Power to the Project

Eine Projektumgebung als Virtual Appliance

Joachim Nelz

Softwareprojekte haben oftmals Defizite im Bereich der Projektumgebung und der Infrastruktur, die das Arbeiten im Team unterstützt. Obwohl Best Practices allgemein bekannt sind, werden sie nicht voll umgesetzt: Der initiale Aufwand, adäquate Werkzeuge auszuwählen und einzuführen, wird gescheut. Weitere Gründe liegen in fehlendem Know-how beim Thema Konfigurationsmanagement und in der mangelnden Aufmerksamkeit für die Infrastruktur. Letzteres schlägt sich auch in einer Kultur des „Jemand anders ist zuständig" nieder. Dabei ist eine gute Infrastruktur enorm wichtig, denn sie hat einen hohen Einfluss auf Produktivität, Qualität und damit auf die Fähigkeit, termin- und budgettreu zu liefern. Es ist also eine lohnende Herausforderung, die hohe Eintrittsbarriere zu überwinden.

Projekte zwischen Einmaligkeit und Schema F

Beides ist wichtig: Die Umsetzung von Best Practices und bewährten Prozessen einerseits, ohne dabei andererseits die Besonderheiten eines Projekts aus dem Auge zu verlieren. So müssen Prozesse auf das Umfeld und die Teamgröße abgestimmt sowie spezielle Werkzeuge angepasst oder zusätzlich integriert werden, zum Beispiel projektspezifische Metriken. Es geht um mehr als nur das Quellcode-Repository, nämlich um eine komplette Infrastruktur inklusive Datenbank, Applikationsserver usw.

Rund um Kollaboration

Gemeinsam ist allen Projekten: Im Mittelpunkt steht die Zusammenarbeit im Team (Kollaboration). Um diese zu unterstützen, sollten Versionierung, Issue-Tracking, Wiki, Automatisierung sowie Continuous-Integration (CI) angewandt und durch die Werkzeuge der Projektumgebung abgedeckt werden.

Es gibt kein „One size fits all" – Anpassung ist notwendig

Was sind die richtigen Werkzeuge? Hier gibt es mehr als nur eine richtige Antwort. Neben den gewünschten Features ist auch zu bedenken, wie gut ein Werkzeug in den Kontext des konkreten Projekts sowie der Organisation passt. Kurz: Die Menschen müssen mit allen Werkzeugen ergonomisch arbeiten können. Gefragt sind also Anpassbarkeit und Flexibilität, denn jedes Projekt ist einzigartig.

Diese Flexibilität kann im Kleinen durch Customizing der einzelnen Werkzeuge erreicht werden, zum Beispiel indem im Issue-Tracker ein spezifischer Workflow konfiguriert oder indem die Funktionalität des Build-Servers durch Plug-ins erweitert wird. Im Großen kann dies die Einführung zusätzlicher Komponenten bedeuten oder das Ersetzen von Werkzeugen durch andere Alternativen, z. B. ein anderes Versionsverwaltungssystem.

Die Werkzeugkette

Eine bewährte Werkzeugkette besteht aus den Komponenten Subversion, Trac und Hudson. *Subversion* [SVN] ist ein Versionsverwaltungssystem, das inzwischen durch weite Verbreitung zum Stand der Technik geworden ist: Zahlreiche Werkzeuge bringen eine Integration mit Subversion mit. Mit Tortoise SVN [Tigris] existiert auch eine sehr gute Integration in den Windows Explorer; mit Smart-SVN [SmartSVN] steht ein rein Java-basierter Client zur Verfügung.

Abb. 1: Trac Timeline View

Subversion arbeitet effizient mit Binärdateien. Damit eignet es sich auch für Szenarien abseits der Quellcode-Verwaltung: zur Versionierung von Dokumenten. Beim Vergleich (Diff) von Word-Dokumenten öffnet der Tortoise SVN Client die Vergleichsansicht von Word, was außerordentlich ergonomisch ist. Somit konnte Subversion erfolgreich zum Dokumentenmanagement in einem reinen Requirements-Engineering-Projekt mit einem großen Team (circa 100 Mitarbeiter) eingesetzt werden.

Subversion ist durch Hook-Skripte erweiterbar, dies sind Unix-Skripte, die der Subversion-Server bei Repository-Events wie Check-in aufruft. Hier können beispielsweise Richtlinien überprüft werden, z. B.: „Kein Check-in ohne Kommentar".

Trac [Trac] vereint mehrere Funktionen: Issue-Tracker, Wiki und Subversion-Web-Frontend. Besonders nützlich ist die Zeitstrahl-Darstellung (engl. Timeline, s. Abb. 1). Sie bietet eine „one to bind them all"-Sicht auf Änderungen im Repository, Wiki und Tickets (Änderungsanträge für neue Leistungsmerkmale oder Fehlerkorrekturen), mit der man schnell einen Überblick über das Projekt gewinnt.

Die intuitive und funktionale grafische Oberfläche erlaubt ein ergonomisches Arbeiten. Durch die Wiki-Engine wird eine hohe Integration erreicht: Jedes Wort in CamelCase wird automatisch zum Wiki-Link, egal ob es sich innerhalb eines Wiki-Dokuments, eines Check-in-Kommentars oder eines Tickets befindet. Spezielle Trac-Links verweisen auf Tickets und Changesets: #42 referenziert Ticket 42 und [4711] das Changeset bzw. die Subversion-Revisionsnummer 4711.

Der Workflow des Ticketing-Systems kann (in den aktuellen Trac-Versionen) selbst konfiguriert werden. Eigene Reports über Tickets werden komfortabel über die Web-GUI erstellt. Hierbei können Tickets zu Meilensteinen und Versionen zugeordnet werden, was deren Einordnung in die Projektplanung ermöglicht.

Durch Plug-ins ist Trac auf vielfältige Weise erweiterbar. TracHacks [TracHacks] listet zahlreiche Plug-ins zur Tool-Integration, SCMs u.v.m. Das Open-Source-Tool Redmine stellt eine Alternative dar [Redmine].

Hudson [HudsonCI] ist ein verbreiteter CI-Server. CI ist eine Best Practice der agilen Softwareentwicklung: Bei jedem Check-in findet automatisch ein kompletter Build und Unit-Test-Lauf statt, der unmittelbares Feedback gibt. Hudson unterstützt auch große Projekte: Komplexe Builds können in separate, von einander abhängige Jobs zerlegt werden. Weiterhin kann Hudson eine verteilte „Build Farm" vieler Build-Server steuern.

Darüber hinaus hilft Hudson auch bei der Automatisierung immer wiederkehrender Aufgaben im Projekt, indem es als Ablaufumgebung für beliebige Skripte genutzt wird. Diese können zeitgesteuert („scheduled"), ausgelöst durch Repository-Check-ins und andere Jobs („triggered") oder manuell gestartet werden. Im letztgenannten Fall kann man Jobs mit Parametern über die Web-GUI versehen. Ein Beispiel für zeitgesteuerte Jobs stellen Health Checks (periodisches Pingen des Zielsystems) dar. Dies könnte man zwar ebenso über einen klassischen Unix Cron Job erledigen, jedoch gewinnt man durch Hudson eine Web-GUI, die sichtbar macht, ob die letzten Läufe erfolgreich waren. Vor allem aber lassen sich die Skripte des Projekts unter Versionskontrolle stellen – analog zum Quellcode mit dem Repository als Master, aus dem vor jedem Lauf der aktuelle Stand geladen wird.

Parametrisierbare Jobs sind nützliche Helfer im Projekt: Der Autor verwendet einen Job namens Datenbank-O-Mat zum Neuaufsetzen und zur Initialbefüllung der Oracle-Datenbank, wobei der SID (System Identifier) der Zieldatenbank als Parameter zu übergeben ist. So steht jedem im Team bei Bedarf innerhalb weniger Minuten eine frische Datenbank mit Stammdaten bereit, was die Produktivität entscheidend nach vorne bringt. Das Wissen um das Setup ist externalisiert als Skript unter Versionskontrolle und für jeden gleichermaßen verfügbar und ausführbar.

Hudsons vielfältige Einsatzmöglichkeiten von CI bis Automatisierungsplattform machen seinen Wert fürs Projekt aus, zudem ist Hudson wegen seiner intuitiven, klaren GUI beliebt. Erweiterbarkeit ist durch zahlreiche Plug-ins für jede denkbare (und undenkbare) Anwendung gegeben.

Bereitstellung der Projektumgebung

Der übliche Weg, eine Projektumgebung bereitzustellen, ist mittels einer zentralisierten Infrastruktur. Diese wird von mehreren Teams gemeinsam benutzt und erfordert ein Administrationsteam zur Wartung, da es sich um eine kritische Infrastruktur handelt.

Dieses Deployment bringt es mit sich, dass die Erweiterbarkeit der Tools nicht voll ausgenutzt werden kann. Erfahrungsgemäß befindet sich mindestens ein Anwenderteam in einer kritischen Projektphase – und ist deshalb an einer möglichst stabilen Umgebung interessiert, ebenso wie das Administrationsteam, dessen Ziel die Betriebssicherheit ist, also „keine Experimente". Als Resultat ergibt sich ein „Lock-in" auf mehreren Ebenen:

- Werkzeuge, Erweiterungen und Versionen sind festgeschrieben. Ein Update auf eine neuere Version betrifft viele Projekte/Anwender und wird dadurch selbst zum schwergewichtigen Projekt.
- Die Werkzeugkette wird von einem Administrationsteam verwaltet, nicht vom Projekt selbst. Die Interessen beider Seiten sind jedoch nicht deckungsgleich.
- Die Prozesse sind starr und nehmen keine Rücksicht auf die Gegebenheiten im Projekt.
- Der Scope, der durch die Werkzeugkette abgedeckt wird, ist fixiert und meist geringer als die Bedürfnisse des Projekts. Oft wird nur die Versionsverwaltung bereitgestellt – das Projekt benötigt aber weit mehr als das.
- Die Lokation des Projekts ist festgelegt, insofern die Umgebung nicht „transportiert" werden kann – z. B. in einem Kundenprojekt, wo die Infrastruktur nicht erreichbar ist.

Positiv formuliert sollte jedes Team seine Umgebung in einer abgeschotteten „Sandbox" haben – ohne Seiteneffekte auf benachbarte Projekte, jedoch ohne den Vorteil einer standardisierten Umgebung aufzugeben, die aus dem Stand heraus lauffähig ist.

Genau dieses ist durch Virtualisierung möglich: Statt die Projektumgebung zentral zu hosten, bekommt jedes Team seine eigene dedizierte Umgebung. Diese besteht aus einer Virtual Appliance, d. h. einem Image einer Virtual Machine (VM), auf dem die genannten Werkzeuge installiert und vorkonfiguriert sind. Unbeeinflusst von anderen, kann jedes Projekt seine Umgebung nach seinen Anforderungen erweitern. Der Lock-in wird damit vermieden. Das VM Image steht als fertige Lösung bereit und kann aus dem Stand heraus eingesetzt werden.

Der Autor hat sich aus folgenden Gründen für VMware [VMWareHome] als Virtualisierungsplattform entschieden:

- VMware ist weit verbreitet, gerade in Rechenzentren.
- VMware ist sowohl als Desktop als auch als leistungsfähige Server-Virtualisierung einsetzbar.
- VMware Server und Player sind lizenzkostenfrei.

Als Betriebssystem der Virtual Appliance dient ein Ubuntu Linux als Server-Installation, die für VMs optimiert ist, das sogenannte „Just enough OS", kurz: JeOS [UbuntuHome]. Wie der Name verspricht, handelt es sich um ein sehr schlankes Betriebssystem mit minimiertem Ressourcenbedarf. Die Installation der oben genannten Werkzeuge ist unkompliziert, da es sich um ein sauberes System handelt und keinerlei Abhängigkeiten zu anderer Software zu beachten sind. Weitere Hinweise zum Bauen von Virtual Appliances finden sich unter [VMWare Home,VMWareServer,VMWareVA].

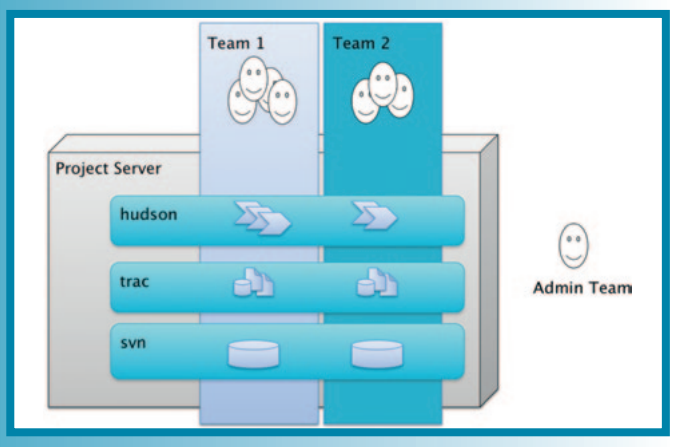

Abb. 2: Alle Projekte auf einem zentralen Server

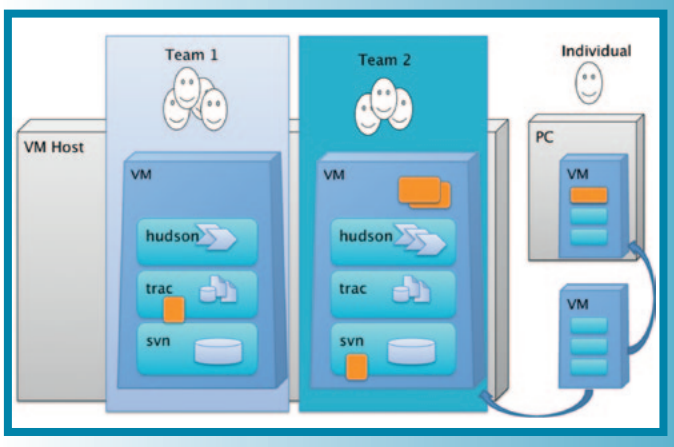

Abb. 3: Jedem Projekt seine eigene VM

Die Hardware-Voraussetzungen dieser Appliance sind überraschend niedrig – die Virtual Appliance läuft problemlos auch auf dem Notebook, mit 256 MB RAM. Somit kann die Umgebung auch offline für Trainings verwendet werden. In einem VM Host können mehrere VMs gleichzeitig betrieben werden. Bei Bedarf kann eine VM auf einen dedizierten VM Host migriert werden. Hierzu muss lediglich das Basis-Verzeichnis dieser VM kopiert werden. Somit kann die komplette Umgebung leicht transportiert werden. Virtual Appliances stellen Deployment Container dar, für die es einen Markplatz gibt [VMWareVA].

Worauf es wirklich ankommt: den Prozess, nicht das Werkzeug

Dieser dezentrale Ansatz erscheint zunächst gewöhnungsbedürftig. Wird damit nicht dem „Wildwuchs" in Projekten Vorschub geleistet? Schließlich ist doch die Durchsetzung von Standards ein schlagendes Argument, das für das zentralisierte Hosting ins Feld geführt wird, mit dem unternehmensweiten Repository als „Single Point of Truth".

Best Practices werden jedoch nicht etabliert, indem man ein Tool vorschreibt. Es besteht ein Unterschied zwischen „Wir arbeiten mit Versionierung" und „Wir benutzen SVN v1.6". Im ersten Fall steht der Prozess im Vordergrund, im zweiten nur ein Werkzeug. Ersteres ist die Essenz, auf die es ankommt, während das konkrete Tool nicht so wichtig ist. Es geht darum, das Tool *richtig* anzuwenden, nicht nur, es überhaupt anzuwenden (und etwa Dateien mit angehängtem Datum einzuchecken). Standards bringen die Qualität und Effektivität voran; der Fokus sollte hier aber auf den Prozessen liegen. Die Durchsetzung solcher Standards gelingt durch Training, Peer-Reviews sowie Projekt-Audits und hat die gelebte Kultur im Projekt im Blick.

Virtualisierung als Innovationstreiber

„*Virtualization can be the catalyst to drive* many fundamental important changes *in architectures,* processes*, and* cultures." [Bitt08]

Durch den dezentralen Ansatz ermöglicht Virtualisierung schlankere Prozesse, nämlich Self Service des Projekt-Teams statt zentraler Administration. Dies führt zu einer besseren, agileren Projektkultur, die darauf setzt, dass Mitarbeiter Verantwortung übernehmen. Statt „Jemand anders ist dafür zuständig" heißt es dann „Wir sind selbst verantwortlich". Die Projekt-Governance fordert die Umsetzung von Best Practices ein („Arbeitet Ihr mit Issue-Tracking?"), statt die Existenz von Werkzeugen abzuhaken („JIRA? – Ok.").

Die Tools in einer Projekt-VM müssen weniger komplex sein als solche, die im ganzen Unternehmen zentral benutzt werden. Viele „Enterprise" Features, wie Mandantenfähigkeit, integrierte Security u. a., werden nicht benötigt. Im Vordergrund stehen Funktionalität und Ergonomie. Die vorgeschlagenen Werkzeuge decken diese adäquat ab.

Fazit

Mit einer Virtual Appliance lassen sich bewährte Werkzeuge zu einer Off-the-Shelf-Lösung kombinieren, um die Infrastruktur für Projektteams bereitzustellen. Diese steht fertig vorkonfiguriert, ohne Setup- bzw. Installationsaufwand zur Verfügung und ist in beliebigen Umgebungen direkt nutzbar. Die Einstiegsbarriere ist damit reduziert: Durch das Anwenden von Best Practices können aus dem Stand heraus Produktivität, Qualität und Transparenz der Projekte gesteigert werden.

Da jede Projektinstanz in einer separaten virtuellen Maschine läuft, werden projektspezifische Anpassungen und Erweiterungen ermöglicht und ein Lock-in (auf bestimmte Produkte, Versionen) vermieden – im Gegensatz zum zentralisierten Hosting.

Eine mögliche Lösung sieht wie folgt aus: Basierend auf VMware, mit Ubuntu Linux als Betriebssystem, laufen die Werkzeuge Subversion als Versionsverwaltungssystem, Trac als Issue-Tracking, Wiki und Subversion Web Frontend sowie Hudson als Automatisierungs- und CI-Plattform. Sämtliche Komponenten sind lizenzkostenfrei.

Dank Virtualisierung können Projekte mehr Kontrolle über ihre Infrastruktur gewinnen. Dies erfordert ein gewisses Umdenken hin zum Self-Service-Prinzip und zu mehr Verantwortung im Sinn einer agileren Kultur.

Literatur und Links

[Bitt08] Th. Bittman, Gartner's VP Distinguished Analyst, in: 27th annual Gartner Data Center Conference, 5. Dezember 2008, Las Vegas

[HaBa08] S. Hansen, N. Barcet, How to Develop Virtual Appliances Using Ubuntu JeOS, in: Linux Magazine, 18.1.2008, `http://www.linux-mag.com/id/4829`

[HudsonCI] Hudson, Extensible Continuous Integration Server, `http://hudson-ci.org/`

[Link06] G.-J. Linker, Defragment and shrink vmdk files: VMWare virtual disk files, Linker IT Software, 9.4.2006, `http://www.oraxcel.com/cgi-bin/yabb2/YaBB.pl?num=1144564156`

[Redmine] `http://www.redmine.org/`

[SmartSVN] Subversion Client SmartSVN, `http://www.syntevo.com/smartsvn/index.html`

[SVN] Apache Subversion, `http://subversion.apache.org/` `http://subversion.tigris.org/`

[Tigris] Subversion Client Tortoise SVN, `http://tortoisesvn.tigris.org/`

[Trac] `http://trac.edgewall.org/`

[TracHacks] `http://trac-hacks.org/`

[UbuntuHome] Ubuntu Homepage, `http://www.ubuntu.com/`

[UbuntuUG] Ubuntu Usergroup, `http://ubuntuusers.de/`

[VMWareHome] VMware Homepage, `http://www.vmware.com/de/`

[VMWareServer] VMware Server, `http://www.vmware.com/products/server/`

[VMWareVA] VMware Virtual Appliances Marketplace, `http://www.vmware.com/appliances/`

Dipl.-Inform. **Joachim Nelz** ist Senior Consultant bei Logica, einem internationalen Beratungs- und IT-Dienstleitungsunternehmen. Seine Themenschwerpunkte sind Softwarearchitektur (Java JEE, Spring, Groovy), Entwicklungsprozesse, Business Intelligence und Projektleitung. In den vergangenen zwölf Jahren war er als Softwarearchitekt in diversen Branchen, wie Medien/TV, Gesundheit, Banking und Consulting, tätig.
E-Mail: joachim.nelz@logica.com