



Softwarepaketierung und Continuous Integration bei Airbus Defence and Space

DB Open Source Workshop, Frankfurt

26. November 2015



Karsten Keil
Kernel Entwickler & Consultant
B1 Systems GmbH
keil@b1-systems.de

Vorstellung B1 Systems

- gegründet 2004
- primär Linux/Open Source-Themen
- national & international tätig
- über 70 Mitarbeiter
- unabhängig von Soft- und Hardware-Herstellern
- Leistungsangebot:
 - Beratung & Consulting
 - Support
 - Entwicklung
 - Training
 - Betrieb
 - Lösungen
- dezentrale Strukturen

Schwerpunkte

- Virtualisierung (XEN, KVM & RHEV)
- Systemmanagement (Spacewalk, Red Hat Satellite, SUSE Manager)
- Konfigurationsmanagement (Puppet & Chef)
- Monitoring (Nagios & Icinga)
- IaaS Cloud (OpenStack & SUSE Cloud & RDO)
- Hochverfügbarkeit (Pacemaker)
- Shared Storage (GPFS, OCFS2, DRBD & CEPH)
- Dateiaustausch (ownCloud)
- Paketierung (Open Build Service)
- Administratoren oder Entwickler zur Unterstützung des Teams vor Ort

Ausgangssituation Airbus Defence and Space Flight Test Ground Station (FTGS)

Ausgangssituation

- Software für Flight Test Ground Station
- Software und Hardware Support für Telemetrie, Mission Monitoring, Data Processing, Analyse und Visualisierung
- geschrieben in C, C++ und FORTRAN
- ca. 100 einzelne Software-Komponenten
- 1993 Projektstart mit Unix (SGI IRIX und HP-UX); um 2002 Migration auf Linux (SuSE 7.x)
- diverse Migrationen auf verschiedene Linux Distributionen (SuSE 7.x → SLES 9.x → SLES 11.x)
- Bei Projektstart noch auf i586; Umstieg auf x86_64 ist in Vorbereitung
- Zielplattform bei Projektstart SLE 11 SP1
- diverse Third Party Libraries in Gebrauch

Alte FTGS Entwicklungsumgebung

Alte FTGS Entwicklungsumgebung

- ca. 100 Software-Projekte
- IDE: proprietär (EOL und nicht mehr unterstützt)
- VCS: CVS
- Build-Umgebung: Kontrolle über IDE und andere Projekte mit Makefiles, imake, qmake, ...
- Betriebssystem: SLE 11 SP1 i586
- Entwickler-Workstations: Fat Clients (Diskless, Distro Image read-only, Homes auf NFS)
- Continuous Integration: Keine
- Deployment: von Hand direkt in einen zentral genutzten NFS-Ordner



Motivation

Motivation

- neue Entwicklungsumgebung für die Entwickler
- Cross-Platform Development (Linux x86, Linux x86_64, Windows, ...)
- Software Maintenance über gesamte Projektlaufzeit (Datenvorhalt für mindestens 10 Jahre)
- konsistente Builds des kompletten Software Stacks
- Bauen und Bereitstellen des SW-Stacks in verschiedenen Variationen (Kundenanpassungen)



Anforderungen

Neue Entwicklungsumgebung

- Entkopplung von IDE und Build-Umgebung
- langfristige Lösung für Build-Umgebung
- Herstellerunabhängigkeit (Produkteinstellung)
- Build-Umgebung muss IDE-Nutzung und Build Automation erlauben
- dezentrale Versionsverwaltung (VCS) für Entwicklung außerhalb des Firmennetzwerks (z. B. bei Testeinsätzen)
 - fortlaufende und konsistente Versionierung
- Verfolgbarkeit

Cross-Platform Development

- Zielplattform: Linux und Windows
- Software Releases unterstützen verschiedene:
 - Linux Distributionen
 - Architekturen
 - Flavors/Subsets (für unterschiedliche Abteilungen/Kunden)

Software-Maintenance (10 Jahre)

- Archivierung von Quellen und Builds
 - von *jedem* Release
- In 10 Jahren muss es möglich sein,
 - exakt den selben Code zu bauen
 - auf die selbe Art zu bauen
 - Builds auszuführen und zu testen
 - an Ort und Stelle Fehler zu suchen und zu beheben
- trotzdem einfache Bedienung, falls sehr selten genutzt



Konzept

Software-Maintenance (10 Jahre)

- statt Entwicklung einzelner Programme: Entwicklung eines konsistenten Produkts
- Einführung eines Maintenance Workflow
- Um Softwarefehler während des Release Life Cycles zu beheben,
 - *alles* automatisieren
 - anstatt alles zu dokumentieren.
- Beibehalten der Automatisierung, damit bei Änderungen sofort auffällt, wenn Fehler auftreten

Ziele

Build Engineering

- *konsistente* Builds
- *reproduzierbare* Builds
- *debugbare* Builds
- Auditing Acceptability/Revision Control
- Traceability
- Unterstützung für Cross-Platform Development
- einfache Migration auf neue:
 - Zielplattformen
 - Third Party APIs

Build Engineering

- vom Code Push zum neuen Software Release in *Minuten!*

Build Engineering

- vom Code Push zum neuen Produkt-Release innerhalb *einer Stunde!*

Build Engineering

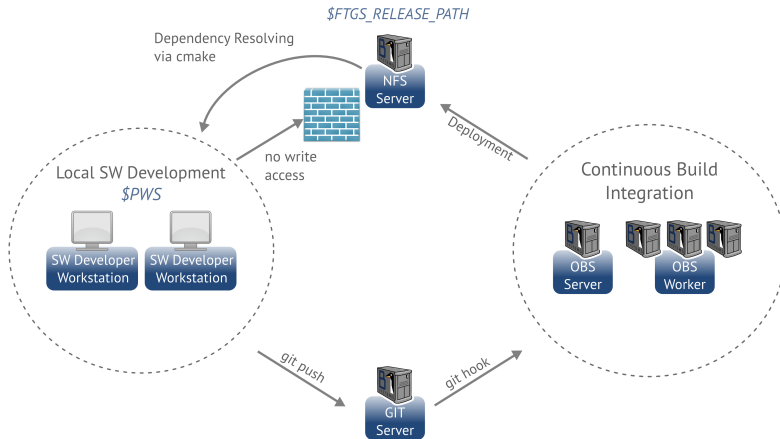
- vom Code Push zum neuen Produkt für **verschiedene Plattformen** innerhalb *einer Stunde!*

Neue FTGS Entwicklungsumgebung

Neue FTGS Entwicklungsumgebung

- ca. 140 Software-Projekte (+40)
- IDE: QtCreator
- VCS: Git
- Build-Umgebung: CMake (exklusiv)
- Betriebssystem: SLE 11 SP1, SLE 11 SP2 - i586/x86_64, SLE 11 SP3 - i568/x86_64
- Entwickler-Workstations: Fat Clients (Diskless, Distro Image read-only, Homes auf NFS)
- Continuous Integration: Git → OBS → (Jenkins/CDash/OBS-Erweiterung) → maßgefertigtes Deployment/Ausrollen
- Deployment: via CI und manuell triggerbar

Neue FTGS Entwicklungsumgebung



Entwicklungsworkflow

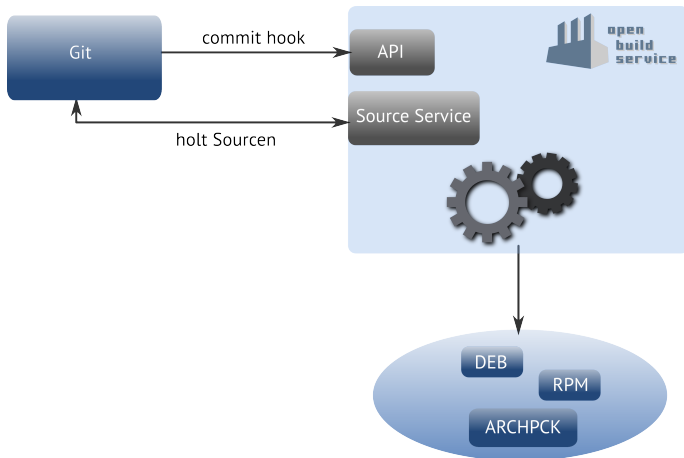
Entwicklungsworkflow

- ① Initial Git Repository vom SW Projekt klonen.
- ② Code mit IDE nach Wahl bearbeiten.
- ③ (Kleine) zusammenhängende Änderungen gemeinsam via Git committen.
- ④ Wenn ein Feature vollständig ist oder ein Fehler behoben wurde: Git push.
- ⑤ ... ab hier übernimmt das CI-System.



CI-Workflow

CI-Workflow



CI-Workflow

- 1 Git Hook löst bei einem Push einen Event im OBS aus.
- 2 OBS baut den aktuellen Source-Stand neu und erstellt ein Repository.
- 3 sofortiges automatisches Deployment bei Release Maintenance
- 4 nächtliches Deployment der Entwicklungs und Test Stages
- 5 Deployment erfolgt direkt auf NFS Share inklusive Snapshot.



Software Staging

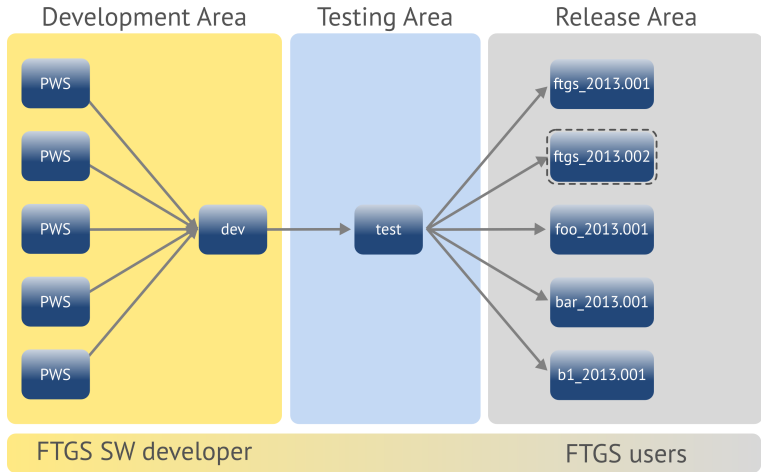
Software Staging

- 1 Entwicklung: Git Master Branch, Feature Branch möglich
- 2 Test: durch eine spezielle Git Tag Notation FTGS_v1.0.0
- 3 Release: Release Manager staged Projekt nach Bedarf aus der Test-Stufe

Maintenance von Releases

- Jede Staging-Stufe und jedes Release ist ein OBS-Projekt.
- Releases sind „abgehängte“ und konsistente Produkt-Snapshots.
- Maintenance eines Release Pakets erfolgt mittels „Wizard“
 - führt durch alle Schritte
 - Abfrage aller notwendigen Informationen
 - automatisiertes Commit, Build und Deployment
 - Dokumentation

FTGS Software Staging mit OBS





Open Build Service

Open Build Service: Intro

- „Distribution Development Platform“
- <https://www.openbuildservice.org>
- seit Januar 2006 unter der GPL verfügbar
- seit Mai 2011 als Open Build Service bekannt (vorher openSUSE Build Service)
- Nachfolger des SUSE internen Build-Systems
- Referenzinstallation: <https://build.opensuse.org>

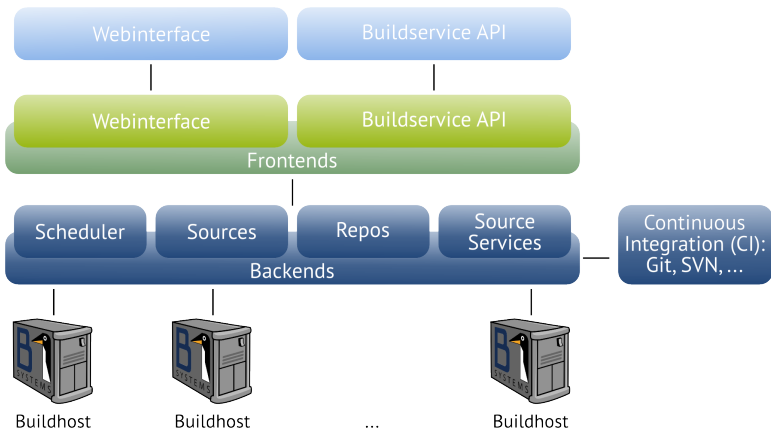
Open Build Service: Features 1/2

- kollaborative Funktionalität: User Management, Merge und Review Funktionalität von SW
- unterstützte Zielplattformen:
 - RPM basierte Distributionen: openSUSE, SLES, RHEL, Fedora, CentOS, Scientific Linux
 - DEB basierte Distributionen: Debian, Ubuntu
 - ARCH Linux
 - Windows
- unterstützt verschiedene Architekturen: x86, x86_64, PPC, S390x, ARM, ...
- Erstellung von Medien (DVD-ISO, Xen/KVM Image, Appliances, ...)
- Maintenanceprozesse (Patchrelease)

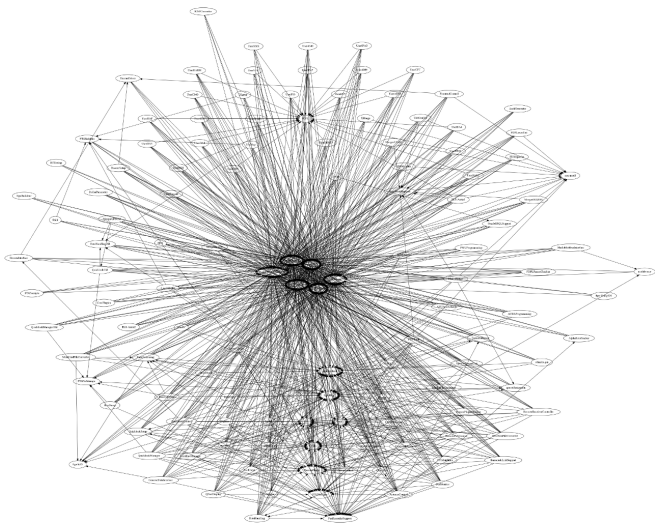
Open Build Service: Features 2/2

- erlaubt konsistente und reproduzierbare Software Builds
- integriert Versionskontrollsysteme
- Skalierbar durch Einsatz mehrerer Worker
- löst Abhängigkeiten selbstständig auf
- eingebaute Revisionsverwaltung mit „Deduplizierung“
- vollautomatisierte Abläufe vom Paketbau bis Repositoryerstellung und Signierung

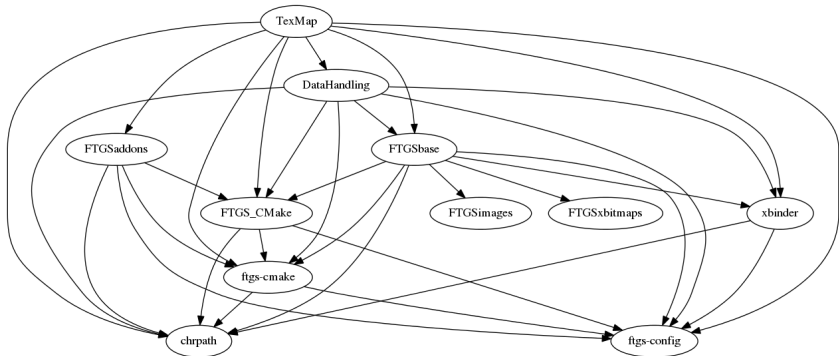
Open Build Service: Architektur



OBS: Build Dependency Handling

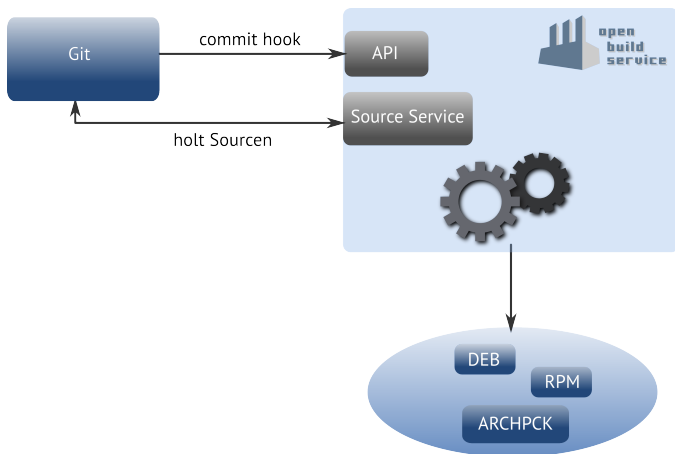


OBS: Build Dependency Handling



Integration des Open Build Service

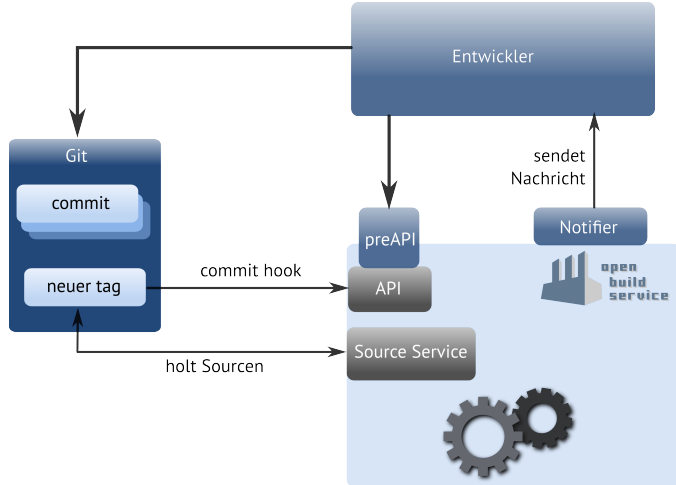
Integration des Open Build Service



OBS in FTGS Entwicklungsumgebung

- Open Build Service Einbindung transparent für Kunden
- Entwicklungsabläufe automatisiert
- Interaktion nur im Fehlerfall (Software baut nicht mehr)
 - Benachrichtigung des Hauptentwicklers der Komponente
 - Benachrichtigung des letzten GIT Committers der Komponente
 - Bereitstellung der Logdatei

FTGS Entwicklungsumgebung aus Entwicklersicht



Fazit

Fazit

DON'Ts:

- Deployment von Tarball/Binary-Blobs via rsync unter der Hand, ohne Kenntnisse des Package Management System
- Deployment durch automatisierte Fetch-Skripte
- das Rad neu erfinden ...
- Bauen von Softwarepaketen (RPM, DEB, ...) auf Entwickler-Workstation von Hand

Fazit

DOs:

- Nutze den Open Build Service
- Die „Power“ von Continuous Integration und der Extreme Programming Ära nutzen!
- Nur einen Software-Stand pflegen:
 - für verschiedene Linux Distributionen, Releases oder Service Packs
 - für verschiedene Architekturen (x86_64, i586, s390x, ia64, ppc64, pcc, ARM, ...)
 - ... in einem Aufwasch: Cross-Distribution-Architecture Packaging
- erhöhte Sicherheits-/Integritätsanforderungen? Eigene Pakete mit eigenem Schlüssel signieren!

Vielen Dank für Ihre Aufmerksamkeit!

Bei weiteren Fragen wenden Sie sich bitte an info@b1-systems.de
oder +49 (0)8457 - 931096