HSLU Hochschule Luzern

Verteilte Systeme und Komponenten

Buildserver

Technologien und Funktionsweise

Roland Gisler

FH Zentralschweiz

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?

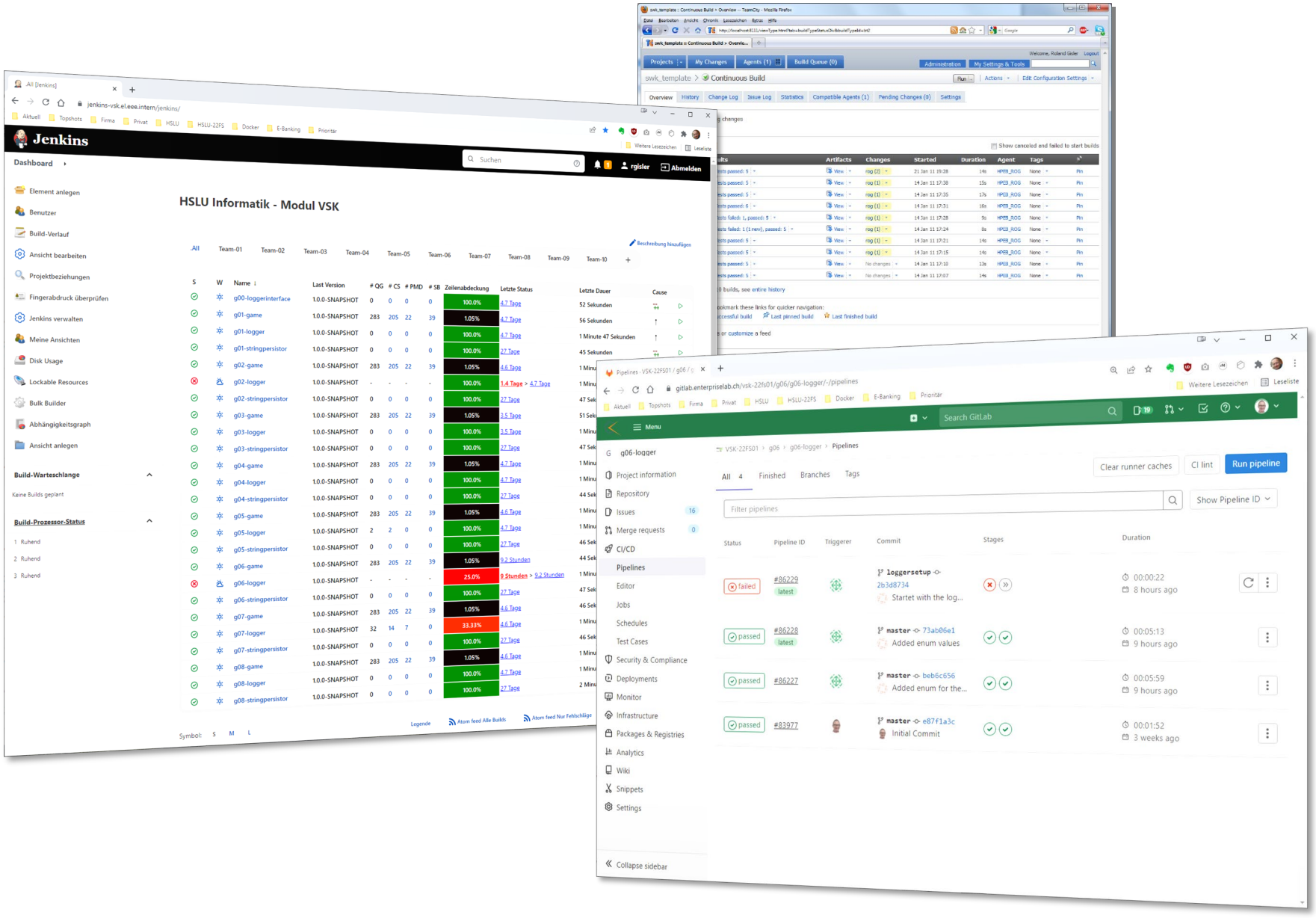
http://www.bobthebuilder.com/de/

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



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 wesentlich **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. **`.yml`**-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 eine Buildservers **setzt** andere Technologien **voraus**:
  - **Automatisierte** Builds, z.B. mit Maven oder Gradle etc.
  - Einsatz eines Versionskontrollsystemes, 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.

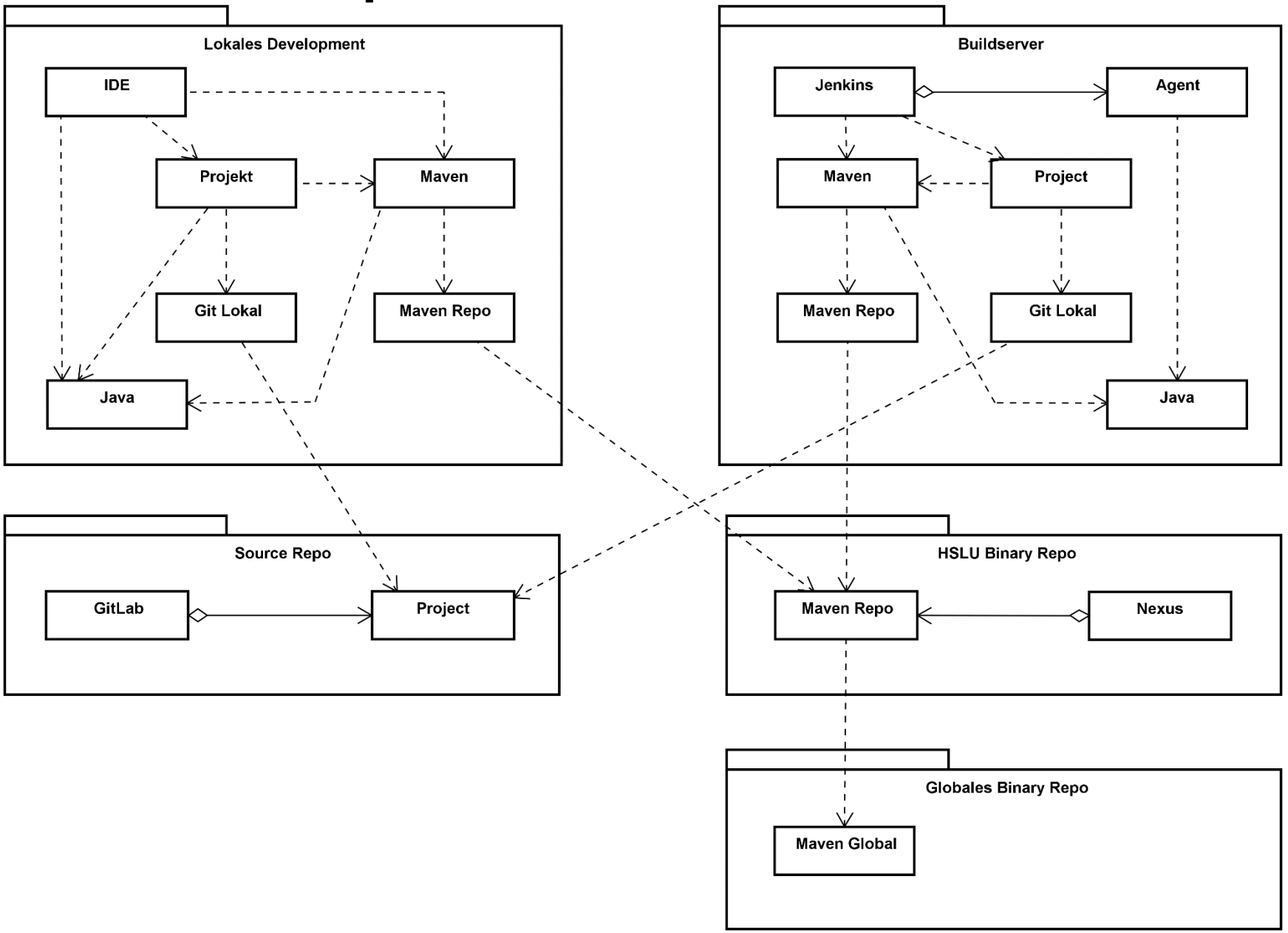
Logger-Projekt

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://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 commited, 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?