

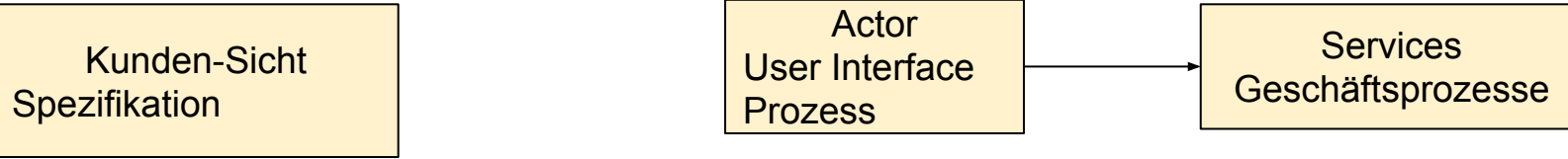
CI/CD in der Praxis

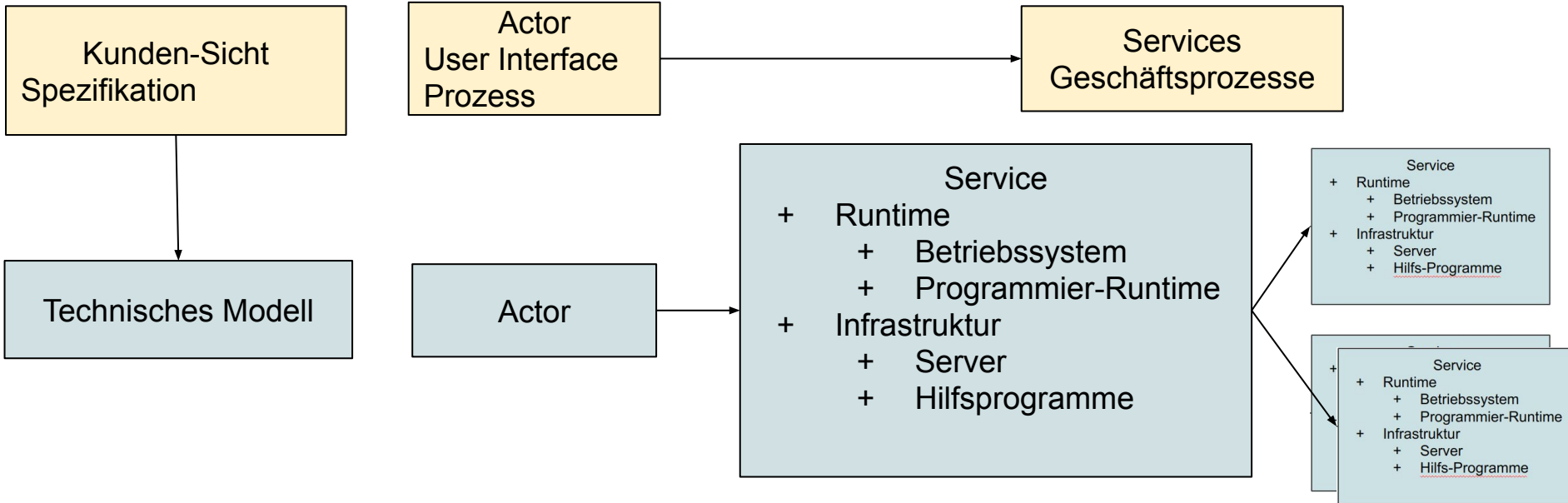
- Name
- Rolle im Unternehmen
- Themenbezogene Vorkenntnisse
- Aktuelle Problemsituation
- Individuelle Zielsetzung

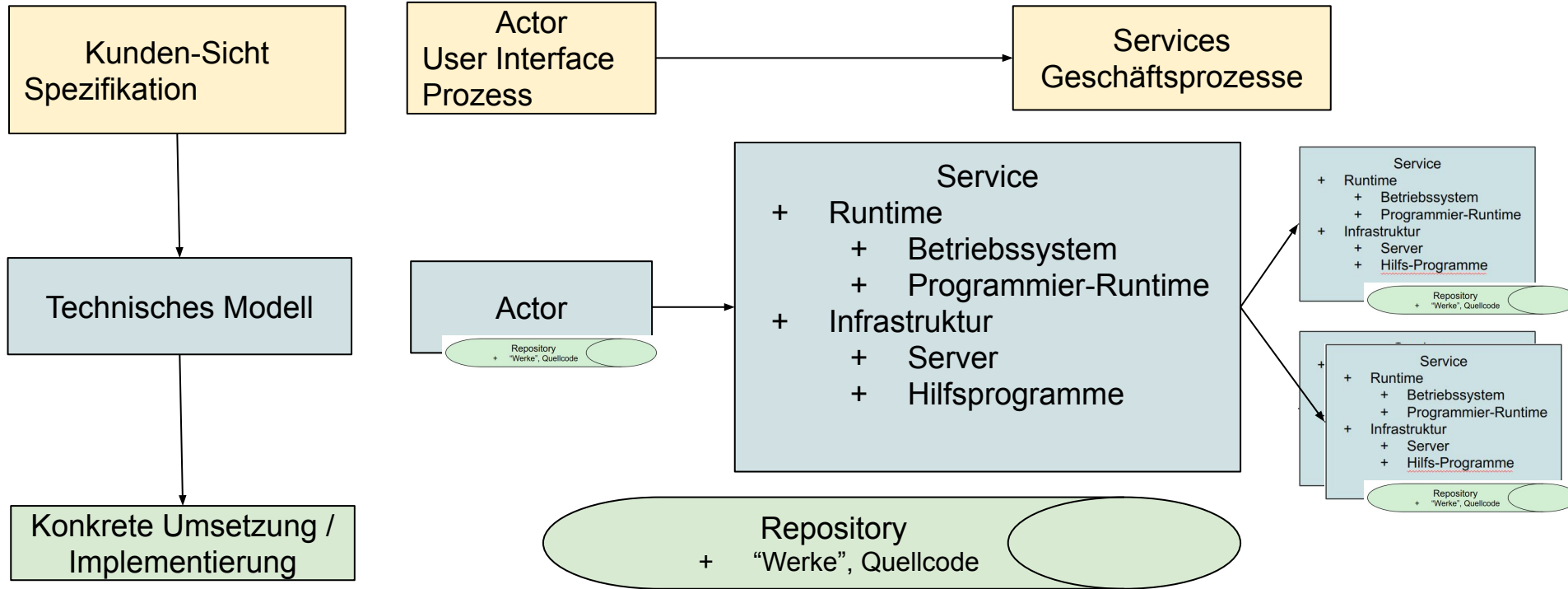
Deskmate-Link: <https://cegos.deskmate.me/>

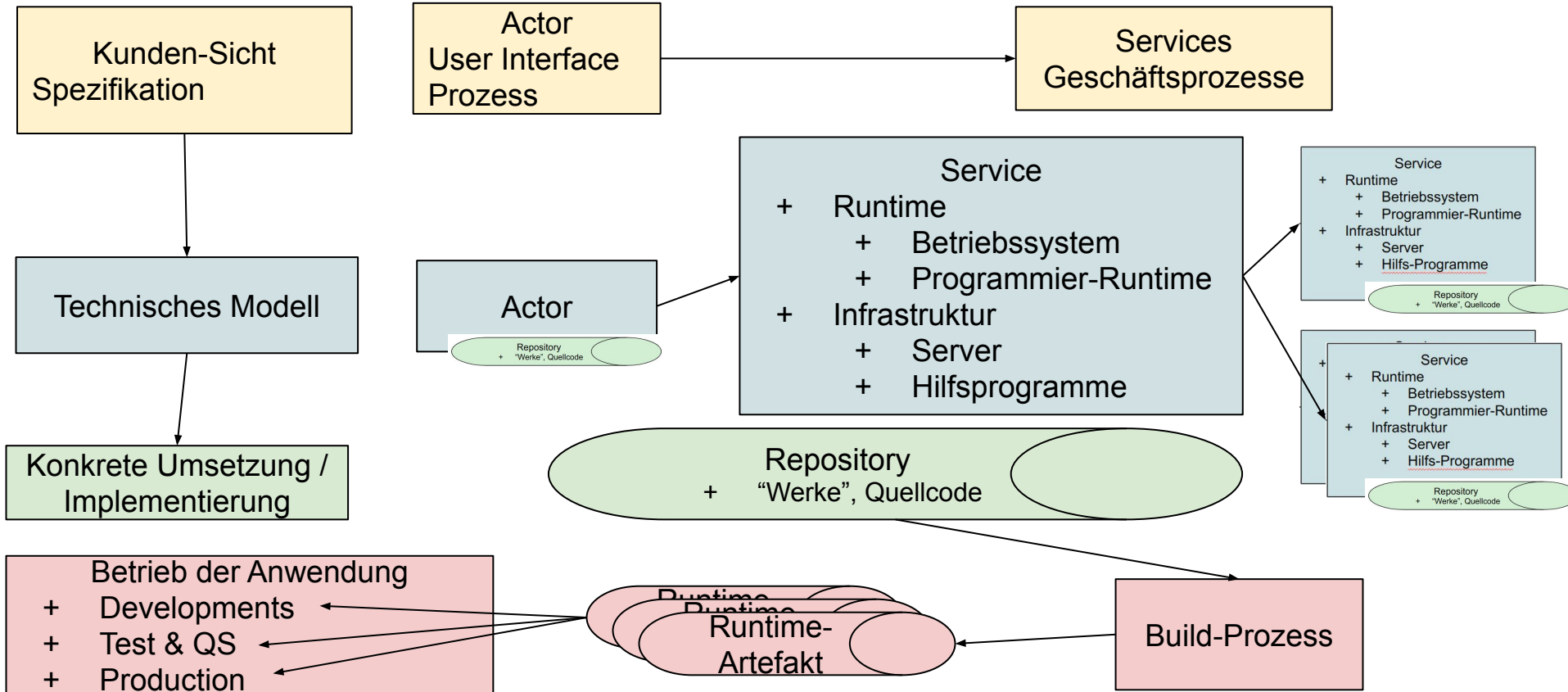
| | 1) Deskmate User | 2) Deskmate-Password | 3) Ubuntu-User | 4) Ubuntu-Kennwort |
|-----------------------|----------------------------------|------------------------|-----------------|--------------------|
| | ----- | ----- | ----- | ----- |
| | tn041.raum01@cegos.de | 33064_tn041 | s101 | s101 |
| Georg Eret | tn042.raum01@cegos.de | 33064_tn042 | s101 | s101 |
| Marc Helmer | tn043.raum01@cegos.de | 33064_tn043 | s101 | s101 |
| Sebastian Huber | tn044.raum01@cegos.de | 33064_tn044 | s101 | s101 |
| Christian Rindlbacher | tn045.raum01@cegos.de | 33064_tn045 | s101 | s101 |

Ausgangssituation









Was ist das Problem gewesen, dass zu CI/CD geführt hat?

- Geschwindigkeit Spezifikation -> Betrieb
 - Früher
 - Release-Zyklen im Monats- / Jahres-Rhythmus
 - Patches / Hot Fixes waren immer Notfall-Szenarien oder waren teilweise in den normalen Release-Zyklus integriert
 - Heute
 - Erwartungshaltung des Kunden- / Auftraggebers ist, dass neue / erweiterte / korrigierte Prozesse “sofort” bereitgestellt werden
 - Hot Fixes -> wenige Stunden
 - Erweiterungen -> wenige Tagen / 2-3 Wochen
 - Hinweis
 - Das ist kein “nice to have”, sondern eine unverhandelbare Anforderung

- “Wir müssen den Prozess der Software-Entwicklung drastisch beschleunigen”
- “wir arbeiten agile”
 - Wichtig: “Agile” ist eine reine Dampfwolke, da steht nichts dahinter
 - Umsetzung erfolgt Digitalisierung, Standardisierung und Automatisierung

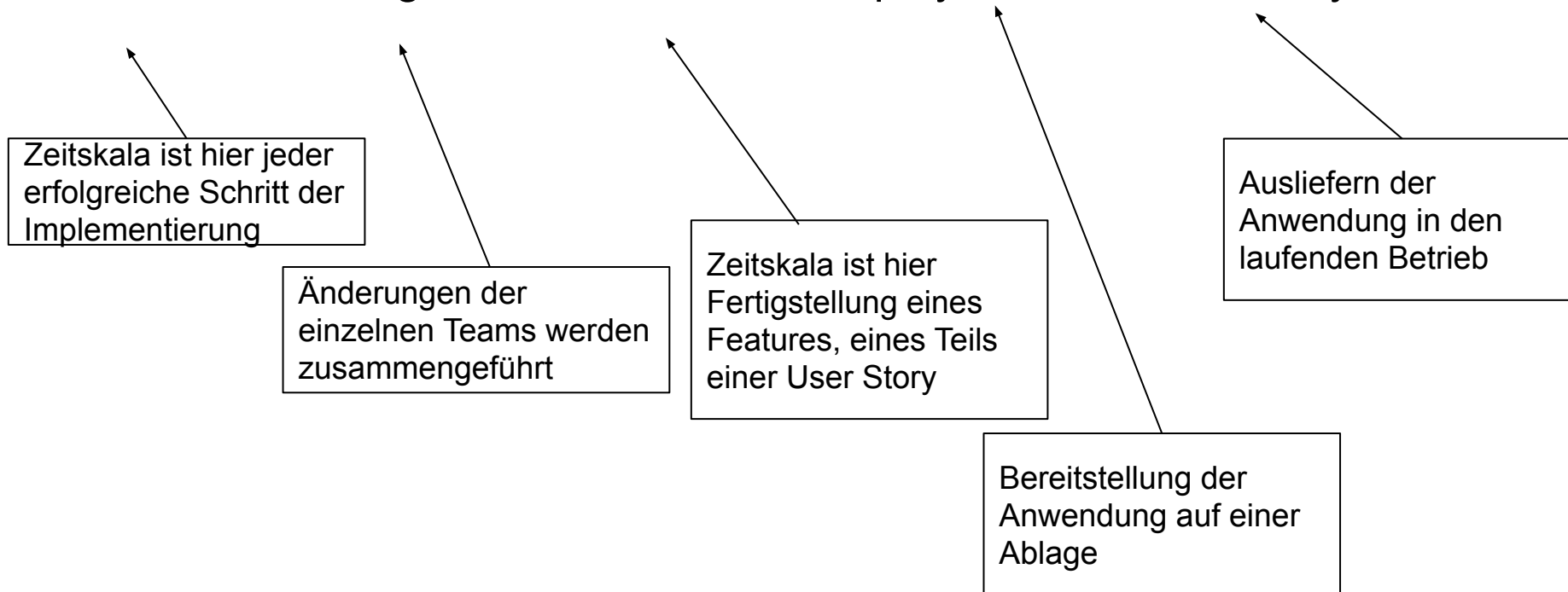
- Fachlichen Anforderungen werden als “User Stories” formuliert
 - Formalisierung “was will der user” und Definition of Done
 - Diese Stories müssen so fein -granular formuliert werden, dass
 - eine Lösung im geforderten Zeitrahmen möglich ist
 - eine Parallelentwicklung von Stories möglich ist

- Technische Modelle werde vordefiniert über einen überschaubaren Satz von Templates realisiert
- Die Umgebung (Plattform, Infrastruktur) wird ebenfalls vordefiniert
- Offener Punkt: Wie können diese Templates formuliert werden?

- Source Code Management Tool, das
 - ein effizientes Entwickeln
 - ein einfaches Dokumentieren
 - effiziente Werkzeuge zur Team-Zusammenarbeit bereitstellt
- Sehr weit verbreitet ist hierfür der Einsatz von Git Servern

- Automatisierter Build-Prozess
- Automatisiertes Testen
-

■ Continuous Integration / Continuous Deployment oder Delivery



- Fachlichen Anforderungen werden als “User Stories” formuliert
 - Formalisierung “was will der user” und Definition of Done
 - Diese Stories müssen so fein -granular formuliert werden, dass
 - eine Lösung im geforderten Zeitrahmen möglich ist
 - eine Parallelentwicklung von Stories möglich ist

Ohne dieses ist der CI/CD-Prozess hilfreich, aber nicht überzeugend

- Technische Modelle werden vordefiniert über einen überschaubaren Satz von Templates realisiert
- Die Umgebung (Plattform, Infrastruktur) wird ebenfalls vordefiniert
- Offener Punkt: Wie können diese Templates formuliert werden?

Templates der technischen Modelle müssen sorgfältig getestet werden

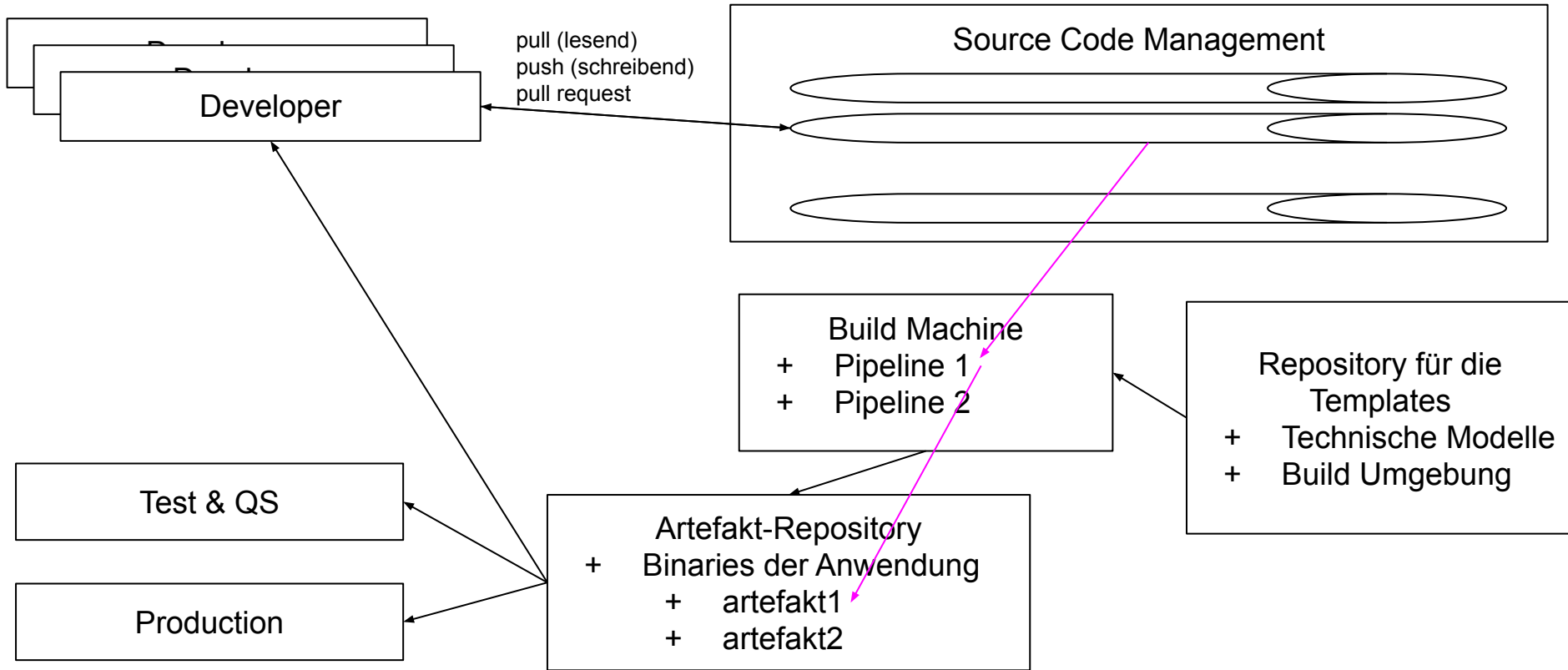
- + reine Funktionalität
- + Security

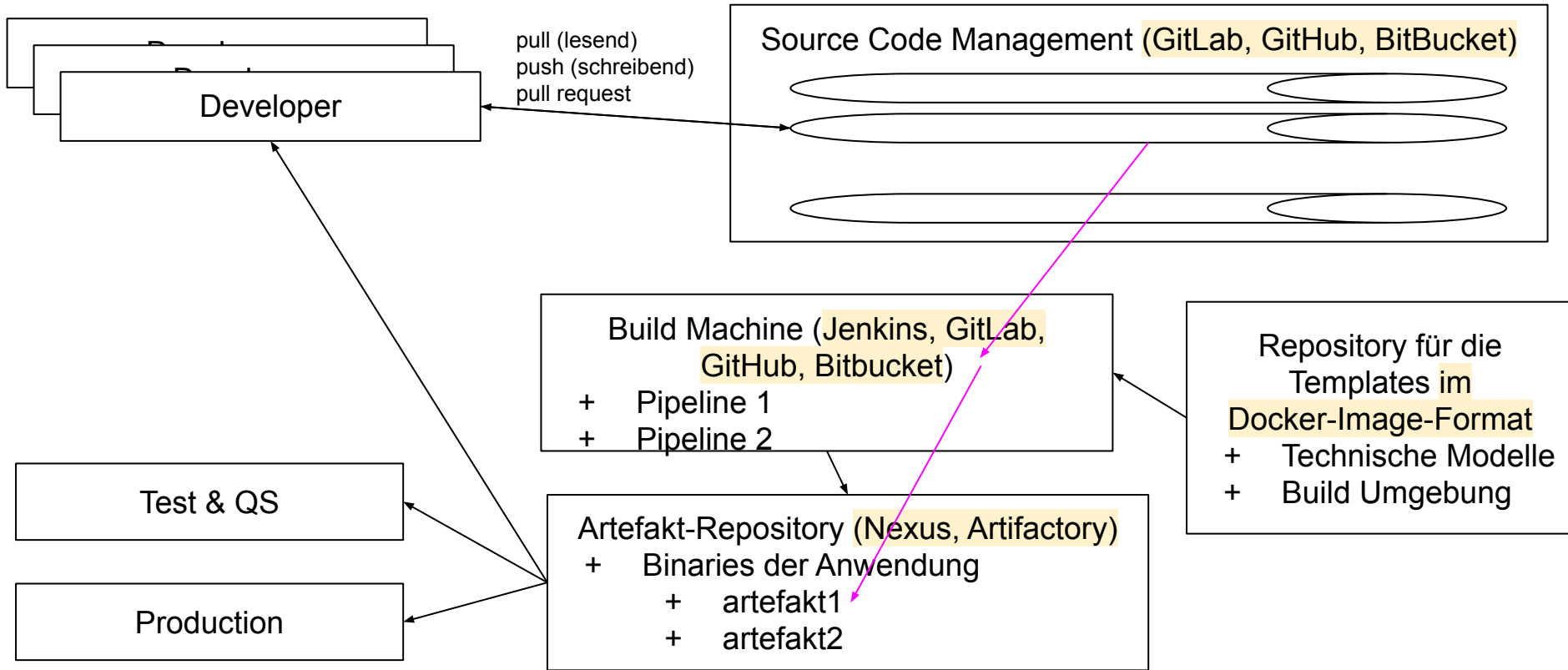
- Source Code Management Tool, das
 - ein effizientes Entwickeln
 - ein einfaches Dokumentieren
 - effiziente Werkzeuge zur Team-Zusammenarbeit bereitstellt
- Sehr weit verbreitet ist hierfür der Einsatz von Git Servern

- + Das liefert die Trigger zum Anstoßen der CI/CD-Prozesse
- + Durch die Dokumentation der Entwicklungs-Historie ist ein Zurücksetzen auf einen älteren Stand möglich

- Automatisierter Build-Prozess
- Automatisiertes Testen

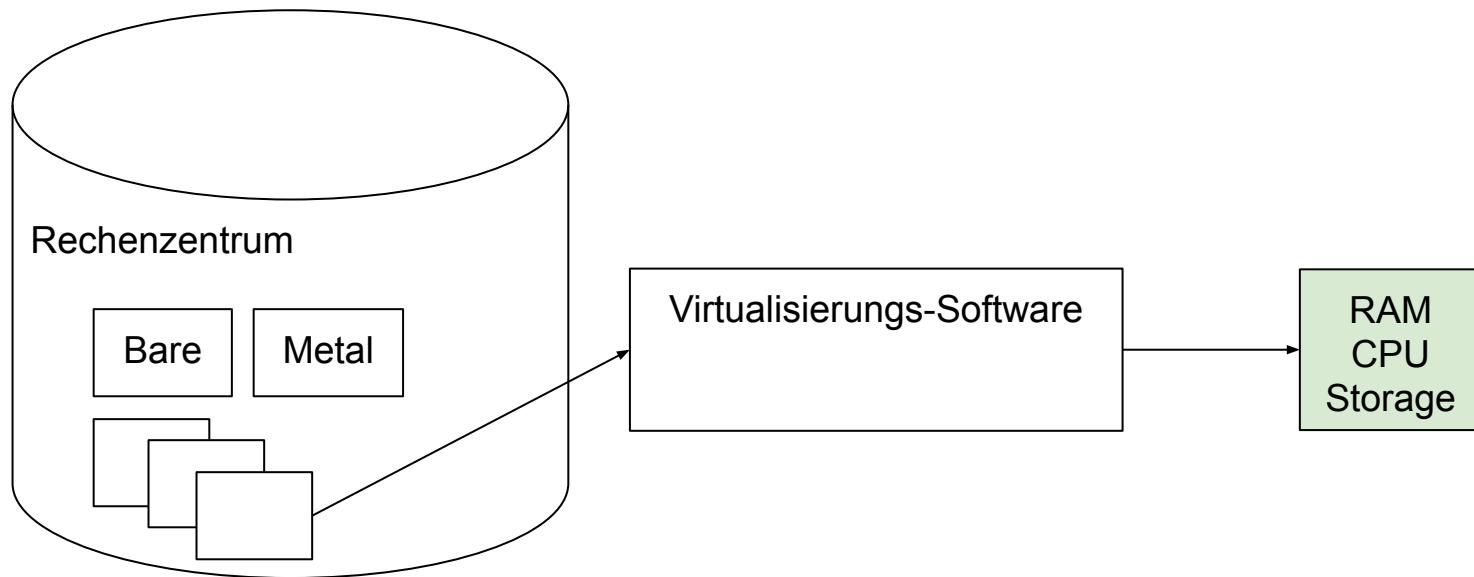
Eine Build-Machine führt im Rahmen einer Pipeline den Build-Prozess + weitere vor- und nachrangige Schritte durch



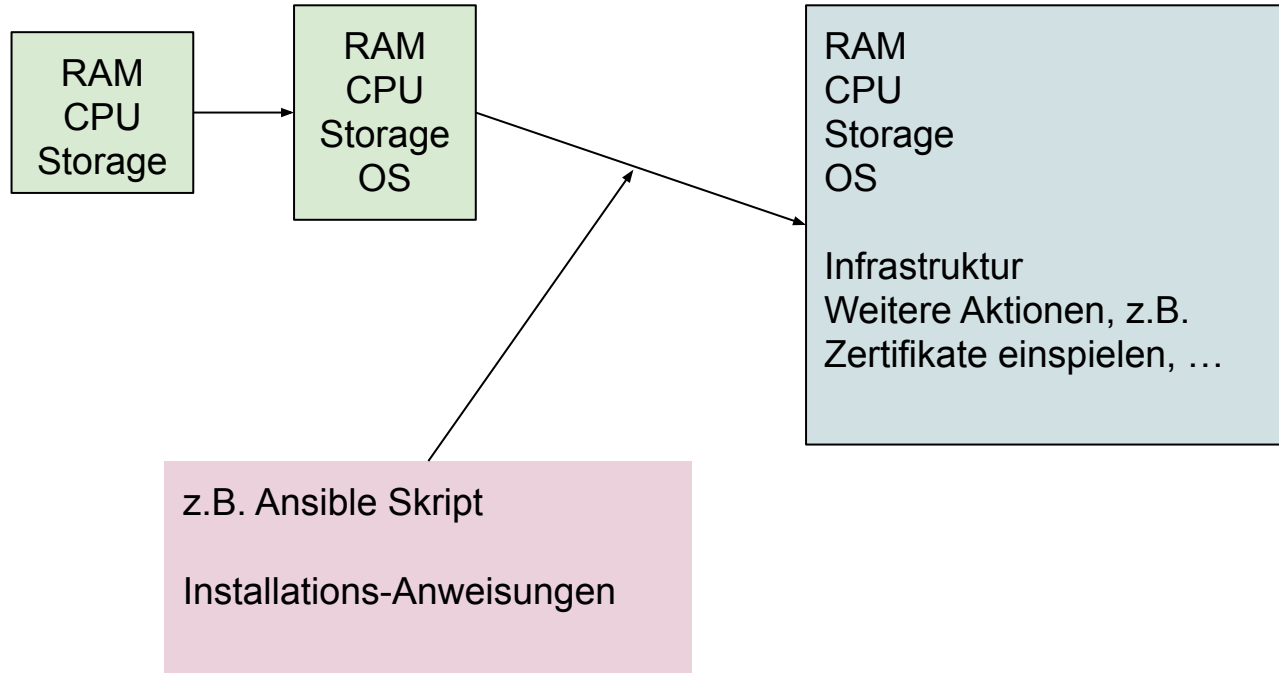


- Beteiligte Produkte werden bereitgestellt
- Pipelines zum Bauen einer Standard-Applikation sind vorhanden

Exkurs: Virtualisierung

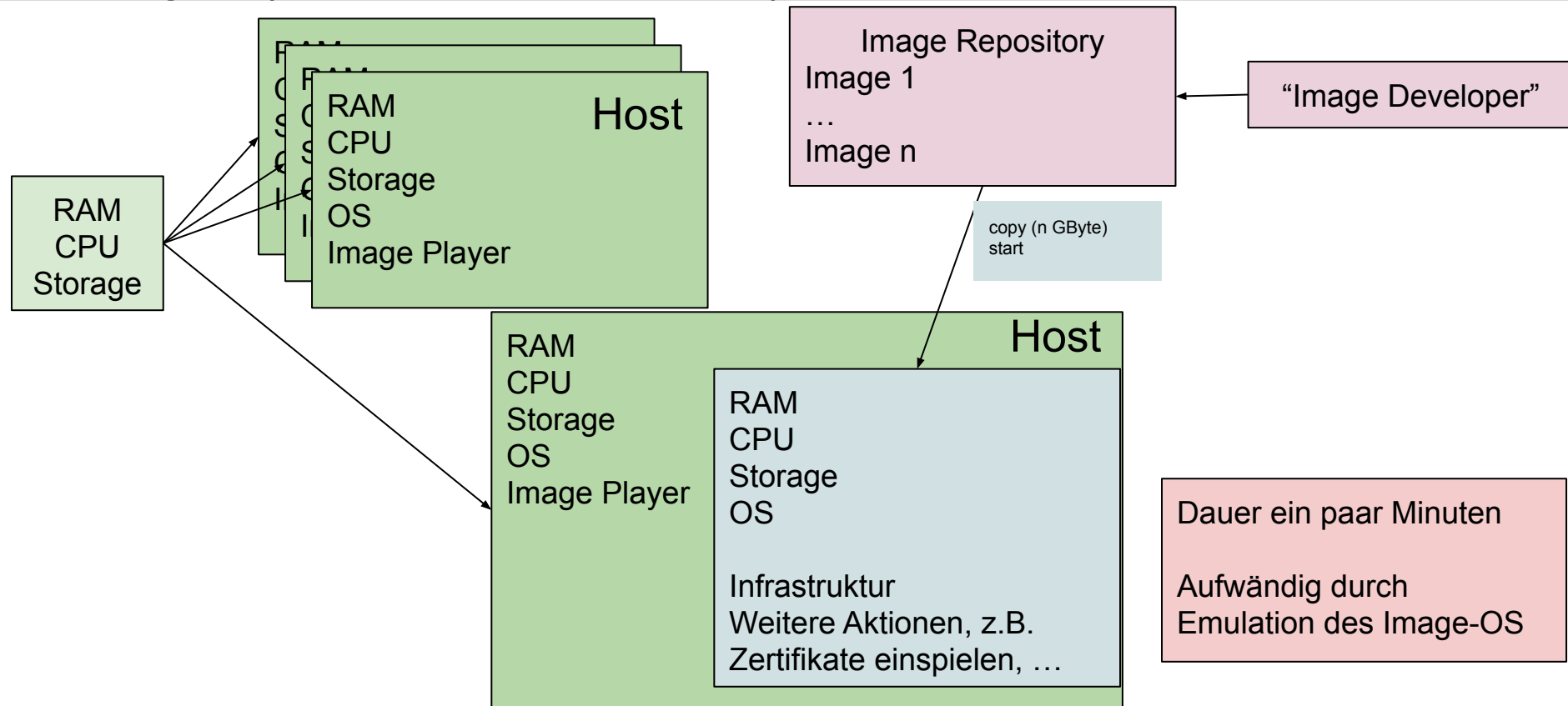


Bereitstellung einer Anwendung: Klassisch

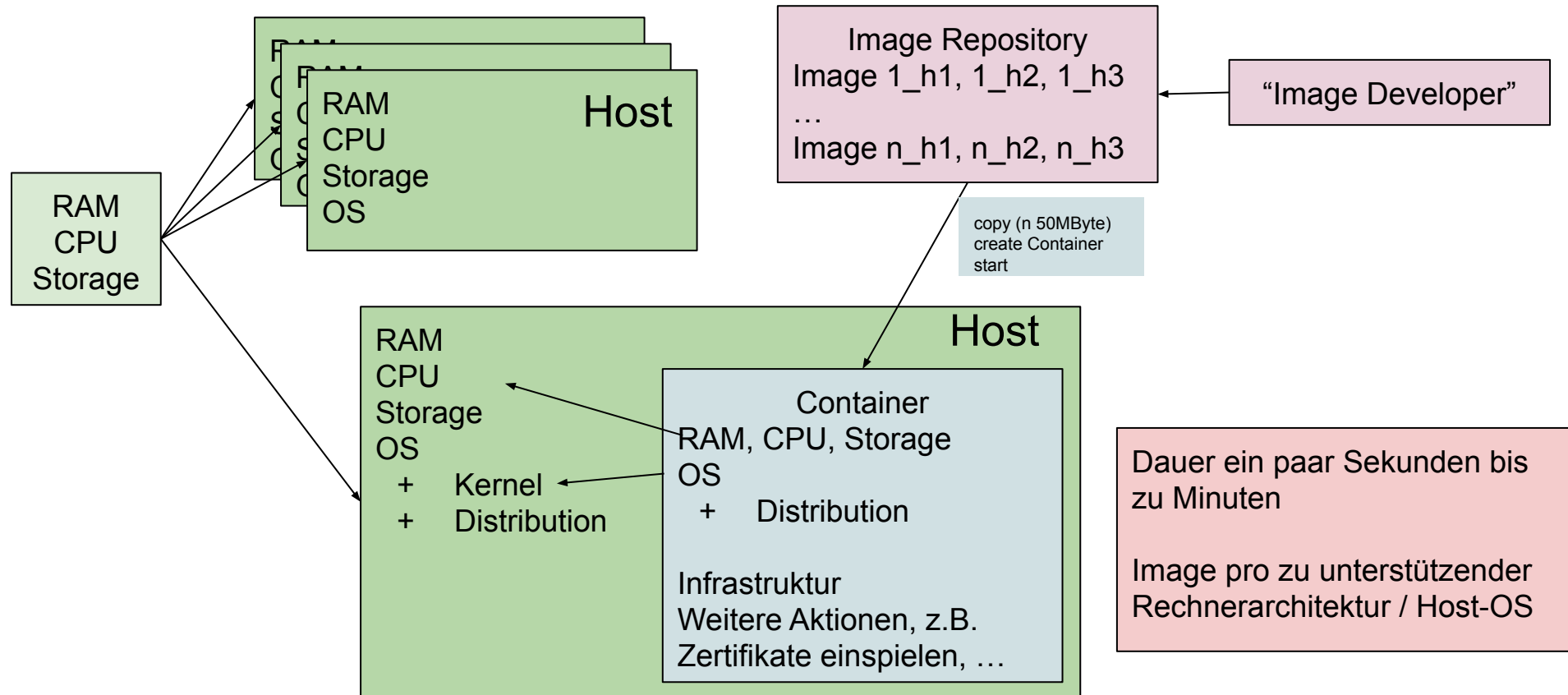


Dauer ist grob geschätzt
30 Minuten
und damit für CI/CD
ungeeignet

Virtualisierung von Anwendungen: Images (VmWare, Virtual Box)



Virtualisierung von Anwendungen: Container (Linux seit 2007, Windows 10+)



- Wir brauchen einen Web Server, z.B. nginx

[CHANGES](#)

[nginx-1.25.3](#) [pgp](#)

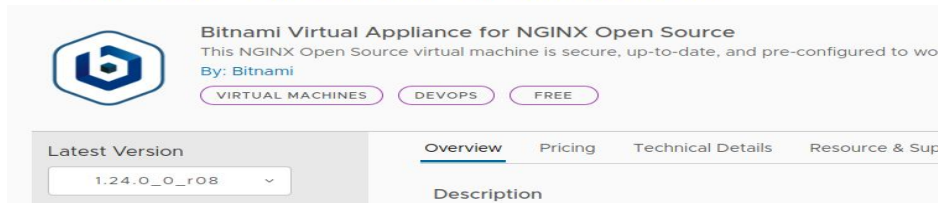
[nginx/Windows-1.25.3](#) [pgp](#)

Stable version

[CHANGES-1.24](#)

[nginx-1.24.0](#) [pgp](#)

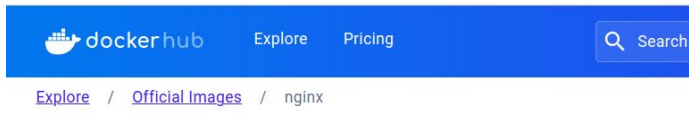
[nginx/Windows-1.24.0](#) [pgp](#)



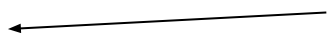
Klassisch



Image



Container



nginx

Docker Official Image · 1B+ · 10K+

Official build of Nginx.

- Klassisch
 - Download, Entpacken, Installieren, Konfigurieren, Starten
- Image
 - Download, Entpacken, Starten
- Container
 - `docker | containerd | podman run --rm -p 8765:80 nginx`

Container im Detail

- Ein Image ist
 - ein Dateisystem
 - speziell und optimiert organisiert
 - Eine Konfiguration
 - Start-Kommando
 - Sind Server-Ports vorhanden (EXPOSE)
- Aus dem Image (und damit dem Dateisystem) wird ein Container erzeugt
 - Dieser kann beim Erzeugen noch angepasst und konfiguriert werden
 - -p HOST-PORT:8080
 - Host-Verzeichnisse können in das Container-Dateisystem eingefügt (“mounted”) werden

- `docker create --name my_container ... image:tag`
- Lifecycle
 - `docker start my_container`
 - `docker stop my_container`
 - `docker rm my_container`
- Operationen innerhalb des Containers
 - `docker exec -it my_container sh`
- Dateitransfer
 - `docker cp src target`
 - `src | target`
 - `path1/path2 -> Host-Pfad`
 - `my_container:/path -> Container-Pfad`
 - genau ein Host- und Container-Pfad muss angegeben werden

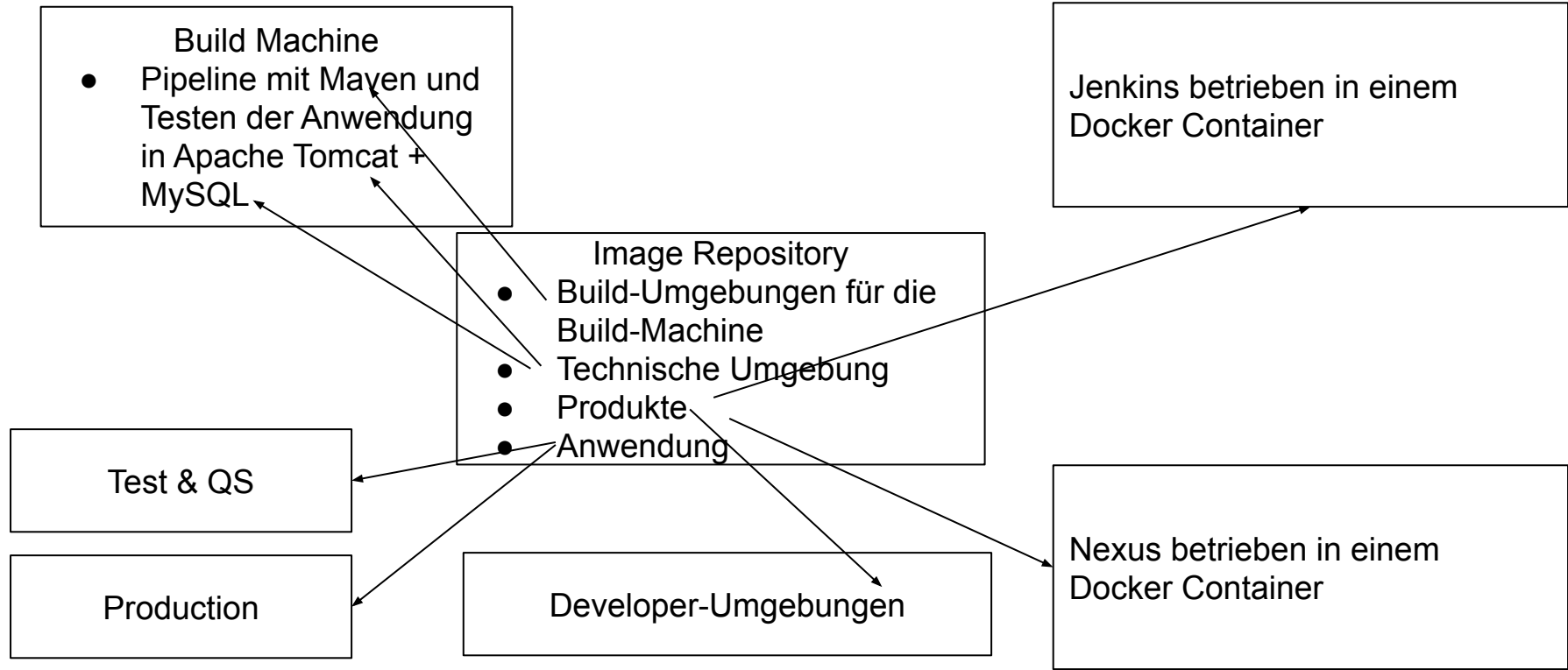
“Image Docker Developer”

“Admin-like”

- Programmatisch über ein Dockerfile

```
FROM tomcat:8  
COPY index.html /usr/local/tomcat/webapps/ROOT/index.html
```

- `docker build -t javacream/custom_tomcat:1.0 .`
- In den meisten Fällen wird anschließend das Image in ein Image-Repository gepushed
 - Hier in der Präsentation: Referent -> Docker Hub
 - später: Ausbringen in ein eigenes Seminar-Repository



- Nachvollziehen der Präsentation
 - Starten eines nginx auf den Deskmates
 - Beispiel einer kleinen Anwendung betrieben von Tomcat
 - `docker create --name tomcat_container -p 8080:8080 tomcat:8`
 - `docker start tomcat_container`
 - `docker exec -it tomcat_container sh`
 - Beenden der Container-Shell mit `exit`
 - Beispiel eines eigenen Docker Images
 - https://github.com/Javacream/org.javacream.training.cicd/tree/audi_29.1.2024/custom_tomcat
 - Weitere Befehle
 - `docker build -t custom_tomcat:1.0`
 - `docker run --rm -p 8081:8080 custom_tomcat:1.0`
 - CHECK: localhost:8081 kommt die / Ihre Webseite

- Source Code Management
- Build-Pipelines in Jenkins
- Aufgabe eines Artefakt-Repositories
- Fragerunde

Source Code Management mit Git

- “Normale” Versionsverwaltung
 - Schreiben von Dateien
 - Entscheidung: Ich möchte einen historisierbaren Stand definieren
- Mit Git
 - git init
 - Initialisierung eines lokalen Repositories

```
sl01@deskmate:~/git_demo$ git add .
sl01@deskmate:~/git_demo$ git config --global user.name "Rainer Sawitzki"
sl01@deskmate:~/git_demo$ git config --global user.email rainer.sawitzki@gmail.com
sl01@deskmate:~/git_demo$ git commit -m "initial project"
[master (Root-Commit) 1b2b881] initial project
2 files changed, 4 insertions(+)
create mode 100644 Readme.md
create mode 100644 content.txt
sl01@deskmate:~/git_demo$ git log
commit 1b2b8818cb47cbd534277e5215108c349d7c9a02 (HEAD -> master)
Author: Rainer Sawitzki <rainer.sawitzki@gmail.com>
Date: Tue Jan 30 09:24:28 2024 +0100

    initial project
sl01@deskmate:~/git_demo$
```

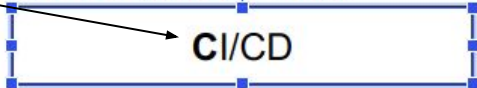
- Branches und Tags verwenden
 - Branches sind fortlaufende Aktionen “ich mache etwas”
 - Tags sind erreicht fixe Stände
- Der Haupt-Branch (master, main) wird nur in bestimmten Situationen benutzt
 - In der Realität: Bei der Fertigstellung eines kompletten Features
 - CI/CD
- Zur Implementierung eines Features wird immer ein Feature-Branch angelegt
 - `git checkout -b feature/<name des features>`
 - In einem Feature-Branch darf beliebig oft committed werden
 - Hinweis: Die Commit-Messages müssen nicht total ausführlich sein
 - Best Practice: Sprechender Einzeiler genügt
 - Formulierung “ich muss folgendes tun...”

CI/CD

- Auf dem Rechner 10.160.1.14:8929 läuft ein GitLab-Server
 - Check
 - Im Browser 10.160.1.14:8929 zeigt Anmeldeseite
- Anmeldung
 - teilnehmer1
 - javacream

- Statt git init (rein lokales Repository) nun ein git clone (lokales Repository aber mit konfiguriertem Server Repo)
- Workflow
 - Feature Branche anlegen
 - Loop
 - Änderungen
 - add
 - commit
 - Neu: Push zum Server

>
ommitted werden
cht total ausführlich sein
t
..



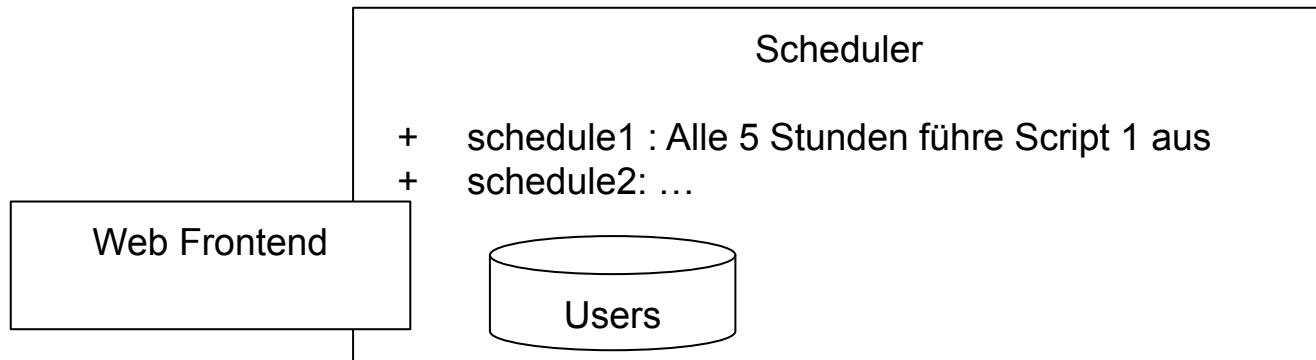
CI/CD

Die Build Machine

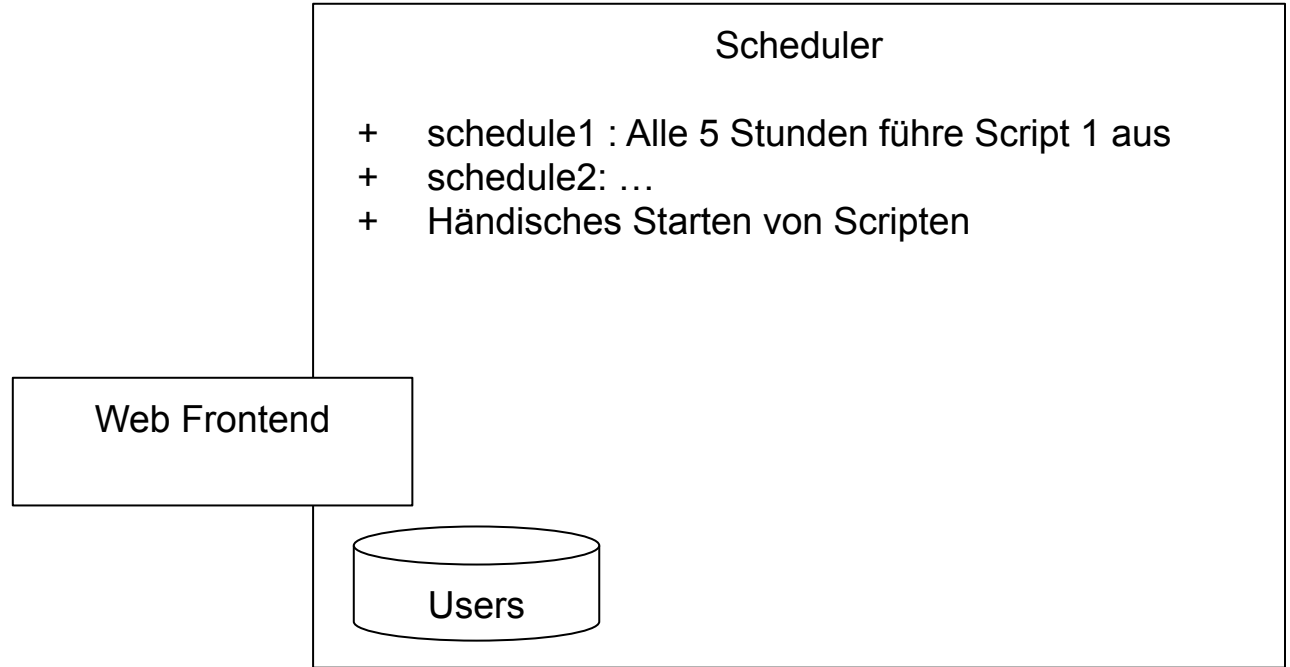
Scheduler

- + schedule1 : Alle 5 Stunden führe Script 1 aus
- + schedule2: ...

Step 2



Step 3



Step 4

Ein Job ist formuliert in der Programmiersprache des Job Executors

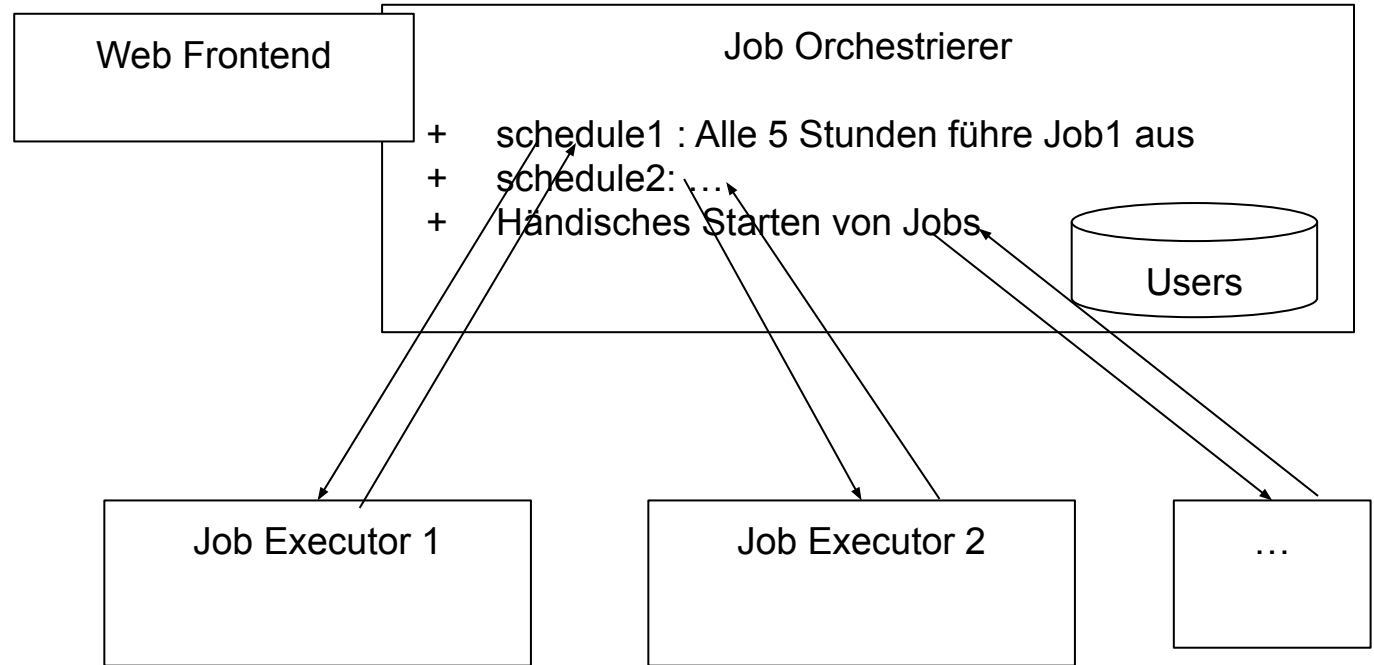
Job Executor

- + schedule1 : Alle 5 Stunden führe Job1 aus
- + schedule2: ...
- + Händisches Starten von Jobs

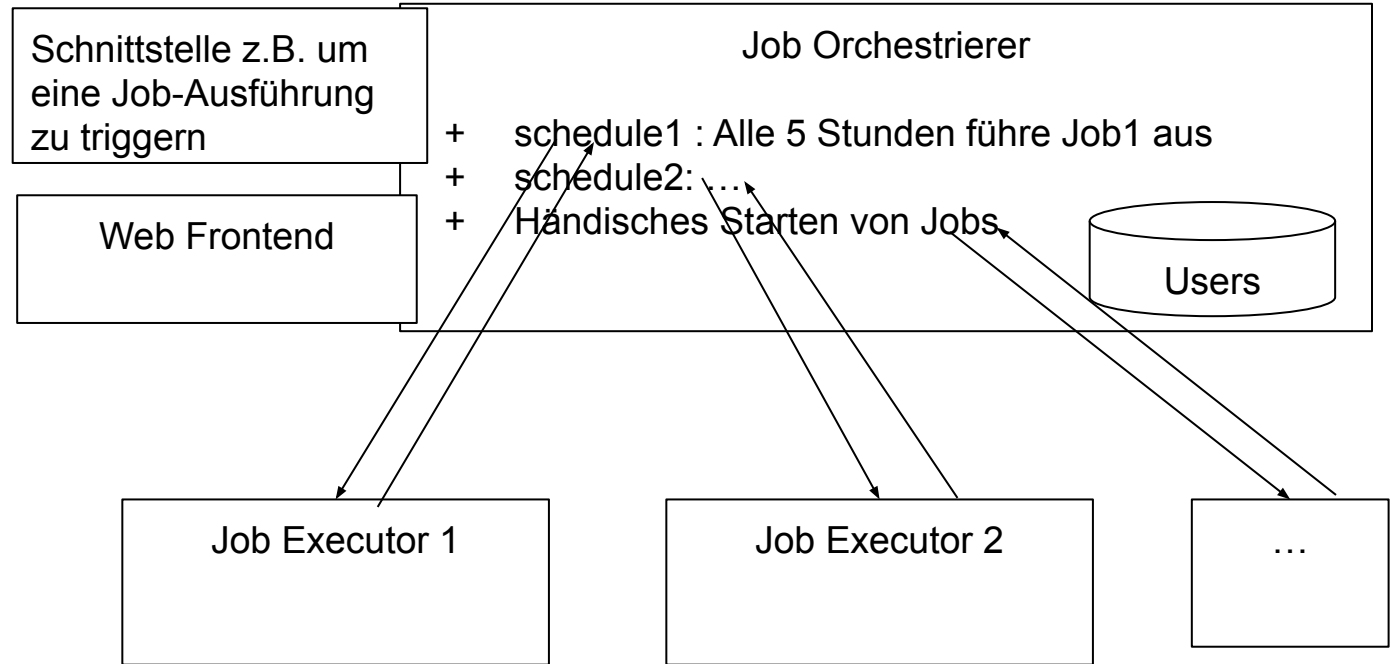
Web Frontend

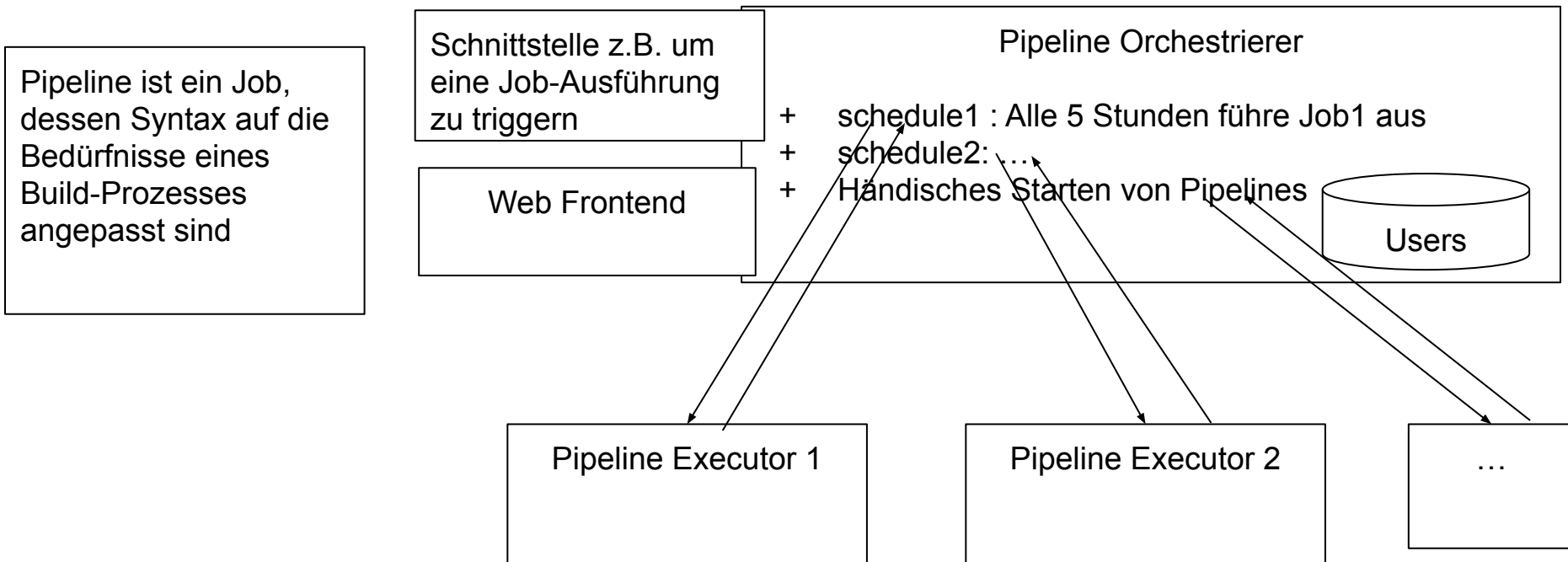
Users

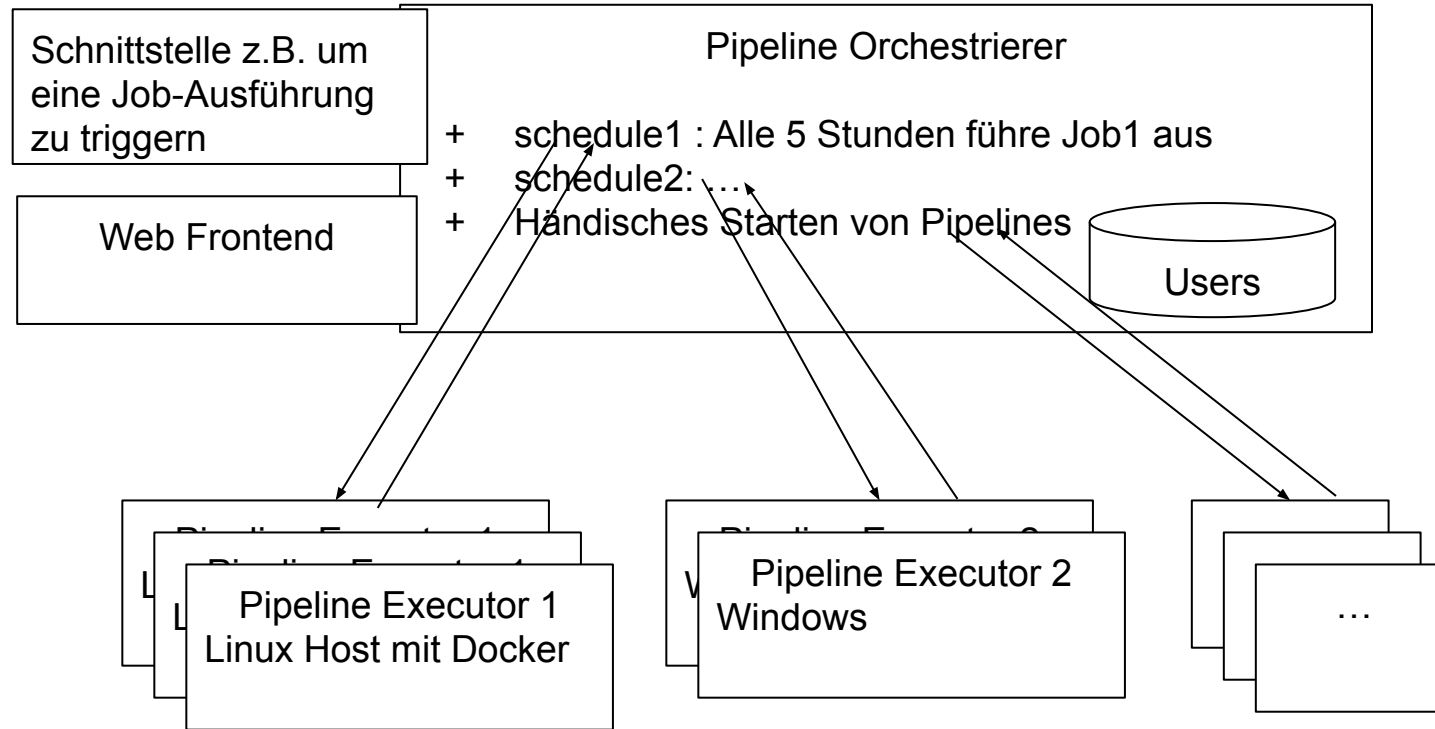
Step 5



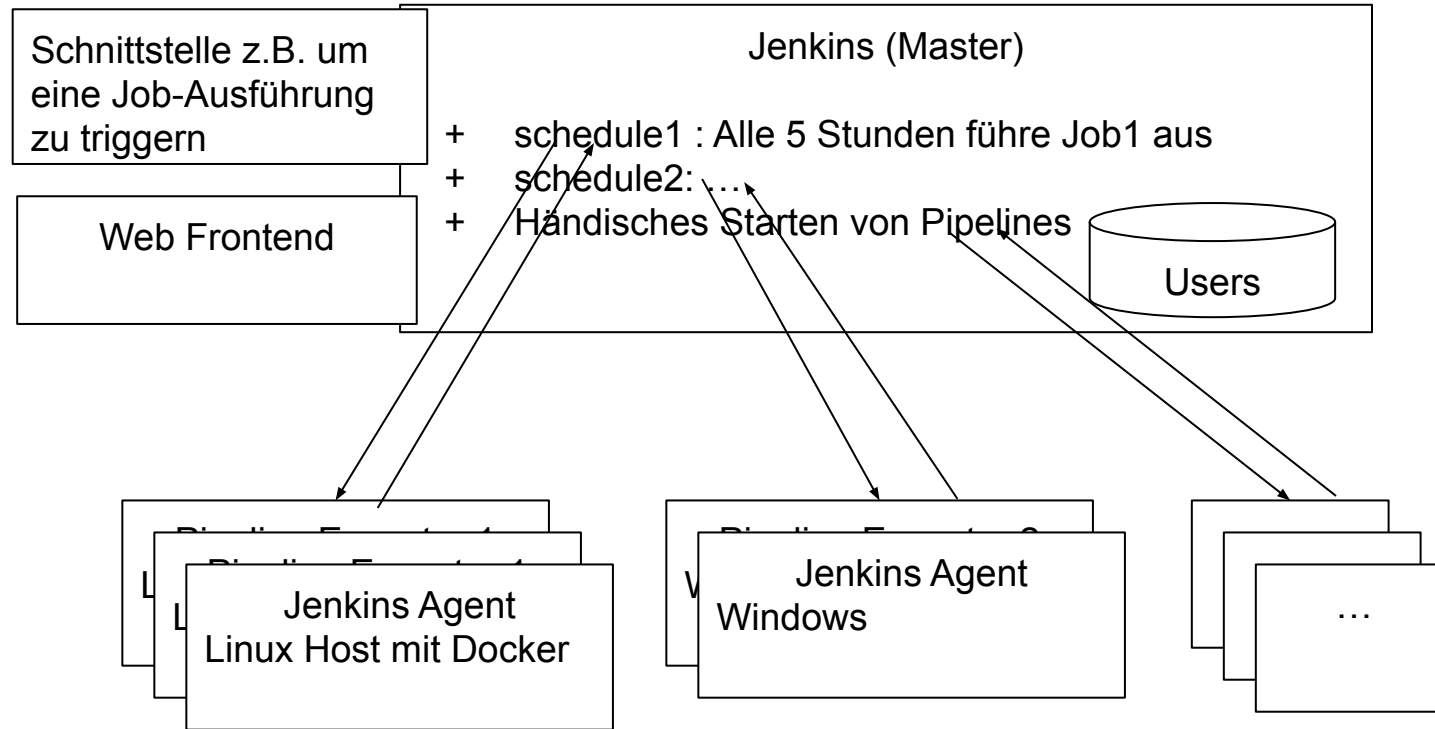
Step 6





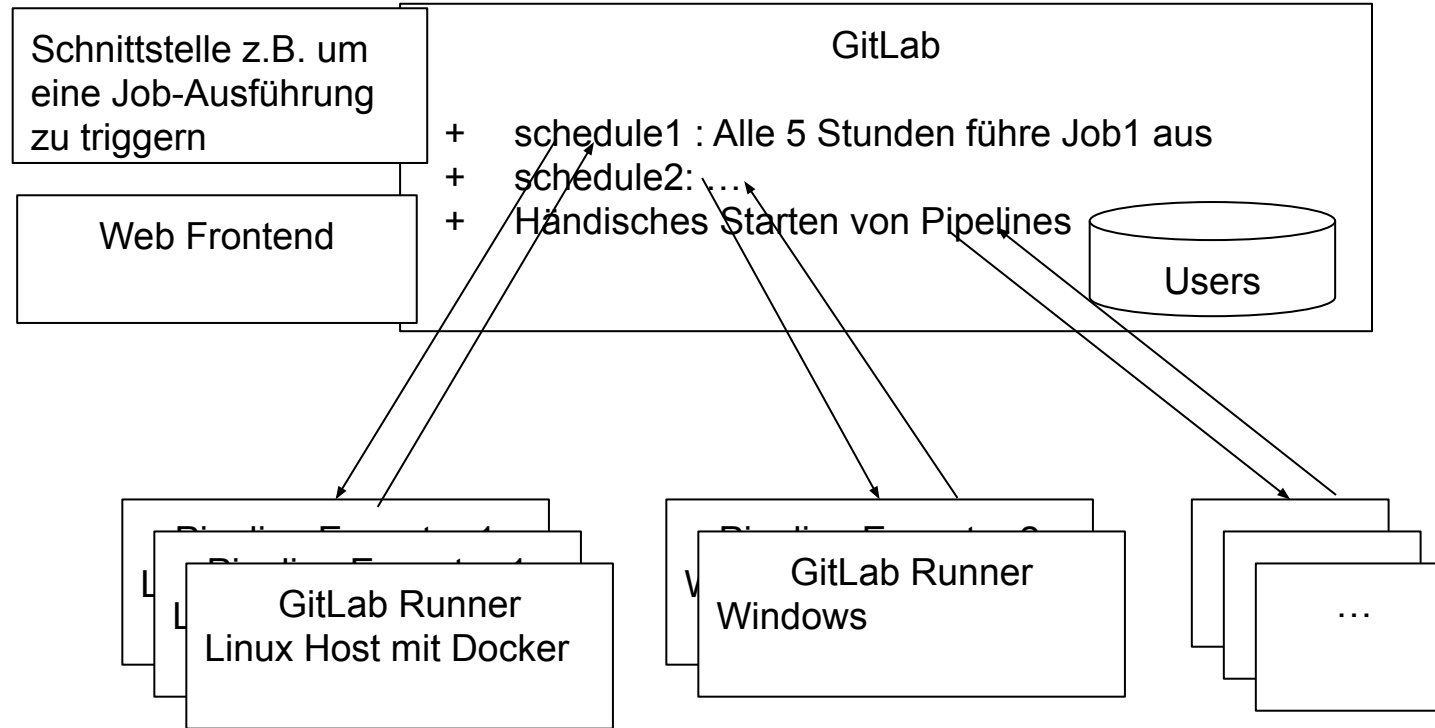


Beispiel: Jenkins als Build Machine



- Jenkins ist eigentlich nicht als reine Build Machine konzipiert
 - Jenkins ist ein Job Executor
- Standalone-Server, der mit anderen Systemen integriert werden kann
 - Konfiguration
 - PlugIn-Konzept

Beispiel: GitLab als Build Machine



- GitLab CI/CD ist eine Ergänzung der Funktionalität “Git Server”
 - Git Server
 - Build Machine
 - Aufgabenverwaltung (-> Jira)
 - Wiki-Plattform (-> Confluence)
- Build-Machine ist intern an die Trigger des Git Workflows gekoppelt

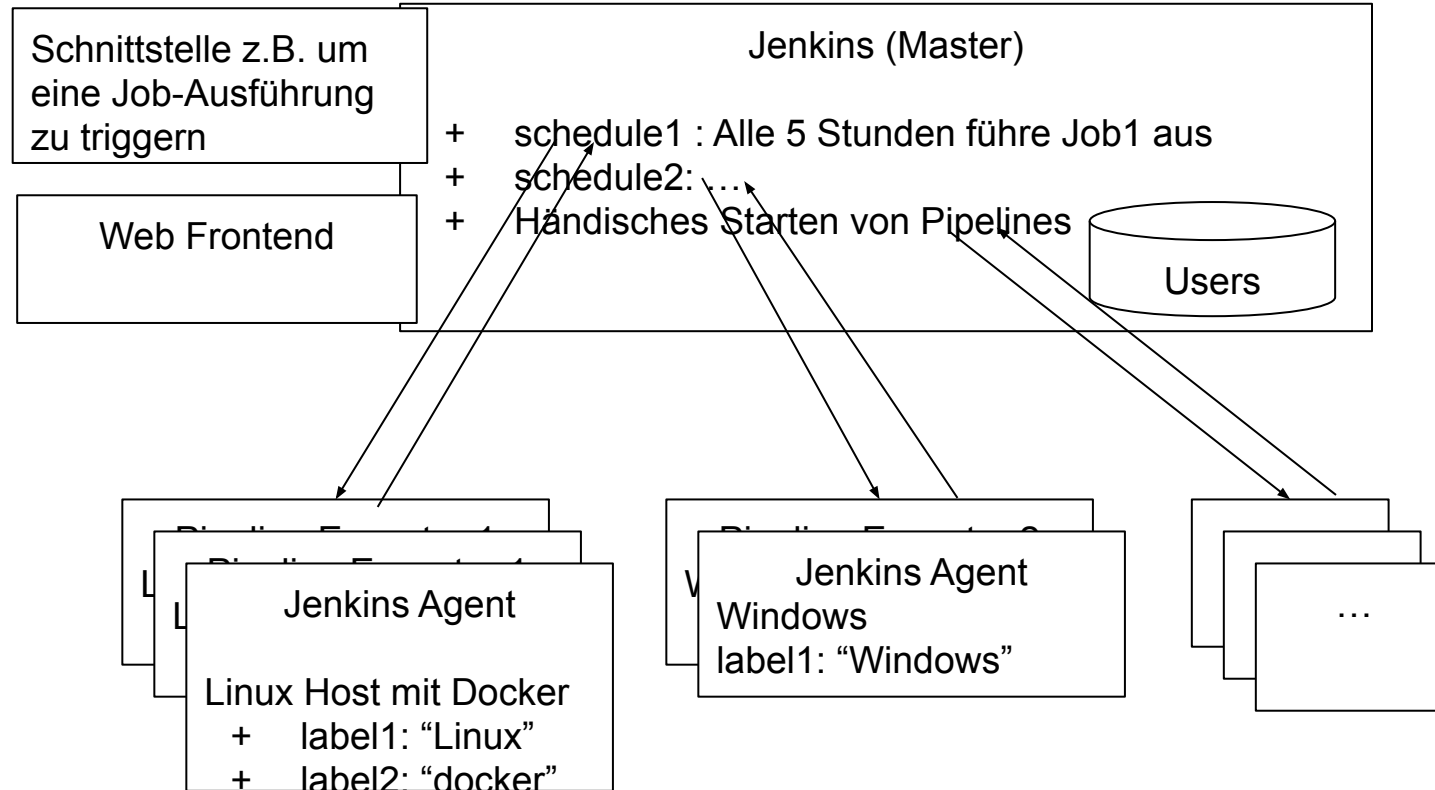
Komplette Suite-Lösungen für
die Anwendungsentwicklung

Jenkins - Ein Überblick

- Ein vorbereiteter Jenkins-Server mit bereits installierten PlugIns steht zur Verfügung
 - javacream.eu:8080
 - Credentials
 - javacream
 - javacream123!
- Vereinfachung: Der Orchestrierer ist selbst ein Agent
- Pipeline-Sprache ist
 - “Scripted Pipelines” werden in Groovy (“Skript-Sprache für Java”) erstellt
 - “Deklarative Pipelines” sind hierarchische Konfigurationen mit eingestreuten Skripten, z.B. in Groovy
 - Hinweis: GitLab CI/CD arbeitet ebenfalls mit einer hierarchischen Konfiguration, .gitlab-ci.yml

Sie sind Admin!

Beispiel: Jenkins als Build Machine, Labels



Der Build-Prozess im Detail

- Aufgabe eines Build-Prozesses ist die Überführung von Werken (Quellcode) in ein Artefakt (ausführbares Programm)
- Beispiel Java
 - Compiler als Build-Werkzeug
 - `.java` -> Compiler-Aufruf mit Optionen -> `.class`
 - Java Archiver
 - `.class` -> Archiver mit Optionen -> `.jar`
- Beispiel Docker
 - `docker build`
 - Beliebige Dateien + Dockerfile -> `docker build` -> Docker-Image

- “Programmierer-Builds”
 - Meistens automatisch ausgeführt in einer Entwicklungsumgebung
 - “nach einem Speichervorgang werden alle Quellcodes compiliert”
- **Developer-Builds**
 - besteht aus mehreren Phasen
 - compile
 - Unit-Tests ausführen
 - package
- **Integration Build**
 - Die Artefakte des Developer-Builds werden als Grundlage zur Ausführung von Integrations-Tests durchgeführt
 - Fließende Grenze zum System-Test

Intern in der Entwicklungsumgebung

Developer Maschinen

Build Machine

- Zusätzliche Phase: “Dependency Management”
 - Services haben Abhängigkeiten
 - Diese Abhängigkeiten werden vom Build-Prozess vor dem Compile aufgelöst
- Beispiel: Maven
 - pom.xml
 - Enthält die Konfiguration eines fixen Build-Ablaufs = Goals
 - https://github.com/Javacream/org.javacream.training.cicd/blob/audi_29.1.2024/projects/org.javacream.util.library/pom.xml

- Pipelines sind sehr einfach, fast trivial
 - Repository clonen
 - Vorhandenen Developer Build aufrufen
 - Gebauten Artefakte auf den Orchestrierer schieben

- Setzen auf auf die Ergebnisse der Developer Pipeline
- Beispiel
 - Erzeugen eines Docker-Images aus den gebauten Artefakten
 - Ablauf
 - Kopiere das Artefakt des Developer Builds
 - Kopiere das Dockerfile aus einem Git Repository
 - `sh 'docker build -t <name> .`

- Wir erstellen nur eine Pipeline
 - git 'repository'
 - Jenkinsfile
 - Quellcodes
 - pom.xml
 - Dockerfile
- Jenkinsfile
 - git 'repo'
 - sh 'mvn ...'
 - docker build ...

- Wir erstellen nur eine Pipeline
 - git 'repository'
 - Jenkinsfile
 - Quellcodes
 - pom.xml
 - Dockerfile
- Jenkinsfile
 - git 'repo'
 - sh 'mvn ...'
 - docker build ...
 - docker push <image>