4. + 5. April 2014 www.linuxtage.at

Continuous Integration mit Debian-/Ubuntu-Paketen

Michael Prokop

Michael „mika“ Prokop

- Unabhängiger IT-Consultant
  - Debian/Linux
  - IT-Forensik
  - Continuous Integration/Delivery
- Debian-Entwickler
- Projektleiter von Grml.org, Grml-Forensic.org + Jenkins-Debian-Glue.org
- Grml Solutions + SynPro Solutions

Agenda

- CI/CD
- Jenkins
- Debian-Paketierung
- Jenkins-debian-glue
- EC2 Autoscaling
- kamailio-deb-jenkins
- Best Practices
- Q/A

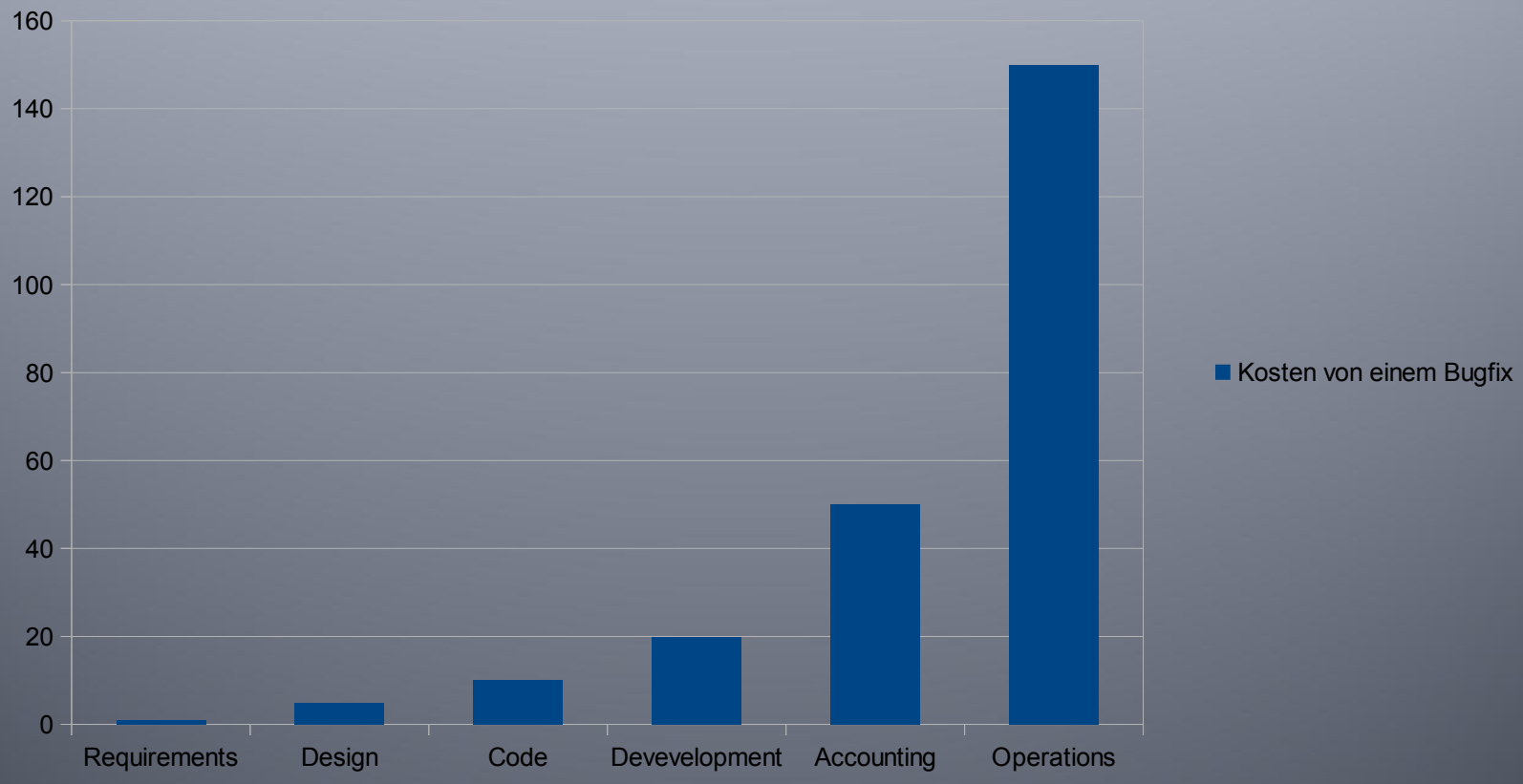
Quelle: http://xkcd.com/303/

CI/CD

Begriffsklärung

- Continuous Integration
  - Aus der SW-Entwicklung bekannt
- Continuous Deployment
  - Q/A-Kriterien erfüllt? → Deploy to production!
- Continuous Delivery
  - Business-Entscheidung!

Warum CI/CD?



Quelle: Barry Boehm's „EQUITY Keynote Address“

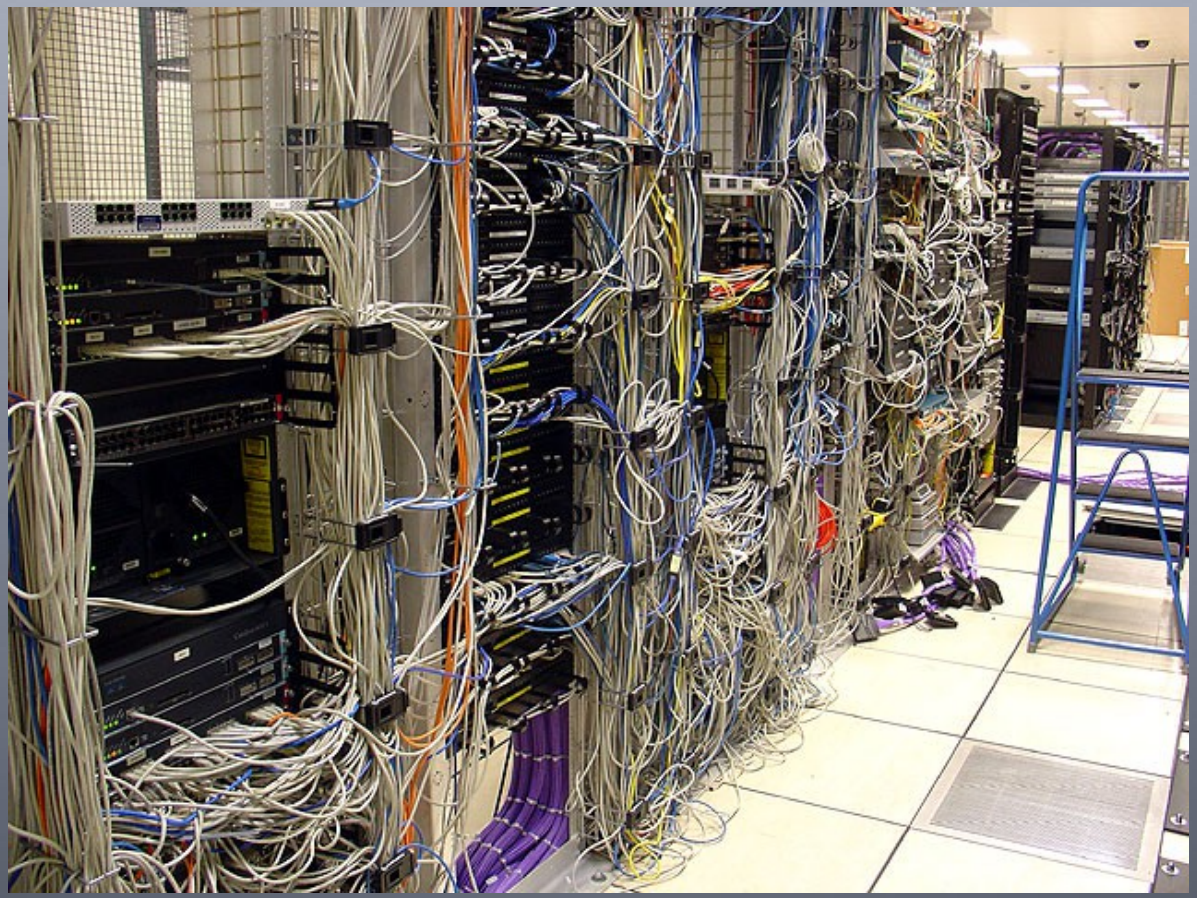
Unabhängig

Quelle: http://decarabia.soup.io/post/241926962/Image

Skalierbar

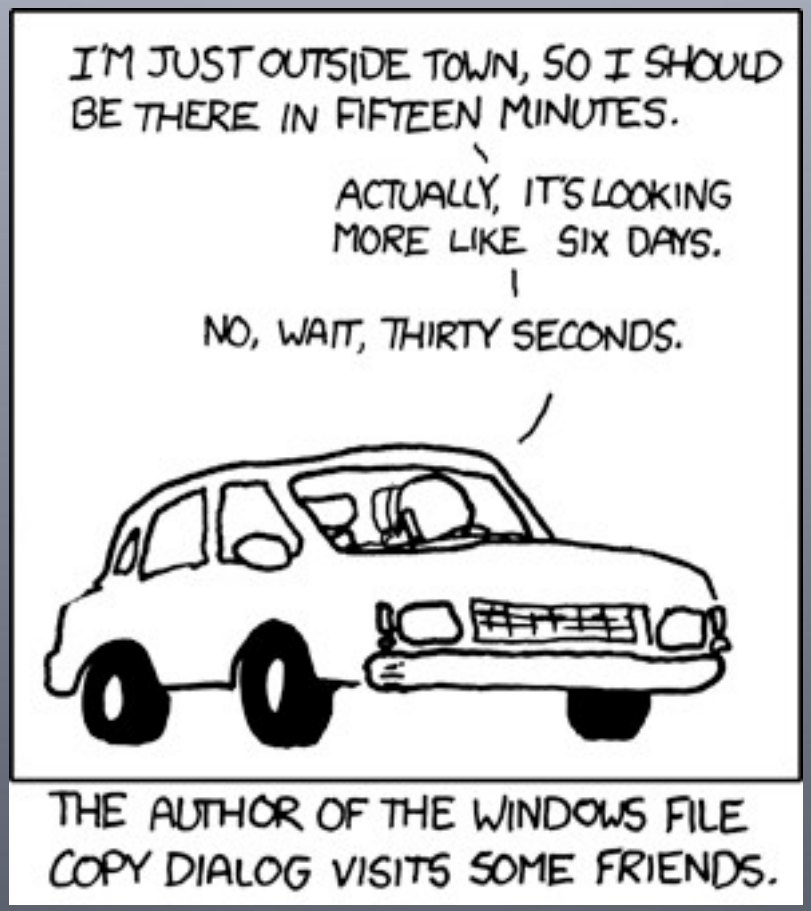
Quelle: http://orlosa.wordpress.com/2013/07/26/leute-die-zeit-haben-bitte-hier-anstellen/

Reproduzierbar



Quelle: http://www.flickr.com/photos/route79/13120127/

Berechenbar

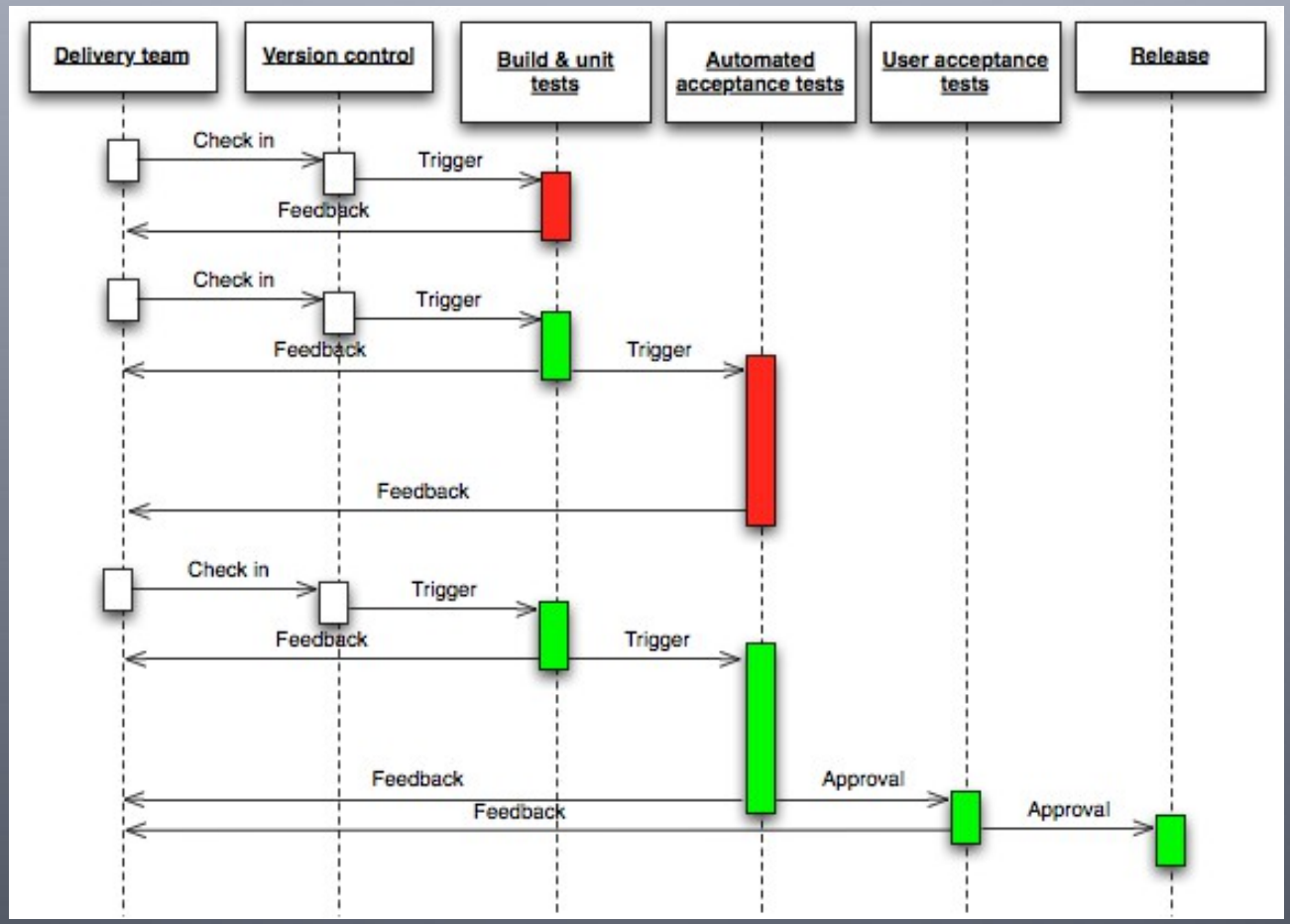


Quelle: http://xkcd.com/612/

Versionskontrolle

- Nur was unter Versionskontrolle ist zählt!
- Dokumentation nicht vergessen
- Distributed VCS ftw!

Deployment Pipeline



Quelle: http://continuousdelivery.com/2010/02/continuous-delivery/

Jenkins

Jenkins

- das „Wordpress der CI-Server“
- Open Source (MIT Lizenz)
- wöchentliche && LTS-Releases
- >78k Installationen (Stand 02/2014)
- >900 Plugins (Stand 02/2014)
- Community

FAQ - Java?!

- äh ja, RAM hilft, schnelle Disks sind auch kein Fehler
- nein, man braucht keinen Javacode anzugreifen
- nein, unterstützt nicht nur Java-Projekte

Debian-Paketierung

Debian-Paketierung – die Tools

- Klassisches (old-school) debhelper vs. CDBS vs dh(7) → dh!
- dh-make, dh-make-perl, dh-make-php, ...
- make-ruby/gem2deb
- fpm
- {cvs,svn,git,...}-buildpackage
- cowbuilder/pbuilder/sbuild/...
- reprepro/dak/freight/...

Debhelper 7

- Verfügbar seit April 2008
- Simples und kurzes debian/rules, trotzdem gut anpassbar:

```
#!/usr/bin/make -f
%:
	dh $@

override_dh_auto_build:
	$(MAKE) world
```

Debian-Paketierung - Einstiegshilfen

Einstieg/Generelles:

- https://wiki.debian.org/IntroDebianPackaging
- https://debian.org/doc/manuals/packaging-tutorial/

Git-Workflow:

- http://www.eyrie.org/~eagle/notes/debian/git.html
- http://pkg-perl.alioth.debian.org/git.html

```
% sudo apt-get install packaging-tutorial
```

jenkins-debian-glue

jenkins-debian-glue (j-d-g)

- Debian-/Ubuntu-Pakete im Eigenbau
  - Jenkins als „Frontend“
  - Auch für Nicht-Debian-Entwickler benutzbar!
- Open Source (MIT)
  - aktuell 26 Contributor
- Unterstützt Subversion + Git ootb
- Vorwiegend Shell, ein wenig Ruby/Python/Perl (je nach Einsatz) → leicht adaptierbar!

j-d-g im Praxiseinsatz

- Grml (hostet u.a. dpkg, FAI, initramfs-tools)
  - http://jenkins.grml.org/
- PostgreSQL
  - https://wiki.postgresl.org/wiki/Apt
- Icinga
  - http://icingabuild.dus.dg-i.net/
- LLVM Debian/Ubuntu
  - http://llvm.org/apt/
- Wikimedia
  - https://integration.wikimedia.org/ci/view/Ops-DebGlue/
- ... und viele weitere

Standard-j-d-g-Setup

${projekt}-source

- Debian Source-Paket erstellen
  - (Upstream-)Source (orig.tar.gz)
  - Debian-Änderungen (debian.tar.xz) [optional]
  - Control-Datei (.dsc)
- Skript generate-{git,svn}-snapshot
- Wichtig: idR nur einmal pro Paket Erstellen (Ausnahme: Multi-Distri in einem Repos)

${projekt}-binaries

- Debian Binary-Pakete
  - Beinhaltet die gebauten Dateien (Binaries, Dokumentation,...)
  - *_all.deb/*_amd64.deb/*_i386.deb
  - *.changes, *.dsc, *.tar.gz
- Skript build-and-provide-package
- Wichtig: pro Architektur/Distribution jeweils einmal Bauen (Ausnahme: „Architecture: all“)

cowbuilder

- Software die für das Bauen der Debian-Pakete verwendet wird
- Pbuilder-ähnliches Interface das auf cowdancer aufsetzt (wtf?)
  - sauberes, minimales chroot
  - schnell (hard-links, COW)
- j-d-g automatisiert auch das cowbuilder-Setup 100% → angreifen idR nicht notwendig

${projekt}-piuparts

- „.deb package installation, upgrading, and removal testing tool“
  - Ist Paket installierbar?
  - Ist Paket von letzter verfügbarer Version auf die aktuelle aktualisierbar?
  - Kann das Paket auch sauber deinstalliert werden?
- automatisiert Schritte die man manuell entweder nicht oder nur unzureichend macht

Repository-Handling?

- Teil vom *-binaries-Job
- Automatisches Erstellen des Repositories ohne manuelle Interaktion
  - reprepro
  - freight
- Auslagerung in separaten *-repos-Job einfach möglich
  - BUILD_ONLY vs PROVIDE_ONLY

Q/A im Debian-Paketbau

- Lintian
  - Common errors + Debian policy checks
  - http://lintian.debian.org/
- Piuparts
  - https://wiki.debian.org/piuparts
- Autopkgtest
  - https://packages.debian.org/autopkgtest
  - http://dep.debian.net/deps/dep8/
- perlcritics/checkbashism/...: Code-Policies
- Resultat als TAP/jUnit/...-Report in Jenkins

lintian

- Beispiele:
  - E: foobar source: version-substvar-for-external-package foobar-ha -> foobar-python-modules
  - E: foobar: maintainer-script-calls-init-script-directly
- Sehr praktisch (nicht nur für Debian-Entwickler) um Fehler in Paketen zu vermeiden

autopkgtest

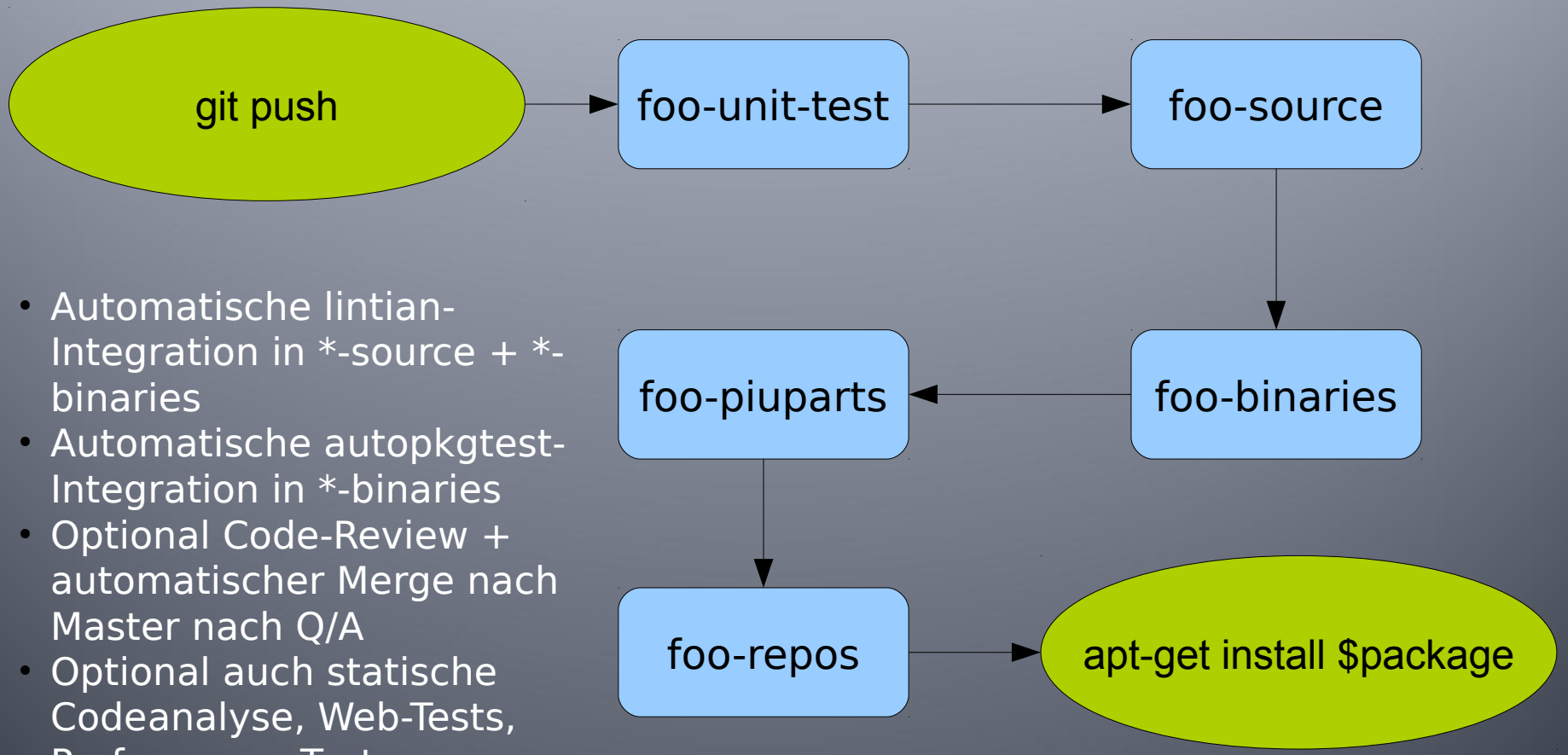
- debian/tests/control

```
Tests: run-tests
Depends: python-foobar
```

- debian/tests/run-tests

```
#!/bin/sh
python -c 'import foobar.foobar3'
```

Paket-Build-Pipeline, ein Beispiel



- Automatische lintian-Integration in *-source + *-binaries
- Automatische autopkgtest-Integration in *-binaries
- Optional Code-Review + automatischer Merge nach Master nach Q/A
- Optional auch statische Codeanalyse, Web-Tests, Performance-Tests,...

Deployment von j-d-g?

- http://jenkins-debian-glue.org/getting_started/automatic/
- in <15 Minuten (auch auf EC2) mit minimalem Aufwand testbar:

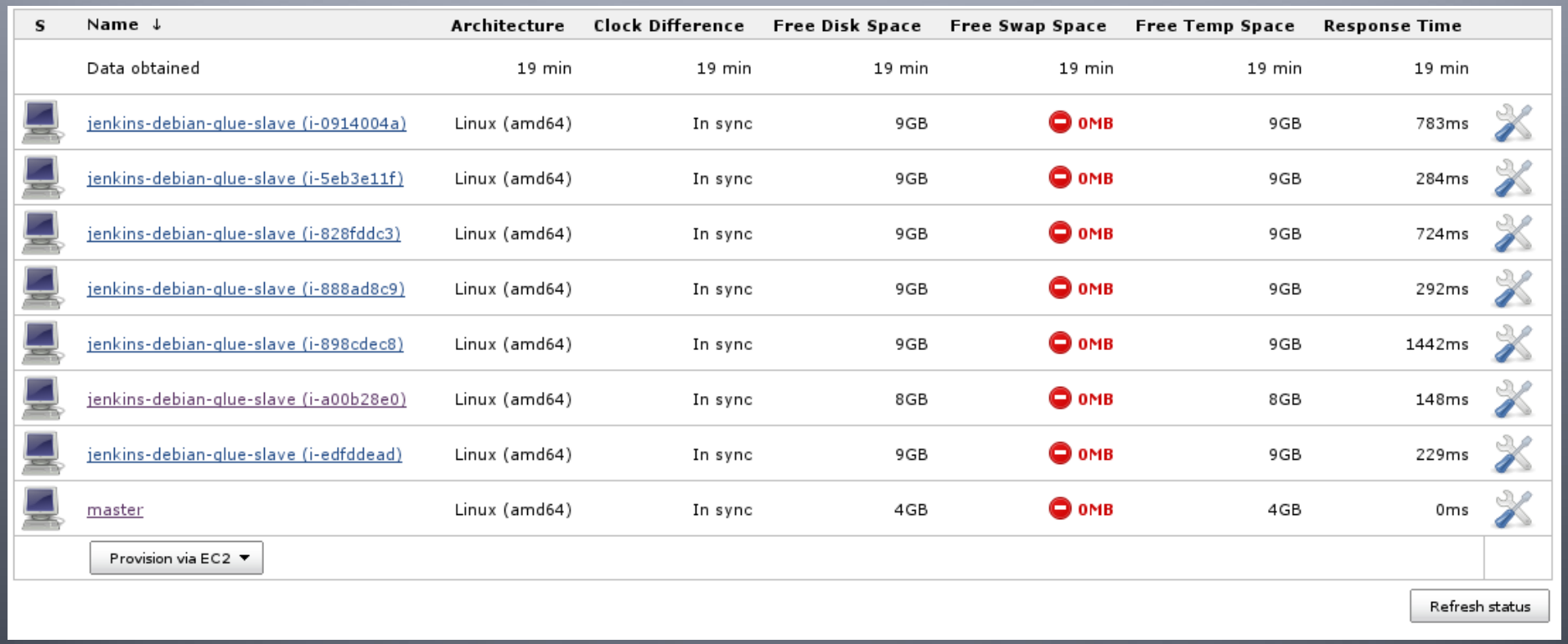
```
% wget --no-check-certificate
https://raw.github.com/mika/jenkins-
debian-glue/master/puppet/apply.sh
% sudo bash ./apply.sh $PASSWORT
```

EC2 Autoscaling

EC2 Autoscaling - die Idee

- Jenkins Master z.B. 24/7 verfügbar
- Jenkins Slaves on demand hochfahren
  - Viele oder aufwendige Pakete schnell auf starken Maschinen bauen lassen (schnelles Feedback!)
  - Starke Maschinen nicht ungenützt idlen lassen
- Disclaimer: Ist natürlich nicht für jeden Einsatzzwerk der richtige Weg

EC2 Autoscaling – Slaves on Demand

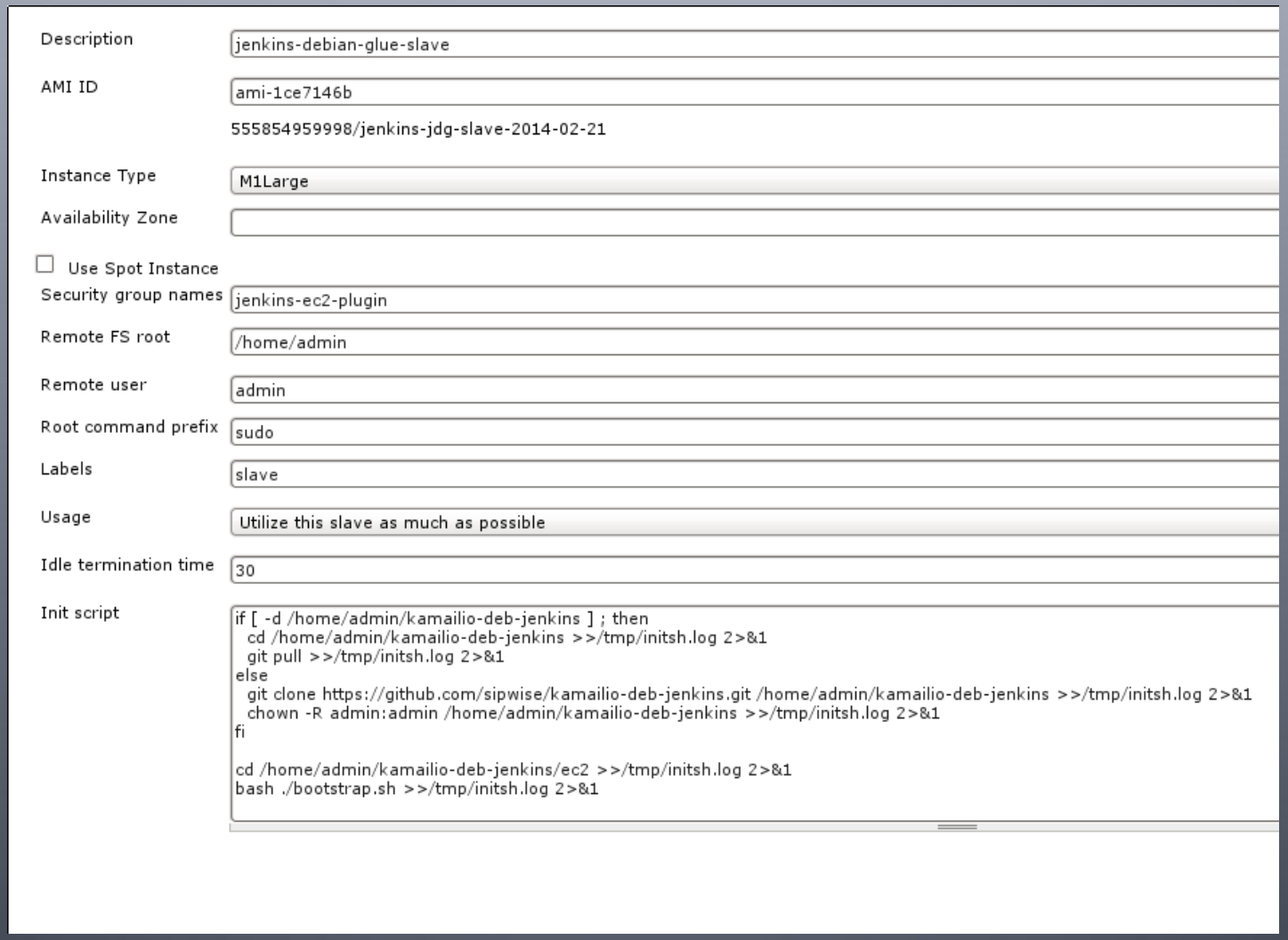


S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Data obtained	19 min	19 min	19 min	19 min	19 min	19 min
	jenkins-debian-glue-slave (i-0914004a)	Linux (amd64)	In sync	9GB	0MB	9GB	783ms
	jenkins-debian-glue-slave (i-5eb3e11f)	Linux (amd64)	In sync	9GB	0MB	9GB	284ms
	jenkins-debian-glue-slave (i-828fddc3)	Linux (amd64)	In sync	9GB	0MB	9GB	724ms
	jenkins-debian-glue-slave (i-888ad8c9)	Linux (amd64)	In sync	9GB	0MB	9GB	292ms
	jenkins-debian-glue-slave (i-898cdec8)	Linux (amd64)	In sync	9GB	0MB	9GB	1442ms
	jenkins-debian-glue-slave (i-a00b28e0)	Linux (amd64)	In sync	8GB	0MB	8GB	148ms
	jenkins-debian-glue-slave (i-edfddead)	Linux (amd64)	In sync	9GB	0MB	9GB	229ms
	master	Linux (amd64)	In sync	4GB	0MB	4GB	0ms

Provision via EC2

Refresh status

EC2 Autoscaling – die Jenkins-Konfiguration



Description: jenkins-debian-glue-slave

AMI ID: ami-1ce7146b

555854959998/jenkins-jdg-slave-2014-02-21

Instance Type: M1Large

Availability Zone:

Use Spot Instance

Security group names: jenkins-ec2-plugin

Remote FS root: /home/admin

Remote user: admin

Root command prefix: sudo

Labels: slave

Usage: Utilize this slave as much as possible

Idle termination time: 30

Init script:

```
if [ -d /home/admin/kamailio-deb-jenkins ] ; then
  cd /home/admin/kamailio-deb-jenkins >>/tmp/initsh.log 2>&1
  git pull >>/tmp/initsh.log 2>&1
else
  git clone https://github.com/sipwise/kamailio-deb-jenkins.git /home/admin/kamailio-deb-jenkins >>/tmp/initsh.log 2>&1
  chown -R admin:admin /home/admin/kamailio-deb-jenkins >>/tmp/initsh.log 2>&1
fi

cd /home/admin/kamailio-deb-jenkins/ec2 >>/tmp/initsh.log 2>&1
bash ./bootstrap.sh >>/tmp/initsh.log 2>&1
```

kamailio-deb-jenkins

kamailio-deb-jenkins – die Idee

- Kamailio = Open Source SIP Server
- Bisheriges Setup für Debian-Pakete:
  - persistente Chroots
  - manuelle Build-Schritte (dpkg-buildpackage)
  - Nicht-reproduzierbare Builds mit Gefahr für kaputte Dependencies ☹
- Neues Setup von Sipwise gesponsert:
  - Know-How aus mehrjähriger Praxiserfahrung eingebracht
  - 100% Open-Source:

https://github.com/sipwise/kamailio-deb-jenkins

kamailio-deb-jenkins – das Setup

- Deployment/Konfiguration von vollständigem CI/CD-Setup für Debian-/Ubuntu-Paketbau von Kamailio
  - Jenkins, jenkins-debian-glue, jenkins-job-builder
- Automatisch gebaute Pakete für:
  - Debian lenny/squeeze/wheezy/jessie
  - Ubuntu lucid/precise
  - → alles davon für jeweils amd64+i386
- EC2 Autoscaling
  - http://michael-prokop.at/blog/2014/03/25/building-debianubuntu-packages-on-ec2/

Best Practices

Best Practices - Misc

- Automatisieren was weh tut
- Schnelle Hardware nutzen → Entwicklerzeit ist teu(r)er!
- Homogenität ist fein
- Dashboards nutzen
- Test-Umgebung + Tests auch für die CI-/CD-Umgebung
- Wartungsfenster für CI-/CD-Umgebung schaffen
- Einheitliche Zeitzone (TZ=UTC)

Best Practices - Jenkins

- Jenkins Master als Controlinstanz
- Jenkins Slaves fürs Bauen
- „Discard old builds“ in Jenkins
- Builds triggern und nicht pollen
- Jenkins Jobs nicht manuell erstellen
  - Jenkins-job-builder
  - SCM Sync Configuration plugin

Best Practices – 3rd Party Dependencies

- Externe Abhängigkeiten beseitigen!
- Beispiele was schiefgehen kann (BTDT):
  - Github
  - PyPI
  - RubyGems
  - CPAN
  - Debian/Percona/Puppetlabs/ $YOUNAMEIT-Repository
  - ...

Best Practices - Speedup

- tmpfs
- eatmydata
- lokaler Package-Mirror

Best Practices - EC2

- NTP nutzen
- Dezidierte Security-Group(s) verwenden
- IAM (AWS Identity and Access Management) nutzen!
- Termination Protection → praktisch um Slaves zu debuggen
- Base-Images nutzen, nicht vergessen sie regelmäßig zu aktualisieren

Best Practices – Anti-Patterns 1/2

- Manuelles SSH
  - Abhilfe: fabric, mcollective,...
- Flaky Tests
  - „sleep X“
  - auf schneller HW entwickelt + auf langsamen System getestet/deployed (hallo EC2!)
- Kein Konfigurationsmanagement

Best Practices – Anti-Patterns 2/2

- Hartcodiert statt Konfigurationsmöglichkeit
- Software wird mehrfach in der Deployment Pipeline gebaut
- Keine Notifications für fehlgeschlagene Builds/Tests/...

Literatur-Tipp zu CD

- „Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation“ von Jez Humble + David Farley
- http://www.continuousdelivery.com/

Danke! Fragen?

Michael Prokop / @mikagrml
prokop (at) grml-solutions.com

Grml-Solutions.com
SynPro-Solutions.com