Spring XD

Introduction to Jobs

Introduction to Jobs

- Workflow
- Batch Jobs Features
- The Lifecycle of a Job in Spring XD
- Deployment manifest support for job
- Launching a job
- Retrieve job notifications
- Lab

Overview

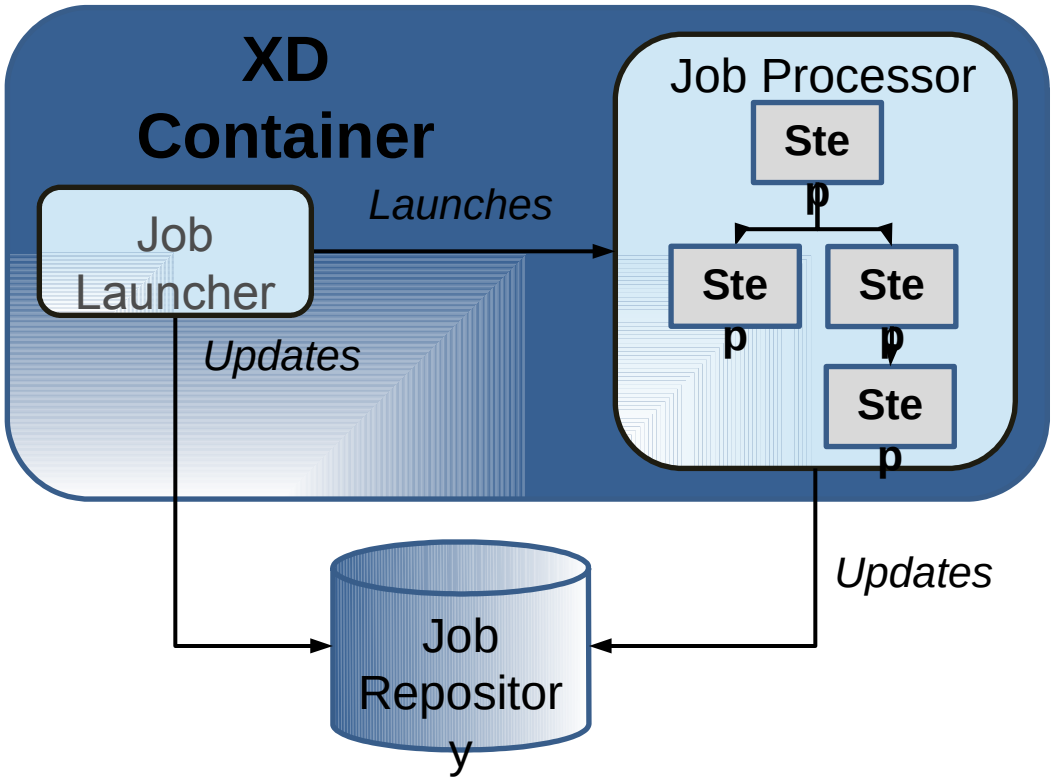
- Spring XD offers the ability to launch and monitor batch jobs based on Spring Batch
- Spring XD builds upon Spring Batch to simplify creating batch workflow solutions

See: Spring Batch Reference
http://docs.spring.io/spring-batch/reference/htmlsingle/

Batch Workflow

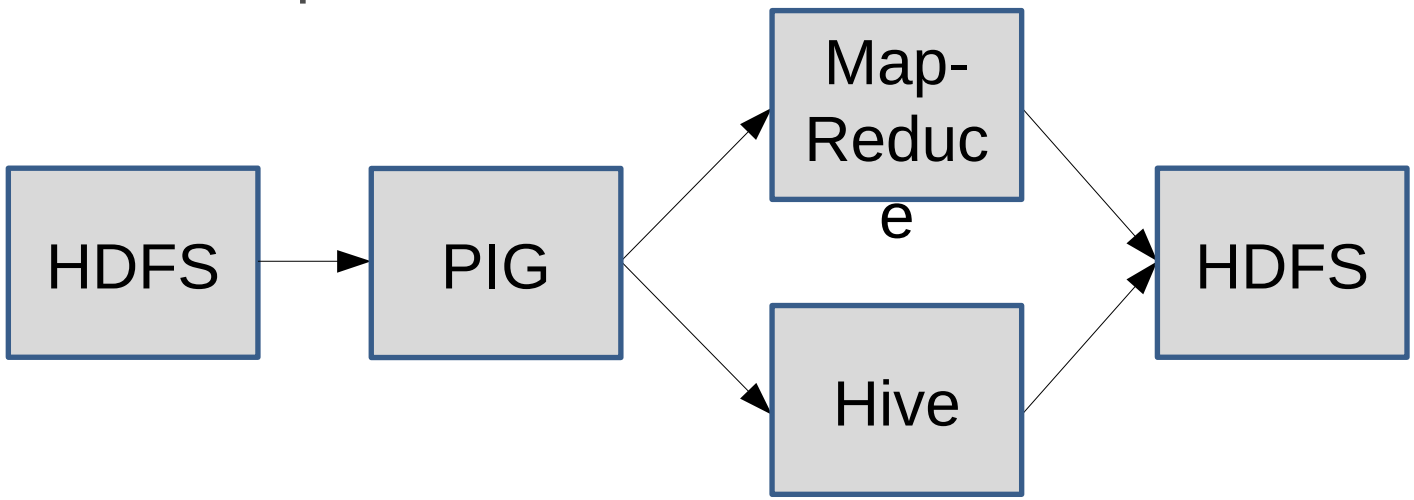
- The concept of a workflow translates to a *Job*
  - Not to be confused with a MapReduce job
- A *Job* is a directed graph, each node of the graph is a processing *Step*
- Steps can be executed sequentially or in parallel, depending on the configuration
- Jobs can be started, stopped, and restarted
- Restarting jobs is possible; the progress of executed steps in a Job is persisted in a database
  - via a JobRepository

Basic Components of a Workflow



Hadoop Specific Steps

- Steps specific to Hadoop in a workflow can be MapReduce jobs, executing Hive/Pig scripts or HDFS operations
- The example below illustrates a job with steps that read from HDFS, execute Pig, Map-Reduce, and Hive steps, and write output to HDFS



Batch Job Features

- Spring XD allows you to create and launch jobs
- Launching can be triggered using a cron expression or in reaction to data on a stream
- Spring XD provides some simple pre-defined Jobs:
  - Poll a Directory and import CSV files to HDFS
  - Import CSV files to JDBC
  - HDFS to JDBC Export
  - JDBC to HDFS Import
  - HDFS to MongoDB Export

The Lifecycle of a Job in Spring XD

- Register a Job Module
- Create a Job Definition
- Deploy a Job
- Launch a Job
- Job Execution
- Un-deploy a Job
- Destroy a Job Definition

Register a Job Module

- Necessary if not using one of the pre-packaged jobs
- Register a Job Module with the Module Registry by using the **`module upload`** command.

Create a Job Definition

- Create a Job Definition from a Job Module by providing a definition name as well as properties that apply to all Job Instances
- At this point the job is not deployed, yet

Deploy the Job

- Deploy the Job Definition to one or more Spring XD containers
- This will initialize the Job Definitions on those containers
- The jobs are now "live" (not launched)
  - Job can be launched by sending a message to a job queue that contains optional runtime Job Parameters

Launch a Job

- Launch a job by sending a message to the job queue with Job Parameters
- A *Job Instance* is created, representing a specific run of the Job
  - Job Instance = Job + runtime Job Parameters
- You can query for the Job Instances associated with a given job name

Job Execution

- A discrete attempt at executing a Job Instance is a *Job Execution*
- A Job Execution object captures the success or failure of the Job Instance
- You can query for Job Executions associated with a given job name

Un-deploy a Job

- This removes the job from the Spring XD container(s) preventing the launching of any new Job Instances
- For reporting purposes, you will still be able to view historic Job Executions associated with the the job

Destroy a Job Definition

- Destroying a Job Definition will
  - Un-deploy the Job (as needed)
  - Remove the Job Definition itself
- Note: The Job XML still exists in the modules/job directory.

Setting Deployment Properties for Jobs

- When deploying batch job you can specify deployment properties
- Deployment properties for jobs are the same as for streams, you can declare:
  - The number of job modules to deploy
  - The criteria expression to use for matching the job to available containers

```
job create --name myjob --definition
  "fooJob --makeUnique=false"

job deploy --name myjob --properties
"module.fooJob.criteria=groups.contains('hdfs-containers-
group'),module.fooJob.count=3"
```

Launching a job

- There are 3 ways to launch a batch job in Spring XD:
  - Ad-hoc
  - Use a named Cron-Trigger
  - As a sink from a stream

Ad-hoc

- Launch a job via command: **`job launch`**.
- Job will run to completion and finish.

```
xd:> job launch helloSpringXD
```

Launch the Batch using Cron-Trigger

- Launch a Job Instance based on a schedule
  - Create a stream with **`trigger --cron`** source:

```
stream create --name cronStream --definition "trigger
--cron='0/5 * * * * *'  > queue:job:myCronJob"
```

- Job Instance can receive parameters from a source (in this case a trigger) or process
- A trigger uses the **`--payload`** option to declare its payload

```
stream create --name cronStream --definition "trigger
--cron='0/5 * * * * *'
--payload={\"param1\":\"Clarence\"}
> queue:job:myCronJob"
```

Launch the Batch using a Fixed-Delay-Trigger

- A fixed-delay-trigger launches a Job on a regular interval
- **`--fixedDelay`**: seconds between executions
- Example: launch myXDJob instance every 5 seconds and pass a payload with a single parameter -

```
stream create --name fdStream --definition "
trigger –fixedDelay=5 --payload={\"param1\":\"holiday\"}
> queue:job:myXDJob"
```

Pause / Stop Scheduled Executions

- To pause/stop future scheduled jobs executions, undeploy the launching stream.
  - Job and stream definitions remain intact.
  - To re-activate, deploy the stream.

```
stream undeploy --name cronStream
```

Launch Job Instance as part of Stream

- A batch job can be used as a stream sink.
  - Can receive messages from sources and processors
- Example: launch job instance on HTTP POST
  - Pass the HTTP payload to the "myHttpJob"

```
stream create --name jobStream --definition
"http > queue:job:myHttpJob" --deploy
```

- Test:
  - Job instance launched with single parameter:

```
http post --target http://localhost:9000
  --data "{"param1":"workday"}"
```

Retrieve Job Notifications

- Spring XD captures notifications sent from executing jobs.
- When a batch job is deployed, by default it registers the following listeners along with pub/sub channels that these listeners send messages to.
  - Job Execution Listener
  - Step Execution Listener
  - Chunk Listener
  - Item Listener
  - Skip Listener

Retrieve Job Notifications

- Example: the job will send notifications to the log.

```
stream create --name jobNotifications --definition
"tap:job:myHttpJob.job >log"
```

Removing Batch Jobs

- Batch Jobs can be deleted by executing

```
xd:> job destroy --name helloSpringXD
```

- Alternatively, one can just undeploy the job, keeping its definition for a future redeployment:

```
xd:> job undeploy --name helloSpringXD
```

Pre-Packaged Batch Jobs

- Spring XD comes with several batch import and export modules
- Run them as-is or use them as a basis for building your own custom modules:
  - CSV Files to HDFS Import (filepollhdfs)
  - CSV Files to JDBC Import (filejdbc)
  - HDFS to JDBC Export (hdfsjdbc)
  - JDBC to HDFS Import (jdbchdfs)
  - HDFS to MongoDB Export (hdfsmongodb)
  - FTP to HDFS Export (ftphdfs)

Import Files to JDBC (filejdbc)

- Loads CSV files into a database table.
- Default settings:
  - Uses **`config/filejdbc.properties`** for configuration
  - Connects to internal HSQL DB used by Spring Batch
    - Probably not what you want!
- Example:

```
job create --name myjob --definition "filejdbc
--resources=/mycsvdir/*.csv
--names=forename,surname,address
--tableName=people"
```

See also: Spring XD Module Configuration
http://docs.spring.io/spring-xd/docs/current/reference/html/#_module_configuration

Import Files to HDFS (filepollhdfs)

- Import data from CSV file into HDFS
  - Expects a list of column names

```
job create --name myjob --definition
"filepollhdfs --names=forename,surname,address"
```

- Designed to be driven by a stream
  - Use file source to scan a directory for files and launch the job instance.
  - Separate job instance launched for each file found:

```
stream create --name csvStream --definition
"file --ref=true --dir=/mycsvdir --pattern=*.csv >
queue:job:myjob"
```

HDFS to JDBC Export (hdfsjdbc)

- Similar to filejdbc except soure files are from HDFS.
  - Similar syntax.

```
job create --name myjob --definition
"hdfsjdbc --resources=/data/*.log
--names=forename,surname,address --tableName=people"
```

- Limitation: database table must be created manually

HDFS to MongoDB Export (hdfsmongodb)

- Exports CSV data from HDFS and stores it in MongoDB
  - MongoDB collection name defaults to stream name. Overridden via **`--collectionName`** option.
- Configured using the file **`config/hdfsmongodb.properties`**

```
job create --name myjob --definition "hdfsmongodb
--resources=/data/*.log
--names=employeeId,forename,surname,address
--idField=employeeId
--collectionName=people"
```

Stream Attributes and Operations

- Source modules have MessageSourceMetrics attributes and operations available
- Processor, sink modules have MessageHandlerMetrics attributes and operations available
- Channels have MessageChannelMetrics attributes and operations available

Monitoring Jobs

- Similar naming convention as stream Mbeans
  - Domain name format is **`xd.<job name>`**
  - Object name is **`<module name>.<module index>`**
- REST API available

Monitoring Jobs

- Similar naming convention as stream Mbeans
  - Domain name format is **`xd.<job name>`**
  - Object name is **`<module name>.<module index>`**
- REST API available to:
  - Return job names (**`jobName`**)
  - Return configurations (**`job/configuration`**)
  - Return execution status (**`jobs/execution`**)
  - Stop or restart jobs
  - Return job steps (**`/jobs/executions/{jobExecutionId}/steps`**)
  - Return job step progress (**`/{stepExecutionId}/progress`**)

Lab