Bob the Builder: Build/Deploy von ADF Enterprise Anwendungen

Torsten Kleiber
IKB Deutsche Industriebank AG
Düsseldorf

Schlüsselworte
ADF, Build, Deployment, Automatisierung, Continous Integration, Erfahrungsbericht.

Einleitung

Das Team Kreditplattform der IKB entwickelt seit etwa 4 Jahren mit ADF. Wir deployen gegen den WebLogic Server.

Gestartet mit einfachen Anwendungen und wenigen Entwicklern wird mittlerweile das Ziel verfolgt, die Eigenentwicklung für das Kreditgeschäft abzulösen. Diese basiert aktuell auf Oracle Forms, Reports und PL/SQL. Alle Neuentwicklungen der Kreditplattform sollen ebenfalls integriert in der ADF Umgebung realisiert werden.

Mittlerweile arbeiten mittlerweile 15 Entwickler parallel an Projekten und Migrationen. Um dies zu gewährleisten wurden auch eine Menge Architekturentscheidungen neu gefällt und durch entsprechendes Refactoring umgesetzt. Wesentlich war die Umstellung auf die Pillar-Architektur, die in einem gesonderten IKB-Vortrag von Timo Veeders „Die Siegessäulen: Der Weg zur erfolgreichen ADF-Architektur“ behandelt wird.

Leider befinden wir uns immer noch auf der Version 11.1.1.5, da die Migration auf 12c wegen eines seit einem Jahr offenen Prio 1 Service Request immer noch nicht erfolgreich umgesetzt werden konnte. Dieses Thema wurde bereits im Vorjahr von mir im Vortrag „Das dreckige Dutzend – ADF Migration nach 12c in der IKB“ vorgestellt.

Bob‘s Geburt

Bereits mit dem ersten Lern-Projekt vor 4 Jahren kam das komplexe Thema Build- und Deployment auf. Da bisher kein Know-How vorhanden war und die meisten Entwickler eigentlich nur für den Fachbereich entwickeln wollen stellte sich schnell die Frage, ob man das automatisieren kann.

Aber wer sollte das machen – wir brauchten also „Bob“.

Bob kümmert sich seitdem um alle Themen rund um Build und Deployment.

Im Vortrag werden seine bisherigen Entscheidungen und Vorgehensweisen besprochen, aber auch offene Ideen und Punkte angerissen.

Build-Tools

Um ADF Sourcen automatisiert zu öffnen, zu bauen und zu paketieren braucht man ein Buildtool. Oracle liefert hier in der JDeveloper Installation das Tool ojdeploy und ab 12c dessen Erweiterung ojserver mit.

Ojdeploy dient entgegen seiner Namensgebung eigentlich nur dem Build und dem Packaging meiner Artefakte. Im Prinzip handelt es sich dabei um eine Kommandozeilen-API für einen headless JDeveloper. Knackpunkt beim ojdeploy ist die Start- und Shutdown-Zeit für diesen headless JDeveloper. Um diese zu minimieren, gibt es seit 12c den ojserver, der im Prinzip eine headless JDeveloper Instanz bereitgestellt. Leider ist aber trotzdem nur eine sequentielle Verarbeitung möglich. Sollen mehrere Build-/Deployment-Prozesse parallel arbeiten können, brauch man mehrere ojserver-Instanzen.

Deployment-Tools

Um die paketierten Bibliotheken und Anwendungen automatisiert auf Standalone-Server zu verteilen benötigt man ein Deployment-Tool. Oracle liefert für den WebLogic Server hier das Tool weblogic.Deployer mit.

Build-Automatisierungs-Tools

Um die Build- und Deployment-Tools automatisierbar immer wieder in einer definierten Reihenfolge aufzurufen, benötigt man ein Build-Automatisierungs-Tool. Hier kommen z.B. Shell-Skripting, Apache Ant oder Maven ins Spiel. Die Entscheidung ist hier stark davon beeinflusst, welche JDeveloper Version, was unterstützt.

Branching

Die IKB stellt keine Software für den Endkunden her, es gibt verschiedene Anforderungen an die Releaseplanung, die viele Softwarehersteller nicht haben:

Wir wissen nicht, wann eine Änderung vom Fachbereich abgenommen wird, da diese parallel dem Tagesgeschäft nachgehen.

Wir müssen Projekte realisieren, deren Einsatztermin fix ist und die z.B. wegen gesetzlichen Vorgaben auch nicht vorher eingesetzt werden dürfen.

Trotzdem wollen wir agil in regelmäßigen Zeitabständen so viel Artefakte wie möglich releasen.

Um dies zu erreichen, haben wir einen speziellen Branchprozess definiert.

Continuous Integration Server

Um die Build-/Deployment-Prozesse automatisch oder auch nur nachvollziehbar manuell zu starten ist es sinnvoll einen Continuous Integration Server einzusetzen. Wir haben uns für Jenkins entschieden.

Wir nutzen den Server um bei Commit auf unseren zentralen Branches automatisch zu bauen und anschließend ebenfalls automatische Unit-Tests und statische Code-Analysen zu starten.

Manuell starten wir das Deployment und anschließende automatische GUI-Test auf verschiedenen Betriebssystem / Browser Kombinationen. Das machen wir deshalb manuell, um dem jeweiligen Tester-Kreis die Umgebung nicht wegzuziehen und nachvollziehbar eine bestimmte Anzahl von User Stories testen zu können.

Konventionen über Konfigurationen

Um schnell neue Programmstrukturen anlegen zu können, die automatisch gebaut werden, braucht man Konventionen für seine Programmierung. Wir wollen in der Entwicklung so wenig wie möglich komplizierte Dokumentationen lesen.

Das beginnt bei Namens-Konventionen für Applikationen und Projekte. Wir kennzeichnen so, welche Applikationen eine Pillaranwendung, eine Rahmenanwendung oder eine Bibliothek darstellen oder nur eine unterstützende Funktion haben, wie Build, Test oder Artefaktablage.

Ein weitere Konvention ist die Benennung und Definition von Deploymentprofilen und -plänen.

Weiterhin sind einige JDeveloper Defaults in der Entwicklung von Enterprise Anwendungen kontraproduktiv und müssen umgestellt werden.

So wird bei der Anlage einer neuen ADF Anwendung automatisch eine Abhängigkeit im ViewController Projekt zum Build Output des Model-Projekts angelegt, was bei der Verwendung von ADF-Libraries zur Erschwerung der Fehler-Analyse führt, da die Klassen dann mehrfach in verschiedenen Versionen im Classpath liegen können.

Ein anderes Beispiel ist, das bei der Anlage einer neuen Bibliotheksabhängigkeit, sei es explizit durch den User oder implizit durch Wizards diese immer im Deployment exportiert, also mit paketiert wird. Einerseits sind ein Großteil dieser Bibliotheken bereits im Container vorhanden z.B. durch die ADF Runtime. Andererseits wird dadurch das Artefakt größer und der Build und das Deployment wird langsamer.

Code-Checks

Wenn die Konventionen nicht eingehalten werden, kommt es zu fehlerhaften oder fehlenden Build oder Deployments. Um hier nicht lange nach der Ursache suchen zu müssen oder in der Fehleranalyse von Bob abhängig zu sein, versuchen wir, für die Konventionen Regeln zu definieren, die bei einer statischen Code-Analyse vor dem Build bereits zu einem Abbruch führen. Hierfür nutzen wir vorrangig PMD XPath Regeln, da diese mit geringem Aufwand erlauben, XML Dateien zu prüfen.

MDS

MDS ermöglicht die Anpassung der Anwendungen für den Endanwender und/oder für verschiedene Kunden/Abteilungen/Endanwender durch den Entwickler. Will man MDS nutzen, kann man derzeit die automatische MDS-Konfiguration nur über WLST realisieren.

Build/Deploy Geschwindigkeit

Umso größer eine Enterprise-Anwendung wird, umso länger dauern Build und Deploy. Meist ist aber ein schnelles Feedback über Fehler gewünscht. Anpassungen an der Anwendung sollen schnell auf die Umgebung gebracht werden, um den Test zu ermöglichen oder dem Endanwender bereitzustellen.

Dazu sind in der Zukunft noch viele Einzelmaßnahmen nötig:

Wiederverwenden von Artefakten aus dem Build

Dieses Kernprinzip aus dem Continuous Delevery ist bei uns aufgrund der besonderen Anforderungen aus dem Branching nur eingeschränkt anwendbar.

So sollen zumindest die Artefakte pro Branch vom Deployment wiederverwendet werden, momentan bauen wir beim Deployment nochmals neu.

Eleminieren von Abhängigkeiten

Dieser Schritt ist oft kaum ohne größeres Refactoring möglich, der Druck muss hier schon sehr groß sein um sich gegen fachliche Neu- oder Migrationsanforderungen durchzusetzen.

Eleminieren von Ringschlüssen in Abhängigkeiten

Das ist die elementare Voraussetzung für nachfolgende Maßnahmen.

Minimieren Build/Deployment

Ziel ist es hier, nur die Sourcen zu bauen, zu verteilen und zu testen, die sich geändert haben oder von der Änderung direkt abhängig sind.

Dazu ist es essentiell die Abhängigkeiten automatisch ermitteln zu können und daraus einen direkten azyklischen Build-Graphen zu ermitteln. Dieser dient dann als Reihenfolge für den Build- und Deployment-Prozess.

Verfügbarkeiten maximieren

Will man die Anwendung möglichst ständig verfügbar zu machen, kommen Konzepte wie Shared Libraries, Hot Deployment und Clustering zum Einsatz.

Entwicklung

Will man auch in der Entwicklung Build und Deploy beschleunigen bietet sich z.B. ein Tool wie JRebel an. Hot Deployment von Oracle direkt gibt es zum großen Teil erst in 12c, und dieses deckt nicht alle Aspekte ab. Schwierig ist das ganze dort speziell bei der Verwendung von ADF Libraries.

Fazit

Der Vortrag deckt verschiedenste Aspekte ab, ohne einen Anspruch auf Vollständigkeit zu haben. Viele der beschriebenen Verfahren sind speziell aus den Gegebenheiten der IKB hergeleitet, sind aber in Teilen so oder anders bestimmt wieder verwendbar.

Kontaktadresse:
Torsten Kleiber
IKB Deutsche Industriebank AG
Wilhelm-Bötzkes-Straße 1
D-40474 Düsseldorf

Telefon:	+49 (211) 8221-4121
Fax:	+49 (211) 8221-2121
E-Mail	torsten.kleiber@ikb.de
Internet:	www.ikb.de