# Play!: Elastische Skalierbarkeit für Web-Anwendungen

Axel Irriger

GB Telecommunications & Media
msg systems ag
Mergenthalerallee 55-59
65760 Eschborn
axel.irriger@msg-systems.com

**Abstract:** Die Entwicklung hochskalierbarer Dienste, die einer Vielzahl von Kunden angeboten werden können, stellt besondere Herausforderungen an die Software-Entwicklung. Das play!-Framework stellt einen Lösungsbaustein für solche skalierbare Architekturen dar.

## 1 Einleitung

Die Entwicklung von skalierbaren Web-Anwendungen wurde meist unter dem Gesichtspunkt einer vertikalen Skalierbarkeit verfolgt. Die Anpassung an sich dynamisch ändernde Last-Szenarien wirft und die Forderung nach horizontaler Skalierbarkeit wird neue Fragestellungen für die Anwendungsarchitektur auf.

Die Möglichkeiten, horizontal zu skalieren, bestehen einerseits in der Integration traditioneller Mechanismen zur Lastverteilung und Integration, oder aber durch eine Vereinfachung des Programmiermodells. Den Weg der Vereinfachung verfolgt das play!-Framework (http://www.playframework.org/), das zustandsbehaftetes Verhalten durch die Prinzipien der atomaren Aufrufe und des (netzwerkweiten) Cachings ersetzt, aber gleichzeitig eine einfache Integrierbarkeit ermöglicht. Für einen umfassenden Nutzen müssen die Besonderheiten aber bereits beim Architekturentwurf berücksichtigt werden.

Durch die starke Ausrichtung auf Web- und Cloud-Anwendungen bietet sich das play!-Framework für die moderne Anwendungsentwicklung an.

## 2 Neue Architekturen durch geänderte Nutzungsverhalten

Eine normale Web-Anwendung ist darauf ausgelegt, dass die Schnittstelle zum Nutzer die Bedienoberfläche ist, die an einen Browser ausgeliefert wird. Im Falle von Cloud-Anwendungen bildet nicht nur die Bedienoberfläche, sondern in wachsendem Maße auch eine Programmierschnittstelle der Anwendung eine Schnittstelle zum Nutzer.

Einerseits wird die Cloud-Anwendung entlastet, indem keine kompletten Web-Seiten mehr ausgeliefert werden, sondern lediglich Daten. Die Aufbereitung und Darstellung findet vollständig auf dem Endgerät statt. Auf der anderen Seite ermöglicht eine veröffentlichte Programmierschnittstelle eine einfache Integration in andere Anwendungen und Nutzungskontexte, wodurch die Anwendung intensiver genutzt wird. Dieser Zusammenhang ist in Abbildung 1 dargestellt.

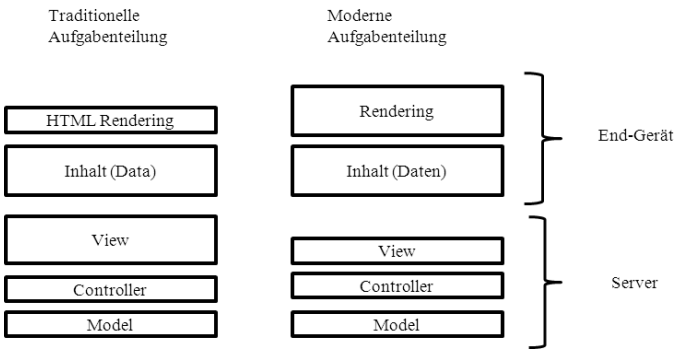


Abbildung 1 Aufgabenverlagerung im Web-Stack

Im Gegensatz zu traditionellen Web-Frameworks, die sowohl Präsentation, Logik als auch Zustand auf den Server verlagern, setzt das play!-Framework an der Stelle der Programmierschnittstelle an und betrachtet jeden Aufruf als atomar und zustandslos. Hierdurch wird Komplexität auf die nutzende Anwendung verlagert und das Programmiermodell für den Server vereinfacht. Diese Vereinfachung ermöglicht es, eine Anwendung sehr einfach horizontal zu skalieren.

## 3 Anforderungen eines skalierbaren Cloud-Service

Um einen Dienst oder eine Anwendung als „Cloud Service“ anbieten zu können, muss man bei der Architektur davon ausgehen, dass Komponenten dynamisch bereitgestellt und miteinander betrieben werden, aber zeitweise nicht verfügbar sein können. Für die Schicht des Applikationsservers bzw. der Geschäftslogik bedeutet dies:

- Eine einzelne Instanz kann ausfallen, die Anwendung funktioniert weiterhin
- Die Zahl der Instanzen der Anwendung ist unbekannt
- Die Verfügbarkeit von Instanzen ist im Vorfeld unbekannt

Diese Flexibilität kann durch die Erfüllung folgender Charakteristiken erreicht werden:

- Wissen ist nie auf nur einer Instanz bekannt („isoliertes Wissen“)
- Operationen sind atomar („Atomizität“)
- Komponenten interagieren nur lose gekoppelt miteinander („lose Kopplung“)
- Fehler in einem Aufruf sind auf diesen Aufruf beschränkt

Die aufgeführten Charakteristiken werden nachfolgend näher erläutert.

### 3.1 Isoliertes Wissen

Bei einer klassischen Web-Anwendung werden Sitzungsinformationen über einen Cookie identifiziert, sind aber auf dem Server gespeichert. Die Anwendung referenziert über einen Schlüssel die Sitzung, wodurch nur wenige Informationen übertragen werden. Im Falle eines Ausfalls des Servers sind damit alle Sitzungsinformationen dieser Instanz verloren.

Dieser Situation wird in der Regel mit einem Replikationsmechanismus begegnet, der die Informationen auf andere Server verteilt. Dieser skaliert aber nur teilweise, da durch die Menge an Servern auch der Kommunikationsaufwand für den Abgleich steigt. Eine optimale Skalierbarkeit einer Anwendung ist durch einen Verzicht auf geteiltes Wissen gegeben. In dieser Situation kann für jede Anfrage individuell entschieden werden, durch welche Instanz eine Anfrage bearbeitet wird.

Der Verzicht auf gemeinsames Wissen bei der Verarbeitung von Aufrufen ist ein zentrales Design-Prinzip der Programmiersprache Erlang (siehe [A03]), die Hochverfügbarkeit in der Telekommunikationsbranche ermöglichte.

### 3.2 Atomizität

Die Forderung nach atomaren Operationen ist sowohl aus Gründen der Skalierbarkeit, als auch der Fehlertoleranz wichtig.

Aus Sicht der Skalierbarkeit ist es wünschenswert, dass jede neue Anfrage von einem beliebigen Server bedient werden kann. Ist es erforderlich, dass mehrere Aufrufe durch den gleichen Server bedient werden, ist dies nur noch eingeschränkt gegeben. In Verbindung mit Datenpersistenz bedeutet diese Forderung jedoch nicht, dass jede Änderung sofort überall sichtbar ist.

Aus Sicht der Fehlertoleranz sollten Verarbeitungen isoliert durchgeführt werden, um schädliche Seiteneffekte zu vermeiden. Im Falle eines Fehlers ist genau dieser Aufruf betroffen, nicht jedoch parallel arbeitende Aufrufe. Durch diese Forderung erhöht sich neben der Skalierbarkeit vor allem die Stabilität der Anwendung. Die positiven Auswirkungen sind als weiteres Design-Prinzip in die Entwicklung der Programmiersprache Erlang eingeflossen (siehe [A03]). Die isolierte Bearbeitung eines Aufrufs ermöglicht ebenso die Entwicklung gut skalierbarer Algorithmen. Das sogenannte „Aktor-Modell“ belegt dies mathematisch (siehe [C81]).

### 3.3 Lose Kopplung

Das Betriebskonzept einer Cloud-Umgebung ist auf die flexible, dynamische Bereitstellung von Komponenten ausgelegt. Dies bedeutet für die Anwendungsarchitektur, dass Verbindungen zu verwendeten Komponenten immer

wieder neu ermittelt werden müssen, da diese potenziell in variabler Instanzenzahl bereitgestellt werden.

## 5 Technologische Umsetzung eines skalierbaren Cloud-Service

Die oben angesprochenen Anforderungen an einen skalierbaren „Cloud Service“ lassen sich mit dem play!-Framework sehr gut umsetzen. Die Abbildung 2 zeigt eine Übersicht über die wesentlichen Komponenten des play!-Frameworks für die Anwendungsentwicklung.

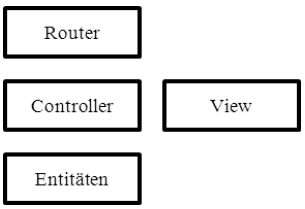


Abbildung 2 Komponenten des play!-Frameworks

Der Eintrittspunkt ist dabei die Konfiguration einer Route. Diese stellt eine URL dar, in der auch Platzhalter verwendet werden können und auf einen Controller verweist. Dieser ist für die Verarbeitung der Anfrage zuständig. Durch die Zustandslosigkeit des Frameworks werden hier statische Methoden verwendet. Falls eine Ausgabe erfolgen soll, verweist dieser auf eine zugehörige Vorlage, die damit durch das Framework gerendert wird. Im Gegensatz zu klassischen Web-Frameworks, die umfangreiche Bibliotheken, wie JSF, verwenden, kann im play!-Framework lediglich HTML, XML, JSON erzeugt werden. Durch die Tatsache, dass viele moderne Bedienoberflächen im Browser durch Verwendung von JavaScript erzeugt werden, reduziert dies weiter die Last auf dem Server.

### Erweiterbarkeit

Die Grundfunktionen des play!-Frameworks lassen sich auf das Bedienen von Web-Anfragen reduzieren. Um zusätzliche Funktionen einzubinden, beispielsweise Anfragen nebenläufig zu bearbeiten, können zusätzliche Module eingebunden werden. Für die nebenläufige Verarbeitung bietet sich die Erweiterung Akka (http://www.typesafe.com/technology/akka) an, die das Aktoren-Modell unter Java bereitstellt. Das play!-Framework stellt einen zentralen Laufzeitcontainer für Aktoren bereit, sodass diese automatisch im Lebenszyklus der Gesamtanwendung betrieben werden.

### Skalierbarkeit

Das play!-Framework verwendet zwei Mechanismen, um Skalierbarkeit zu erreichen. Zum einen ist durch die Verwendung statischer Controller-Methoden gewährleistet, dass

die Anwendung durch die Bereitstellung weiterer Instanzen in Kombination mit einem Lastverteiler horizontal skaliert. Zum anderen kann durch die Verwendung eines netzwerkfähigen Caches die Datenabfrage beschleunigt werden. Obwohl in der Standardauslieferung ehcache (http://ehcache.org/) verwendet wird, können andere Cache-Implementierungen verwendet werden.

**Zusammenspiel und Integration in Bestandsanwendungen**

Um einen skalierbaren Cloud-Service auf Basis des play!-Frameworks zu erstellen, kann die in Abbildung 3 aufgeführte Architektur-Vorlage dienen.

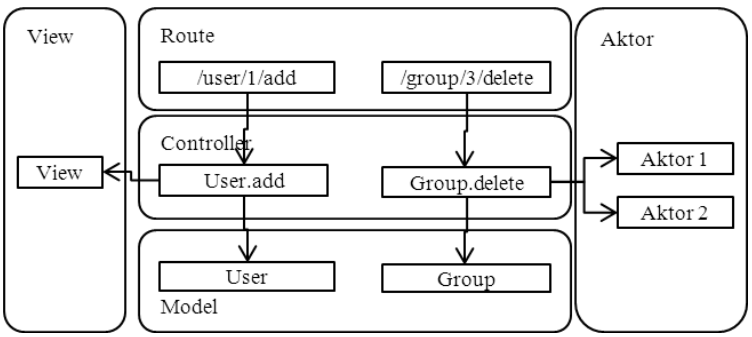


Abbildung 3Vorlage für play!-Planung

Diese verknüpft die wesentlichen Aspekte des Frameworks und bietet eine schnelle Orientierung. Anhand der Vorlage sind die wesentlichen Merkmale einfach ersichtlich:

- Die Definition der Programmier- und Aufrufschnittstelle anhand von Routen
- Die Ablage der Geschäftslogik in zugehörige Controller-Klassen
- Die Verarbeitung von komplexen Abläufen und Berechnungen in nebenläufigen Aktoren

Bei der Migration einer bestehenden Anwendung muss man die Besonderheiten berücksichtigen, die sich durch die geänderte Architektur ergeben. Für die Verwendung von Aktoren müssen implementierte Algorithmen evtl. angepasst werden, um Nebenläufigkeit zu ermöglichen. Des Weiteren sollte berücksichtigt werden, dass die Programmierschnittstelle eine einfache Abbildung in eine URL ermöglicht, um die Vereinfachungen nutzen zu können, die das Framework anbietet.

Für den Betrieb einer play!-Anwendung muss beachtet werden, dass ein Betrieb in modernen Applikationsservern möglich ist, sofern die Servlet 3.0 Spezifikation erfüllt ist. Frühere Servlet-Spezifikationen werden aktuell nicht unterstützt.

## 6 Vergleich zu anderen Frameworks

Die Frage, ob der Einsatz des play!-Frameworks sinnvoll, hängt überwiegend von den Anforderungen an. Ein Vergleich mit einigen bestehenden Frameworks bietet sich daher an. Nachfolgend werden einzelne Frameworks beispielhaft in Relation zu play! gesetzt, um eine Einsatzentscheidung zu vereinfachen.

#### Ruby on Rails

Das Framework „Ruby on Rails“ legt den Schwerpunkt auf Standardvorgaben („convention over configuration“) und eine hohe Ausdrucksfähigkeit. Das play!-Framework setzt diese Prinzipien für die Programmiersprache Java um. Von daher ist es sehr gut vergleichbar und Rails-Entwickler finden sich schnell in play! zurecht.

#### Spring MVC

Spring MVC ist ein Web-Anwendungs-Framework, das auf den Grundzügen von Spring aufbaut. Es ist auf die Entwicklung von komplexen serverseitigen Web-Anweendungen ausgelegt. Das play!-Framework benötigt diese Flexibilität nicht, da es auf clientseitige Anwendungen ausgelegt ist. Im Gegensatz zu Spring werden bei der Verwendung von play! sehr viel mehr Konventionen vorgegeben, was die Entwicklung stark vereinfacht.

## 7 Zusammenfassung

Das vorgestellte play!-Framework bietet eine schlanke Möglichkeit, skalierbare Cloud-Services zu entwickeln. Die Verlagerung der Darstellung auf den Browser vereinfacht das Programmiermodell, was insbesondere datengetriebenen Diensten zu Gute kommt. Insbesondere in Verbindung mit einem Aktoren-Framework wie Akka bildet es eine umfassende Laufzeitplattform für moderne Anwendungen, die über eine Programmierschnittstelle in dritte Anwendungen eingebunden werden sollen.

## Literaturverzeichnis


[A03] Armstrong, J.: Making reliable distributed systems in the presence of software errors. SICS Dissertation Series 34, Stockholm, 2003

[C81] Clinger, W.D.: Foundations of Actor Semantics. MIT Artificial Intelligence Laboratory, Massachusetts, 1981.

[JG04] Jeffrey, D.; Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Sixth Symposion on Operation System Design and Implementation, San Francisco, CA, 2004.

[ST97] Shavit, N.; Touitou, D.: Software transactional memory. Distributed Computing Volume 10, 1997.