Sandro Lehmann
(sandro.lehmann@mimacom.com)
ist als Senior Software Engineer bei der mimacom ag tätig. Seine Schwerpunkte sind Projekte in der Entwicklung von Individualsoftware. Seit mehreren Jahren arbeitet er mit diversen Spring-Projekten und Webtechnologien.

Der Elefant im Frühling

Apache Hadoop mit seinem schnell wachsenden Ökosystem (z. B. Pig, Hive, HBase u. a.) genießt seit einiger Zeit viel Aufmerksamkeit. Doch wie einfach lässt sich damit entwickeln und wie lassen sich herkömmliche Applikationen damit verbinden? Dieser Artikel zeigt, wie die beiden Projekte *Spring for Apache Hadoop* [Spring1] und *Spring XD* [Spring2] den Einstieg in die Welt von Hadoop erleichtern. Spring ist ein quelloffenes Framework für die Java-Plattform und bietet eine Vielzahl von Funktionalitäten und Erweiterungen (z. B. Spring Security [Spring3], Spring Batch [Spring4], Spring Cloud [Spring5] und viele weitere). Generell geht es darum, die bekannten Elemente von Spring [Spring6], wie z. B. Dependency-Injection, Scheduling, Workflows mit Hadoop zusammenzuführen. Versierte Benutzer von Spring werden sich sofort wie zu Hause fühlen. Und für UNIX-Benutzer hat Spring XD das Verknüpfen von Funktionskomponenten mit dem Pipe-Symbol | parat.

Rund um Hadoop

Hadoop ist nach wie vor DIE Plattform für „Big Data“ und verteiltes Rechnen. Jedoch besitzt Hadoop ein Low-Level-Programmiermodell, mit dem aufwendig zu programmieren ist. Man kommt häufig nicht darum herum, viel Boilerplate-Code (Infrastruktur-Code) zu schreiben. Schon für das Konfigurieren eines einfachen MapReduce-Jobs müssen einige Zeilen Java programmiert werden (siehe Listing 1), ganz abgesehen von der Programmierung der Map- und der Reduce-Funktionen selbst.

Deswegen gibt es eine Reihe von Erweiterungen, die auf Hadoop aufbauen, auf einer höheren Abstraktionsebene sind und somit die Komplexität vermindern. Ein Beispiel einer solchen Erweiterung ist Pig [Pig1]. Mit Pig können SQL-ähnliche Scripts (Pig Latin genannt) erstellt und auf Hadoop ausgeführt werden. Intern werden die Scripts in MapReduce-Jobs übersetzt.

Spring for Apache Hadoop

Unter anderem aus oben genannten Gründen wurde das Spring-Projekt *Spring for Apache Hadoop* ins Leben gerufen. Das Ziel von *Spring for Apache Hadoop* ist die Vereinfachung der Entwicklung von Hadoop-Applikationen. Es bietet ein bekanntes und konsistentes Programmierungs- und Konfigurationsmodell an.

Unterstützung erhält der Programmierer zum Beispiel zur Nutzung des HDFS (*Hadoop Distributed File System*) sowie zur Definition von Workflows und Integration von Datenanalyse-Komponenten (z. B. Pig und Hive). Das Spring-Projekt

```
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setMapperClass(WordCountMapper.class);
job.setReducerClass(WordCountReducer.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

Listing 1: Beispiel für eine Konfiguration eines MapReduce-Jobs per Java.

```
<hadoop: job id="wordcountJob"
          input-path="${inputPath}" output-path="${outputPath}"
          mapper="com.mimacom.wordcount.WordCountMapper"
          reducer="com.mimacom.wordcount.WordCountReducer"/>
```

Listing 2: Beispiel für eine Deklaration eines MapReduce-Jobs per Spring Konfiguration.

baut zudem auf existierenden Service-Layer-Abstraktionen auf. Das Spring-Framework und die Erweiterungen *Spring Batch*, *Spring Integration* und *Spring Data* sind integrierte Bestandteile.

Die Vorteile liegen auf der Hand. Mit all den Komponenten können große und komplexe Applikationen gebaut werden.

MapReduce-Job per Spring

Ein gutes Beispiel, wie man einen MapReduce-Job per Spring-Konfiguration deklariert und Boilerplate-Code vermeidet, ist in Listing 2 dargestellt. Der Job muss hier nicht wie bei Listing 1 per Java deklariert werden.

Parameter wie z. B. InputFormatClass und OutputFormatClass (vgl. Listing 1) sind optional. Spring erkennt und ermittelt fehlende Parameter zur Laufzeit durch Introspektion. Zudem können Werte, was bei Spring-Applikationen weit verbreitet ist, parametrisiert und externalisiert werden (z. B. ${inputPath}). Das vereinfacht das Deployen und Stagen der Software, da sich gewisse Parameter auf den verschiedenen Umgebungen häufig unterscheiden können.

Der deklarierte MapReduce-Job kann anschliessend per Dependency-Injection oder über den Application-Context von Spring referenziert und gestartet werden (vgl. Listing 3).

```
JobRunner runner = (JobRunner)
context.getBean(„wordcountJob“);
runner.call();
```

Listing 3: Starten eines per Spring-Konfiguration deklarierten MapReduce-Jobs.

Testen

Den Entwicklern von Spring war es immer sehr wichtig, dass Code auch getestet werden kann – unabhängig von der Umgebungskomplexität. Auch das Testmodul vom Projekt *Spring for Apache Hadoop* bietet einigen Support an. So kann man beispielsweise mit der Annotation *@MiniHadoopCluster* eine Testklasse anweisen, einen Hadoop-Cluster im Speicher zu starten.

Für das Testen bieten diverse Hilfsklassen zusätzliche Funktionalitäten an. Vor allem MapReduce-Jobs können relativ einfach getestet werden. Beispielsweise kann ohne viel Aufwand eine Testklasse geschrieben werden, die auf das Ende von MapReduce-Jobs wartet und danach dessen Status und Resultat verifiziert.

Für Pig, welches im gezeigten Beispiel benutzt wird, gibt es leider noch keine zusätzliche Testfunktionalität.

Spring XD

Es macht Sinn, Spring XD (XD steht für Extreme Data) für die Data-Ingestion (validieren, verarbeiten und importieren von Daten) in Hadoop zu benutzen, da es eines der Hauptziele des Projekts ist. Die lang erwartete finale Version von Spring XD ist im Juli 2014 in der Version 1.0 erschienen.

Kurz gesagt ist Spring XD eine Laufzeitumgebung für Big-Data-Applikationen. Es ist ein verteilter und erweiterbarer Service für Data-Ingestion, Echtzeitanalyse, Batch-Jobs und Datenexport. Zudem eignet es sich für die Zusammenarbeit mit HDFS und Hadoop.

Ein sogenannter *Stream* definiert in der Welt von Spring XD die ereignisgesteuerte Datenaufnahme von einer *Source* (Datenquelle) zu einem *Sink* (Datenausgabe) mit einer beliebige Anzahl von *Prozessoren* (z. B. Filter und Transformatoren) dazwischen. Streams haben eine lange Lebensdauer und müssen manuell gestoppt werden, falls man sie nicht mehr braucht. Die Module (Sourcen, Sinks und Prozessoren) werden mit dem Pipe-Symbol | verbunden (analog dem Verbinden von einzelnen Befehlen auf einer Unix-Shell). Das vereinfacht den Einstieg für gewohnte Unix-Benutzer.

Auf der XD-Konsole kann man Streams definieren und starten. Diese Befehle werden von der XD-Konsole an die REST-Schnittstelle des Spring-XD-Services gesendet. Listing 4 beschreibt HTTP als Source und HDFS als Sink. Der Befehl startet einen HTTP-Server der auf dem Port 9000 HTTP-Requests empfängt und diese in einer Datei auf dem HDFS ablegt (standardmäßig unter dem Pfad /xd/<Streamname>/,). Mit --deploy wird der Stream gestartet.

```
xd:> stream create ⏎
--name myStream ⏎
--definition “http | hdfs” ⏎
--deploy
```

Listing 4: Stream welcher über HTTP Daten gesendete Daten in eine Datei auf dem HDFS schreibt.

Die einzelnen Module basieren wiederum auf Spring-Programmen. Oft besitzen sie diverse Konfigurationsmöglichkeiten wie z. B. setzen eines Ports oder eines Ordnernamens. Module kann man auch gut selber modifizieren oder neu entwickeln.

Zusätzlich zu Streams, Sinks und Prozessoren bietet Spring XD auch die Möglichkeit, Spring-Batch-Jobs zu benutzen, welche im folgenden Beispiel gezeigt wird. Diese können manuell, durch einen Cron-Job oder einen Sink gestartet werden.

Ein End-to-End-Beispiel

Nachfolgend wird ein Beispiel erläutert, welches Hadoop, Pig, Spring for Apache Hadoop, Spring XD und Spring-Batch benutzt. Die benötigten Daten für dieses Beispiel sind Wetterdaten. Diese sind für Big-Data-Analysen prädestiniert, da Wetterstationen eine Vielzahl von Daten generieren.

Wetterdaten findet man zum Beispiel im Archiv von NOAA (National Oceanic and Atmospheric Administration) [NOAA1]. Die amerikanische Behörde unterhält das weltweit größte Klimadatenarchiv. Das folgende Beispiel ist auf die Datenstruktur dieses Archivs ausgerichtet [NOAA2].

Das Beispiel kann auf GitHub (vgl. [Mima1]) angesehen und heruntergeladen werden. Voraussetzungen dafür sind ein laufendes Apache Hadoop System in der Version 2.x und Spring XD in der Version 1.0.

Folgender Zyklus wird durchlaufen:

1. Ein Stream schaut in einem Ordner periodisch nach neuen Dateien mit Wetterdaten.
2. Ein Spring Batch-Job wird gestartet, der
   a. die neue Datei auf das HDFS kopiert;
   b. mittels eines Pig-Scripts Durchschnittstemperaturen der einzelnen Tage ermittelt;
   c. das Resultat zurück auf das Dateisystem schreibt.
3. Ein weiterer Stream versendet das Resultat per E-Mail.

Die Grafik zeigt die verschiedenen Schritte des Beispiels.

Der Batch-Job

Der Batch-Job (siehe Grafik) besteht aus den oben beschriebenen drei Schritten (Listing 5).

Erwähnenswert sind vor allem der Import- (Listing 6) und Durchschnittstemperatur-Schritt (Listing 7). Die neue Datei mit den Wetterdaten wird unter einem bestimmten Pfad auf dem HDFS abgelegt.

Innerhalb des *Script-Tags* wird in diesem Beispiel Groovy verwendet. Werden

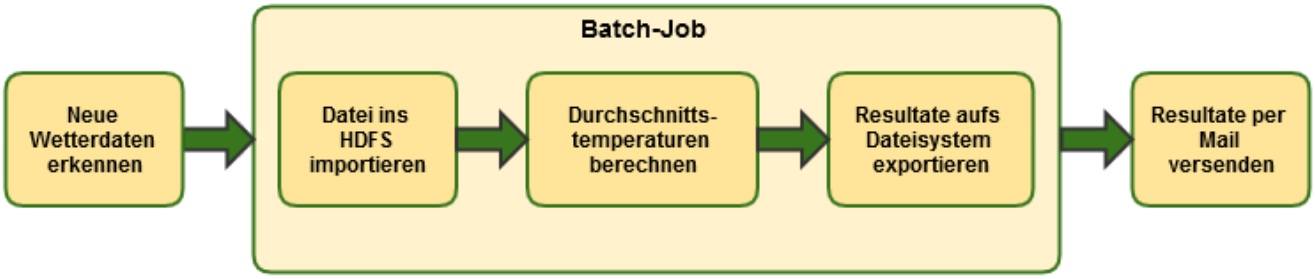

Abb.: Grafische Übersicht des Beispiels.

```
<batch:job id="averageTemperatureJob">
    <batch:step id="import" next="averageTemperature">
        <batch:tasklet ref="importScriptTasklet"/>
    </batch:step>
    <batch:step id="averageTemperature" next="export">
        <batch:tasklet ref="averageTemperaturePigTasklet"/>
    </batch:step>
    <batch:step id="export">
        <batch:tasklet ref="exportScriptTasklet"/>
    </batch:step>
</batch:job>
```

Listing 5: Batch-Job mit drei Schritten

```
<hadoop:script-tasklet id="importScriptTasklet" script-ref="importScript" scope="step"/>
<hadoop:script id="importScript" language="groovy" scope="step">
    <hadoop:property name="localSourceFile"
                     value="${resources:file:///#{jobParameters['absoluteFilePa
th']}}"/>
    <hadoop:property name="hdfsInputDir" value="${weather.input.path}"/>
    <hadoop:property name="hdfsOutputDir" value="${weather.output.path}"/>

  if (!fsh.test(hdfsInputDir)) {
     fsh.mkdir(hdfsInputDir);
  }

  fsh.rm(hdfsInputDir + „*");
  fsh.copyFromLocal(localSourceFile, hdfsInputDir);

  if (fsh.test(hdfsOutputDir)) {
     fsh.rmr(hdfsOutputDir)
  }

</hadoop:script>
```

Listing 6: Schritt, welcher die neue Datei auf das HDFS speichert.

```
<hadoop:pig-tasklet id="averageTemperaturePigTasklet">
    <hadoop:script location="averageTemperature.pig">
        <hadoop:arguments>
            inputfile=${weather.input.path}
            outputpath=${weather.output.path}
        </hadoop:arguments>
    </hadoop:script>
</hadoop:pig-tasklet>
```

Listing 7: Schritt, welcher die Durchschnittstemperaturen der neuen Wetterdaten ausrechnet.

die Scripts länger, kann man sie auch auslagern. fsh (File System Shell) ist eine implizite Variable mit welcher man auf das HDFS zugreifen kann.

Anschließend werden die Tagesdurchschnittstemperaturen durch ein Pig-Script ausgerechnet (Listing 7).

Das Pig-Script wird hier nicht weiter erläutert, kann aber bei GitHub [Mima1] angeschaut werden.

Wenn der Batch-Job fertig programmiert ist, werden die nötigen Bibliotheken und die Job-Definition in den erforderlichen Verzeichnissen von XD abgelegt. Jetzt fehlt noch das Erstellen und Deployen des Jobs per Spring XD (Listing 8). Dies geht am einfachsten über die Spring-XD-Konsole.

```
xd:> job create ⏎
--name weatherJob ⏎
--definition „averageTemperatureJob" ⏎
--deploy
```

Listing 8: Anlegen und deployen des Wetter-Jobs.

Neue Wetterdateien erkennen

Die Stream-Definition unter Listing 9 achtet kontinuierlich auf Veränderungen in einem Order auf dem Dateisystem (standardmäßig unter dem Pfad /tmp/xd/input/<Streamname>/). Sobald neue Wetterdateien im Ordner /tmp/xd/input/weatherFiles/ erscheinen, wird der Batch mit der neuen Datei als Eingabeparameter gestartet. Die Option --ref bedeutet, dass eine Referenz auf die Datei und nicht der Dateiinhalt weitergegeben werden soll (vgl. Listing 9).

```
xd:> stream create ⏎
--name weatherFiles ⏎
--definition „file ⏎
--ref=true > queue:job:weatherJob" ⏎
--deploy
```

Listing 9: Anlegen des Streams der neue Dateien an den Wetter-Job weiterleitet.

Resultat per Mail versenden

Sobald neue Resultate vom Batch-Job geliefert werden, werden diese an einen E-Mail-Empfänger gesendet (Listing 10).

```
xd:> stream create mailstream ↵
--definition „file ↵
--dir=/tmp/xd/output/weatherJob/ ↵
--outputType=text/plain | ↵
mail ↵
--to='\"name@domain.ch\"' ↵
--from='\"sender@domain.ch\"' ↵
--subject='\"Neue Durchschnitts-
temperaturen\"'" ↵
--deploy
```

Listing 10: Resultate werden per Mail versendet.

In dem Beispiel werden die Durchschnittstemperatur-Dateien nicht als Anhang sondern als Text der E-Mail versendet. Will man die Dateien als Anhang versenden, muss das vordefinierte Mail-Sink-Modul angepasst werden. In der entsprechenden XML-Konfigurationsdatei müsste die Zeile <int-mail:attachment-filename value="${filename:null}" /> hinzugefügt werden. In Spring XD ist es üblich Module zu modifizieren oder neue zu schreiben.

Fazit Spring XD

Wo steht Spring XD heute? Spring XD bietet eine Vielzahl von vordefinierten Sourcen, Sinks und Prozessoren und stellt eine einfache DSL zur Verfügung, um Streams zu definieren. Will man die vorhandenen Module erweitern, muss häufig nur die Konfiguration angepasst werden.

Zusätzlich bietet Spring XD die Möglichkeit von Taps an. Taps werden dazu benutzt, um Daten von Streams abzuhören, ohne dass dieser davon etwas mitbekommt. Taps sind empfehlenswert, um Metriken zu sammeln und den Datenstrom zu analysieren.

Ein primärer Anwendungsfall ist die Echtzeitanalyse des Datenstroms, der über den primären Stream läuft. Zum Beispiel werden Taps häufig für Logging benutzt.

Beim Arbeiten mit Spring XD muss man achtgeben, welche Bibliotheken bereits zur Verfügung gestellt werden, denn diese kann man beim Deployen weglassen (Achtung Versionskonflikte). Zum Beispiel sind Hadoop- und Spring-for-Apache-Hadoop-Bibliotheken bereits integriert.

Fazit allgemein

Die beiden vorgestellten Spring-Projekte sind für gewohnte Anwender von Spring schnell erschließbar und bieten viele attraktive Funktionalitäten. Spring vereinfacht und beschleunigt die Entwicklung erheblich. Komplexität wird verborgen und der Entwickler kann sich mehr den fachlichen Aufgaben widmen.

Mit geringem Aufwand lassen sich bereits vielschichtige Applikationen ableiten, für deren Entwicklung man ohne Spring viel mehr Zeit investieren müsste. Die Projekte von Spring sind gut dokumentiert und Tutorials gibt es genügend.

Mit einem passenden Zitat von Rod Johnson möchte ich diesen Artikel beenden.

"I believe that Spring is unique, for several reasons:
It addresses important areas that many other popular frameworks don't. Spring is both comprehensive and modular.
Spring is designed from the ground up to help you write code that's easy to test.
Spring is an increasingly important integration technology."

Rod Johnson,
The ServerSide.com, 2005

Links

[Spring1] Spring for Apache Hadoop http://projects.spring.io/spring-hadoop/
[Spring2] Spring XD http://projects.spring.io/spring-xd/
[Spring3] Spring Security http://projects.spring.io/spring-security/
[Spring4] Spring Batch http://projects.spring.io/spring-batch/
[Spring5] Spring Cloud http://projects.spring.io/spring-cloud/
[Spring6] Spring http://spring.io/
[Pig1] Pig http://pig.apache.org/
[Mima1] Beispielprojekt https://github.com/mimacom/spring-hadoop-xd-example
[NOAA1] NOAA (National Oceanic and Atmospheric Administration) www.ncdc.noaa.go
[NOAA2] Wetterdaten von NOAA http://cdo.ncdc.noaa.gov/qclcd_ascii/