



Tag 2: Git-Workflows, CI/CD, GitLab CI

09.07.2024, Daniel Krämer

© Copyright 2024 anderScore GmbH

HECKER
CONSULTING

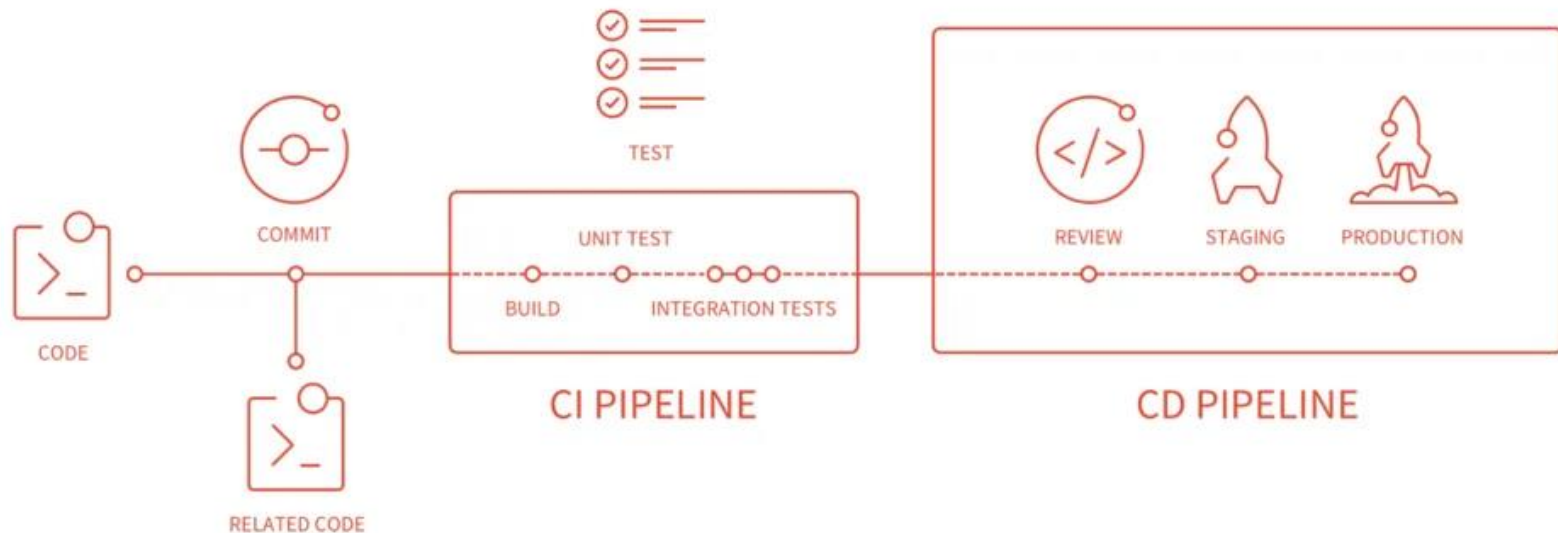
- **Tag 1 – Einführung in Git und GitLab**
 - Einführung & Kursüberblick
 - Grundlagen von Git
 - Git Rebase und Merge-Strategien
 - Git Remote
 - Grundlagen von GitLab
- **Tag 2 – Git-Workflows, CI/CD, GitLab CI**
 - Git-Workflow im Team
 - Gitflow-Workflow
 - Tags, Releases & deren Verwaltung
 - Einführung in GitLab CI/CD & gitlab-ci.yml
 - GitLab Runner
- **Tag 3 – Docker, GitOps, Deployment-Strategien**
 - Entwicklung mit Docker
 - Container/Docker-Registry
 - Erstellen von Release- und Tagged-Images
 - GitOps Grundlagen
 - Möglichkeiten des Deployments & Verwaltung von Konfiguration
 - Abschlussübung & Diskussion

- **Tag 1 – Einführung in Git und GitLab**
 - Einführung & Kursüberblick
 - Grundlagen von Git
 - Git Rebase und Merge-Strategien
 - Git Remote
 - Grundlagen von GitLab
- **Tag 2 – Git-Workflows, CI/CD, GitLab CI**
 - Git-Workflow im Team
 - Gitflow-Workflow
 - Tags, Releases & deren Verwaltung
 - Einführung in GitLab CI/CD & gitlab-ci.yml
 - GitLab Runner
- **Tag 3 – Docker, GitOps, Deployment-Strategien**
 - Entwicklung mit Docker
 - Container/Docker-Registry
 - Erstellen von Release- und Tagged-Images
 - GitOps Grundlagen
 - Möglichkeiten des Deployments & Verwaltung von Konfiguration
 - Abschlussübung & Diskussion

Grundlagen der **GitLab Runner**



GitLab CI



Quelle: <https://medium.com/@mosiko1234/optimizing-gitlab-ci-cd-pipelines-for-high-efficiency-f2ebbc046a89>

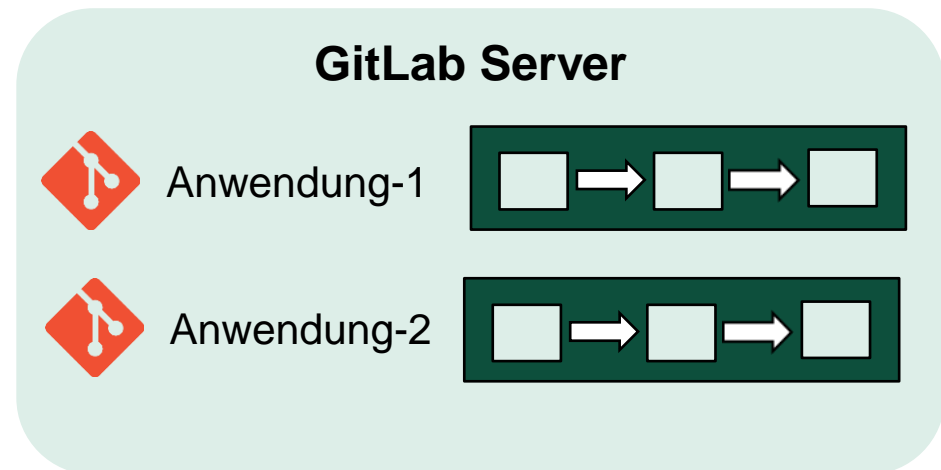
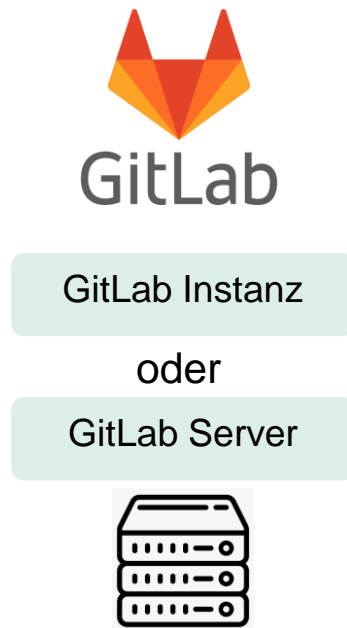
Basics

- Arbeiten Aufträge (Jobs) in einer Pipeline ab
- Varianten
 1. GitLab-hosted Runners
 2. Self-managed Runners
- GitLab-hosted Runners
 - GitLab.com oder „GitLab Dedicated“*
 - Default: enabled
- **Self-managed Runners**
 - GitLab Runner auf Infrastruktur installieren
 - Im Anschluss in GitLab registrieren

*GitLab Enterprise DevSecOps Platform as a single-tenant SaaS deployment

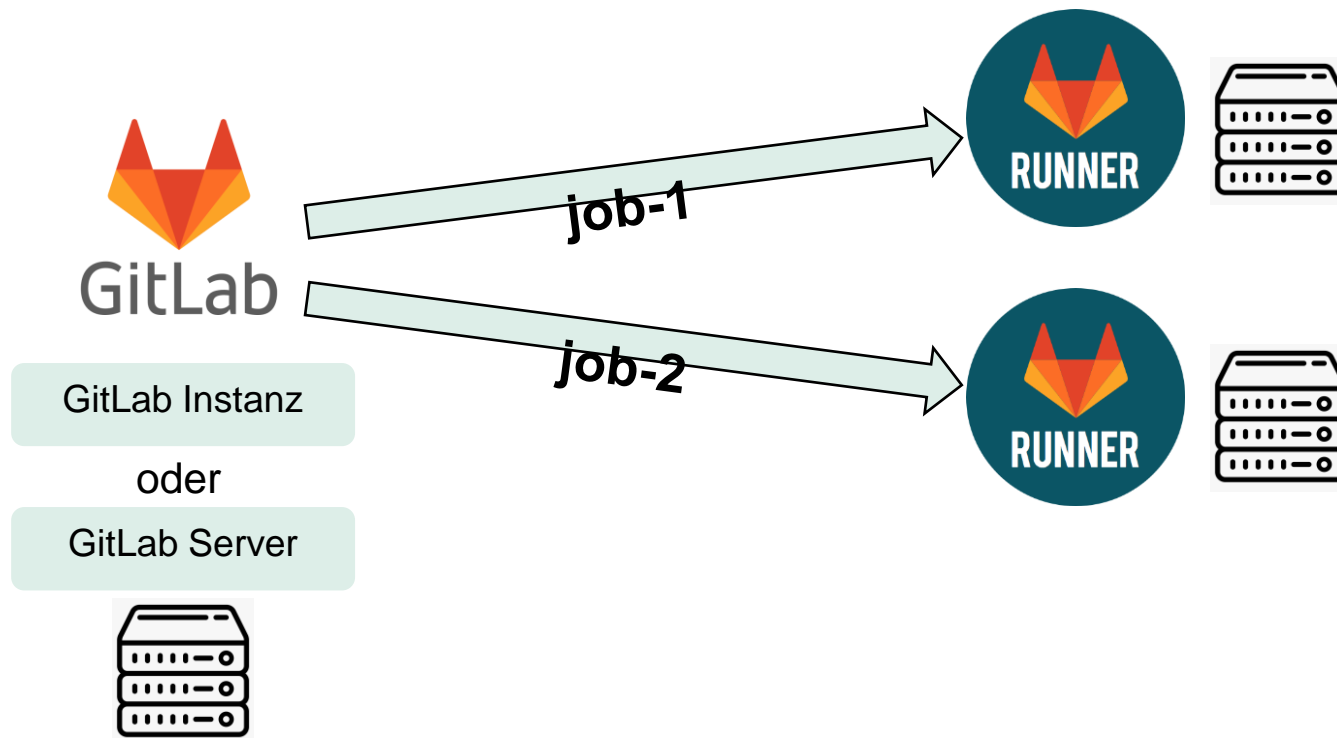
GitLab Architektur

- GitLab
 - Repos enthalten Anwendungscode und Pipeline-Definitionen
 - Weitere GitLab-Konfiguration über GUI
 - Steuerung die Pipeline-Ausführung



GitLab Architektur

- GitLab-Runners
 - Agents führen CI/CD Jobs aus
 - Zuweisung Pipeline Jobs an Runner durch GitLab



Arten von Runnern

- Project Runners
 - Mit Projekt(en) verknüpft
 - Maintainer-Rolle für Projekt benötigt
- Group Runners
 - Projekte und Untergruppen einer Gruppe
 - Owner-Rolle für Gruppe benötigt
- Instance Runners
 - Für alle
 - Adminrechte auf GitLab benötigt

Runner Status

Runner Status	Beschreibung
online	Runner hat sich innerhalb der letzten 2 Stunden mit GitLab verbunden und ist bereit für Jobs.
offline	Runner hat sich seit über 2 Stunden nicht mit GitLab verbunden und ist nicht verfügbar. Überprüfen Sie den Runner.
stale	Runner hat seit über 3 Monaten keinen Kontakt zu GitLab. Wenn er vor über 3 Monaten erstellt, aber nie verbunden wurde, gilt er als veraltet.
never_contacted	Runner hat sich nie mit GitLab verbunden. Führen Sie gitlab-runner run aus, um ihn zu verbinden.

Wie Instance Runners ihre Jobs auswählen

- Fair-Queuing
- Jobs: auf Basis von Projekten zugeordnet
- Projekt: geringste Anzahl laufender Runner → erhält Runner
- Beispiel-Queue
 - Job 1 für Project 1
 - Job 2 für Project 1
 - Job 3 für Project 1
 - Job 4 für Project 2
 - Job 5 für Project 2
 - Job 6 für Project 3

Wie Instance Runners ihre Jobs auswählen

Reihenfolge bei paralleler (unabhängiger) Ausführung der Jobs

- Beispiel-Queue
 - Job 1 für Project 1
 - Job 2 für Project 1
 - Job 3 für Project 1
 - Job 4 für Project 2
 - Job 5 für Project 2
 - Job 6 für Project 3

Wie Instance Runners ihre Jobs auswählen

Reihenfolge bei sequenzieller (abhängiger) Ausführung der Jobs

- Beispiel-Queue
 - Job 1 für Project 1
 - Job 2 für Project 1
 - Job 3 für Project 1
 - Job 4 für Project 2
 - Job 5 für Project 2
 - Job 6 für Project 3

Self-managed Runner



Self-managed Runner

1. Runner auf Infrastruktur installieren
2. (Project) Runner auf GitLab anlegen
3. (Project) Runner verwalten

Live Demo: Self-managed Project Runner

- Runner Executable installieren
- Neuen Project Runner einrichten
- Möglichkeiten zur Konfiguration explorieren





Live Demo: Self-managed Project Runner

- Runner Executable installieren
- Neuen Project Runner einrichten
- Möglichkeiten zur Konfiguration explorieren



Runner Executable installieren

- Erinnerung: GitLab Runner führen unsere CI/CD Jobs aus
- Folgende Installationen sind möglich
 - Eigene Infrastruktur
 - Docker Container
 - Kubernetes Cluster
- Hier: Infrastruktur → lokale Maschine
 - Linux oder Windows



Linux

1. Offizielles GitLab Repository hinzufügen
2. Aktuellstes GitLab Runner Paket installieren
3. Installation einer bestimmten GitLab Runner Version
4. GitLab Runner aktualisieren



Linux: Offizielles GitLab Repository hinzufügen

```
# Für Debian/Ubuntu/Mint (Achtung bei Debian: APT-Pinning!)  
curl -L  
https://packages.gitlab.com/install/repositories/runner/git  
lab-runner/script.deb.sh | sudo bash
```

```
# Für RHEL/CentOS/Fedora  
curl -L  
https://packages.gitlab.com/install/repositories/runner/git  
lab-runner/script.rpm.sh | sudo bash
```



Linux: Offizielles GitLab Repository hinzufügen

- Debian-Benutzer sollten APT Pinning verwenden

```
cat <<EOF | sudo tee /etc/apt/preferences.d/pin-gitlab-runner.pref
```

Explanation: Prefer GitLab provided packages over the Debian native ones

Package: gitlab-runner

Pin: origin packages.gitlab.com

Pin-Priority: 1001

EOF



Linux: Aktuellstes GitLab Runner Paket installieren

Für Debian/Ubuntu/Mint

```
sudo apt-get install gitlab-runner
```

Für RHEL/CentOS/Fedora

```
sudo yum install gitlab-runner
```



Linux: Installation einer bestimmten GitLab Runner Version

```
# Für DEB basierte Systeme (Debian, Ubuntu, Mint..)
apt-cache madison gitlab-runner
sudo apt-get install gitlab-runner=10.0.0
```

```
# Für RPM basierte Systeme (RHEL, CentOS, Fedora...)
yum list gitlab-runner --showduplicates | sort -r
sudo yum install gitlab-runner-10.0.0-1
```



Linux: GitLab Runner aktualisieren

Für Debian/Ubuntu/Mint

```
sudo apt-get update
```

```
sudo apt-get install gitlab-runner
```

Für RHEL/CentOS/Fedora

```
sudo yum update
```

```
sudo yum install gitlab-runner
```



Windows

1. Systemordner erstellen, z.B.: C:\GitLab-Runner
2. Executable herunterladen, in erstellten Ordner kopieren und in gitlab-runner.exe umbenennen



Live Demo: Self-managed Project Runner

- Runner Executable installieren
- Neuen Project Runner einrichten
- Möglichkeiten zur Konfiguration explorieren



Neuen Project Runner einrichten

- Neues Projekt erstellen
- Projekt-Pipeline erstellen
- Project Runner erstellen und registrieren
- Pipeline triggern, um den Runner zu starten



Neuen Project Runner einrichten

- Neues Projekt erstellen
- Projekt-Pipeline erstellen
- Project Runner erstellen und registrieren
- Pipeline triggern, um den Runner zu starten



Neues Projekt erstellen (optional, falls eins besteht)

1. Linke Sidebar oben
→ „+“ (Create new)
→ New project/repository
2. Create blank project
3. Projektdetails eingeben
 - Project name
 - Project slug
4. Abschließend: Create project



Eigenen Project Runner benutzen

- Neues Projekt erstellen
- Projekt-Pipeline erstellen
- Project Runner erstellen und registrieren
- Pipeline triggern, um den Runner zu starten



Projekt-Pipeline erstellen

- .gitlab-ci.yml Datei im Projekt erstellen
- = Konfiguration der CI/CD Pipeline
- Inhalt:
 - Struktur und Reihenfolge der Jobs, welche durch den Runner ausgeführt werden
 - Entscheidungen, welche der Runner bei bestimmten Bedingungen (conditions) treffen soll



Projekt-Pipeline erstellen

1. Linke Sidebar → „Search or go to“ → Projekt suchen
2. „Project overview“ auswählen
3. „+“ Icon in der Projektübersicht (nicht Sidebar!) auswählen → „New file“
4. „Filename“: .gitlab-ci.yml
5. Beispielkonfiguration:

stages:

- build
- test

job_build:

stage: build
script:

- echo "Building the project"

job_test:

stage: test
script:

- echo "Running tests"

6. „Commit changes“



Eigenen Project Runner benutzen

- Neues Projekt erstellen
- Projekt-Pipeline erstellen
- Project Runner erstellen und registrieren
- Pipeline triggern, um den Runner zu starten



Project Runner erstellen und registrieren

Voraussetzung: Maintainer-Rechte für das Projekt

1. Eigenes Projekt auswählen
2. „Settings“ → „CI/CD“
3. Runners-Sektion aufklappen (Expand)
4. „Project runners“ → „New project runner“
5. „Tags“ → „Run untagged jobs“ auswählen
6. „Create runner“
7. On-screen Anweisungen befolgen für das OS, auf dem der Runner läuft (lokaler Rechner)
 1. Runner registrieren
 2. Executor auswählen (shell)
 3. Runner starten



Eigenen Project Runner benutzen

- Neues Projekt erstellen
- Projekt-Pipeline erstellen
- Project Runner erstellen und registrieren
- Pipeline triggern, um den Runner zu starten



Pipeline triggern, um den Runner zu starten

1. Eignes Projekt auswählen
2. „Build“ → „Pipelines“
3. „Run pipeline“
4. Job selektieren → Man sieht die dazugehörigen Logs



Live Demo: Self-managed Project Runner

- Runner Executable installieren
- Neuen Project Runner einrichten
- Möglichkeiten zur Konfiguration explorieren



Möglichkeiten zur Konfiguration explorieren

- Anhalten und Fortsetzen eines Runners
- Einen Project Runner löschen
- Project Runner für ein anderes Projekt aktivieren
- Project Runner für andere Projekte sperren



Möglichkeiten zur Konfiguration explorieren

- Anhalten und Fortsetzen eines Runners
- Project Runner löschen
- Project Runner für ein anderes Projekt aktivieren
- Project Runner für andere Projekte sperren



Anhalten und Fortsetzen eines Runners

Voraussetzung: Administrator oder Maintainer-Rechte für das Projekt

1. Gewünschtes Projekt in GitLab auswählen
2. „Settings“ → „CI/CD“
3. „Runners“ aufklappen
4. Unter „Assigned project runners“ den Runner finden
 - Pause-Symbol zum Pausieren
 - Play-Symbol zum Fortsetzen



Möglichkeiten zur Konfiguration explorieren

- Anhalten und Fortsetzen eines Runners
- Project Runner löschen
- Project Runner für ein anderes Projekt aktivieren
- Project Runner für andere Projekte sperren



Project Runner löschen

Voraussetzung: Administrator oder Maintainer-Rechte für das Projekt

- Achtung: Runner wird permanent gelöscht!
- 1. Gewünschtes Projekt in GitLab auswählen
- 2. „Settings“ → „CI/CD“ auswählen
- 3. „Runners“ aufklappen
- 4. Unter „Assigned project runners“ den Runner finden
- 5. „Remove runner“ auswählen
- 6. Mit „Remove“ das Löschen bestätigen



Möglichkeiten zur Konfiguration explorieren

- Anhalten und Fortsetzen eines Runners
- Project Runner löschen
- Project Runner für ein anderes Projekt aktivieren
- Project Runner für andere Projekte sperren



Project Runner für ein anderes Projekt aktivieren

Voraussetzung: Mindestens die Maintainer-Rechte für

- Projekt, in dem der Runner bereits aktiviert ist
- Projekt, in dem der Runner aktiviert werden soll
- Project Runner darf nicht gesperrt sein

1. Gewünschtes Projekt auswählen
2. „Settings“ → „CI/CD“
3. „Runners“ aufklappen
4. Im Bereich „Project runners“
 1. Gewünschten Runner auswählen und
 2. „Enable for this project“ selektieren



Möglichkeiten zur Konfiguration explorieren

- Anhalten und Fortsetzen eines Runners
- Project Runner löschen
- Project Runner für ein anderes Projekt aktivieren
- Project Runner für andere Projekte sperren



Project Runner für andere Projekte sperren

- Project Runner können für andere Projekte gesperrt werden
 1. Gewünschtes Projekt auswählen
 2. „Settings“ → „CI/CD“
 3. „Runners“ aufklappen
 4. Zu (ent)sperrenden Project Runner auswählen
 5. „Edit“ (Stift-Icon) anklicken
 6. „Lock to current projects“ auswählen
 7. Mit „Save changes“ bestätigen



Aufgabe 1: Eigenen Project Runner installieren

1. Ziel: Erstes Gefühl für GitLab Runner erhalten

2. Schritte:

- GitLab Runner herunterladen
- GitLab Runner nach der Anleitung für Ihr Betriebssystem installieren

Aufgabe 2: Eigene Pipeline mit einem Project Runner starten

1. Ziel: Verständnis über Runner schaffen

2. Schritte:

- Erstellen oder nutzen Sie Ihr eigenes Projekt
- Erstellen Sie eine Pipeline für Ihr Projekt
- Erstellen und registrieren Sie den Runner
- Starten Sie Ihre Pipeline



Live Demo: Group Runners

- Einen Group Runner mit Authentication Token erstellen
- Group Runners anzeigen lassen
- Anhalten und Fortsetzen eines Runners
- Einen Group Runner löschen
- Alte Group Runners bereinigen





Group Runners

- Einen Group Runner mit Authentication Token erstellen
- Group Runners anzeigen lassen
- Anhalten und Fortsetzen eines Runners
- Einen Group Runner löschen
- Alte Group Runners bereinigen



Einen Group Runner mit Authentication Token erstellen

Voraussetzung: Owner-Rechte für die Gruppe

1. Gewünschte Gruppe auswählen in GitLab
2. „Build“ → „Runners“ auswählen
3. „New group runner“ auswählen
4. Tags auswählen bzw. erstellen, falls nicht vorhanden, dann „Run untagged“ auswählen
5. Optional: Beschreibung ausfüllen
6. Optional: Konfiguration ausfüllen
7. „Create runner“ auswählen
8. Die Anweisungen von GitLab folgen, um den Runner zu registrieren



Group Runners

- Einen Group Runner mit Authentication Token erstellen
- Group Runners anzeigen lassen
- Anhalten und Fortsetzen eines Runners
- Einen Group Runner löschen
- Alte Group Runners bereinigen



Group Runners anzeigen lassen

Voraussetzung: Maintainer- oder Owner-Rechte für die Gruppe

Alle Runner einer Gruppe und dessen Sub-Gruppen sowie Projekte kann man wie folgt einsehen:

1. Gewünschte Gruppe in GitLab auswählen
2. „Build“ → „Runners“ auswählen
3. Filter, um nur Sub-Gruppen zu sehen:
 - „Show only inherited“ toggeln



Group Runners

- Einen Group Runner mit Authentication Token erstellen
- Group Runners anzeigen lassen
- Anhalten und Fortsetzen eines Runners
- Einen Group Runner löschen
- Alte Group Runners bereinigen



Anhalten und Fortsetzen eines Runners

Voraussetzung: Administrator oder Owner-Rechte für die Gruppe

- Group Runner anhalten, damit dieser keine Jobs mehr von Sub-Gruppen und Projekten annimmt
 - Bei Benutzung durch mehrere Projekte → für alle pausiert
1. Gewünschte Gruppe in GitLab auswählen
 2. „Build“ → „Runners“ auswählen
 3. Den gewünschten Runner suchen
 4. In der Liste von Runnern
 - Pause-Symbol zum Pausieren
 - Play-Symbol zum Fortsetzen



Group Runners

- Einen Group Runner mit Authentication Token erstellen
- Group Runners anzeigen lassen
- Anhalten und Fortsetzen eines Runners
- Einen Group Runner löschen
- Alte Group Runners bereinigen



Einen Group Runner löschen

Voraussetzung: Administrator oder Owner-Rechte für die Gruppe

- Achtung: Runner wird permanent gelöscht!
- 1. Gewünschte Gruppe auswählen
- 2. „CI/CD“ → „Runners“ auswählen
- 3. Entsprechenden Runner suchen
- 4. Löschen des Group Runners
 - Einzelnen Runner zu löschen → „Delete runner“ (Löschen-Symbol)
 - Mehrere Runner zu löschen → Checkbox selektieren neben dem Runner und „Delete selected“ auswählen
 - Alle Runner zu löschen → Die Checkbox für alle Runner auswählen und „Delete selected“ auswählen
- 5. „Permanently delete runner“ auswählen



Group Runners

- Einen Group Runner mit Authentication Token erstellen
- Group Runners anzeigen lassen
- Anhalten und Fortsetzen eines Runners
- Einen Group Runner löschen
- Alte Group Runners bereinigen



Alte Group Runners bereinigen

Voraussetzung: Owner-Rechte für die Gruppe

- Inaktive (> 3 Monate) Group Runner (= „stale“) können automatisch bereinigt werden
 - Group Runners = erstellt auf Gruppenebene
1. Gewünschte Gruppe in GitLab auswählen
 2. „Settings“ → „CI/CD“ auswählen
 3. „Runners“ aufklappen
 4. „Enable stale runner cleanup“ toggeln



Aufgabe 3: Group Runner kennenlernen

1. Ziel: Verständnis über Runner schaffen

2. Schritte:

- Erstellen oder nutzen Sie eine Gruppe
- Erstellen Sie einen Group Runner
- Lassen Sie sich Ihren Group Runner anzeigen
- Stoppen und Starten Sie einen Group Runner



I need a
< br />

TL;DR

- GitLab Job
 - Kleinste Komponente einer Pipeline
 - 1-n auszuführende Befehle
- GitLab Runner
 - Agent, oft auf anderer Infrastruktur
 - GitLab Server gibt Anweisung über nächste Job-Ausführung
- Runner Executor
 - Jeder Runner hat mindestens einen Executor
 - Executor = Umgebung für die Ausführung des GitLab Jobs

Executors

- Verfügbare Executors in GitLab
 - Shell
 - SSH
 - VirtualBox
 - Parallels
 - Docker
 - Kubernetes
- Abhängig vom Use Case!
 - Es gibt nicht den „besten“ Executor!

Executor Chart

Executor	SSH	Shell	VirtualBox	Parallels	Docker	Kubernetes
Clean build environment for every build	×	×	✓	✓	✓	✓
Reuse previous clone if it exists	✓	✓	×	×	✓	×
Runner file system access protected (5)	✓	×	✓	✓	✓	✓
Migrate runner machine	×	×	partial	partial	✓	✓
Zero-configuration support for concurrent builds	×	× (1)	✓	✓	✓	✓
Complicated build environments	×	× (2)	✓ (3)	✓ (3)	✓	✓
Debugging build problems	easy	easy	hard	hard	medium	medium

- (1) Möglich, aber problematisch, wenn der Build auf der Build Maschine installierte Services nutzt
- (2) Erfordert manuelle Dependency Injection
- (3) Beispielsweise mit Vagrant

Compatibility Chart

Executor	SSH	Shell	VirtualBox	Parallels	Docker	Kubernetes	Custom
Secure Variables	✓	✓	✓	✓	✓	✓	✓
.gitlab-ci.yml: image	x	x	✓ (1)	✓ (1)	✓	✓	✓ (via <code>\$CUSTOM_ENV_CI_JOB_IMAGE</code>)
.gitlab-ci.yml: services	x	x	x	x	✓	✓	✓
.gitlab-ci.yml: cache	✓	✓	✓	✓	✓	✓	✓
.gitlab-ci.yml: artifacts	✓	✓	✓	✓	✓	✓	✓
Passing artifacts between stages	✓	✓	✓	✓	✓	✓	✓
Use GitLab Container Registry private images	n/a	n/a	n/a	n/a	✓	✓	n/a
Interactive Web terminal	x	✓ (UNIX)	x	x	✓	✓	x

Shell Executor

- Umgebung des Runners = Umgebung der Jobausführung
 - Analog Jenkins o.ä.
- Dependencies müssen auf OS installiert sein
- Docker Images in .gitlab-ci.yml werden ignoriert!
 - Selbst wenn Docker installiert ist
- Alles zur Laufzeit vorhanden
 - → Umgehende Ausführung der Jobs

Shell Executor

- Use Case
 - Native Umgebung
 - Einheitliche Builds
- Zu beachten
 - Umgebung
 - Versionen installierter Software und Dependencies
 - Abweichungen in der Infrastruktur
 - Andere Jobs
 - Keine Isolation
 - „Left-overs“ → keine saubere Build-Umgebung
 - Vertrauen (voller Zugriff auf Projekt & Secrets)

Docker Executor

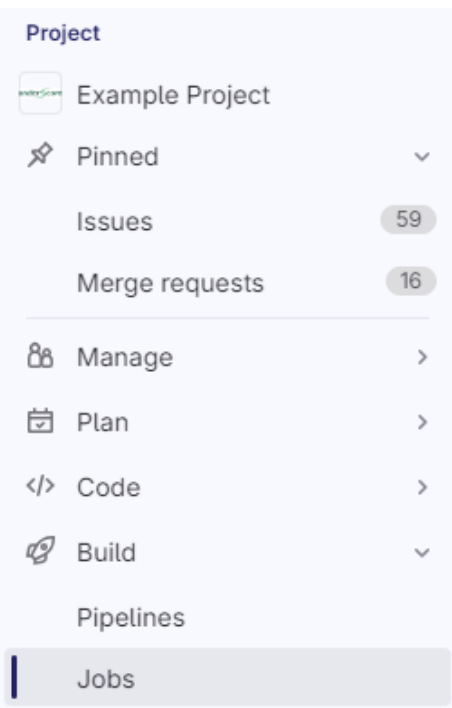
- Images auf Ebene von Jobs individuell konfigurierbar
- Simple Laufzeitumgebungen
- Mehrere Jobs gleichzeitig
 - Ohne Interferenzen (außer Systemlast/Performance)
- Für die meisten Projekte sinnvoll
- Alle Abhängigkeiten definiert

Docker Executor

- Use Case
 - Saubere, isolierte Umgebung für jeden Job
 - Projekte unabhängig voneinander
- Zu beachten
 - Overhead vom „Pulling“
 - Docker Image Download pro Job
 - Dependencies (z.B. Maven)
 - Austausch von Build-Artefakten zwischen Jobs (Containern) erschwert

Welcher Executor wird genutzt?

- Steht in den Logs vom GitLab Job



```
1 Running with gitlab-runner 15.3.0 (bbcb5aba)
2   on docker-default HFitoxxJ
3 Preparing the "docker" executor
4 Using Docker executor with image gitlab.ads.anderscore.com:5006/example/project:latest ...
5 Authenticating with credentials from job payload (GitLab Registry)
6 Pulling docker image gitlab.ads.anderscore.com:5006/example/project:latest ...
7 Using docker image sha256:4b078e4f3a50b99dfbfc1e92c5736eb598a7662fd918f0ef83f3df2e79b5ba0e
  76778b0f8ba01b48962ad4ebb2657ed520c30ad6774f6a ...
8 Preparing environment
9 Running on runner-hfitoxxj-project-63-concurrent-0 via ed568276aa82...
10 Getting source from Git repository
11 Fetching changes...
12 Reinitialized existing Git repository in /builds/example/project/.git/
13 Checking out 3ff31f42 as master...
14 Removing Gemfile.lock
```