

SpringOne Platform

Spring and Big Data

Thomas Risberg

@trisberg

Pivotal

About me

Thomas Risberg

- Member of the Spring Cloud Data Flow engineering team at Pivotal
- Lead for the Spring for Apache Hadoop project
- Joined the Spring Framework open source project in 2003 working on JDBC support
- co-author of “Professional Java Development with Spring Framework” from Wrox 2005 and “Spring Data” book from O'Reilly 2012



Agenda

- Introduction
 - Big Data
 - Microservices with Spring Boot and Spring Cloud Stream
- Spring projects for “Big Data”
 - Spring Integration, Spring Batch, Spring Data etc.
- Data Ingestion
- Data Analysis
- Cloud
 - Local laptop, Cloud Foundry, Google Cloud etc.

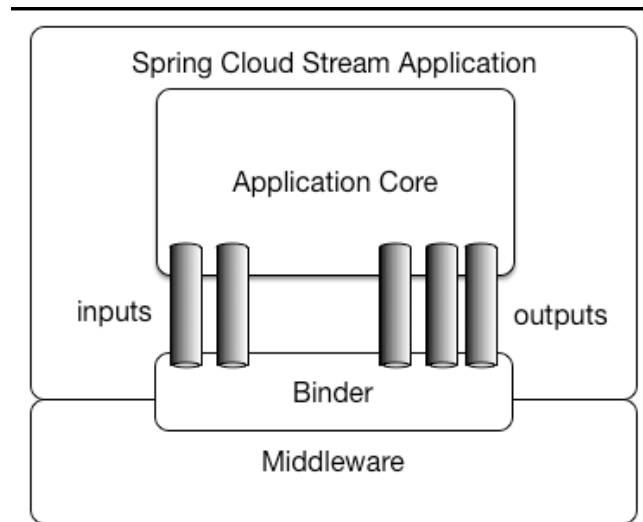
Introduction

What is “Big Data”

- Existing definitions are very vague:
 - Wikipedia” “Big data is a term for data sets that are so large or complex that traditional data processing applications are inadequate.”
- We can look at it in terms of new technologies used to process data:
 - Kafka
 - Hadoop
 - HBase
 - Cassandra
- What these technologies have in common:
 - Open Source projects usually started at “big” web companies
 - Ability to scale
 - Support structured and un-structured data

Boot Apps for Big Data

- Cloud Native
- Spring Cloud Stream
 - Rabbit / Kafka binder
- Spring Data
 - Spring for Apache Hadoop
 - Spring Data Cassandra



Spring Projects for Big Data

Spring Projects for Big Data

- Spring Projects for “Big Data”
 - Spring for Apache Kafka [1]
 - Spring Data
 - **Spring for Apache Hadoop - HDFS, YARN, Hive, Spark, HBase**
 - Spring Data Cassandra [2]
 - Spring Data GemFire (GemFire and Apache Geode) [3]
 - Spring Cloud
 - Spring Cloud Stream/Task - writing cloud native data microservices
 - Spring Cloud Data Flow - orchestrating data microservices

[1] [Spring For Apache Kafka - Wed Aug 3rd 11:30am](#)

[2] [Sneak Peek at Spring Data Cassandra - Wed Aug 3rd 3:20pm](#)

[3] [Spring Data and In-Memory Data Management in Action - Thu Aug 4th 9:00am](#)

Spring for Apache Hadoop

“*Spring for Apache Hadoop provides extensions to Spring, Spring Boot, Spring Batch, and Spring Integration to build manageable and robust pipeline solutions around Hadoop.*”

Consistent Programming Model

- Configure, and parameterize Hadoop connectivity and all job types
- Support for running MapReduce jobs, streaming, tool, jars
- Configure Hadoop's distributed cache
- Support for working with Hive, Pig, HBase, Sqoop2, Spark and MapReduce
- Writing to HDFS – partitioning, many data formats
- Support for YARN programming
- Relies on standard Spring Framework features
 - Configuration and property files
 - Environment profiles – easily move application from dev to qa to prod

Developer Productivity

- Create well-formed applications, not spaghetti script applications
- Simplify HDFS access:
 - FsShell API with support for JVM scripting
 - Powerful and flexible DataStoreWriter implementations
- Helper “Template” classes for Hive/Pig/HBase
- Runner classes for Hive/Pig/MapReduce for small workflows
- Tasklet implementations for larger Spring Batch flows
 - Hive, Pig, Spark, Sqoop2, MapReduce

Common Use Cases

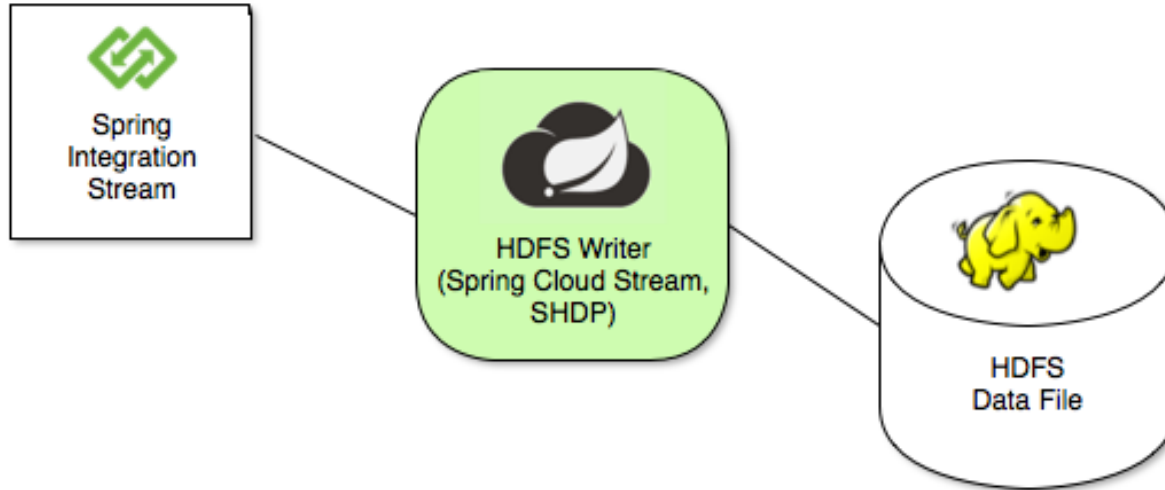
- Apply across a wide range of use cases
 - Ingestion: Events/JDBC/NoSQL/Files to HDFS
 - Orchestrate: Hadoop Jobs
 - Export: HDFS to JDBC/NoSQL
- Spring Integration and Spring Batch make this possible
- Spring Boot simplifies it
- Spring XD/Spring Cloud Data Flow makes it even easier

A unified model for different Hadoop distros

- Spring for Apache Hadoop provides several “flavors” to match dependencies with Hadoop distributions from:
 - Apache Hadoop
 - Cloudera CDH
 - Hortonworks HDP
 - Pivotal HD
- See Wiki page for details:
 - ➔ <https://github.com/spring-projects/spring-hadoop/wiki#supported-distributions>
 - ➔ <https://github.com/spring-projects/spring-hadoop/wiki#building-using-supported-distributions>

Data Ingestion

Demo #1 - Ingestion



The sink implementation

```
1 package com.springdeveloper.demo;~
2 ~
3 import java.io.IOException;~
4 ~
5 import org.springframework.beans.factory.annotation.Autowired;~
6 import org.springframework.cloud.stream.annotation.EnableBinding;~
7 import org.springframework.cloud.stream.messaging.Sink;~
8 import org.springframework.data.hadoop.store.DataStoreWriter;~
9 import org.springframework.integration.annotation.ServiceActivator;~
10 ~
11 @EnableBinding(Sink.class)~
12 public class IngestSink {~
13     ~
14     @Autowired~
15     private DataStoreWriter<String> writer;~
16     ~
17     private long counter = 0;~
18     ~
19     @ServiceActivator(inputChannel=Sink.INPUT)~
20     public void writeData(String payload) {~
21         System.out.println("*** PROCESSING ... " + ++counter);~
22         try {~
23             writer.write(payload);~
24         } catch (IOException e) {~
25             e.printStackTrace();~
26             throw new IllegalStateException("Unable to write to HDFS", e);~
27         }~
28     }~
29     ~
30 }
```


The configuration

```
32 @Configuration~
33 public class IngestConfiguration {~
34     ~
35     @Value("${app.basePath:/tmp/demo}")~
36     private String basePath;~
37     ~
38     @Value("${app.fileName:data}")~
39     private String fileName;~
40     ~
41     @Value("${app.fileExtension:dat}")~
42     private String fileExtension;~
43     ~
44     @Bean~
45     DataStoreWriter<String> dataStoreWriter(org.apache.hadoop.conf.Configuration hadoopConfiguration) {~
46         > TextFileWriter writer = new TextFileWriter(hadoopConfiguration, new Path(basePath), null);~
47         > ChainedFileNamingStrategy namingStrategy = new ChainedFileNamingStrategy(~
48             > > > Arrays.asList(new FileNamingStrategy[] {~
49                 > > > > new StaticFileNamingStrategy(fileName),~
50                 > > > > new UuidFileNamingStrategy(),~
51                 > > > > new StaticFileNamingStrategy(fileExtension, ".")});~
52         > writer.setFileNamingStrategy(namingStrategy);~
53         > return writer;~
54     }~
55 }
```

The POM

```
14 <parent>
15 > <groupId>org.springframework.boot</groupId>
16 > <artifactId>spring-boot-starter-parent</artifactId>
17 > <version>1.3.6.RELEASE</version>
18 > <relativePath> <!-- lookup parent from repository -->
19 </parent>
20
21 <properties>
22 > <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
23 > <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
24 > <java.version>1.8</java.version>
25 > <spring-data-hadoop.version>2.4.0.RELEASE</spring-data-hadoop.version>
26 > <stream-binder>kafka</stream-binder>
27 </properties>
28
29 <profiles>
30 > <profile>
31 > > <id>rabbit</id>
32 > > <properties>
33 > > > <stream-binder>rabbit</stream-binder>
34 > > </properties>
35 > </profile>
36 </profiles>
```

```
38 <dependencies>
39 > > <dependency>
40 > > > <groupId>org.springframework.cloud</groupId>
41 > > > <artifactId>spring-cloud-starter-stream-${stream-binder}</artifactId>
42 > > </dependency>
43
44 > > <dependency>
45 > > > <groupId>org.springframework.data</groupId>
46 > > > <artifactId>spring-data-hadoop-boot</artifactId>
47 > > > <version>${spring-data-hadoop.version}</version>
48 > > </dependency>
49 > > <dependency>
50 > > > <groupId>org.springframework.data</groupId>
51 > > > <artifactId>spring-data-hadoop-store</artifactId>
52 > > > <version>${spring-data-hadoop.version}</version>
53 > > </dependency>
54
55 > > <dependency>
56 > > > <groupId>org.springframework.cloud</groupId>
57 > > > <artifactId>spring-cloud-stream-test-support</artifactId>
58 > > > <scope>test</scope>
59 > > </dependency>
60 </dependencies>
61
62 <dependencyManagement>
63 > <dependencies>
64 > > <dependency>
65 > > > <groupId>org.springframework.cloud</groupId>
66 > > > <artifactId>spring-cloud-dependencies</artifactId>
67 > > > <version>Brixton.SR3</version>
68 > > > <type>pom</type>
69 > > > <scope>import</scope>
70 > > </dependency>
71 > </dependencies>
72 </dependencyManagement>
```

The Integration Tests

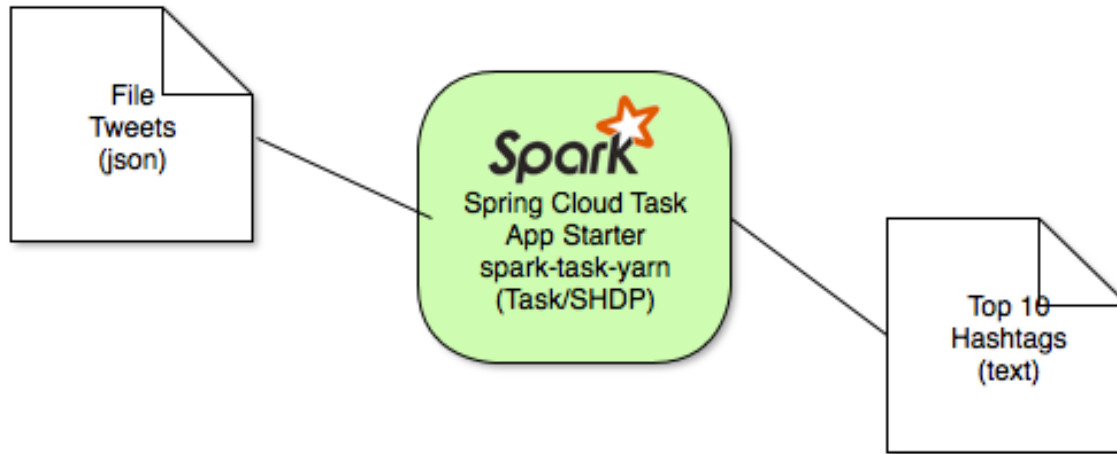
```
47 @RunWith(SpringJUnit4ClassRunner.class)~
48 @SpringApplicationConfiguration(classes = IngestSinkIntegrationTests.IngestApplication.class)~
49 @IntegrationTest({"spring.hadoop.fsUri=file:///",
50 > > "app.basePath=${java.io.tmpdir}/test"})~
51 public class IngestSinkIntegrationTests {~
52 ~
53 > @Autowired~
54 > ConfigurableApplicationContext applicationContext;~
55 ~
56 > @Value("${app.basePath}")~
57 > private String testDir;~
58 ~
59 > @Autowired~
60 > private FsShell fsShell;~
61 ~
62 > @Autowired~
63 > @Bindings(IngestSink.class)~
64 > private Sink sink;~
65 ~
66 > @Before~
67 > public void setup() {~
68 > > if (fsShell.test(testDir)) {~
69 > > > fsShell.rm(testDir);~
70 > > }~
71 > > fsShell.mkdir(testDir);~
72 > }~
73 ~
74 > @Test~
75 > public void testWritingSomething() throws IOException {~
76 > > sink.input().send(new GenericMessage<>("Foo"));~
77 > > sink.input().send(new GenericMessage<>("Bar"));~
78 > > sink.input().send(new GenericMessage<>("Baz"));~
79 > }
```

```
81 > @After~
82 > public void checkFilesClosedOK() throws IOException {~
83 > > applicationContext.close();~
84 > > File testOutput = new File(testDir);~
85 > > assertTrue(testOutput.exists());~
86 > > File[] files = testOutput.listFiles((dir, name) -> name.endsWith(".dat"));~
87 > > assertTrue(files.length > 0);~
88 > > File dataFile = files[0];~
89 > > assertNotNull(dataFile);~
90 > > Assert.assertThat(readFile(dataFile.getPath(), ~
91 > > > Charset.forName("UTF-8")), equalTo("Foo\nBar\nBaz\n"));~
92 > }~
93 ~
94 > private String readFile(String path, Charset encoding) throws IOException {~
95 > > byte[] encoded = Files.readAllBytes(Paths.get(path));~
96 > > return new String(encoded, encoding);~
97 > }
```

```
98 > @SpringBootApplication~
99 > static class IngestApplication {~
100 > > public static void main(String[] args) {~
101 > > > SpringApplication.run(IngestApplication.class, args);~
102 > > }~
103 > }~
104 }
```

Data Analysis

Demo #2 - Spark Task



The task implementation

```
17 package com.springdeveloper.demo;↵
18 ↵
19 import org.springframework.boot.SpringApplication;↵
20 import org.springframework.boot.autoconfigure.SpringBootApplication;↵
21 import org.springframework.cloud.task.app.spark.yarn.SparkYarnTaskConfiguration;↵
22 import org.springframework.context.annotation.Import;↵
23 ↵
24 @SpringBootApplication↵
25 @Import(SparkYarnTaskConfiguration.class)↵
26 public class SparkTaskApplication {↵
27 ↵
28     public static void main(String[] args) {↵
29         SpringApplication.run(SparkTaskApplication.class, args);↵
30     }↵
31 }↵
32 ↵
```

The task application properties

```
1  # Hadoop config-
2  spring.hadoop.fs-uri=hdfs://localhost:8020-
3  spring.hadoop.resource-manager-host=localhost-
4  spring.hadoop.resource-manager-port=8032-
5  spring.hadoop.job-history-address=localhost:10020-
6  -
7  # Spark application configuration-
8  # (override spark.app-args if needed)-
9  spark.app-name=hashtags-
10 spark.app-class=Hashtags-
11 spark.app-jar=hdfs:///app/spark/spark-hashtags_2.10-0.1.0.jar-
12 spark.assembly-jar=hdfs:///app/spark/spark-assembly-1.6.2-hadoop2.6.0.jar-
13 spark.app-args=/demo/input,/demo/testout-
14
```

The POM

```
21 > <properties>
22 > > <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
23 > > <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
24 > > <java.version>1.8</java.version>
25 > > <start-class>com.springdeveloper.demo.SparkTaskApplication</start-class>
26 > </properties>
27
28 > <dependencies>
29 > > <dependency>
30 > > > <groupId>org.springframework.cloud.task.app</groupId>
31 > > > <artifactId>spring-cloud-starter-task-spark-yarn</artifactId>
32 > > > </dependency>
33 > > <dependency>
34 > > > <groupId>org.springframework.boot</groupId>
35 > > > <artifactId>spring-boot-starter-test</artifactId>
36 > > > <scope>test</scope>
37 > > </dependency>
38 > </dependencies>
39
40 > <dependencyManagement>
41 > > <dependencies>
42 > > > <dependency>
43 > > > > <groupId>org.springframework.cloud.task.app</groupId>
44 > > > > <artifactId>spring-cloud-task-app-dependencies</artifactId>
45 > > > > <version>1.0.1.RELEASE</version>
46 > > > > <type>pom</type>
47 > > > > <scope>import</scope>
48 > > > </dependency>
49 > > </dependencies>
50 > </dependencyManagement>
```


The Starter

```
53 private class SparkAppYarnRunner implements CommandLineRunner {
54
55     private final Log logger = LoggerFactory.getLog(SparkAppYarnRunner.class);
56
57     @Autowired
58     private Configuration hadoopConfiguration;
59
60     @Autowired
61     private SparkYarnTaskProperties config;
62
63     @Override
64     public void run(String... args) throws Exception {
65         SparkConf sparkConf = new SparkConf();
66         sparkConf.set("spark.yarn.jar", config.getAssemblyJar());
67
68         List<String> submitArgs = new ArrayList<String>();
69         if (StringUtils.hasText(config.getAppName())) {
70             submitArgs.add("--name");
71             submitArgs.add(config.getAppName());
72         }
73         submitArgs.add("--jar");
74         submitArgs.add(config.getAppJar());
75         submitArgs.add("--class");
76         submitArgs.add(config.getAppClass());
77         if (StringUtils.hasText(config.getResourceFiles())) {
78             submitArgs.add("--files");
79             submitArgs.add(config.getResourceFiles());
80         }
81         if (StringUtils.hasText(config.getResourceArchives())) {
82             submitArgs.add("--archives");
83             submitArgs.add(config.getResourceArchives());
84         }
85         submitArgs.add("--executor-memory");
86         submitArgs.add(config.getExecutorMemory());
87         submitArgs.add("--num-executors");
88         submitArgs.add(" " + config.getNumExecutors());
89         for (String arg : config.getAppArgs()) {
90             submitArgs.add("--arg");
91             submitArgs.add(arg);
92         }
93         logger.info("Submit App with args: " + Arrays.asList(submitArgs));
94         ClientArguments clientArguments =
95             new ClientArguments(submitArgs.toArray(new String[submitArgs.size()]), sparkConf);
96         clientArguments.isClusterMode();
97         Client client = new Client(clientArguments, hadoopConfiguration, sparkConf);
98         System.setProperty("SPARK_YARN_MODE", "true");
99         try {
100             client.run();
101         } catch (Throwable t) {
102             logger.error("Spark Application failed: " + t.getMessage(), t);
103             throw new RuntimeException("Spark Application failed", t);
104         }
105     }
106 }
107 }
```

Improve the Spark Starters

- Provide for better extensibility
- protected methods for
 - creating the `SparkContext`
 - adding Spark properties and client submit args
 - configuring the `SparkClient`

See: <https://github.com/spring-cloud/spring-cloud-task-app-starters/issues/57>

Cloud

Using Hadoop in the Cloud

- Amazon Elastic MapReduce
- Microsoft Azure HDInsight
- Google Cloud - Dataproc
- IBM BigInsights
- Hortonworks/SequenceIQ Cloudbreak
- Cloudera on AWS / Cloudera Live
- Your own Docker image
- Your own AWS installation



Common issues with Hadoop Cloud Clusters

- Hadoop has a cluster centric view
 - easier to run apps from inside the cluster
 - you should have core-site.xml, yarn-site.xml etc accessible
 - some insights into internal configs might be necessary
- Spring for Apache Hadoop tries to work around this
 - creating its own Hadoop Configuration
 - pulling from environment and config properties
- Cloud clusters usually configured for internal network
 - hard/impossible to reach from outside - have to use proxies/tunnels
 - easiest to run your apps on the same network as the Hadoop cluster

Use Hadoop with Pivotal Cloud Foundry



- Deploy Hadoop separately on the same network as PCF
- Use a user-provided service:

```
cf create-user-provided-service hadoop  
  -p '{"fs":{"defaultFS":"hdfs://172.28.128.3:8020"},  
      "yarn":{"resourcemanager":{"host":"172.28.128.3","port":"8032"}}}'
```

- Refer to the VCAP_SERVICES env var values in Boot config file:

```
---  
spring:  
  profiles: cloud  
  hadoop:  
    fsUri: ${vcap.services.hadoop.credentials.fs.defaultFS}  
    resourceManagerHost: ${vcap.services.hadoop.credentials.yarn.resourcemanager.host}  
    resourceManagerPort: ${vcap.services.hadoop.credentials.yarn.resourcemanager.port}
```

Use Hadoop with Google Cloud Platform



- Create a Hadoop cluster on Dataproc:

```
gcloud dataproc clusters create hadoop-demo --master-boot-disk-size-gb 100 \
  --worker-boot-disk-size-gb 100 --zone us-central1-b
```

- Provide the setting in env var using SPRING_APPLICATION_JSON:

env:

```
- name: SPRING_APPLICATION_JSON
  value: '{"spring.hadoop.fs-uri":"hdfs://hadoop-demo-m:8020",
    "spring.hadoop.resource-manager-host": "hadoop-demo-m",
    "spring.hadoop.resource-manager-port": 8032,
    "spring.hadoop.job-history-address": "hadoop-demo-m:10020"}'
```

Spring Cloud Data Flow



- Supports what we have done so far to deploy apps from a GUI/Shell environment
- Allows you to register apps you need - from a curated list and also your own custom apps
- Supports multiple runtime environments CF, YARN, Kubernetes and Mesos plus “local” for testing/demos
- See the following presentations
 - [Data Microservices in the Cloud - Tue Aug 2nd 11:30am \(watch replay\)](#)
 - [Task Madness - Modern On Demand Processing - Tue Aug 2nd 2:40pm \(watch replay\)](#)
 - [Orchestrate All the Things! with Spring Cloud Data Flow - Thu Aug 4th 11:10am](#)

Spring Cloud Stream/Task - Some Big Data Apps

Source	Processor	Sink	Task
gemfire	bridge	cassandra	spark-client
jdbc	filter	counter	spark-yarn
jms	httpclient	gemfire	
mongodb	pmml	gpfdist	
s3	splitter	hdfs	
tcp	tcp-client	hdfs-dataset	
twitterstream	transform	jdbc	<i>(more coming)</i>

Take aways

- Run “Big Data” apps as cloud native microservices
 - locally with `java -jar` or using Docker
 - in the “cloud”
 - Kubernetes on Google Cloud
 - Cloud Foundry with PCF
- Same app can move un-changed from one environment to another
- We used Spring/Spring Boot passing of config override settings using `--spring.xxx`
- We also used Spring Cloud Stream for communication between apps
- In addition we utilized Spring Cloud Data Flow for orchestrating the source app for our ingest pipeline
- We used Spark for analysis running as a task with Spring Cloud Task

SpringOne Platform

Learn More. Stay Connected.

Demo Source & Slides: <https://github.com/trisberg/springone-2016>

Hadoop Install: <https://github.com/trisberg/hadoop-install>

Spring for Apache Hadoop Project: <http://projects.spring.io/spring-hadoop/>

Spring Cloud Data Flow Project: <http://cloud.spring.io/spring-cloud-dataflow/>

Questions: <http://stackoverflow.com/questions/tagged/spring-data-hadoop>

Twitter: @trisberg



@springcentral
spring.io/blog



@pivotal
pivotal.io/blog



@pivotalcf
<http://engineering.pivotal.io>