Operation am offenen Herzen

Dirk Ehms
GameDuell GmbH
Berlin

Schlüsselworte
Migration, High Availability, Zero Downtime, Glassfish, JEE7, Continuous Delivery, Maven

Einleitung
Dieser Praxisbericht basiert auf einem Projekt bei GameDuell, einem der größten deutschen Anbieter für Online-Spiele. Er beschreibt die technischen Maßnahmen und Anpassungen des Build- und Release-Prozesses, um die Umstellung der hochverfügbaren Live-Plattform mit mehr als 100 Application-Servern von JEE-6 auf JEE-7 und die Migration von 300 Maven-Modulen erfolgreich abzuschließen.

Das Besondere an diesem Migrationsprojekt waren vor allem die Randbedingungen, unter denen es durchgeführt werden musste. Zum einen sollte die Weiterentwicklung des bestehenden Systems von der Migration weitgehend unbeeinflusst bleiben. Dies war insbesondere deshalb eine Herausforderung, weil 70 Softwareentwickler parallel zum Projekt in der Lage sein mussten, kontinuierlich Erweiterungen am laufenden System durchzuführen. Das bedeutete, dass alle Änderungen des Migrationsprojekts am Quelltext über die gesamte Projektlaufzeit vollständig abwärtskompatibel implementiert werden mussten, um den Betrieb der Live-Plattform nicht zu beeinträchtigen.
Weiterhin musste die Umstellung auf den neuen Application-Server ohne Downtimes des hochverfügbaren Live-Systems durchgeführt werden.

Maven Dependency Management und Profile

Der Quelltext des Systems ist in über 300 Maven-Modulen organisiert, die wiederum in 10 unterschiedlichen Deployment-Artefakten gebündelt sind. Dabei handelt es sich sowohl um Enterprise-Archive als auch um Web-Archive. Alle Maven-Module verwenden einen gemeinsamen Parent-POM (Project Object Model) wie in Listing 1 dargestellt.

```
</project>
    <parent>
        <groupId>de.gameduell</groupId>
        <artifactId>jee-parent</artifactId>
        <version>2.8.0</version>
    </parent>
    ...
    <dependencies>
        ...
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-api</artifactId>
            <scope>provided</scope>
        </dependency>
    </dependencies>
</project>
```

Listing 1: Ausschnitt Module-POM

Durch den Einsatz eines Parent-POMs können modul-übergreifende Konfigurationen und die zu verwendenden Versionen von Dependencies leicht angepasst werden. Die Entscheidung darüber, welche Version der `javax:javaee-api` Bibliothek beim Build verwendet werden soll, wird in die Maven-Profile `jee6` und `jee7` ausgelagert (Listing 2).

```
<project>
    ...
    <profiles>
        <profile>
            <id>jee6</id>
            <activation>
                <activeByDefault>true</activeByDefault>
            </activation>
            <dependencyManagement>
                <dependencies>
                    <dependency>
                        <groupId>javax</groupId>
                        <artifactId>javaee-api</artifactId>
                        <version>6.0</version>
                        <scope>provided</scope>
                    </dependency>
                </dependencies>
            </dependencyManagement>
        </profile>
        <profile>
            <id>jee7</id>
            <dependencyManagement>
                <dependencies>
                    <dependency>
                        <groupId>javax</groupId>
                        <artifactId>javaee-api</artifactId>
                        <version>7.0</version>
                        <scope>provided</scope>
                    </dependency>
                </dependencies>
            </dependencyManagement>
        </profile>
        ...
```

Listing 2: Ausschnitt Parent-POM

Die standardmäßige Aktivierung des Maven-Profils jee6 erlaubt es allen Entwicklern ihre Maven-Builds wie bisher durchzuführen. Die Verwendung des optionalen Parameters `-P jee7` aktiviert das Maven-Profil `jee7` und ermöglichte es den Teammitgliedern des Migrationsprojekts, vorhandene Kompatibilitätsprobleme in den einzelnen Maven-Modulen zu erkennen und aufzulösen (Listing 3).

```
$ mvn clean install

$ mvn clean install -P jee7
```

Listing 3: Maven Build durchführen

Versionskontrolle mit Branches

Für die normale Entwicklung gilt die Vorgabe, dass alle Änderungen direkt auf dem Trunk durchgeführt werden. Allerdings sind nicht alle für die Migration notwendigen Anpassungen

abwärtskompatibel möglich. In solchen Fällen muss auf Branches der Versionskontrolle zurückgegriffen werden. Auf diese Weise können alle Änderungen unabhängig von der aktiven Produktentwicklung durchgeführt werden. Die einzelnen Teilschritte sind in Abb. 1 dargestellt.

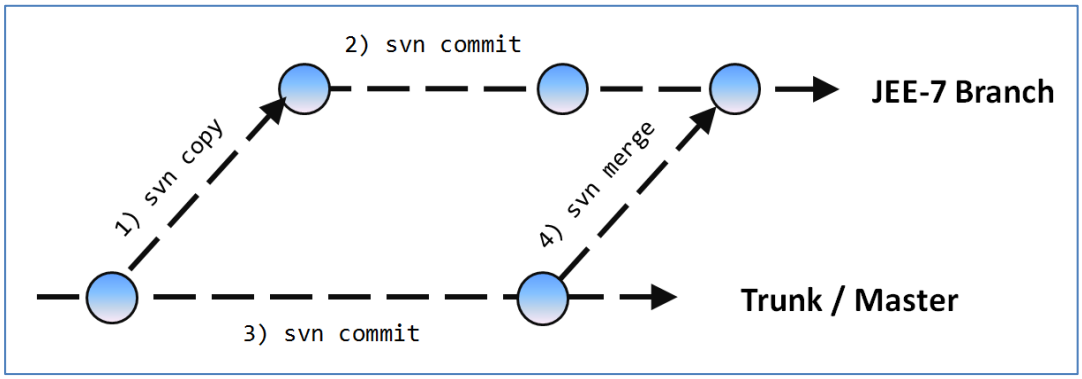

Abb. 1: Änderungen verwalten

Maven-Classifier

Die Unterstützung von Continuous Integration für JEE-7 und die Verwendung von Branches erfordert eine Erweiterung der vorhandenen Jenkins-Konfigurationen (Abb. 2)

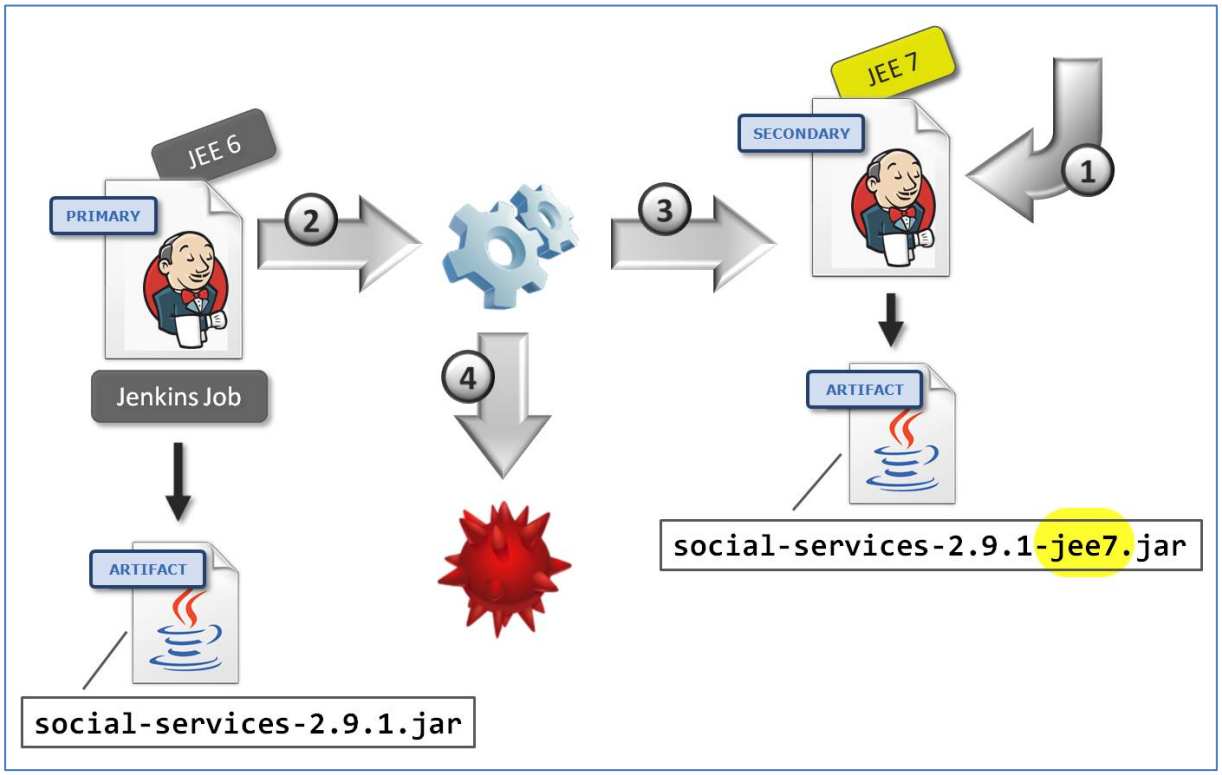

Abb. 2: Änderungen verwalten

1. Alle bereits vorhandenen Jenkins-Jobs (Primär-Jobs) werden dupliziert. Die neuen Jobs (Sekundär-Jobs) werden so konfiguriert, dass das Maven-Profil jee7 bei der Ausführung verwendet wird (Listing 3). Bei Maven-Modulen, die aus Kompatibilitätsgründen einen Branch benötigen, muss zusätzlich die URL zur Versionskontrolle angepasst werden.

2. Nach der erfolgreichen Ausführung eines Primär-Jobs werden alle Codeänderungen des betreffenden Maven-Moduls vom Trunk in den Branch überführt (Abb. 1).
3. Der Sekundär-Job wird als Nachfolger in Jenkins gestartet. Nichtkompatible JEE-7 Artefakte werden zur eindeutigen Kennzeichnung mit einem Maven-Classifier versehen.
4. Ggf. auftretende Merge-Konflikte müssen manuell aufgelöst werden. Nach der Konfliktlösung werden die durchgeführten Änderungen im Branch eingecheckt und der erfolglose Jenkins-Jobs neu gestartet.

Die unter Punkt 3. genannte Verwendung von Maven-Classifiers sind ein essentieller Bestandteil des in diesem Artikel vorgestellten Migrationskonzepts. Durch diesen Ansatz können Artefakte für unterschiedliche JEE-Zielumgebungen, bei identischen Artefaktnamen und Versionsnummern, erzeugt und verwendet werden. Die hierfür notwendige Konfiguration innerhalb einer POM-Datei ist exemplarisch in Listing 4 dargestellt.

```
</project>
    ...
    <profiles>
        <profile>
            <id>jee6</id>
            <dependencies>
                <dependency>
                    <groupId>de.gameduell.social</groupId>
                    <artifactId>social-services-jpa</artifactId>
                    <version>2.9.1</version>
                </dependency>
            </dependencies>
        </profile>
        <profile>
            <id>jee7</id>
            <dependencies>
                <dependency>
                    <groupId>de.gameduell.social</groupId>
                    <artifactId>social-services-jpa</artifactId>
                    <version>2.9.1</version>
                    <classifier>jee7</classifier>
                </dependency>
            </dependencies>
        </profile>
    </profiles>
    ...
```

Listing 4: Maven-Artefakte mit Classifier konfigurieren

Zero Downtime Deployment

Ein- bis zweimal täglich werden alle integrierten Codeänderungen auf der GameDuell Live-Plattform online gestellt, wobei die hundertprozentige Verfügbarkeit des Systems oberstes Ziel ist. Abb. 3 zeigt den Ablauf eines Deployments.

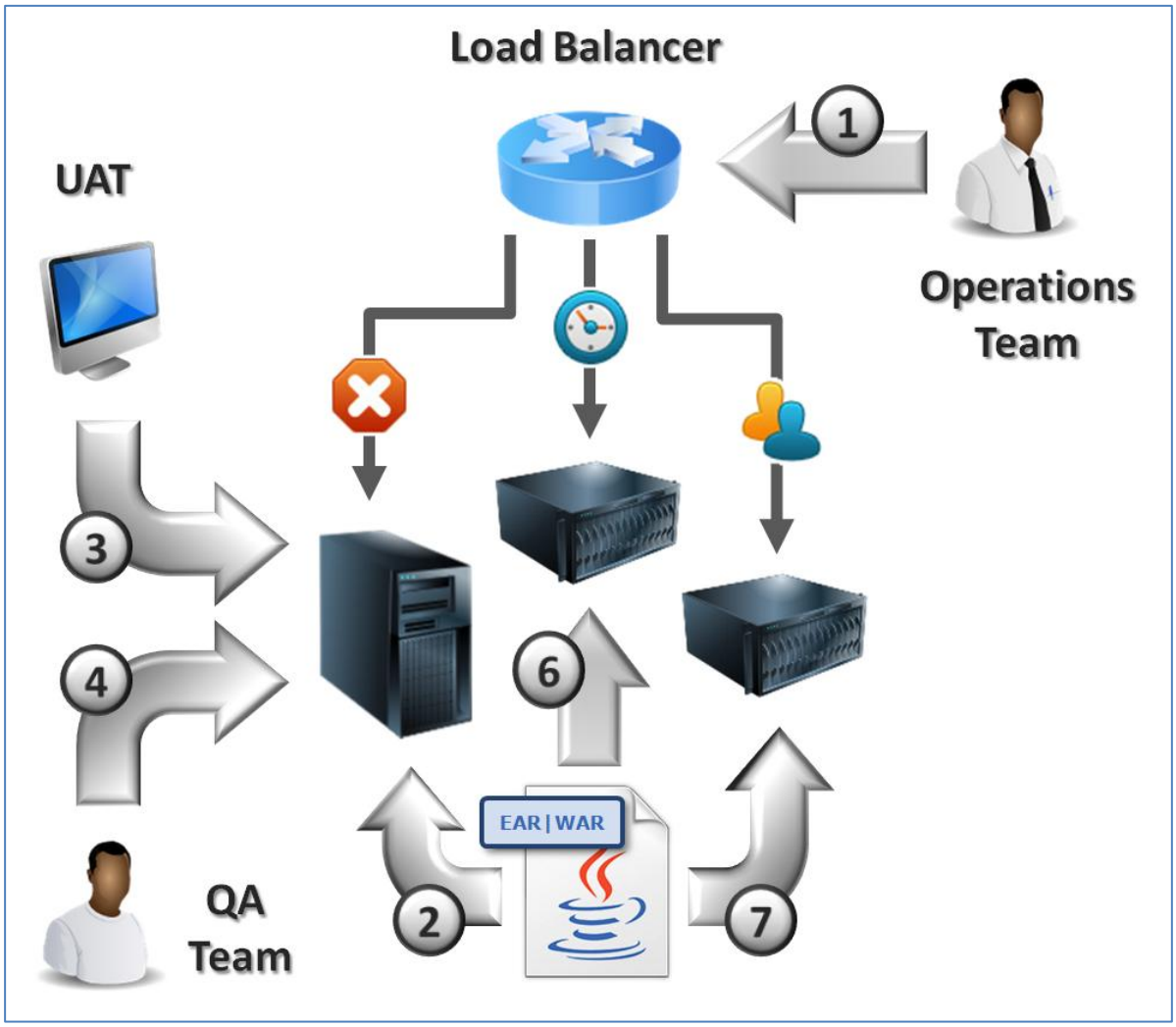

Abb. 3: Deployment auf der Live-Plattform

1. Ein Teil der Frontend-Server wird vom Operations-Team in den Standby Modus geschaltet. Alle laufenden Spiele können noch beendet werden. Neue Anfragen werden auf aktive Frontends weitergeleitet.
2. Das initiale Deployment erfolgt auf einem einzelnen Frontend, auf dem sich nun keine aktiven Benutzer mehr befinden.
3. Eine weitere Sammlung von automatischen User Acceptance Tests wird gegen das einzelne Frontend gestartet.
4. Ein Mitglied des QA-Teams sichtet alle in das Deployment eingeflossenen JIRA-Tickets und führt zusätzliche manuelle Tests durch. Bei erfolgreichem Test wird der Status der Tickets entsprechend fortgeschrieben.
5. Das einzelne Frontend wird wieder in die Verteilung durch den Loadbalancer genommen, wodurch „echte“ User auf den Server zugreifen können. Für die nächsten 15-20 Minuten erfolgt ein verstärktes Monitoring des Frontends, um evtl. auftretende Anomalien zu erkennen.
6. Wenn das Frontend ohne Auffälligkeiten funktioniert, wird der Rollout auf alle weiteren im Standby laufenden Frontends ausgeführt.
7. Abschließend werden alle noch nicht aktualisierten Frontends sukzessive in den Standby-Mode gesetzt, aktualisiert und anschließend wieder in die Verteilung genommen.

Unter Anwendung des beschriebenen Prozesses können einzelne Server-Instanzen alternativ auch mit einem JEE-7 kompatiblen Application-Server betrieben werden. Dieser Ansatz minimiert das Risiko, da nicht alle Server auf einmal umgestellt werden.

Aufräumarbeiten

Nach der vollständigen Migration der Live-Plattform nach JEE-7, heißt es nun aufzuräumen. Alle angelegten Branches müssen wieder in den Trunk integriert werden, da die Weiterentwicklung für JEE-7 ab diesem Zeitpunkt ausschließlich auf dem Trunk erfolgt (Abb. 4). Die Branches können anschließend gelöscht oder stillgelegt werden.

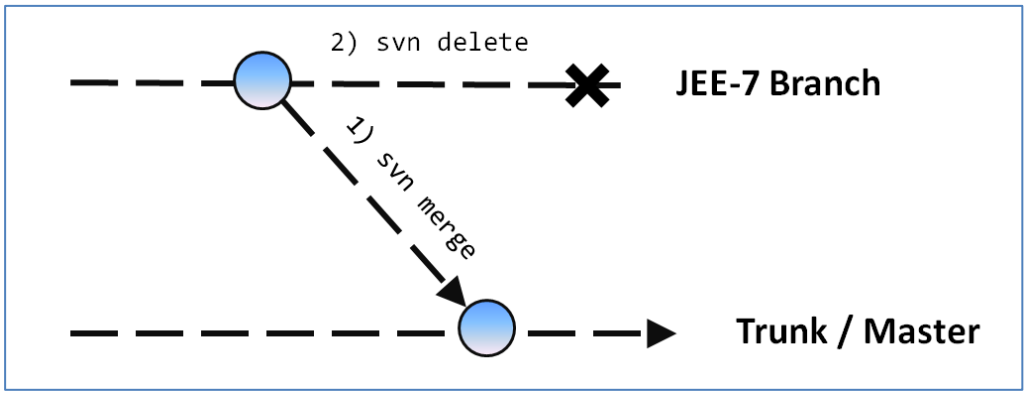

Abb. 4: Branches der Versionskontrolle integrieren

Im nächsten Schritt werden die nicht länger benötigten Maven-Profile aus den POM-Dateien entfernt, um auf diese Weise die Komplexität zu verringern. Außerdem wird Jenkins in seinen ursprünglichen Zustand zurückgesetzt, indem alle sekundären Jenkins-Jobs gelöscht werden.

Fazit

Das in diesem Manuskript beschriebene Vorgehen ist sicher nicht die einzige Lösung, um unter den gegebenen Randbedingungen, die Migration von JEE-6 auf JEE-7 erfolgreich abzuschließen. Allerdings überzeugt es durch seine unkomplizierte Struktur und kann ggf. mit kleinen Anpassungen auch in anderen Bereichen angewendet werden. Es bietet somit einen guten Ansatz zur Durchführung von Migrationsprojekten im Umfeld hochverfügbarer Systeme und der Verwendung von Continuous Delivery.

Kontaktadresse:
Dirk Ehms
GameDuell GmbH
Taubenstrasse 24-25
D-10117 Berlin

Telefon:	+49 (0) 30-288 768 210
Fax:	+49 (0) 30-288 768 299
E-Mail	dirk.ehms@gameduell.de
Internet:	inside.gameduell.de