



CI/CD & DevOps

Struktur der Veranstaltung

 Let's create a Startup

Continuous Integration

Continuous Delivery

Continuous Deployment

Best Practices

Fazit

Noch Fragen?

Lernziele der Veranstaltung

Was solltest du mitnehmen?

 Integration, Delivery, Deployment

CI/CD Konzepte

Sinnhaftigkeit von CI/CD

Einsatzmöglichkeiten im Beleg



**Let's create a
Startup**

Dresden Tech Designs

- Startup neben der Uni
- Maßgeschneiderte Softwareprojekte
- Von Organisation über Entwicklung bis Bereitstellung und Betrieb
- Alles aus einer Hand



Unsere Situation

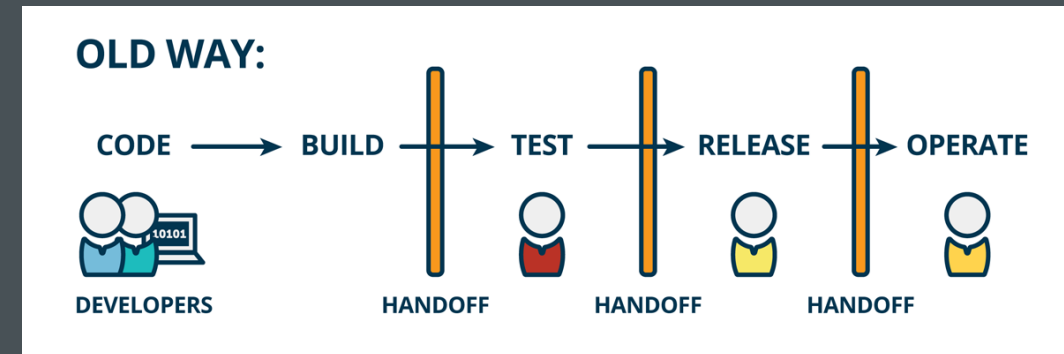
Das sind wir

- Wir
 - sind ein großes Team
 - arbeiten kollaborativ
 - arbeiten an mehreren Funktionen gleichzeitig
 - haben mehrere Arbeitsstände, je nach Feature
 - arbeiten nach der „DevOps“ Kultur
 - wollen schnelles Feedback und kontinuierlich hohe Qualität für unsere Software

Entwickeln und Betreiben

„Gängiges“ Arbeiten

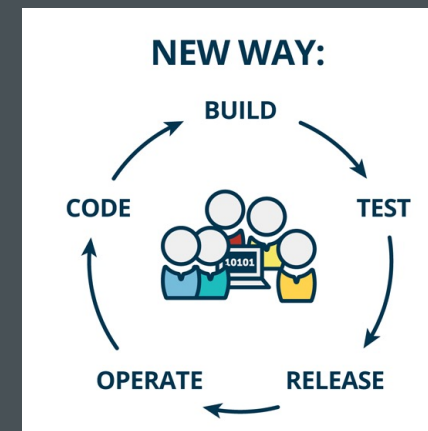
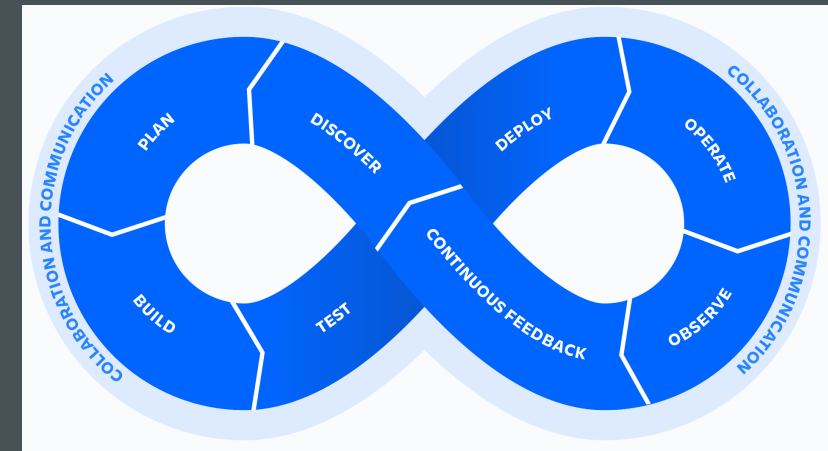
- Oft wird in einzelnen „Silos“ gearbeitet
- Entwicklungs- und Betriebsteams sind nur für ihren Bereich zuständig
- Wenig Prozesse werden automatisiert
- Bereitstellung erfolgt am Ende der Entwicklung
- Das führt möglicherweise zu
 - Kommunikationsproblemen
 - Langsamer Bereitstellung der entwickelten Software
 - Ineffizienten Prozessen



DevOps

Unser Prinzip

- Philosophie für Entwicklung und Betrieb von Software
- Entwicklung und Betrieb vereinen
- Verwendung unterschiedlicher Tools und Praktiken
- Dauerhafte Zusammenarbeit von Entwicklungs- und operativen Teams
- Wir wollen Prozesse der Entwicklung und Bereitstellung automatisieren / beschleunigen
- Etablieren von ständiger Kommunikation / Zusammenarbeit



Unsere Situation

Kundenauftrag!

- Der Kunde möchte eine API zum Verwalten von TO-Dos
- Als Programmiersprache soll Python gewählt werden
- TO-Dos sollen in einer Datenbank gespeichert werden
- TO-Dos sollen hinzugefügt, abgerufen, gelöscht und geupdatet werden
- -> CRUD-Operationen



Auf was müssen wir uns vorbereiten?

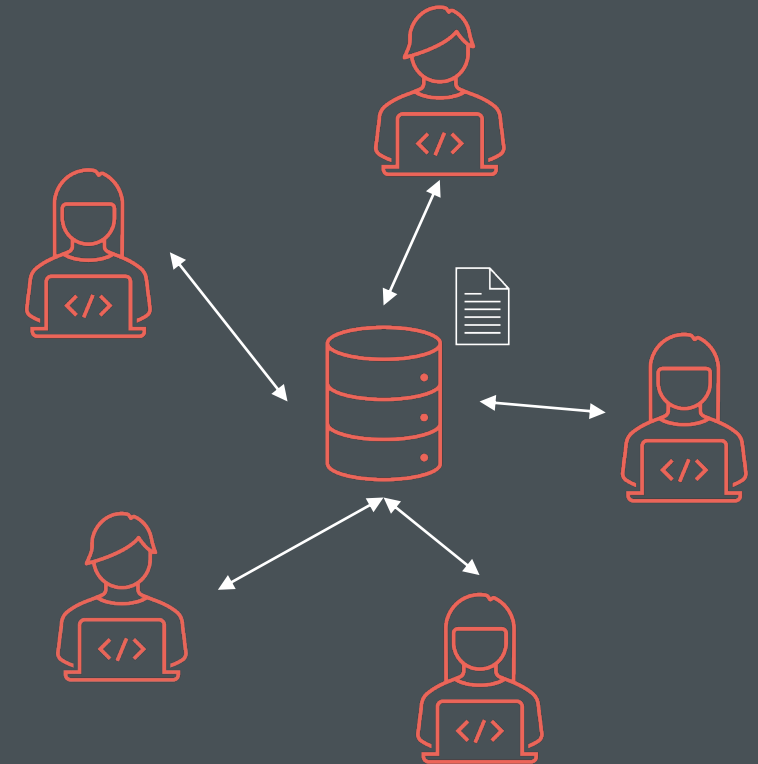
- Kollaborative Arbeit
- Schnelles Feedback zur Qualität unserer Arbeit
- Händische Arbeit minimieren
- Denken in kleinen Releases / Versionen
- Entwickeln und Testen (Integration)
- Bereitstellen (Delivery)
- Installieren / Ausliefern (Deployment)

- Was kann bei kollaborativer Arbeit an Software schief gehen?
- Welchen Herausforderungen können wir begegnen?
- Was macht uns die Arbeit leichter?

Herausforderungen

Kollaboratives Arbeiten

- Synchronisierung untereinander
- Austausch von Dateien
- Aktuellsten Stand abrufen
- Simultanes Arbeiten mit möglichst geringem Aufwand
- Wer hat wann, was geändert?
- Funktioniert mein Code noch nach Änderung?





Kick-Off!

GitHub Repository

Der Startpunkt

- Ausgangspunkt für uns ist ein Repository
- Repository ist Code- und Dokumentenverwaltung
- Soll unsere Arbeit vereinfachen
- Wir nutzen GitHub Actions
- [Beispiel Repository](#)

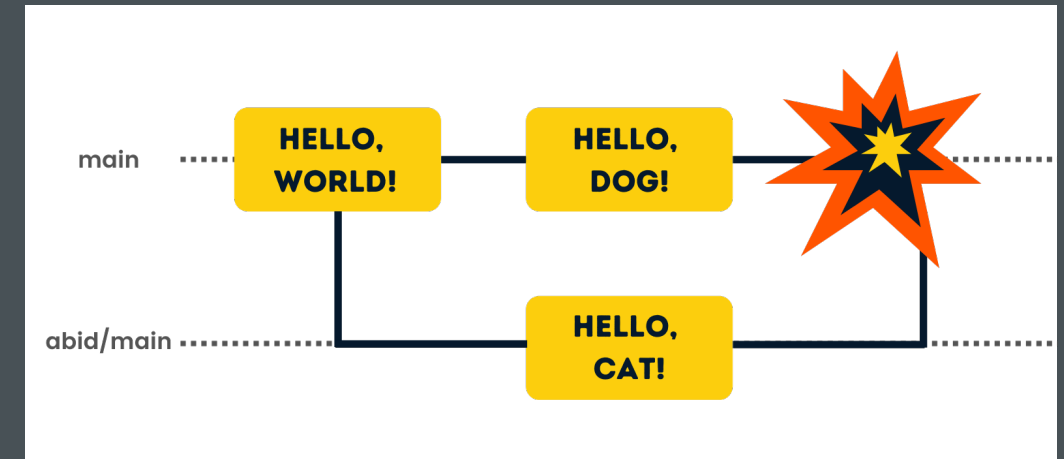


- Was kann bei kollaborativer Arbeit in einem Repository „schief“ gehen?
- Auf welche Probleme seid ihr bereits gestoßen?

GitHub Repository

Was kann passieren?

- Alle arbeiten gleichzeitig am Projekt
- Vielleicht ändern zwei Personen die gleiche Datei und pushen ihren Stand
- Oder es gab Änderungen auf einem Branch
- Dann gibt es zwei unterschiedliche Versionen des Codes
- -> Es kommt zu einem Konflikt (Merge Konflikt)
- -> Dateien im Repository passen also nicht immer "zusammen"





Continuous Integration

Continuous Integration

Was ist das?

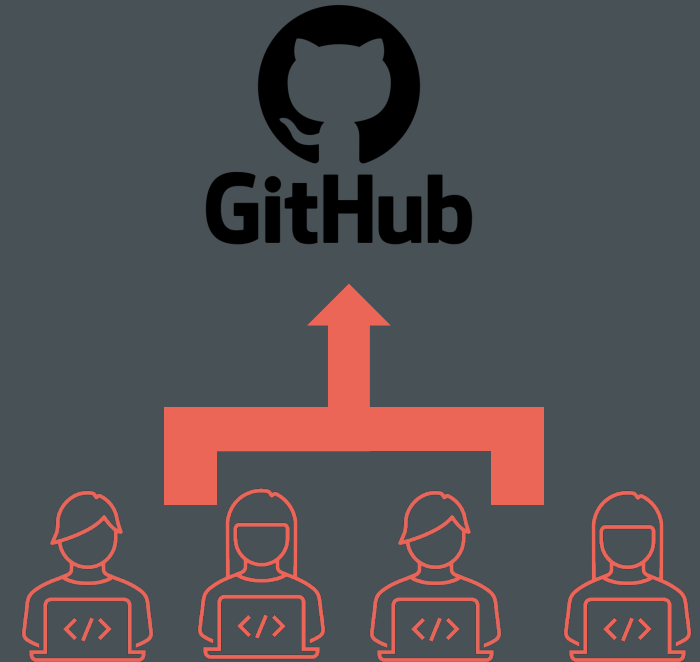


- Continuous Integration
- Änderungen werden in zentralen Stand „integriert“
- Integration von Änderungen stets geprüft (-> Konflikte werden erkannt)
- Zentraler Ort für Dokumente & Code ist das Repository (GitHub, GitLab, BitBucket, ...), ein Versionsverwaltungssystem
- Jeder kann zeitgleich mitarbeiten
- Wir commiten unsere Änderungen regelmäßig
- Testen von Änderungen an unserem Code

Continuous Integration

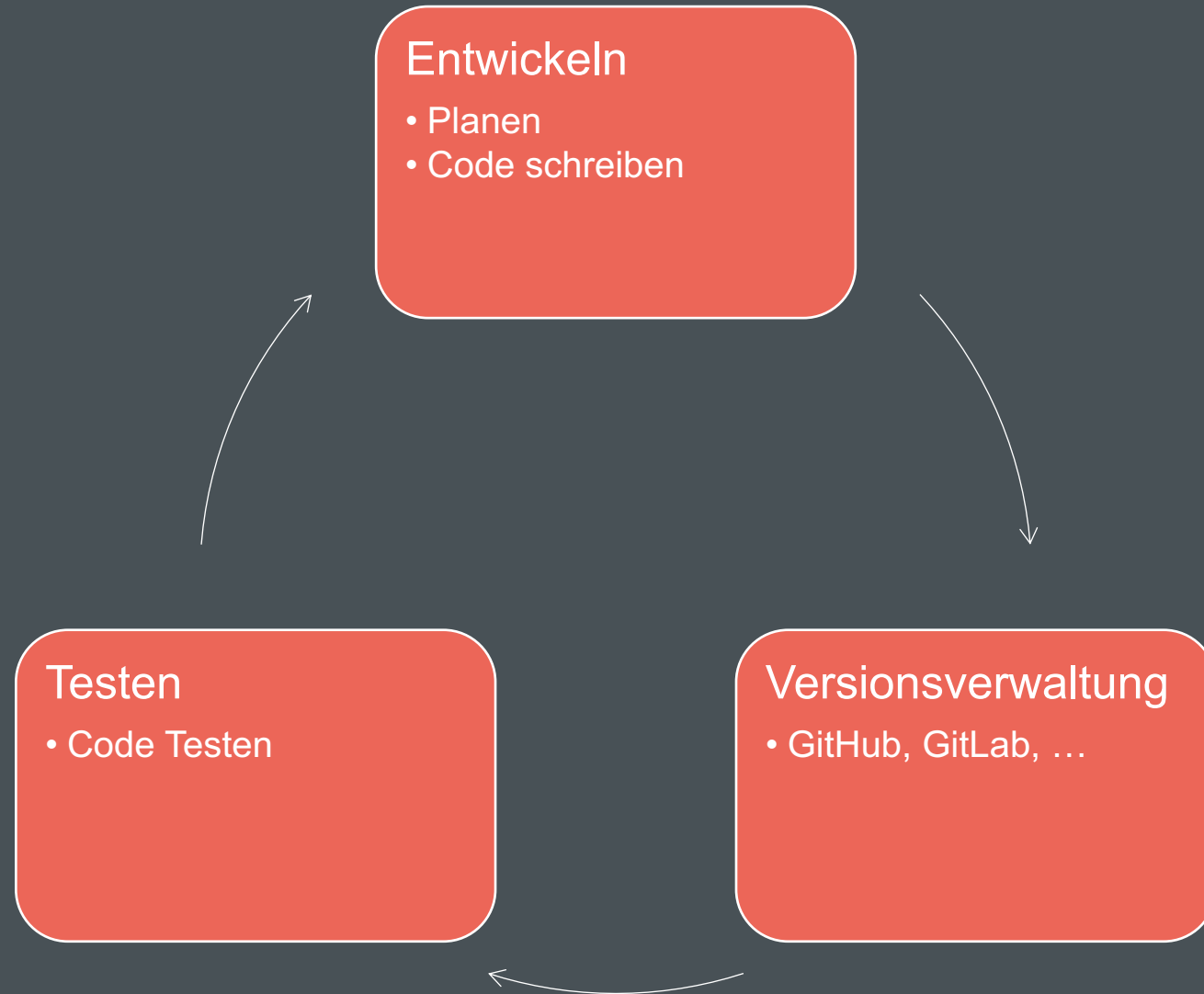
Was führen wir ein

- Nutzen eines Repositorys auf GitHub (Unser Repository)
- Entwickeln von Tests für unsere Software (Tests)
- Aktualisierung unserer Dokumentation / Dokumente
- Häufige Synchronisierung unserer Änderungen



Continuous Integration

Zyklus



Continuous Integration

Zyklus – Versionsverwaltung (GitHub)

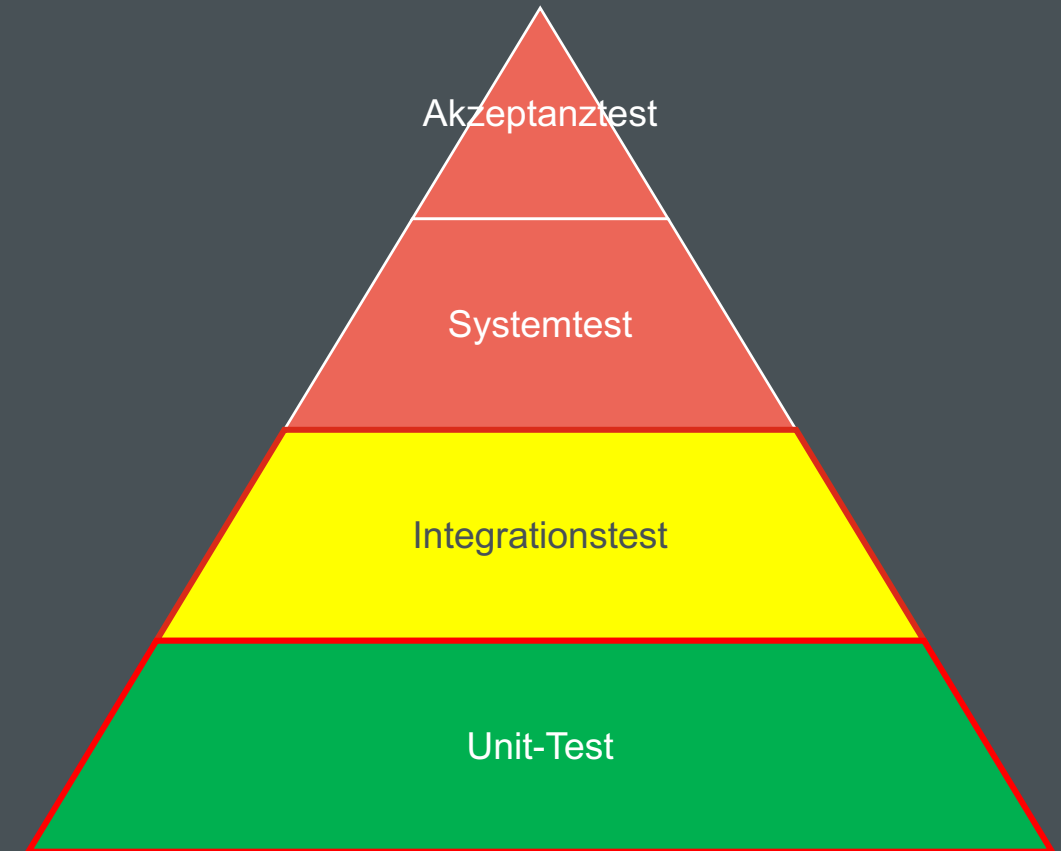
- Nutzen eines Repositorys auf GitHub ([Unser Repository](#))
- Wir entwickeln fleißig Features
- Wir pushen unsere Änderungen ins Repository oder Mergen einen Branch
- GitHub prüft ob unser Code integriert werden kann -> Passt der Code zum bestehenden Code
 - (Dokumente wie *.adoc*, *.md*, *.txt*, ... sind ebenfalls „Code“)
- Zusätzlich werden die Änderungen geprüft und durch Tests validiert -> geschieht automatisch



Continuous Integration

Zyklus - Tests

- Wir wollen unseren Code regelmäßig testen
- Unser Ziel ist automatisiertes Testen
- Dafür Nutzen wir
 1. Unit-Tests (Modul-Tests)
 2. Integrationstests



Continuous Integration

Zyklus – Tests – Unit-Test

- Prüfung einzelner Komponenten unseres Systems in Isolation
- Ziel: Fehler finden und Lauffähigkeit validieren
- Oft betrachten einzelner Klassen in einem Unit-Test

```
1 class ToDoBase(BaseModel):
2     title: str
3     description: str
4
5
6 class ToDoCreate(ToDoBase):
7     title: str
8     description: str
9
10
11 class ToDo(ToDoBase):
12     id: int
13     title: str
14     description: str
15     is_done: bool
16
17     class Config:
18         from_attributes = True
```

```
1 def test_create_todo_with_valid_data():
2     todo_data = {
3         "id": 1,
4         "title": "Test Todo",
5         "description": "This is a test todo",
6         "is_done": False
7     }
8     todo = ToDo(**todo_data)
9     assert todo.id == 1
10    assert todo.title == "Test Todo"
11    assert todo.description == "This is a test todo"
12    assert todo.is_done == False
13
```

Continuous Integration

Zyklus – Tests – Integrationstest

- Überprüfung des Zusammenspiels von Konfiguration mehrerer Module / Subsystemen
- Beispielsweise Datenbankkommunikation
- Zielstellung: Die einzelnen Teile meines Systems funktionieren miteinander
- -> Testen von Interaktionen mit anderen Modulen / (Sub)-Systemen

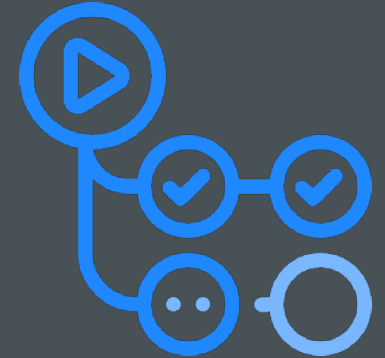
```
1 def get_todo(db: Session, todo_id: int):
2     """
3     Retrieves a todo item from the database based on the specified todo_id.
4
5     Parameters:
6         db (Session): The database session.
7         todo_id (int): The ID of the todo item to retrieve.
8
9     Returns:
10        Todo | None: The todo item with the specified ID, or None if not found.
11    """
12    return db.query(models.ToDo).filter(models.ToDo.id == todo_id).first()
```

```
1 def test_get_todo(client, db: Session):
2     # Test getting a todo
3     todo = models.ToDo(title="Test Todo", description="Test Description")
4     db.add(todo)
5     db.commit()
6
7     response = client.get(f"/api/todo/{todo.id}")
8     assert response.status_code == 200
9     assert response.json()["title"] == "Test Todo"
10    assert response.json()["description"] == "Test Description"
11    assert response.json()["is_done"] is False
```


GitHub Actions

Teil der CI

- Wir Nutzen GitHub Actions zur Automatisierung von Aufgaben und Prozesse im Kontext von Softwareentwicklung
- GitHub Actions bietet uns die Plattform um Aufgaben automatisiert durchzuführen
- Prozesse / Aufgaben sind beispielsweise:
 - Generieren von Dokumentation
 - Prüfen unserer Änderungen am Code -> Tests
 - Anlegen / Ändern von Issues
 - Bauen unserer Anwendung



GitHub Actions

Funktionsweise

- Wir definieren Schritte, welche ausgeführt werden sollen
- Alle Schritte bilden einen Workflow (auch genannt Pipeline)
- Dieser wird durch Events ausgelöst
- Tritt ein Event ein werden alle oder ein Teil der Schritte ausgeführt, je nach Konfiguration
- Events sind beispielsweise:
 - Commits
 - Merge Requests
 - Issue Aktionen
- Schritte können dann erfolgreich / nicht erfolgreich sein -> direktes Feedback

Continuous Integration

Gelöste Probleme / Gewinn

- ✓ Zentraler Ort -> Keine Dateien hin und her schieben
- ✓ Neuester Stand immer sichtbar -> Keine Frage wer die neuesten Änderungen hat
- ✓ Synchronisierung der Stände immer möglich -> Jeder sieht sofort den Stand der anderen
- ✓ Gleichzeitiges Arbeiten möglich -> Jeder kann simultan Arbeiten, ohne andere zu hindern
- ✓ Änderungsverlauf (Commits/Pull Requests) -> Änderungsverlauf stets präsent
- ✓ Funktionstest nach Änderung -> Fehler bei neuen Funktionen / Änderungen direkt erkannt

Unser Stand

GitHub Repository eingeführt, Tests

Das können wir

- ✓ Kollaborativ erfolgreich Arbeiten
- ✓ Softwarequalität durch Tests halten
- ✓ Fehler direkt erkennen
- ✓ Dokumentation generieren

Wir wollen zusätzlich

- ✗ Software bauen
- ✗ Software ausliefern

Fragen?



Continuous Delivery

- Wie kommen wir von Code zu etwas ausführbaren?
- Wie bringe ich den Code „zum laufen“?

Build Step

Wat is dat?

- Verpacken / transformieren unseres Codes in eine kleine ausführbare Einheit
- Zum Beispiel:
 - Kompilieren des Codes in Maschinencode (Java)
 - Ausführen von Transpilern zur Transformation von einer Sprache in eine andere (Typescript zu Javascript)
 - Installieren von Abhängigkeiten und Einrichten der Laufzeitumgebung (Python)
 - Verpacken des Codes in Docker Images (quasi alle Sprachen)



```
1 # Bauen einer NPM-Installation
2 npm run build
```



```
1 # Bauen einer Python Anwendung
2 python -m pip install --upgrade pip
3 pip install -r requirements.txt
4 python setup.py build
```



```
1 # Bauen einer Java Anwendung mit maven
2 mvn clean install
3 mvn package
```

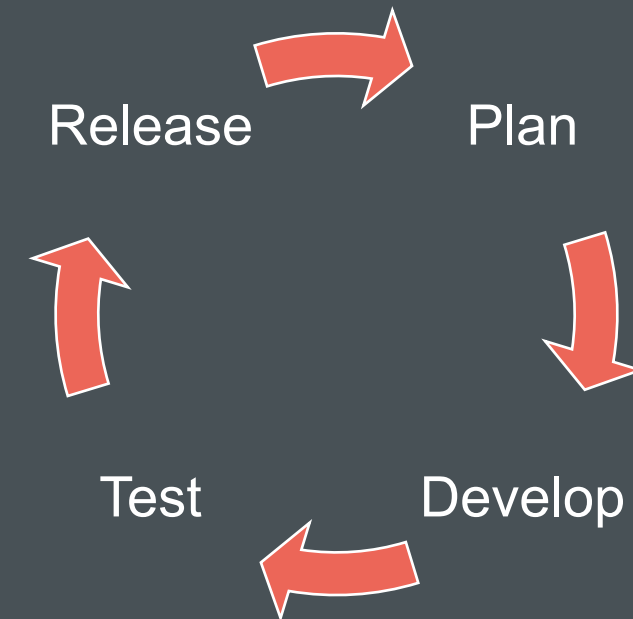


```
1 # Bauen unserer Anwendung als Image
2 docker build -t todo-api-python:latest .
```


Denken in Releases

Unser Ziel

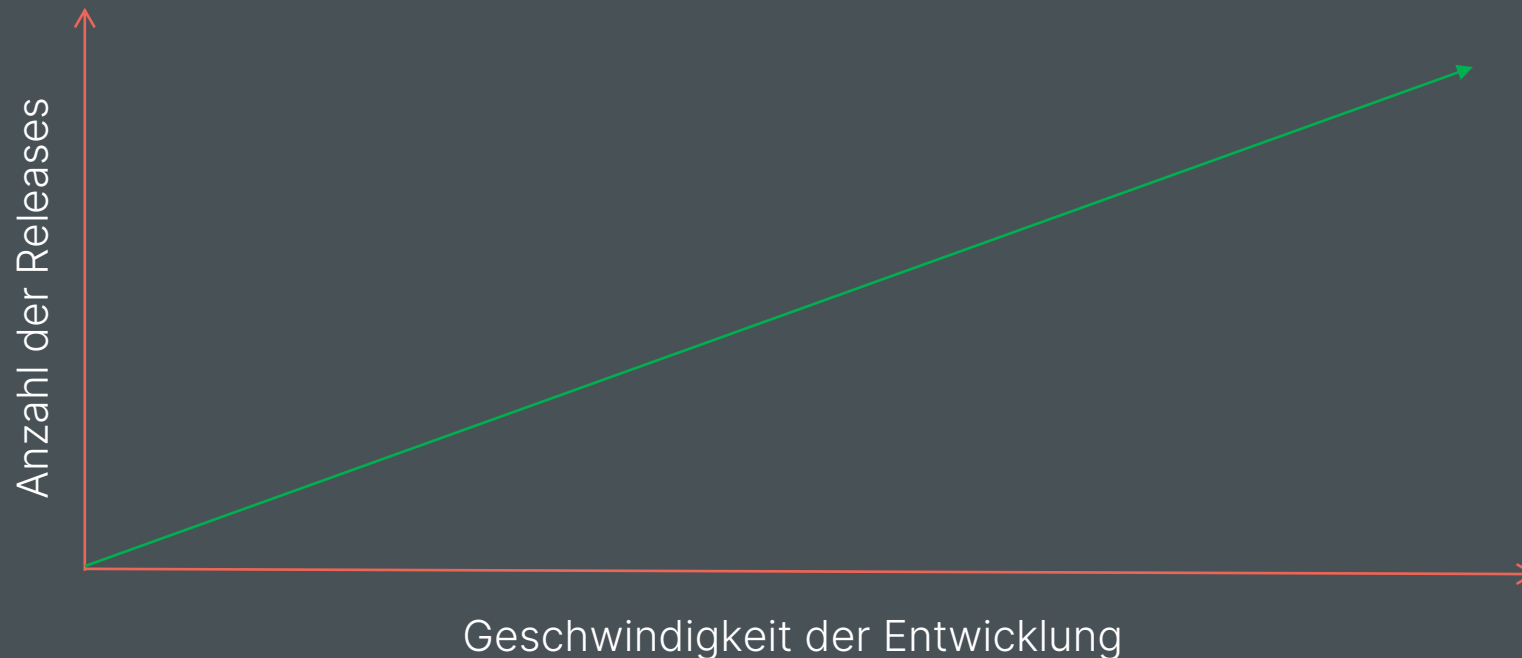
- Möglichst zügig wissen, ob wir in die richtige Richtung arbeiten
- -> Schnell Feedback einholen
- Technische Schulden verringern
- Qualität je Release dauerhaft halten, am besten sogar steigern
- Schnell Funktionen „an den Mann bringen“
- Unsere Software möglichst einfach und stressfrei ausliefern



Denken in Releases

Klein aber oho!

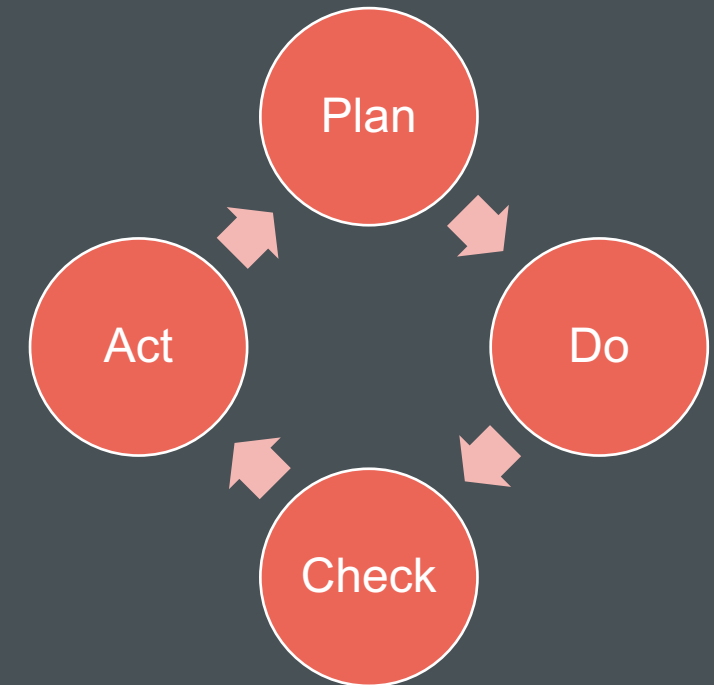
- Wir wollen oft neue Funktionen / Verbesserungen veröffentlichen
- Wir denken in „kleinen“ Releases -> Release = neue Version der Software



Denken in Releases

Schnelles Feedback

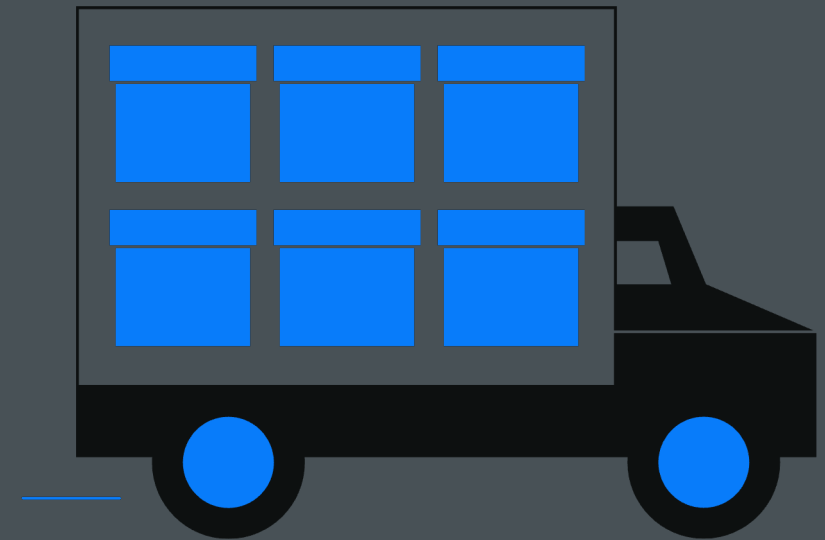
- Schnelles Feedback durch schnelle kleine Releases
- Der Kunde gibt uns direkt Meldung, ob wir gute Arbeit leisten
- Wir bekommen Feedback aus erster Hand
- Dadurch reduzieren wir unsere technischen Schulden



Continuous Delivery

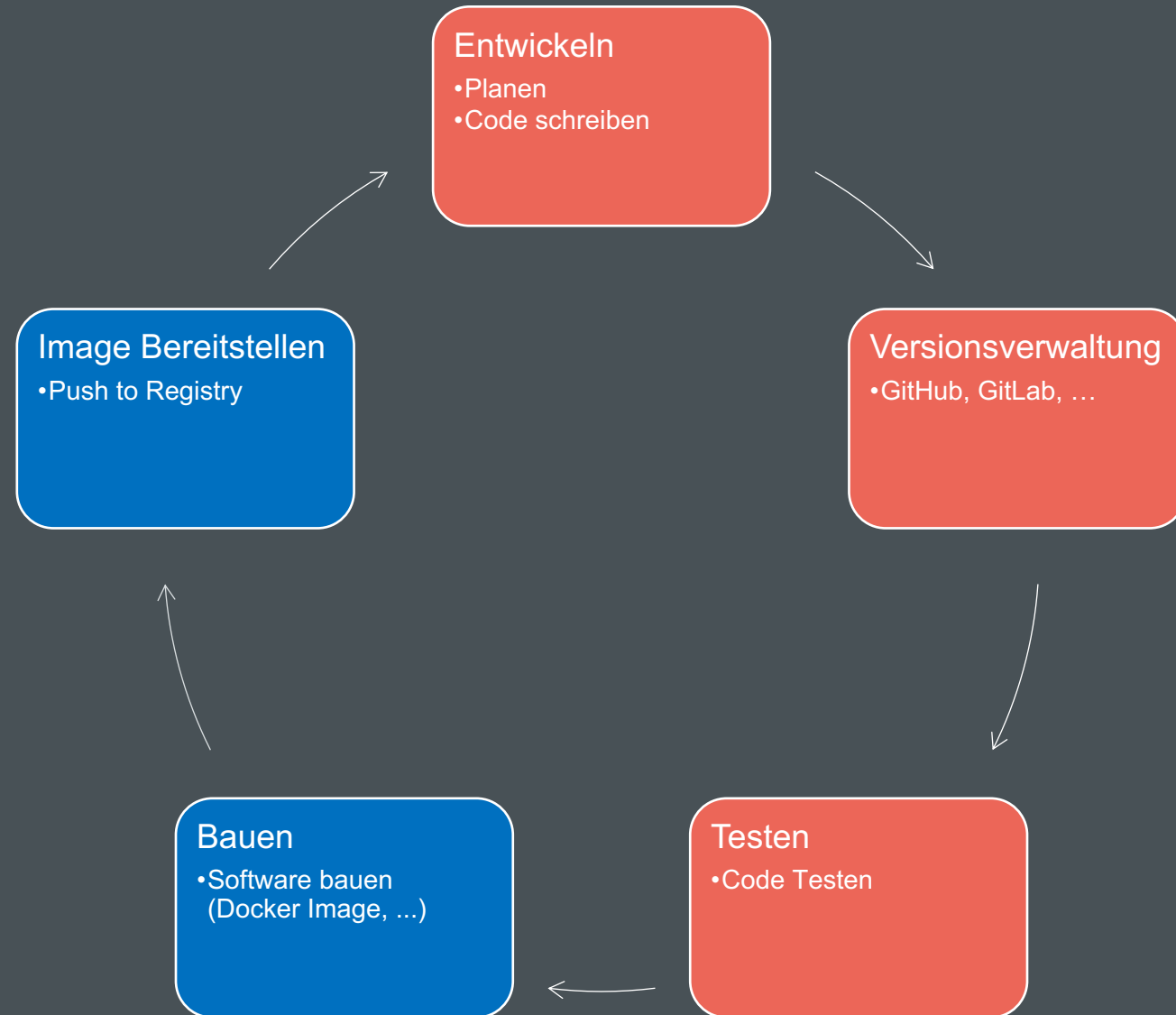
Aufbauen auf der CI

- Continuous Delivery
- Wir liefern die Software and den / die Kunden aus -> Delivery
- Wir nutzen dafür automatisierte Build-Prozesse
- Wir „verpacken“ oder „bauen“ unsere Software also in ein Packet
- Das Packet kann dann schnell ausgeliefert und abgerufen werden
- Alles geschieht automatisiert



Continuous Delivery

Zyklus



Continuous Delivery

Beispiel Build

```
build-and-push-image:
  runs-on: ubuntu-latest
  needs: test
  steps:
    - name: Checkout repo
      uses: actions/checkout@v4

    - name: Set up QEMU
      uses: docker/setup-qemu-action@v3

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v3

    - name: Login to GitHub Container Registry
      uses: docker/login-action@v3
      with:
        registry: ghcr.io
        username: ${github.repository_owner}
        password: ${secrets.REGISTRY_TOKEN}

    - name: Extract metadata (tags, labels) for Docker
      id: meta
      uses: docker/metadata-action@v5.5.1
      with:
        images: ghcr.io/${github.repository}

    - name: Build Image and push to registry
      uses: docker/build-push-action@v5
      with:
        context: ./todo-api-python
        platforms: linux/amd64,linux/arm64
        push: true
        tags: ${steps.meta.outputs.tags}
        labels: ${steps.meta.outputs.labels}
```

Build Schritt

Dockerfile

```
#
FROM python:3.11

#
WORKDIR /code

#
COPY ./requirements.txt /code/requirements.txt

#
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

#
COPY ./app /code/app

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

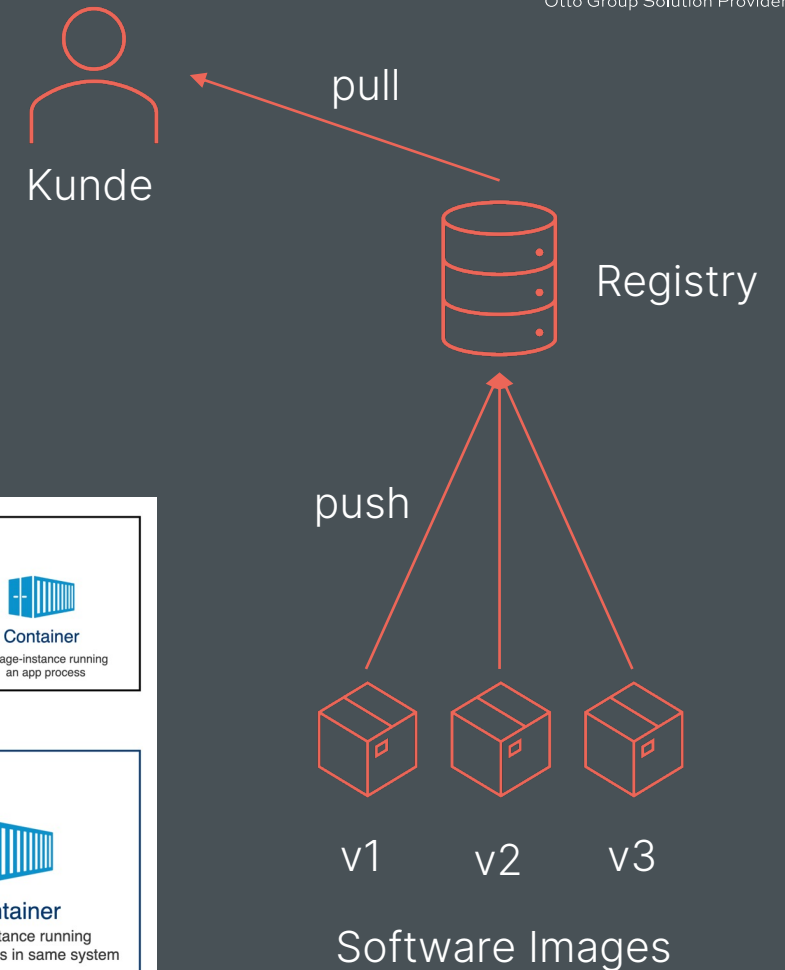
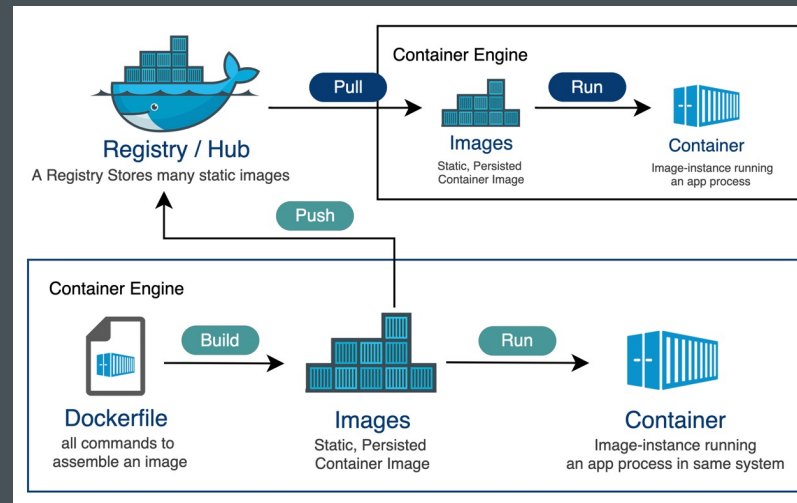
#
CMD wait-for-it -s db:5432 -t 5 && uvicorn app.main:app --host 0.0.0.0 --port 8000
```

- Wir verpacken die API als Docker Image
- Ausführen aller notwendigen Befehle zum Einrichten und Betreiben der Software

Continuous Delivery

Image bereitstellen - Registry

- Das Image wird in eine Registry hochgeladen
- Registry ist eine Server-Anwendung welche Images speichert
- Dort können Images von Clients abgerufen werden
- Beispiele sind
 - [Docker Hub](#)
 - [Amazon ECR](#)
 - [GitHub Packages](#) (unser Tool)



Continuous Delivery

Image Abrufen

- Unser Image wurde erfolgreich durch die CI/CD-Pipeline bereitgestellt
- Der Kunde kann unser Image dann nach Belieben aus der Registry abrufen
- Es kann gezielt eine bestimmte Version des Images abgerufen werden

```
○ ruben@OSP3737:~/dev$ docker login ghcr.io -u USERNAME --password-stdin
```

```
○ ruben@OSP3737:~/dev$ docker pull ghcr.io/edlaser/cicd-example:main
```

Befehl zum Abrufen (Vorher muss ein Login erfolgen)

Continuous Delivery

Gelöste Probleme / Gewinn

- ✓ Wir denken in kleinen Releases -> Hohe Geschwindigkeit, schnelles Feedback
- ✓ Wir bauen unsere Images automatisiert -> Weniger Handarbeit
- ✓ Unser Kunde kann neue Funktionalitäten schnell abrufen -> Schnelles Feedback
- ✓ Wir haben keinen händischen Aufwand neue Releases auszuliefern -> Noch weniger Arbeit

Unser Stand

GitHub Repository, GitHub Actions eingeführt

Das können wir

- ✓ Kollaborativ erfolgreich Arbeiten
- ✓ Software durch Tests Qualität halten
- ✓ Fehler direkt erkennen
- ✓ Dokumentation generieren
- ✓ Software bauen
- ✓ Software „bereit halten“

Wir wollen zusätzlich

- ✗ Software ausliefern

Fragen?



Continuous Deployment

- Wie kriegt der Kunde Zugriff auf unsere Software?
- Wie kommt unsere Software zum Kunden?

Continuous Deployment

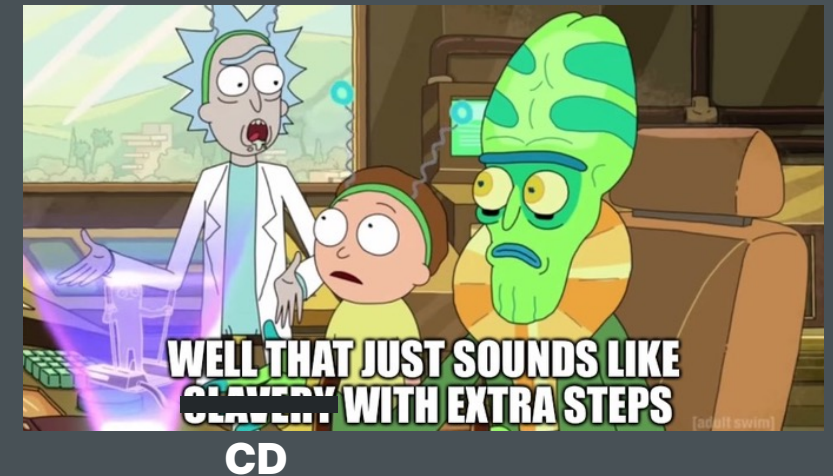
Was möchten wir erreichen?

- Wir wollen unsere Software für den Kunden bereitstellen
- Die Software soll direkt nutzbar sein, ohne dass der Kunde Images abrufen muss
- Der Kunde soll die Software nicht erst selbst installieren müssen
- Bereitstellen mit möglichst wenig Aufwand

Continuous Deployment

CD with extra Steps

- Continuous Deployment
- Wir deployen unsere Software
- Von Integration / Validierung (CI) über Lieferung (CD) bis zum produktiven Einsatz (CD)
- Software ist in einer fertigen Version direkt nutzbar
- Updates werden direkt im Betrieb eingespielt
- Der Kunde muss nicht installieren / einrichten



Continuous Deployment

Zyklus



Continuous Deployment

Gelöste Probleme / Gewinn

- ✓ Wir haben ohne Aufwand eine neue Version produktiv gesetzt
- ✓ Unser getesteter Code kann direkt verwendet werden
- ✓ Features sind direkt „sichtbar“
- ✓ Mehrere Releases am Tag möglich

Unser Stand

GitHub Repository, GitHub Actions eingeführt

Das können wir

- ✓ Kollaborativ erfolgreich Arbeiten
- ✓ Software durch Tests Qualität halten
- ✓ Fehler direkt erkennen
- ✓ Dokumentation generieren
- ✓ Software bauen
- ✓ Software „bereit halten“
- ✓ Software ausliefern

Wir wollen zusätzlich

- ✓ Nichts, wir sind happy!

- Haben wir durch CI/CD eventuell Nachteile?
- Gibt es Schwachstellen?

Fragen?

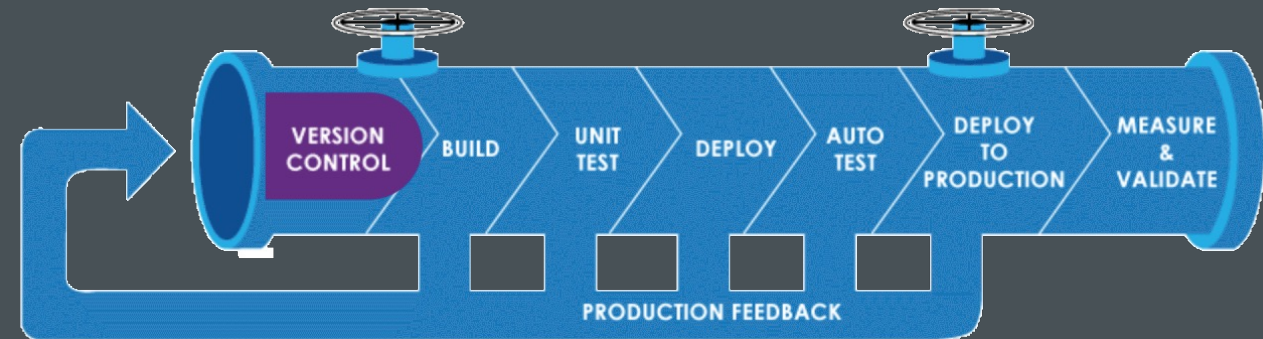


Best Practices

- Häufig Committen / Mergen
 - Alle bauen auf gleichen Grundlagen auf
 - Keine Aufwendige „große“ Integration
- Fehler direkt beheben
 - Keine Fehlerhafte Grundlage im Code
- Nicht zu viel und nicht zu wenig testen
 - Zu viele Tests führen zu schlechter Performance
 - Balance zwischen Abdeckung / Performance finden
- Alle aus dem Team nehmen teil
 - Jeder bekommt einen Überblick über das Geschehen und kann mithelfen
 - Gemeinsame Verantwortung schafft Vertrauen

CI/CD Pipeline

- Kombination alle Schritte wird auch Pipeline genannt
- Da alle Schritte einer Abfolge nach ausgeführt werden
- Sie laufen durch eine „Röhre“ / „Pipeline“





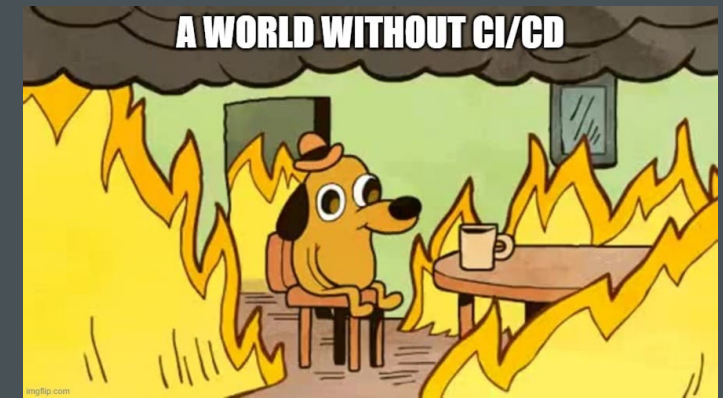
Fazit

- Nutzen eines Repositorys zur Verwaltung von Code und Dokumenten -> Continuous Integration
 - Vereinfacht unsere kollaborative Arbeit
- Bereitstellen unserer Software als Images -> Continuous Delivery
 - Software kann schnell bei neuen Versionen abgerufen werden
- Bereitstellen unserer Umgebung in Produktion (Live) -> Continuous Deployment
 - Neue Features/Fehlerbehebungen sind direkt nutzbar
- CI/CD automatisiert gängige Aufgaben in der Entwicklung und Betreiben von Software
 - Weniger händische Arbeit
 - Fusion von Entwicklung und Betreiben

Was haben wir erreicht?

Benefits in unserem Projekt

- Wir:
 - können unsere Dokumentation automatisiert bei Änderungen generieren
 - testen unsere Software bei Änderungen ohne händischen Aufwand
 - bauen unsere Software und stellen sie in einer Registry bereit
 - haben nervige händische Aufgaben automatisiert
 - erhalten direkt Feedback über unsere Codequalität





Wir haben es geschafft! Noch Fragen?

When you get a blue screen after you run your code but it says hello world.



ProgrammerHumor.io