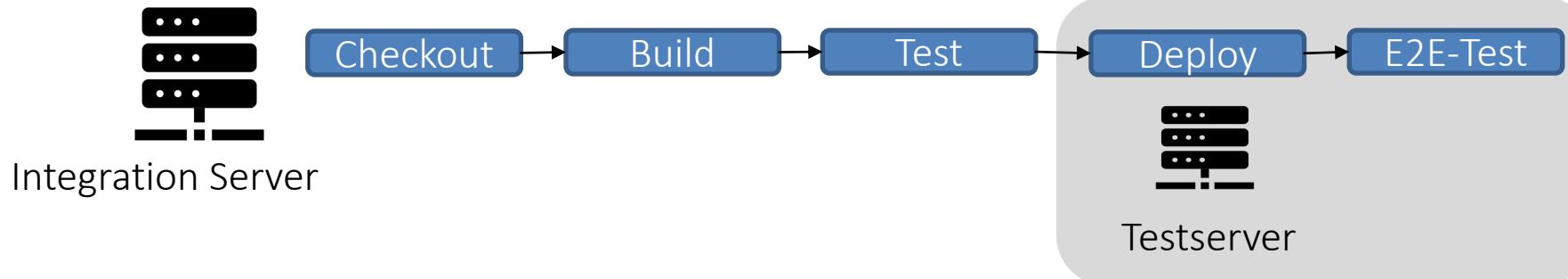
Alle sehen was gerade passiert – CI Server Frontend



9 Praktiken von CI

1. Ein einziges Repository für den Quellcode
2. Automatisiertes Bauen (Build) der Anwendung
3. Automatisiertes Testen der Anwendung
4. Bei jedem Commit wird der Main-Branch auf einer Integrationsmaschine gebaut und getestet
5. Das Bauen der Anwendung geht schnell und Entwicklerinnen erhalten schnell Feedback
6. „Kaputte“ Builds werden sofort gefixt oder rückgängig gemacht (Fix or Revert)
7. Alle sehen was gerade passiert und welchen Status der Build hat
8. Die Anwendung wird in einem Klon der Produktionsumgebung getestet

Testen in einem Klon der Produktionsumgebung



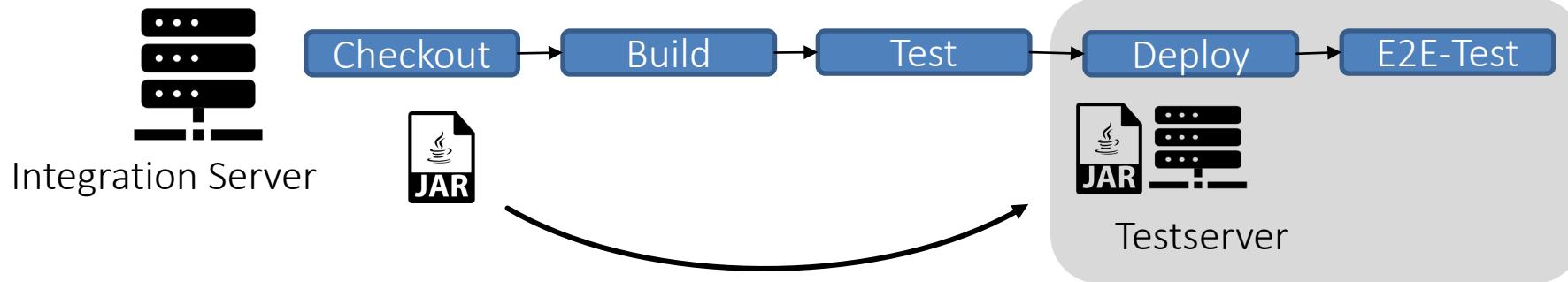
- Testet man in einer anderen Umgebung als der Produktivumgebung, birgt jede Abweichung das Risiko, dass Testergebnisse nicht der Produktivumgebung entsprechen.
- Also: Setze die Testumgebung so auf, dass sie möglichst ähnlich wie die Produktivumgebung ist, dh. nutze das selbe Betriebssystem, Datenbanktechnologie, Bibliotheksversionen, ...
- Häufig werden heute **Virtualisierungs- und Containerisierungstechnologien** verwendet um möglichst exakte Replikationen der Umgebung zu generieren



9 Praktiken von CI

1. Ein einziges Repository für den Quellcode
2. Automatisiertes Bauen (Build) der Anwendung
3. Automatisiertes Testen der Anwendung
4. Bei jedem Commit wird der Main-Branch auf einer Integrationsmaschine gebaut und getestet
5. Das Bauen der Anwendung geht schnell und Entwicklerinnen erhalten schnell Feedback
6. „Kaputte“ Builds werden sofort gefixt oder rückgängig gemacht (Fix or Revert)
7. Alle sehen was gerade passiert und welchen Status der Build hat
8. Die Anwendung wird in einem Klon der Produktionsumgebung getestet
9. Die Anwendung wird nur einmal gebaut

Anwendung wird nur einmal gebaut

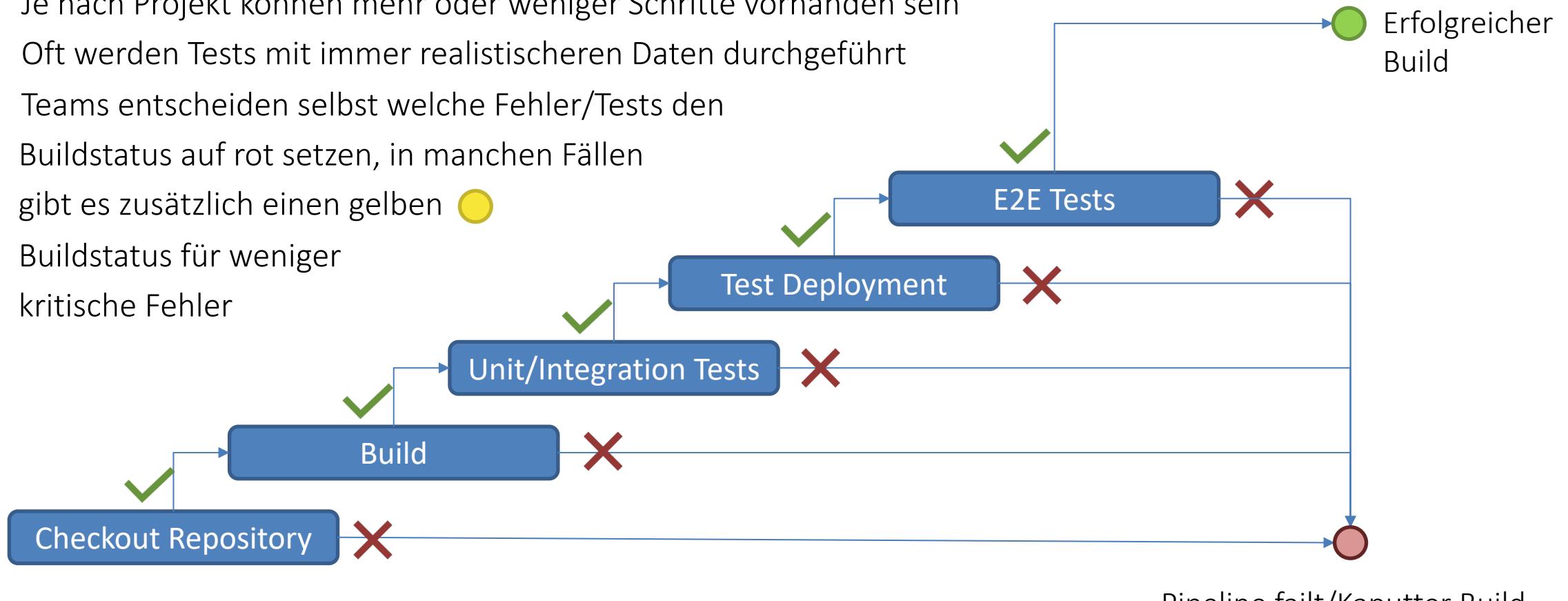


- Die Anwendung wird nur einmal gebaut und dann in verschiedenen Schritten wiederverwendet

```
build-backend:  
  stage: build  
  script:  
    - cd backend/  
    - mvn package  
  artifacts:  
    paths:  
      target/  
  
deploy-job1:  
  stage: deploy  
  script:  
    - java -jar target/packaged-app.jar
```

Zusammenfassung: CI-Pipeline

- Je nach Projekt können mehr oder weniger Schritte vorhanden sein
- Oft werden Tests mit immer realistischeren Daten durchgeführt
- Teams entscheiden selbst welche Fehler/Tests den Buildstatus auf rot setzen, in manchen Fällen gibt es zusätzlich einen gelben ● Buildstatus für weniger kritische Fehler



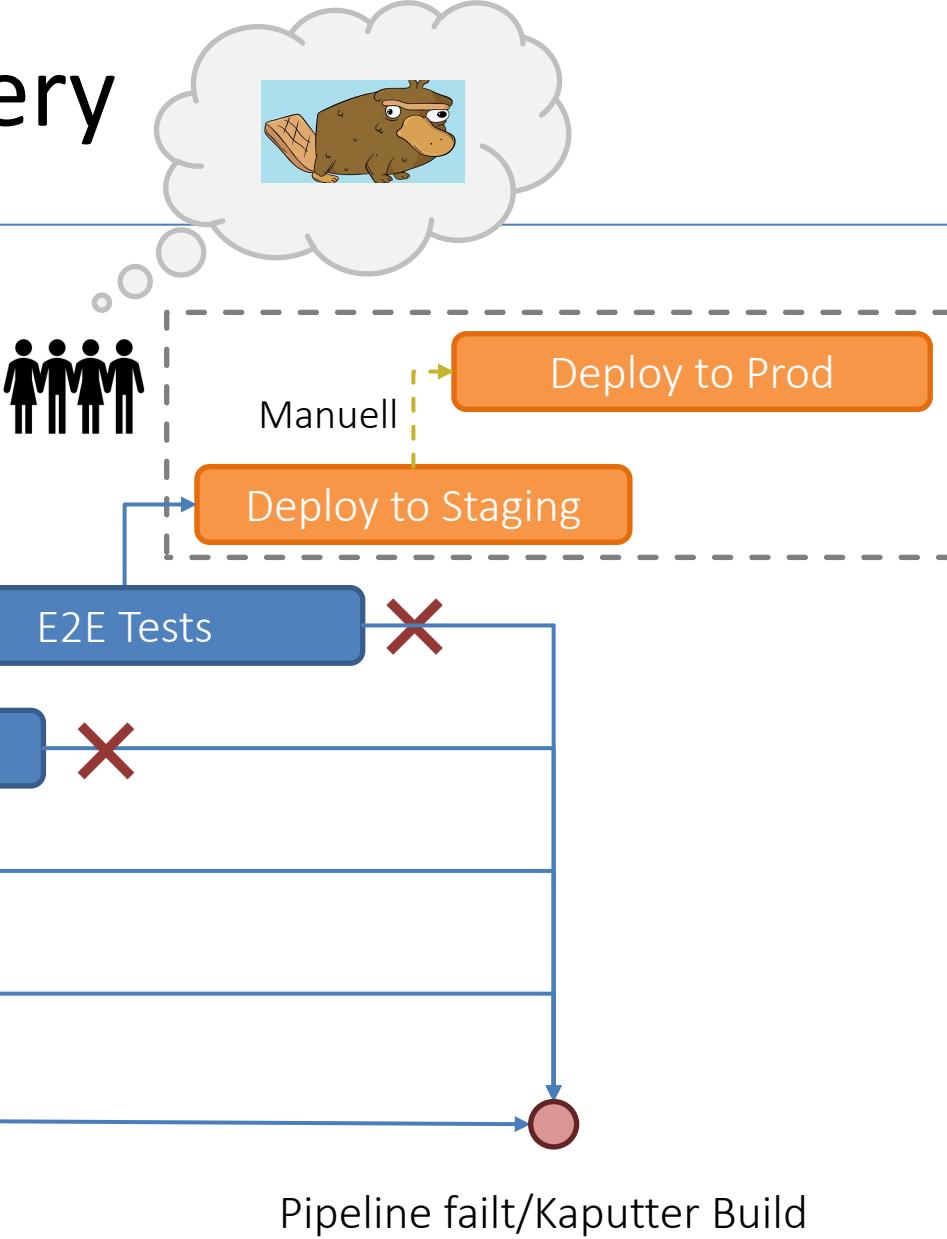
Continuous Delivery

Setzt dort an wo CI endet und zielt darauf ab, dass jederzeit eine funktionsfähige Version des Produkts deployt werden kann.

3 Voraussetzungen von CD

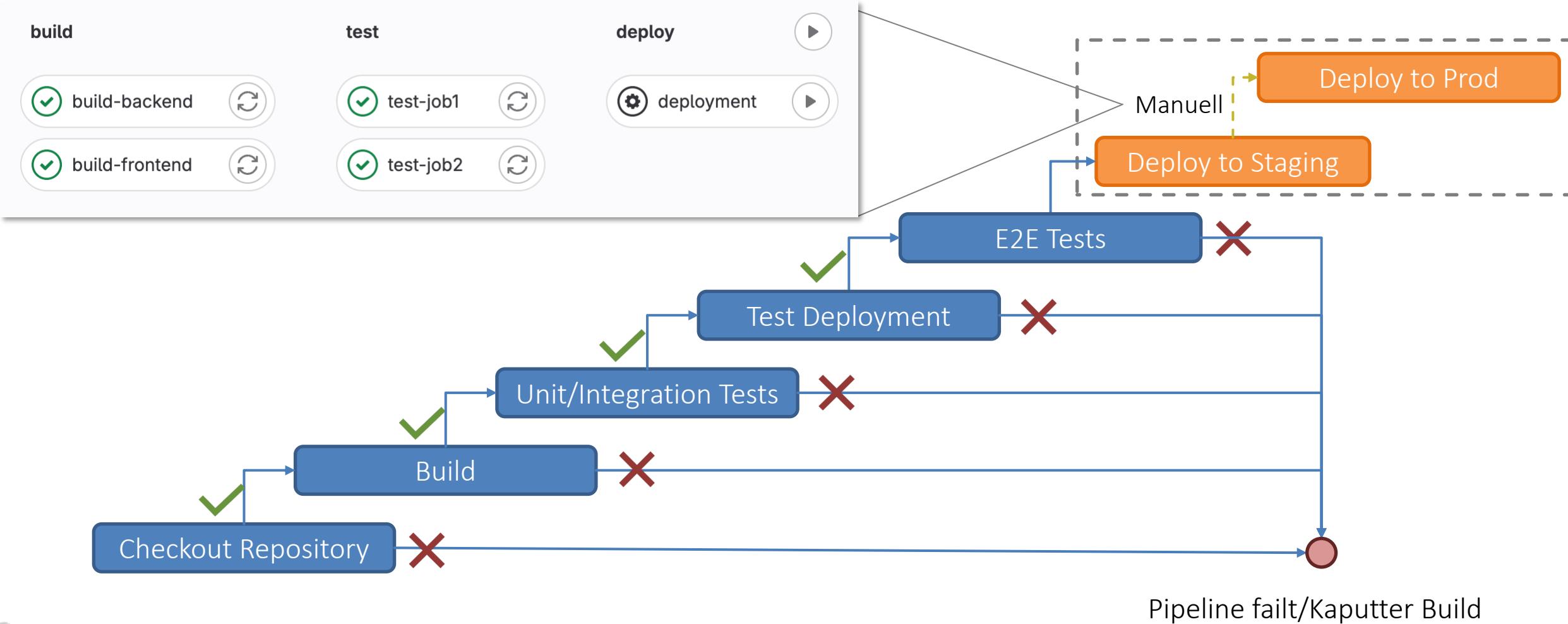
- Continuous Integration
- Continuous Testing
 - Am besten von Anfang an testen, nicht erst wenn das Feature fertig entwickelt ist
 - Tests müssen automatisiert bei jedem Commit ausgeführt werden
 - Automatisierte Tests müssen von jeder Entwicklerin auch lokal auf ihrem Rechner ausgeführt werden können
 - Kein Feature ist fertig entwickelt, wenn nicht alle automatisierten Unit-/Integration-/E2E-Tests entwickelt und erfolgreich durchlaufen sind
- Umfassendes Konfigurationsmanagement
 - Die Konfiguration der Umgebungen und Technologien, die zum Bauen, Testen und Deployen der Anwendung nötig sind, müssen im Repository vorhanden sein

CI/CD-Pipeline: Continuous Delivery



- An die CI-Pipeline schließt ein Schritt zum automatischen Deployment der Anwendung in eine Staging Umgebung an
- Stakeholder haben Zugriff auf die Anwendung und können so schnell und direkt Feedback geben
- In einem manuellen Schritt kann das automatisierte Deployment ausgelöst werden

CI/CD Pipeline: Continuous Delivery

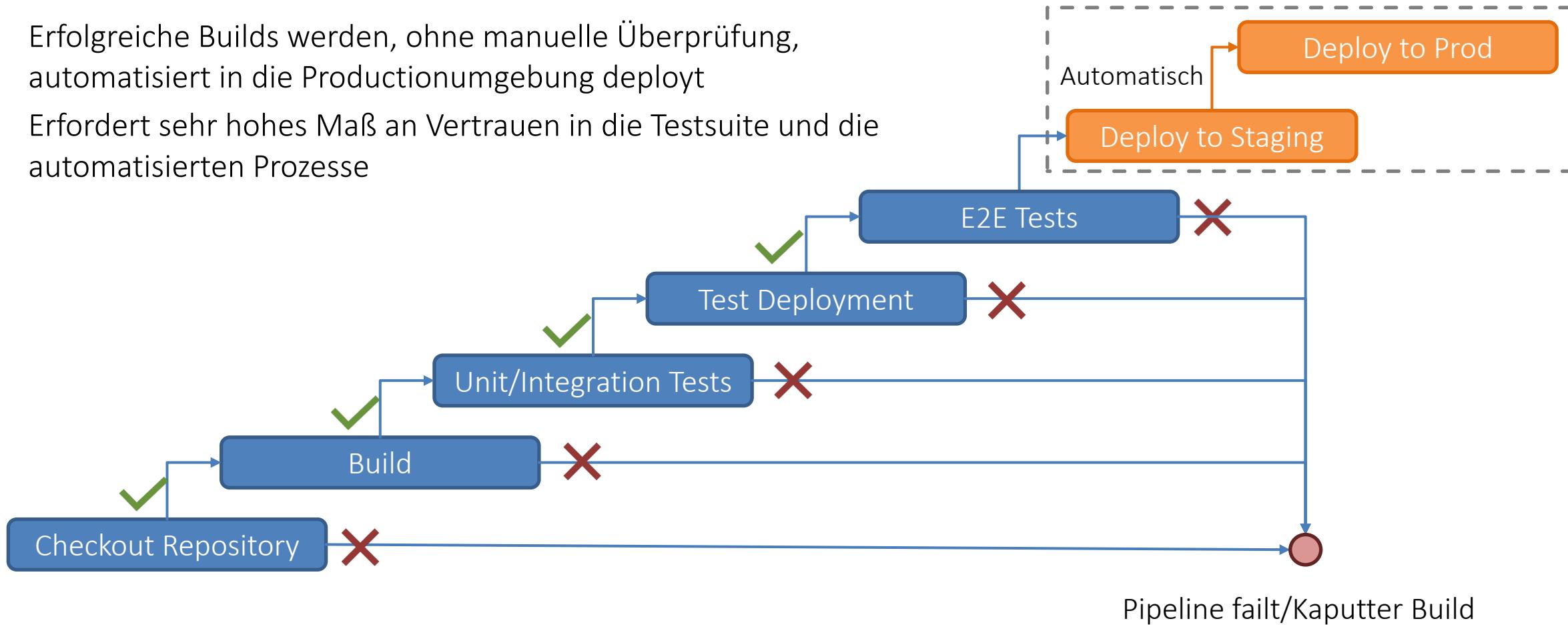


Continuous Deployment

Alle erfolgreichen Builds werden direkt in die Produktionsumgebung deployt.

CI/CD Pipeline: Continuous Deployment

- Erfolgreiche Builds werden, ohne manuelle Überprüfung, automatisiert in die Productionsumgebung deployt
- Erfordert sehr hohes Maß an Vertrauen in die Testsuite und die automatisierten Prozesse



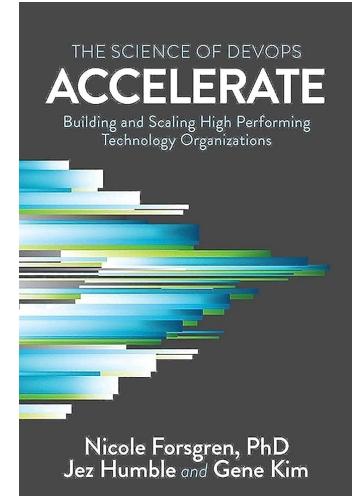
Und in der Praxis?

Häufige Probleme & Nachteile

- Lange Laufzeit der Pipeline, besonders wenn ausgiebig getestet wird
- Wenig oder kein Vertrauen in den Buildstatus = wenig bis keine Vorteile durch CI/CD
- Andererseits: Zuviel Vertrauen in den Buildstatus kann dazu führen, dass, vor allem bei Codereviews, weniger genau kontrolliert wird
- Der Aufwand, der durch zusätzliche Infrastruktur und deren Konfiguration entsteht, kann enorm sein und erfordert zusätzliche personelle und finanzielle Ressourcen

Viele Wege führen zum Ziel...

- In der Praxis ist mit CI/CD oft ausschließlich die Pipeline gemeint
- Die vorgestellten Prinzipien und Praktiken entsprechen Idealen bzw. Richtlinien
- Es existieren viele verschiedene Formen CI/CD zu implementieren:
 - Jedes Team entscheidet selbst welches Setup, welche Branching-Strategie und welcher Automatisierungsgrad richtig ist
 - Entscheidungen hängen von verschiedenen Faktoren ab: Was für ein Produkt wird entwickelt? Wer ist beteiligt? Sind alle Teammitglieder bereits sehr erfahren oder ist es ein „junges“ Team, mit wenig Erfahrung?
- Allgemein gilt: viele der vorgestellten Aspekte stehen in engem Zusammenhang mit verbesserter Performance von Teams und Organisationen



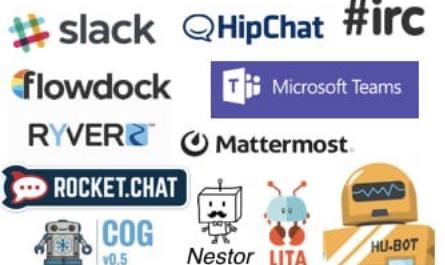
Tools

Collaborate

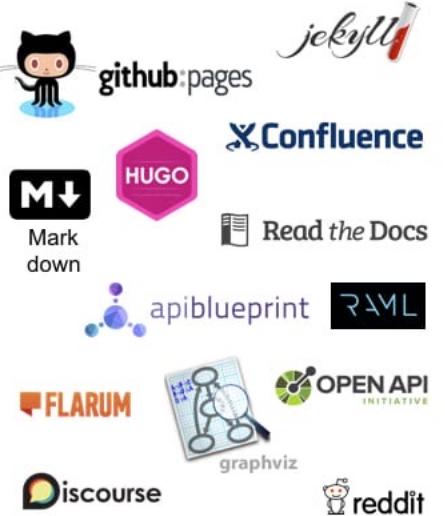
Application Lifecycle Mgmt.



Communication & ChatOps



Knowledge Sharing



Build

SCM/VCS



CI



Build



Database Management

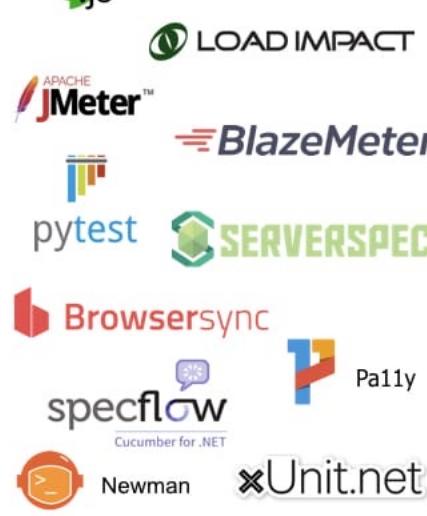


Test

Testing



Artefact Management



Deploy

Deployment



Config Mgmt./Provisioning



Cloud / IaaS / PaaS



Orchestration & Scheduling

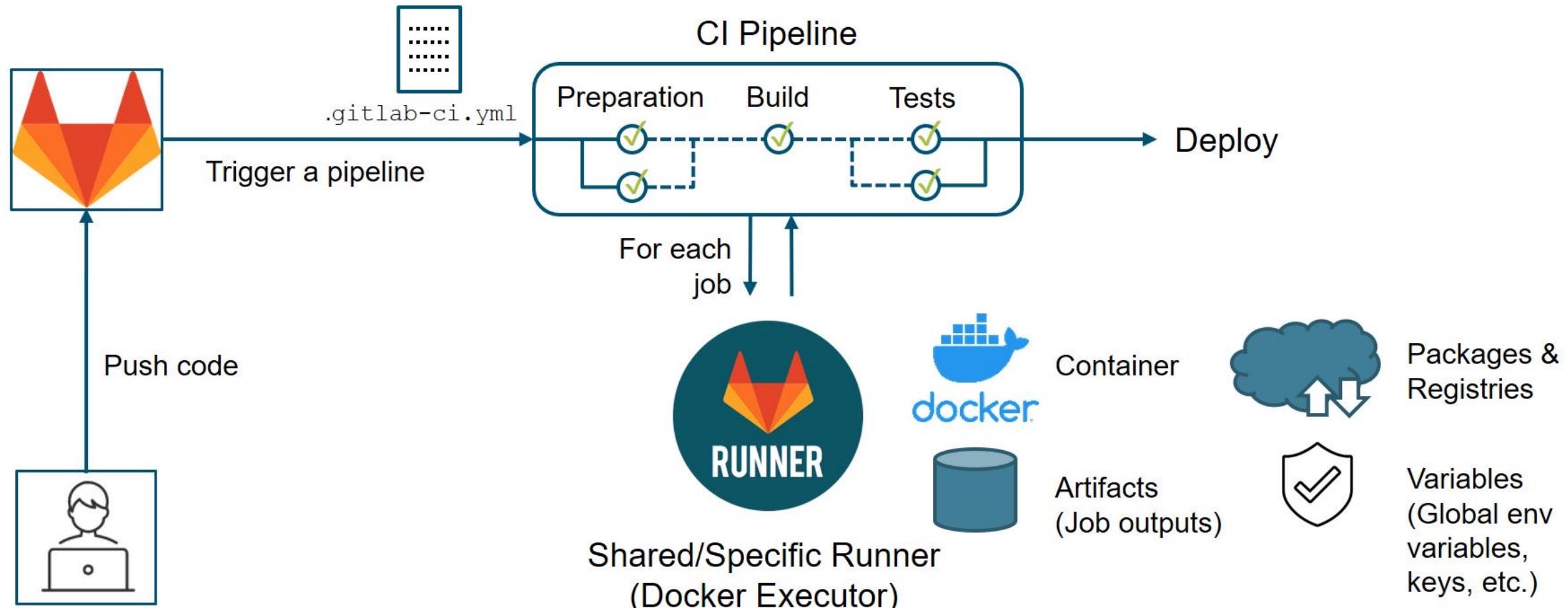


BI / Monitoring / Logging



Run

GitLab CI (1): Workflow



GitLab CI (2): Pipelinedefinition

Jobs

```
↳ .gitlab-ci.yml
1   image: maven:3-openjdk-11-slim
2   stages:
3     - build
4     - test
5     - deploy
6
7   build-frontend:
8     stage: build
9     script:
10    - cd frontend/
11    - npm run build
12
13   build-backend:
14     stage: build
15     script:
16    - cd backend/
17    - mvn package
18
19   test-job1:
20     stage: test
21     script:
22    - mvn test
23
24   deployment:
25     stage: deploy
26     image: nginx:alpine
27     when: manual
28     script:
29       - ./deploy.sh
```

Umgebung in der die Jobs ausgeführt werden die kein anderes Image angegeben haben

Stages die nach einander ausgeführt werden

Jobs sind definiert durch die stage zu der sie gehören und ein Script das ausgeführt wird. Zusätzlich können viele verschiedene Eigenschaften konfiguriert werden, z.B.: wie der Job ausgeführt wird (hier: manuell), für welche Branches, Variablen, etc.

<https://docs.gitlab.com/ee/ci/yaml/>

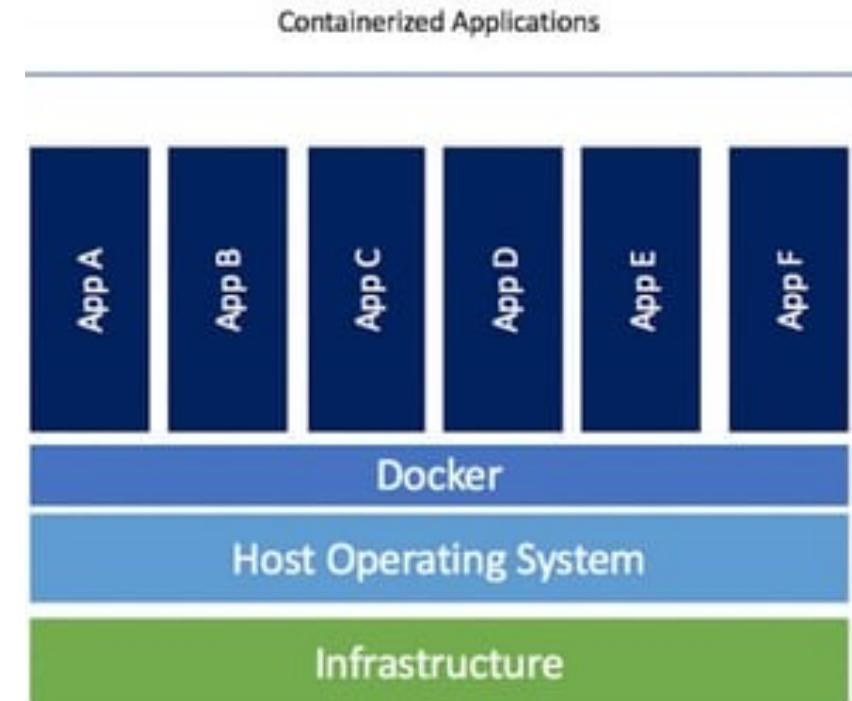
Docker



Docker

Docker ist eine offene Plattform zum Erstellen, Ausführen und Verwalten von verteilten Anwendungen mit Hilfe von Containern.

- Container sind ein standardisierter Weg um eine Anwendung mit all ihren Abhängigkeiten so zu verpacken, dass sie überallhin (wo die Container-Engine unterstützt wird) bewegt und ausgeführt werden kann
- Docker-Container virtualisieren die Betriebssystemebene, im Gegensatz zu VMs, die Hardware virtualisieren.
- Container teilen sich das gleiche Betriebssystemkern und isolieren die Anwendungen in einer benutzerdefinierten Umgebung.



Docker (2): Kernkonzepte

Dockerfile: eine Textdatei mit Anweisungen zum automatischen Erstellen eines Dockerimages.

Dockerimage: eine unveränderliche Datei, die eine Momentaufnahme einer Anwendung und ihrer Umgebung enthält, aus der Dockercontainer erstellt werden.

Dockercontainer: eine laufende Instanz eines Dockerimages, die die Anwendung und ihre Umgebung in einer isolierten Umgebung ausführt.

Docker Engine: die Laufzeit- und Verwaltungsumgebung, die zum Erstellen und Ausführen von Dockercontainern benötigt wird.

Docker Registry: ein Speicher- und Verteilungssystem für Dockerimages, das das Hochladen und Herunterladen von Images ermöglicht.

Docker (3): Dockerfile & Image

Definieren des Images im Dockerfile

```
FROM <baseimage>
WORKDIR <working directory>
COPY <copy local files to image>
RUN <command>
EXPOSE <port>
ENTRYPOINT <executed on container start>
```

👉 Dockerfile > ...

```
1  FROM node:lts-alpine
2  WORKDIR /app
3  COPY . .
4  RUN npm install
5  EXPOSE 5173
6  ENTRYPOINT ["npm", "run", "dev"]
```

Bauen des Images

```
docker build -t <image_name> <dockerfile location>
```

Anzeigen von Images:

```
docker image ls oder docker images
```

```
[+] vue-frontend git:(main) ✘ sudo docker build -t vue-dev-example .
[+] Building 24.8s (9/9) FINISHED
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 143B
=> [internal] load metadata for docker.io/library/node:lts-alpine
=> [1/4] FROM docker.io/library/node:lts-alpine
=> [internal] load build context
=> => transferring context: 497.18kB
=> CACHED [2/4] WORKDIR /app
=> [3/4] COPY . .
=> [4/4] RUN npm install
=> exporting to image
=> => exporting layers
=> => writing image sha256:73a0662badd86a7a7b72adcefadf8cf71033f3bd21185ff381dfe5601dc3912
=> => naming to docker.io/library/vue-dev-example
[+] vue-frontend git:(main) ✘ docker image ls
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
vue-dev-example     latest   73a0662badd8  20 seconds ago  908MB
```

Docker (4): Container

Erstellen und Ausführen eines Containers

```
docker run -p <host port>:<container port> <image>
```

```
[→ vue-frontend git:(main) ✘ docker run -p 127.0.0.1:5173:5173 vue-dev-example
> vue-frontend@0.0.0 dev
> vite --host

VITE v3.1.3  ready in 327 ms
→ Local:  http://localhost:5173/
→ Network: http://172.17.0.2:5173/
```

Anzeigen von laufenden Containern

```
docker container ls oder docker ps
```

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
|--------------|-----------------|---------------|--------------------|-------------------|--------------------------|-----------------|
| 26305a292aa3 | vue-dev-example | "npm run dev" | About a minute ago | Up About a minute | 127.0.0.1:5173->5173/tcp | agitated_liskov |

Docker (3): Docker-Compose

- Docker Compose ist ein Werkzeug zur Definition und Ausführung von Mehrcontainer-Dockeranwendungen
- In YAML-Dateien werden Dienste und deren Konfiguration spezifiziert
- Ermöglicht das simultane Starten, Stoppen und Wiederaufbauen aller Dienste einer Anwendung mit einem einzigen Befehl:
docker-compose up und **docker-compose down**

```
git docker-compose-prod.yaml
1   version: "3.7"
2
3   services:
4     frontend:
5       container_name: vue-prod
6       image: mailach/vue-prod-example
7       ports:
8         - 127.0.0.1:5124:80
9       restart: unless-stopped
10
11    backend:
12      container_name: spring-prod
13      image: mailach/spring-prod-example
14      ports:
15        - 127.0.0.1:8124:8080
16      restart: unless-stopped
17      networks:
18        - backend
19
20    db:
21      container_name: postgres_db
22      image: postgres
23      volumes:
24        - ./postgres-database:/var/lib/postgresql/data
25      restart: always
26      ports:
27        - 127.0.0.1:1432:5432
28      networks:
29        - backend
30
31    networks:
32      backend:
```

Was Sie mitgenommen haben sollten:

- Was ist DevOps und aus welcher Ausgangssituation heraus wurde es entwickelt?
- Diskutieren Sie X Ziele/Prinzipien von DevOps
- Welche Probleme können in Projekten ohne Continuous Integration auftreten?
- Was sind die Voraussetzungen von Continuous Delivery?
- Was ist der Unterschied zwischen Continuous Delivery und Continuous Deployment?
- Welche Probleme und Nachteile gibt es in der Praxis?
- Was ist Docker? Wofür wird es eingesetzt und was ist der Unterschied zu virtuellen Maschinen?

Zum Nachdenken:

- Wann sollte die CI/CD Pipeline im Projekt stehen?
- In welchen Szenarien ist Trunk Based Development sinnvoll und wann nicht?
- Wer profitiert von CI/CD und warum?