

Verteilte Systeme und Komponenten

# **Buildserver**

**Technologien und Funktionsweise**

Roland Gisler



# Inhalt

- Was ist ein Buildserver?
- Beispiele von Buildserver-Produkten
- Einsatzszenarien
- VSK: Buildserver im Logger-Projekt

# Lernziele

- Sie wissen was ein Buildserver ist.
- Sie kennen die Vorteile beim Einsatzes eines Buildservers.
- Sie kennen beispielhafte Produkte von Buildservern und können diese als Anwender\*in nutzen.
- Sie kennen das Potential von automatisierten CI/CD-Prozessen.

# Bob the Builder?



<http://www.bobthebuilder.com/de/>

# Was ist ein Buildserver?

- Serversoftware, welche einen bereits automatisierten Build eines Softwareprojektes ausführt, und die Resultate allen Entwickler\*innen im Team zur Verfügung stellt.
- Auslösung eines Builds aufgrund verschiedener Events:
  - Automatisch durch Änderungen im Versionskontrollsystem.
  - Automatisch durch Zeitsteuerung.
  - Manuell durch Anwender\*in.
- Positive Effekte
  - Entlastung von Entwickler\*innen von repetitiven Aufgaben.
  - Häufigere Verifikation (Buildprozess, Tests, Deploying etc.).
  - Statistische Information über Entwicklungsprozess.
  - Offensive (automatische) Information über den Zustand der Projekte.

# **Beispiele von Buildservern**

# Beispiele von Buildservern

The image displays three screenshots of build server interfaces:

- Jenkins Dashboard (Top Left):** Shows the Jenkins web interface for 'HSLU Informatik - Modul VSK'. It includes a sidebar with navigation links and a main table of build jobs.
- Jenkins Configuration (Top Right):** Shows the 'Continuous Build' configuration page for a Jenkins job, including tabs for Overview, History, Change Log, Issue Log, Statistics, Compatible Agents, Pending Changes, and Settings.
- GitHub Actions Pipelines (Bottom Right):** Shows the GitHub Actions 'Pipelines' page for a repository, displaying a list of pipelines with their status, ID, trigger, commit, stages, and duration.

Name	Last Version	# OG	# CS	# PMD	# SB	Zielerreichung	Letzte Status
g00-loggerinterface	1.0.0-SNAPSHOT	0	0	0	0	100.0%	4.7 Tage
g01-game	1.0.0-SNAPSHOT	283	205	22	39	100.0%	4.7 Tage
g01-logger	1.0.0-SNAPSHOT	0	0	0	0	100.0%	4.7 Tage
g01-stringpersistor	1.0.0-SNAPSHOT	0	0	0	0	100.0%	4.7 Tage
g02-game	1.0.0-SNAPSHOT	283	205	22	39	100.0%	4.6 Tage
g02-logger	1.0.0-SNAPSHOT	-	-	-	-	100.0%	1.4 Tage > 4.7 Tage
g02-stringpersistor	1.0.0-SNAPSHOT	0	0	0	0	100.0%	4.7 Tage
g03-game	1.0.0-SNAPSHOT	283	205	22	39	100.0%	4.6 Tage
g03-logger	1.0.0-SNAPSHOT	0	0	0	0	100.0%	4.6 Tage
g03-stringpersistor	1.0.0-SNAPSHOT	0	0	0	0	100.0%	4.7 Tage
g04-game	1.0.0-SNAPSHOT	283	205	22	39	100.0%	4.7 Tage
g04-logger	1.0.0-SNAPSHOT	0	0	0	0	100.0%	4.7 Tage
g04-stringpersistor	1.0.0-SNAPSHOT	0	0	0	0	100.0%	4.7 Tage
g05-game	1.0.0-SNAPSHOT	283	205	22	39	100.0%	4.6 Tage
g05-logger	1.0.0-SNAPSHOT	2	2	0	0	100.0%	4.7 Tage
g05-stringpersistor	1.0.0-SNAPSHOT	0	0	0	0	100.0%	4.7 Tage
g06-game	1.0.0-SNAPSHOT	283	205	22	39	100.0%	8.1 Stunden > 8.2 Stunden
g06-logger	1.0.0-SNAPSHOT	-	-	-	-	25.0%	8.1 Stunden > 8.2 Stunden
g06-stringpersistor	1.0.0-SNAPSHOT	0	0	0	0	100.0%	4.6 Tage
g07-game	1.0.0-SNAPSHOT	283	205	22	39	100.0%	4.6 Tage
g07-logger	1.0.0-SNAPSHOT	32	14	7	0	33.33%	4.6 Tage
g07-stringpersistor	1.0.0-SNAPSHOT	0	0	0	0	100.0%	4.6 Tage
g08-game	1.0.0-SNAPSHOT	283	205	22	39	100.0%	4.6 Tage
g08-logger	1.0.0-SNAPSHOT	0	0	0	0	100.0%	4.7 Tage
g08-stringpersistor	1.0.0-SNAPSHOT	0	0	0	0	100.0%	4.7 Tage

Status	Pipeline ID	Triggerer	Commit	Stages	Duration
failed	#86229	latest	2b348734	logger-setup Startet with the log...	00:00:22 8 hours ago
passed	#86228	latest	73ab06e1	master Added enum values	00:05:13 9 hours ago
passed	#86227	latest	beb6c656	master Added enum for the...	00:05:59 9 hours ago
passed	#83977	latest	e87f1a3c	master Initial Commit	00:01:52 3 weeks ago

# Buildserver - Ausgewählte Produkte (Beispiele)

- Open Source
    - Jenkins/Hudson – einer der aktuell populärsten Buildserver.
    - Go – Moderne Ansätze, vereinfacht, Pipelines, CD (status?).
    - CruiseControl – einer der ältesten Buildserver (retired)
    - Continuum – Ursprüngliche speziell für Maven-Projekte (retired).
  - Kommerzielle Server (häufig auch freie Community-Editions)
    - TeamCity – sehr funktionaler und gut skalierender Buildserver.
    - Bamboo – eng verknüpft mit JIRA (Issue-Tracking).
- ➔ Ideal für den Einsatz in Unternehmen (on-site/closed source).



# Buildserver - Cloud-Dienste (Beispiele)

- Hostingplattformen zur Projekt- und Codeverwaltung haben sich in der Cloud ja bereits schon lange etabliert.
  - Beispiele: Sourceforge, GitHub, BitBucket, GitLab etc.
- Dienste für Buildserver (CI) in der Cloud zogen nach, z.B.:
  - <https://travis-ci.com/>
  - <http://www.cloudbees.com/>
  - Nun häufig direkt in Hostingplattformen integrierte Services.
- Für Open Source Projekte häufig gratis!
  - Meist (sehr) gute, transparente Integration.
  - Projekte müssen aber oft «public» verfügbar sein (OSS).
- Derzeit findet eine Konsolidierung statt. Einige Dienste «sterben» gerade oder werden übernommen und integriert.

# Konfiguration von Buildservern

- Es gibt im wesentlichen **zwei** komplett **gegensätzliche** Ansätze!
- **Variante 1:** (typisch für «on-site» Produkte)
  - Konfiguration ist vom Projekt vollständig getrennt.
  - Wird (meist) interaktiv direkt auf dem Buildserver vorgenommen.
  - Infrastruktur (Buildagents etc.) wird «geschützt».
  - ➔ Es resultiert eine Art «Gewaltentrennung».
- **Variante 2:** (eher für Cloud- und Hosting-Plattform)
  - Konfiguration ist direkt im Projekt abgelegt (z.B. `.ym1`-Datei).
  - Wird direkt durch Entwickler\*innen konfiguriert.
  - Weniger Restriktiv, sehr häufig mit Docker (ad-hoc Agents).
  - ➔ Mehr Freiheit und Eigenverantwortung.
- Variante **1** eher in Organisationen, Variante **2** eher bei OSS.
  - Individuelle Ausnahmen bestätigen die Regel...

# **Einsatz von Buildservern**

# Einsatz von Buildserver

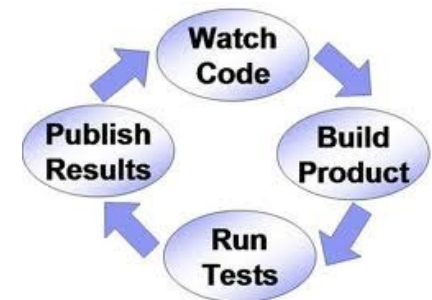
- Einsatz eines Buildservers **setzt** andere Technologien **voraus**:
  - **Automatisierte** Builds, z.B. mit Maven oder Gradle etc.
  - Einsatz eines Versionskontrollsystems, z.B. Git, Subversion etc.
- Man sollte auf eine saubere Aufgabentrennung zwischen den einzelnen Systemen und Technologien achten:
  - **Wann** wird ein Build ausgeführt: Buildserver / Anwender.
  - **Was** wird gebaut: Versionskontrollsystem.
  - **Wie** wird gebaut: Buildautomatisation.
  - **Wohin** gehen die Artefakte: Buildserver / Binary-Repo.
- Speziell das '**wie**' sollte **nie** im Buildserver umgesetzt, sondern immer durch den automatisierten Build abgedeckt werden.
  - Durch Buildtools wie z.B. Maven oder Gradle.

# Verschiedene Buildarten/-szenarien

- **Continuous** Builds: Automatisch bei einer Änderungen (Push/Merge-Request/Pull-Request) im Versionskontrollsystem.
  - Schnelle, möglichst kurze Builds.
  - Ziel: Schnelles Feedback für Entwickler\*innen.
- **Nightly** Builds: Automatisch nach Zeitsteuerung, typisch Nachts.
  - Eher umfangreiche, lange Builds.
  - Auch für zeitintensive Tests und Metriken geeignet.
  - Ziel: Am Morgen stehen umfassende Resultate zur Verfügung.
- **Release** Build: Manuell ausgelöst.
  - Build einer auslieferbaren Version, im VCS getagged.
  - Ziel: Reproduzierbarkeit gewährleisten.
  - Alternative: Build Pipeline.

# Integration und Verknüpfung

- Moderne Buildserver-Technologien zeichnen sich durch eine hohe Integrationsfähigkeit aus (analog zu Buildtools und IDE's).
  - Typisch über Plugin-Mechanismen realisiert.
- Integration von
  - verschiedenen Buildtools.
  - verschiedenen Versionskontrollsystemen.
  - verschiedenen Kommunikationstechnologien zur Notifikation.
  - verschiedene Visualisierungen / Plugins für IDE's.
  - etc.
- Verknüpfung mit
  - Issue-Tracking Systemen.
  - Code-Review Werkzeugen.
  - etc.



# **Logger-Projekt**

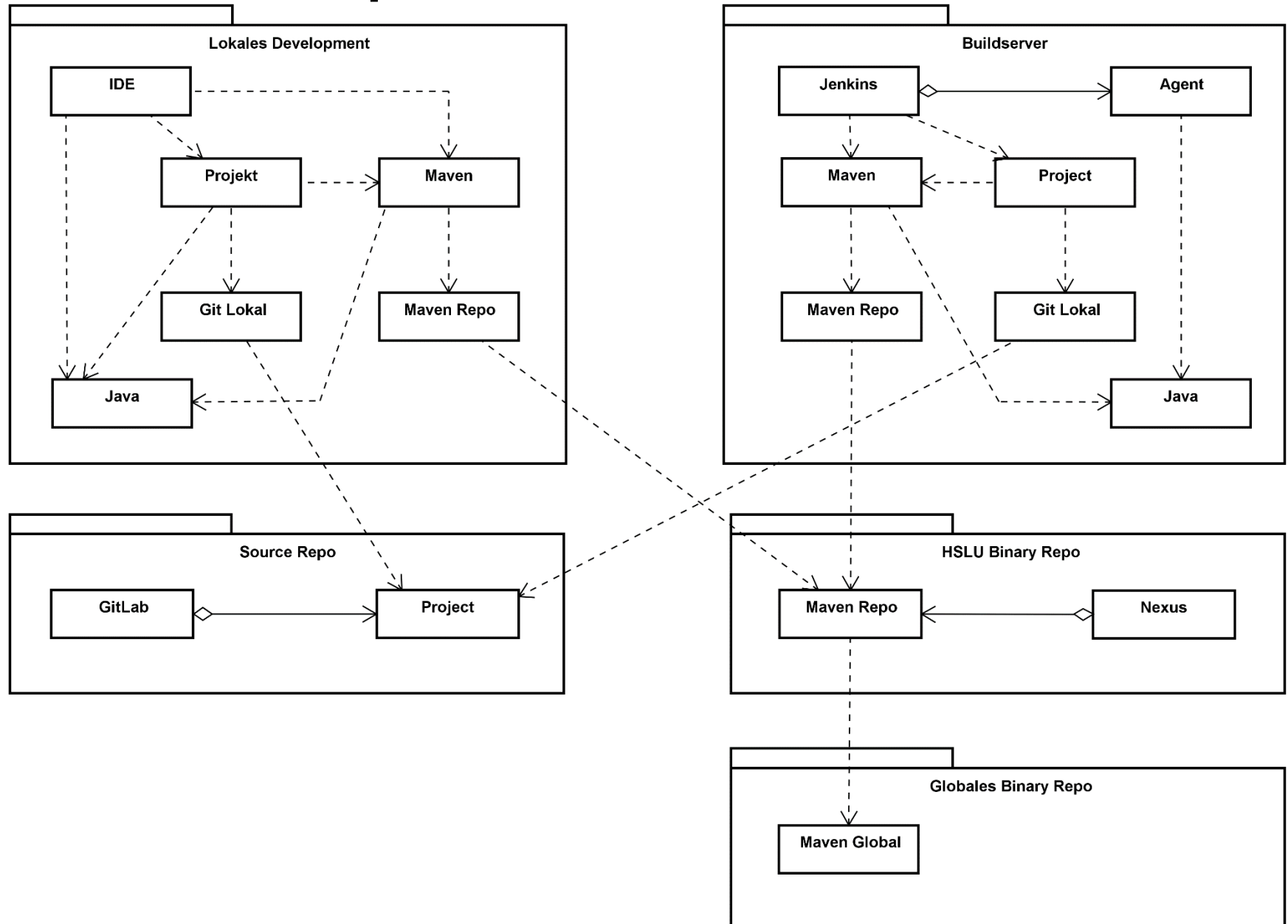
# Zugriff auf Buildserver

- Buildserver stehen **nur** im **internen** Netz der HSLU zur Verfügung
  - Zugriff von ausserhalb nur über **VPN** möglich.
- Buildserver VSK HS22: <https://jenkins-vsk.el.eee.intern/jenkins>
  - Login mit Enterpriselab-Login (analog GitLab)
  - **Achtung**: «**Neues**» HSLU-internes Root-Zertifikat (SSL), siehe:  
[https://wiki.enterpriselab.ch/el/public:help:certificate\\_import](https://wiki.enterpriselab.ch/el/public:help:certificate_import)  
<https://discuss.enterpriselab.ch/t/el-root-certificate-change/197>
- Konfiguriert für **Continuous Builds** sämtlicher Projekte
  - Automatischer Build nach Commit/Push auf Repository.
  - Build führt die Goals **deploy** und **site** aus.
  - Unüblich viel in einem Build, aber es geht ja relativ schnell... 😊
  - Besser wäre: Aufteilen in mehrere Steps, «fail fast»-Prinzip.
    - Höherer Konfigurationsaufwand 😞



# **Demo (Jenkins & GitLab)**

# Übersicht – Komplette Infrastruktur



# VSK Logger Projekt - Ziele

- Projekte auf dem Buildserver sollen möglichst «**grün**» sein.
  - Projekte müssen fehlerfrei gebaut werden können.
  - Jenkins gilt als «Master» für die Beurteilung!
- Ziel/Empfehlung: Alle Projekte sind **immer** fehlerfrei buildbar.
  - ➔ Höchstes Ziel.
  - Sobald ein Build «**rot**» ist, wird kein neuer Code mehr committed, sondern zuerst der Build gefixt.
  - Gemeinsames Ziel (für das Team) ist es, allfällig rote Builds wieder «**grün**» zu machen.
- Das ist ein zentraler Schritt zur «Continuous Integration» (CI).
  - Abschliessender Input in SW07.

**Fragen?**