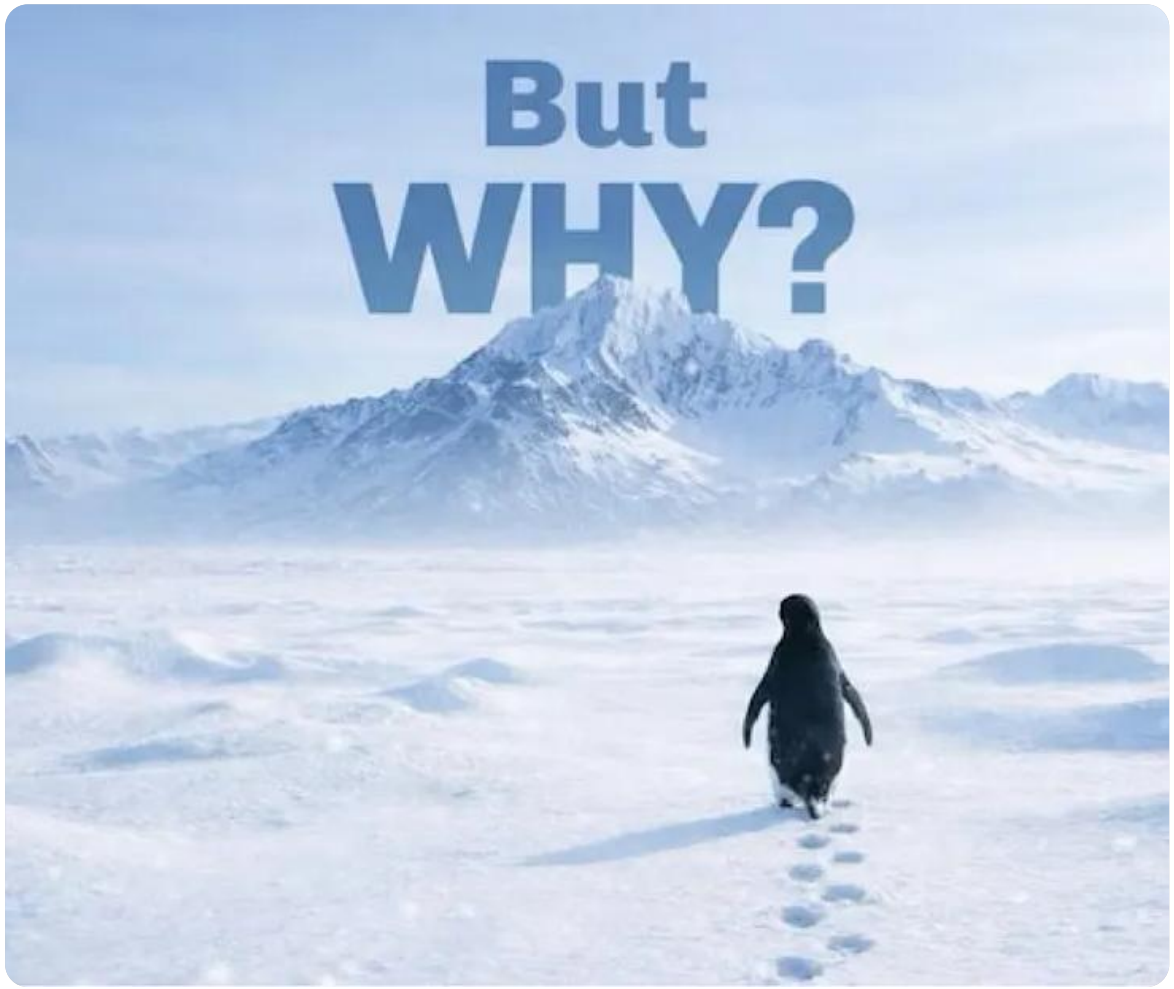
Adieu Jenkins: warum wir über 100 Jenkins-Pipelines auf GitLab CI migriert haben

Java Forum Stuttgart 2026

09.07.2026

Gründe für Migration



Werner Herzog: Encounters at the End of the World

- CD → Build-Infrastruktur wie Prod!
- Security- & Compliance-Anforderungen an Build-Infra
- Hoher Cognitive Load & time-sink für Teams → Zentralisierungsbedarf

Zentralisierung: GitLab CI vs CloudBees Jenkins

- GitLab bereits als SCM im Einsatz, dediziertes Platform-Team mit klarem Betriebs-Auftrag
  - Gutes Alignment mit GitOps-Strategie
  - Zu dem Zeitpunkt (~ 2017) war CI aber noch in den Kinderschuhen
- Enterprise-Support bei beiden möglich (Premium/Ultimate vs. bspl. CloudBees)
- Elastische Skalierungsfähigkeit für 20+ Teams (100+ parallele Builds mit klaren peak times) ebenfalls bei beiden möglich (Runners vs. Master-Nodes-Modell)

Migrationsvorgehen: Shared Pipeline Library → CI/CD Component

```groovy
#!groovy
@Library('pipeline-library@2.1.12') _

serviceDeploymentPipeline([
    serviceName             : "[redacted]",
    dockerImageBaseName     : "[redacted]",
    pipelines               : ["[redacted]
    deployWithGateway       : true,
    sbomEnabled             : true,
    sbomBreaking            : true,
    gitlabProjectId         : "[redacted]",
    publishCdc              : { ->
      dir('./api-contract-test') {
        script {
          pom = readMavenPom(file: 'pom.xml')
          groupId = pom.parent.groupId
          artifactId = pom.artifactId
        }
```

Migrationsvorgehen: Shared Pipeline Library → CI/CD Component

```yaml
include:
  - component: $CI_SERVER_FQDN/[redacted]/components/pipelines/maven-library@1.27.0
    inputs:
      sbom_breaking_license_check: "true"
      java_version: jdk21
      build_job_tag: large
```

Jenkins Pipeline Library

```
3  def call(args) {
4
5    assert args.mavenSettings: "Error: Maven settings must be provided."
6    assert args.sonarQubeEnv: "Error: SonarQube Environment must be provided."
7
8    withSonarQubeEnv("${args.sonarQubeEnv}") {
9      withMaven(mavenSettingsFilePath: "${args.mavenSettings}") {
10       withEnv([    MAVEN_PATH_CORRECTION]) {
11         sh "$    .MAVEN_COMMAND} -e org.sonarsource.scanner.maven:sonar-maven-plugin:5.0.0.4389:sonar"
12       }
13     }
14   }
15 }
```

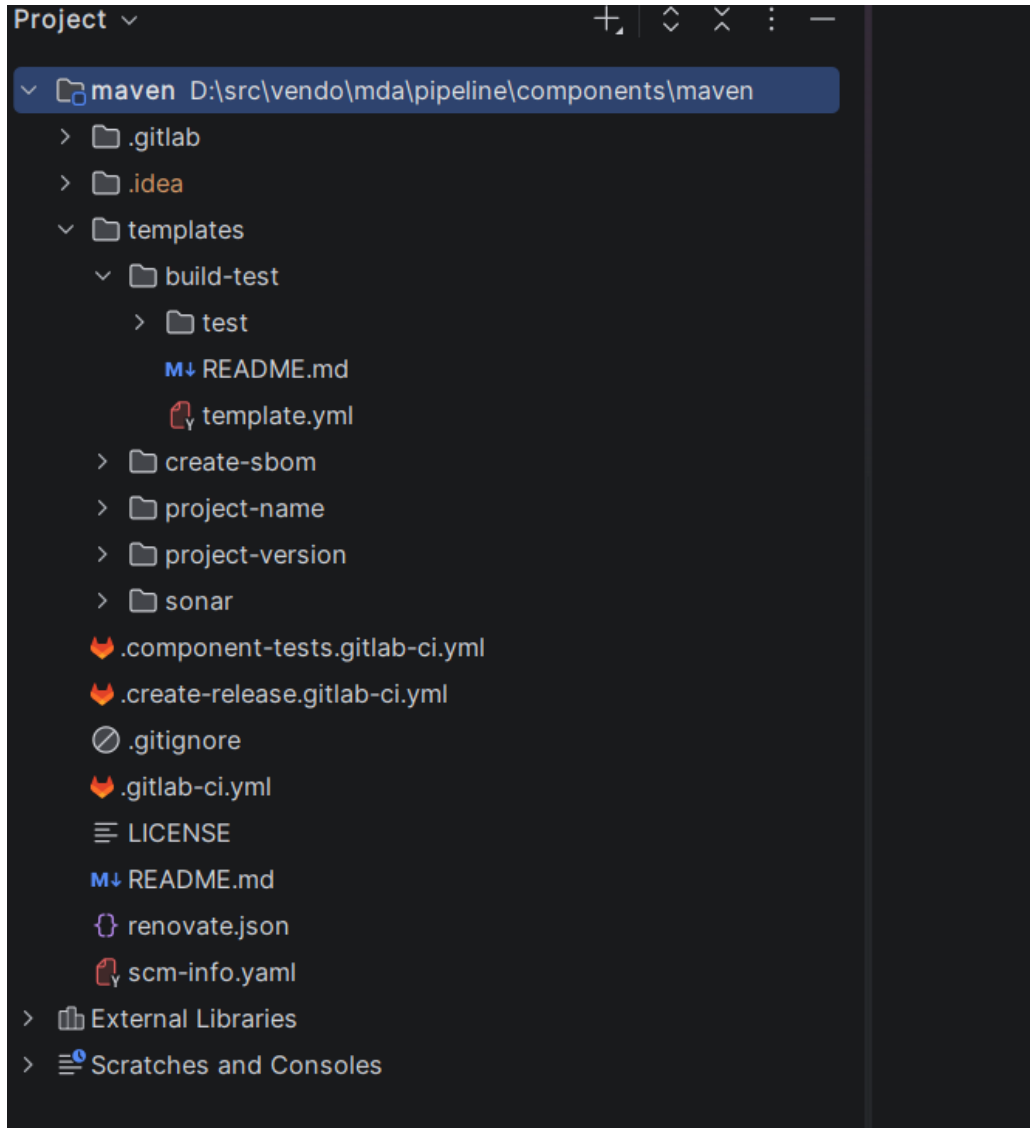
GitLab Component

```yaml
spec:
  inputs:
    stage:
      default: sonar-analysis
    sonarqube_host:
      type: string
      default: "[redacted]"
      description: SonarQube host url
    sonarqube_token:
      type: string
      description: SonarQube access token
      default: "${SONAR_TOKEN}"
    build_version:
      type: string
      description: Build version
      default: "${BUILD_VERSION}"
    maven_cli_opts:
      type: string
      description: Maven cli opts
    java_version:
      type: string
      default: jdk17
      options:
        - jdk17
        - jdk21
      description: Java version to be used
    job_name:
      type: string
      default: sonar
      description: Job name
---

include:
  - local: /templates/defaults.yml
  - local: /templates/utils.yml

$[[ inputs.job_name | expand_vars ]]:
  stage: $[[ inputs.stage ]]
  image:
    name: [redacted]/builder-cicdtools:$[[ inputs.java_version | expand_vars ]]-latest
    pull_policy: if-not-present
  variables:
    GIT_DEPTH: "0" # Tells git to fetch all the branches of the project, required by the analysis task
  extends: .job-medium
  before_script:
    - !reference [ .log_utils, logger ]
  script:
    - mvn $[[ inputs.maven_cli_opts ]] -s ${MAVEN_SETTINGS_XML} versions:set -DnewVersion=$[[ inputs.build_version ]]
    - |
      mvn $[[ inputs.maven_cli_opts ]] -s ${MAVEN_SETTINGS_XML} org.sonarsource.scanner.maven:sonar-maven-plugin:5.0.0.4389:sonar \
      -Dsonar.qualitygate.wait=true \
```

GitLab Component

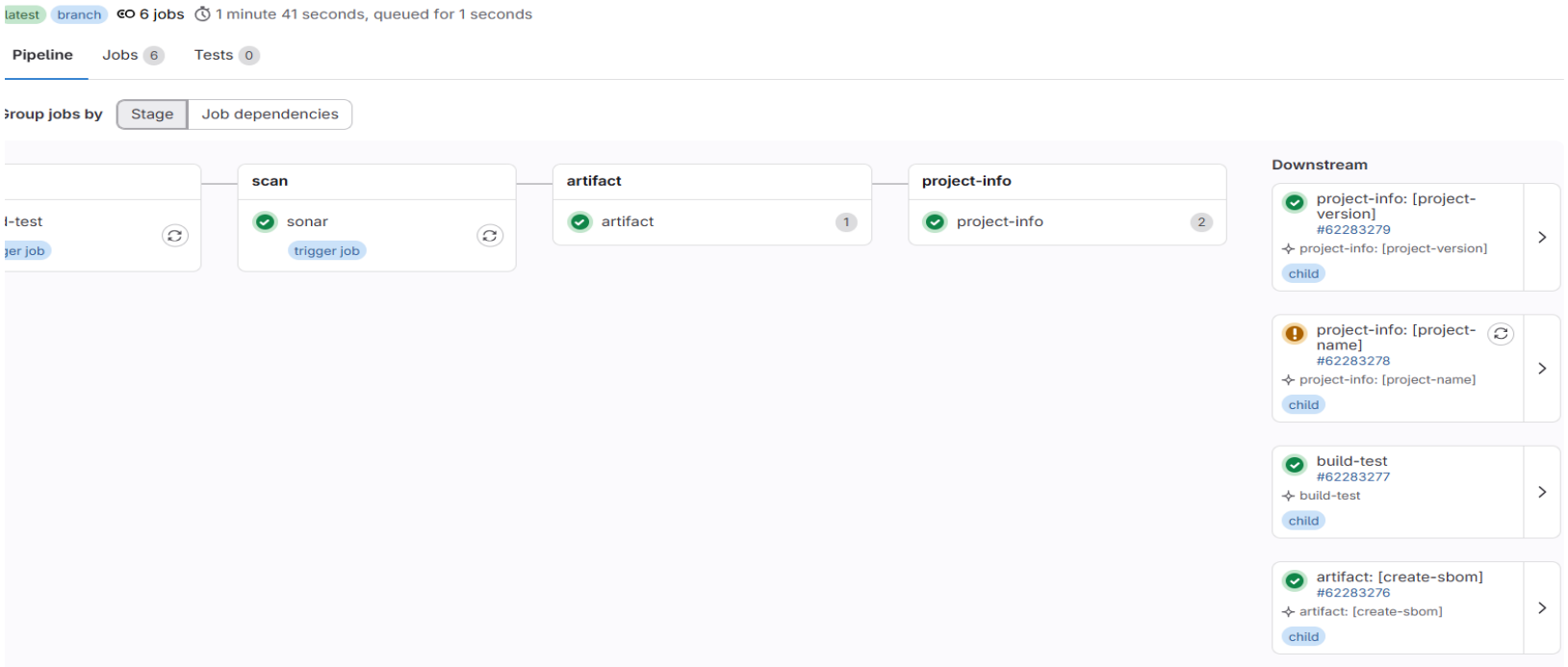
```
31  ---
32
33  include:
34    - local: /templates/defaults.yml
35    - local: /templates/utils.yml
36
37  $[[ inputs.job_name | expand_vars ]]:
38    stage: $[[ inputs.stage ]]
39    image:
40      name: [redacted]/builder-cicdtools:$[[ inputs.java_version | expand_vars ]]-latest
41      pull_policy: if-not-present
42    variables:
43      GIT_DEPTH: "0" # Tells git to fetch all the branches of the project, required by the analysis task
44    extends: .job-medium
45    before_script:
46      - !reference [ .log_utils, logger ]
47    script:
48      - mvn $[[ inputs.maven_cli_opts ]] -s ${MAVEN_SETTINGS_XML} versions:set -DnewVersion=$[[ inputs.build_version ]]
49      - |
50        mvn $[[ inputs.maven_cli_opts ]] -s ${MAVEN_SETTINGS_XML} org.sonarsource.scanner.maven:sonar-maven-plugin:5.0.0.4389:sonar \
51        -Dsonar.qualitygate.wait=true \
```

Struktur von CI/CD Component Projekt

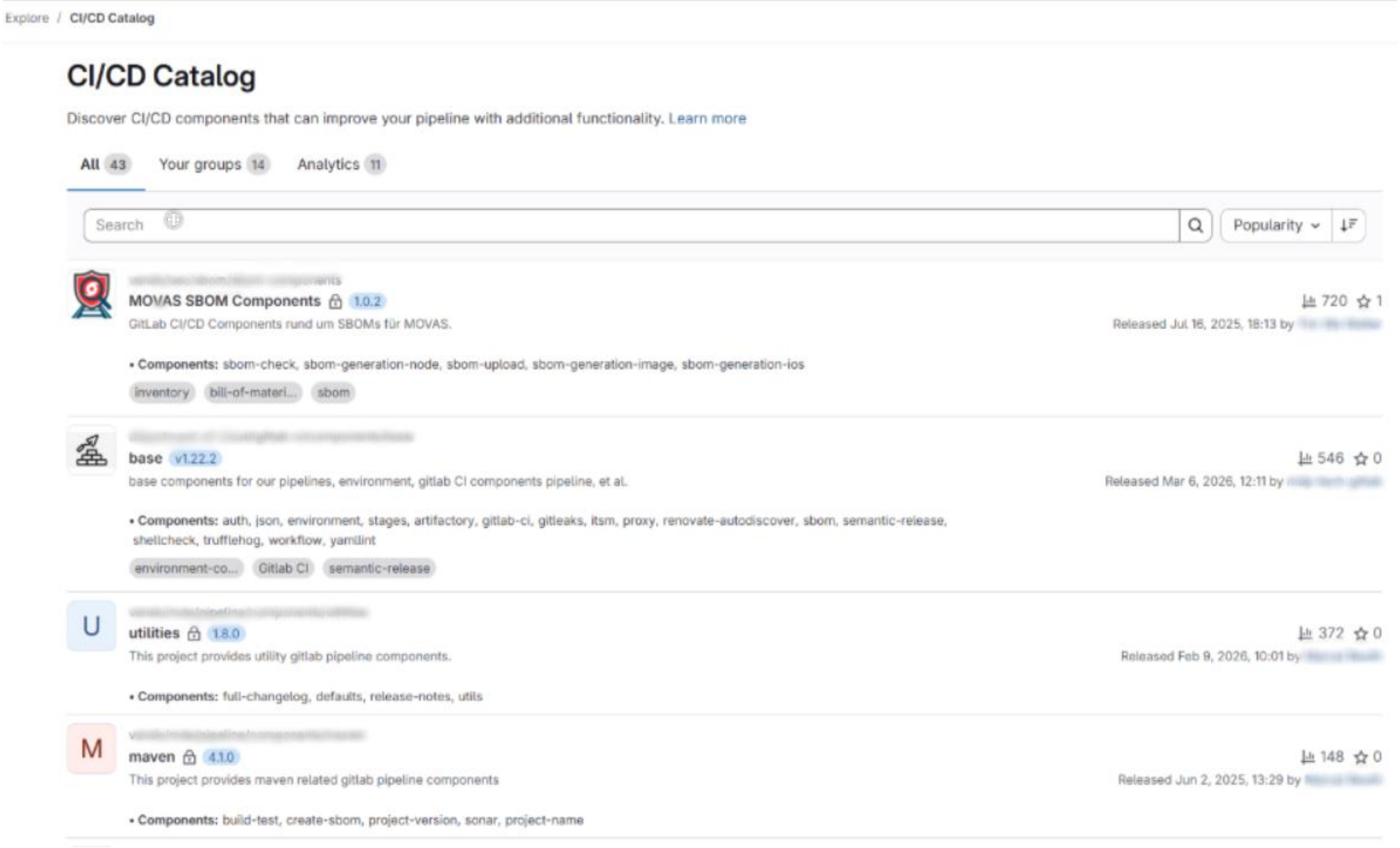


Testing von Pipeline-Library vs CI/CD Component

- Für Jenkins z.B. mittels jenkins-pipeline-unit Library möglich
- Für GitLab CI/CD Component:
  - Isolierter Test der Component
  - sowie Test im „Verbund“ mehrere Components in größerer Pipeline (insb. relevant für Components), die aufeinander aufbauen
- mittels .gitlab-ci.yaml im Wurzelverzeichnis des Component Projekts
- Entweder direkt mit Test-Stage oder über Downstream-Pipelines



GitLab CI/CD Catalog

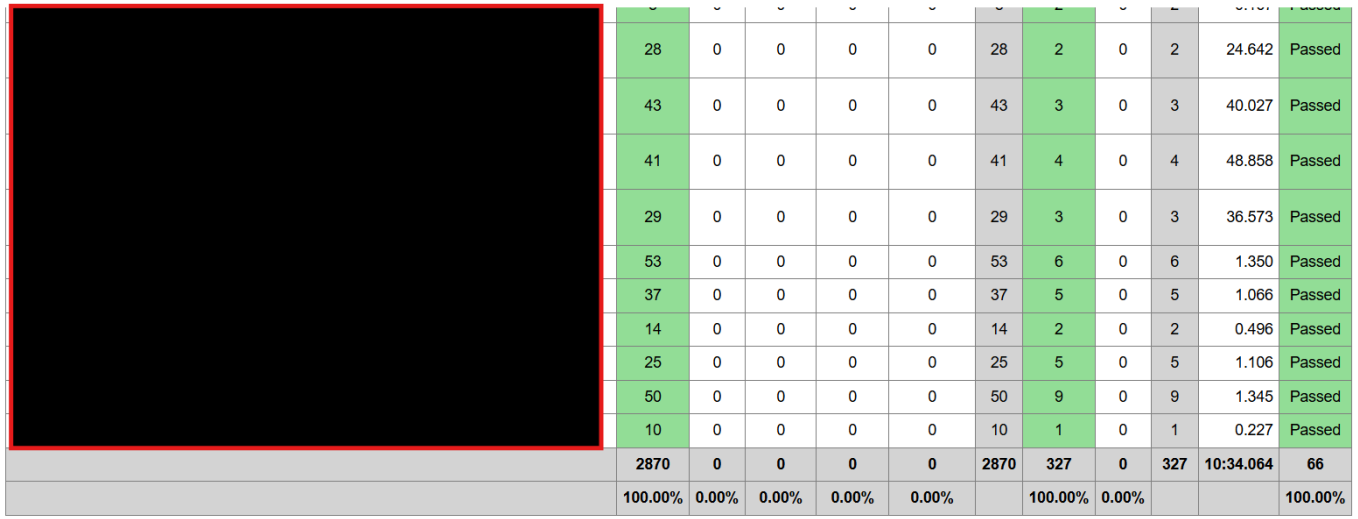


Lessons learned & Zwischenfazit zur Migration

- Viele Jenkins-Instanzen abgeschaltet → beträchtliche Ersparnis auch bei Berücksichtigung kompensierender GitLab Runner
- Spür- und messbare Performance-Steigerung in Teams durch Zentralisierung der Build-Infra (weniger Maintenance-Aufwände und klarer Fokus auf Anwendungscode)
- Weniger Downtimes des Build-Servers (wg. dediziertem Plattform-Team), sowie „Schluckauf“
- Pipeline-Laufzeiten reduziert* trotz ca. 1 Mio. Pipeline-Runs pro Monat
- Schrittweise Migration je Stage klappt oft recht gut mit bekannten AI-Tools bei ausreichend Kontext
- Zentralisierte Lösung für alle Teams ist zwar stabiler, aber auch behäbiger (z.B. keine Beta-Features, Aktivierung optionaler Features dauert z.T. lange wg. Abwägung der Auswirkungen)
- Enterprise Support in aller Regel nur mit „Durchlauferhitzer“ (starker Ownership-Anspruch Platform-Team)
- Aufwands-Investition war erheblich, Gesamt-Aufwand eher unterschätzt, dadurch Erklärungsbedarf (Business Case lässt sich berechnen)
- Migrations-Dauer mehrere Monate (teilw. noch ongoing)!

Was wir an Jenkins vermissen

- Plugins & Grad der „Customization“
- Möglichkeit, Bugs und Features selbst durch Contributions beizukommen*
- Doku/Troubleshooting-Datenbasis
- Groovy
- … dass er einem keine tollen neuen Features aufschwatzen will
- 1-Klick grafische Test-Reports für z.B. Cucumber oder Gatling



	28	0	0	0	0	28	2	0	2	24.642	Passed
	43	0	0	0	0	43	3	0	3	40.027	Passed
	41	0	0	0	0	41	4	0	4	48.858	Passed
	29	0	0	0	0	29	3	0	3	36.573	Passed
	53	0	0	0	0	53	6	0	6	1.350	Passed
	37	0	0	0	0	37	5	0	5	1.066	Passed
	14	0	0	0	0	14	2	0	2	0.496	Passed
	25	0	0	0	0	25	5	0	5	1.106	Passed
	50	0	0	0	0	50	9	0	9	1.345	Passed
	10	0	0	0	0	10	1	0	1	0.227	Passed
	2870	**0**	**0**	**0**	**0**	**2870**	**327**	**0**	**327**	**10:34.064**	**66**
	100.00%	**0.00%**	**0.00%**	**0.00%**	**0.00%**		**100.00%**	**0.00%**			**100.00%**

Was wir an Jenkins NICHT vermissen

- Plugins
- Flaky/Fehlschlagende „Restart from Stage“
- Groovy

Zusammenfassung

- Sollte ich jetzt sofort auf GitLab CI umsteigen?
- Was ist die bessere Engine – Jenkins oder GitLab CI?
- Ist Jenkins tot?
- Würden wir den hohen Aufwand für die Migration rückblickend erneut investieren?
- Wie sieht tragfähige Build-Infrastruktur für große Softwareorganisationen aus?
- Bei Greenfield, welche Engine sollte ich wählen?

Vielen Dank


DB