Spring XD

Lab Instructions

Data Ingestion Using Spring XD

spring Pivotal

Copyright Notice

This Page Intentionally Left Blank

Table of Contents

1. Installing Spring XD 1
1.1. Objective 1
1.2. Installation 1
1.2.1. Generic Installation 1
1.3. Verifying the Installation 1
1.3.1. Starting the Spring XD Server 1
1.3.2. Connecting to the Server 3
2. Introduction to Streams 4
2.1. Objective 4
2.2. A Simple Time Logger 4
2.3. Using Processors 4
2.4. Solutions 5
3. Customizing Streams 8
3.1. Objective 8
3.2. Installing a Custom Module 8
4. Advanced Streams 10
4.1. Objective 10
4.2. Transform Processor 10
4.3. TCP Source 11
4.4. Creating a Composite Module 12
5. Introduction to Jobs 14
5.1. Objective 14
5.2. Installing and Running a Basic Batch Job 14
6. Customizing Jobs 16
6.1. Objective 16
6.2. Creating a Custom Spring XD Job 16
7. Advanced Jobs 18
7.1. Objective 18
7.2. Setting Up Hadoop 18
7.3. Ingesting CSV Files into HDFS 20
8. Installing Spring XD in Distributed Mode 22
8.1. Objective 22
8.2. Installation 22
8.2.1. Configuring HSQLDB 22
8.2.2. Configuring Redis 23
8.2.3. Manual Installation of ZooKeeper 24

8.2.4. Spring XD Configuration ……24
8.2.5. Starting Spring XD ……26

Chapter 1. Installing Spring XD

1.1. Objective

To have a functioning version of Spring XD single-node instance installed and running on your workstation, and to be able to use the Spring XD shell to successfully connect to it.

1.2. Installation

For classroom purposes, you can run Spring XD in standalone mode on any platform that supports Java, such as Windows, Mac, Linux, etc.. For production instances, Pivotal recommends running on Linux-based operating systems. For Mac OSX, there is a Homebrew installation available, and Pivotal also provides a yum install for RedHat/CentOS. These latter repositories (Homebrew and Yum) may not contain the most recent release - the latest version can always be obtained from the Spring XD project website.

1.2.1. Generic Installation

Since Spring XD is written in Java, it can be installed on essentially any platform that supports a JVM. Also, performing a generic installation (as opposed to using Homebrew or Yum) will result in getting the latest version of the product running on your system.

1. Download the latest version of Spring XD from the Spring website, http://projects.spring.io/spring-xd/ or http://repo.spring.io/release/org/springframework/xd/spring-xd/1.1.0.RELEASE/spring-xd-1.1.0.RELEASE-dist.zip. All releases can be found at http://repo.spring.io/release/org/springframework/xd/spring-xd/, there you will find subdirectories containing various builds.
2. Extract the archive into the directory of your choice.
3. Set the `XD_HOME` environment variable to the `xd` subdirectory of the installation:

```
<installation-directory>/spring-xd-<version>.RELEASE/xd
```

1.3. Verifying the Installation

Now that you've completed the installation, let's verify that you can start the server and connect to it.

1.3.1. Starting the Spring XD Server

1. Start the server by running the `xd-singlenode` server startup script. On Windows you'll have to switch to

the `<installation-directory>/spring-xd-<version>.RELEASE/xd/bin` directory first or provide the full path to the `xd-singlenode.bat` batch file. On a Linux-based system you can simply execute `xd-singlenode`, assuming the script has execute permissions (chmod 777).

You should see the server start up, with the Spring XD ASCII-art logo output onto your console.

```
C:\apps\spring-xd\spring-xd-1.1.0.RELEASE\xd\bin>xd-singlenode

  _____                           __   _______
 /  ___|          (-)             \ \ / /  _  \
 \ `--. _ __  _ __ _ _ __   __ _   \ V /| | | |
  `--. \ '_ \| '__| | '_ \ / _` |  / ^ \| | | |
 /\__/ / |_) | |  | | | | | (_| | / / \ \ |/ /
 \____/| .__/|_|  |_|_| |_|\__, | \/   \/___/
       | |                   __/ |
       |_|                  |___/
1.1.0.RELEASE                      eXtreme Data


Started : SingleNodeApplication
Documentation: https://github.com/spring-projects/spring-xd/wiki

17:12:00,906 1.1.0.RELEASE  INFO main server.SingleNodeApplication - Starting SingleNodeApplication v1.1.0.RELEASE on dbuchk
3.RELEASE\xd\lib\spring-xd-dirt-1.1.0.RELEASE.jar started by dbuchko in C:\apps\spring-xd\spring-xd-1.1.0.RELEASE\xd\bin)
17:12:01,350 1.1.0.RELEASE  INFO main server.SingleNodeApplication - Started SingleNodeApplication in 0.934 seconds (JVM run
[2015-01-05 17:12:06.777] boot - 1596  INFO [NIOServerCxn.Factory:0.0.0.0/0.0.0.0:60284] --- ZooKeeperServer: Client attempt
[2015-01-05 17:12:06.777] boot - 1596  INFO [SyncThread:0] --- FileTxnLog: Creating new log file: log.1
[2015-01-05 17:12:06.837] boot - 1596  INFO [SyncThread:0] --- ZooKeeperServer: Established session 0x14abc27b5770000 with n
1
[2015-01-05 17:12:06.837] boot - 1596  INFO [main-SendThread(0:0:0:0:0:0:0:1:60284)] --- ClientCnxn: Session establishment c
 sessionid = 0x14abc27b5770000, negotiated timeout = 60000
[2015-01-05 17:12:06.837] boot - 1596  INFO [main-EventThread] --- ConnectionStateManager: State change: CONNECTED
17:12:12,427 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - XD Home: C:\apps\spring-xd\spring-xd-1.1.0.RELEASE\x
17:12:12,427 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - Transport: local
17:12:12,427 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - XD config location: file:C:\apps\spring-xd\spring-xd
17:12:12,427 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - XD config names: servers,application
17:12:12,427 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - XD module config location: file:C:\apps\spring-xd\sp
17:12:12,427 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - XD module config name: modules
17:12:12,437 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - Admin web UI: http://dbuchko1-e6420:9393/admin-ui
17:12:12,437 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - Zookeeper at: localhost:60284
17:12:12,437 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - Zookeeper namespace: xd
17:12:12,437 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - Analytics: memory
17:12:12,477 1.1.0.RELEASE  INFO LeaderSelector-0 server.DeploymentSupervisor - Leader Admin singlenode:default,admin,single
ment requests.
17:12:12,497 1.1.0.RELEASE  INFO main server.AdminServerApplication - Started AdminServerApplication in 5.73 seconds (JVM ru
[2015-01-05 17:12:12.527] boot - 1596  INFO [DeploymentSupervisorCacheListener-0] --- ContainerListener: Path cache event: t
17:12:14,690 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - XD Home: C:\apps\spring-xd\spring-xd-1.1.0.RELEASE\x
17:12:14,691 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - Transport: local
17:12:14,691 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - XD config location: file:C:\apps\spring-xd\spring-xd
17:12:14,692 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - XD config names: servers,application
17:12:14,692 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - XD module config location: file:C:\apps\spring-xd\sp
17:12:14,693 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - XD module config name: modules
17:12:14,693 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - Hadoop Distro: hadoop22
17:12:14,697 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - Hadoop version detected from classpath: 2.2.0
17:12:14,697 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - Zookeeper at: localhost:60284
17:12:14,698 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - Zookeeper namespace: xd
17:12:14,698 1.1.0.RELEASE  INFO main util.XdConfigLoggingInitializer - Analytics: memory
17:12:14,725 1.1.0.RELEASE  INFO main server.ContainerRegistrar - Container {groups=, host=dbuchko1-e6420, id=1698c0f3-7fc4-
d cluster
17:12:14,728 1.1.0.RELEASE  INFO main server.ContainerServerApplication - Started ContainerServerApplication in 0.829 second
17:12:14,730 1.1.0.RELEASE  INFO DeploymentSupervisorCacheListener-0 server.ContainerListener - Path cache event: path=/cont
ADDED
17:12:14,733 1.1.0.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Path cache event: type=INITIALIZ
17:12:14,740 1.1.0.RELEASE  INFO DeploymentSupervisorCacheListener-0 server.ContainerListener - Container arrived: Container
={groups=, host=dbuchko1-e6420, id=1698c0f3-7fc4-4230-8cf2-3f58d4417fb1, ip=10.74.17.227, pid=1596}}
```

Find the installation directory (based on where you installed it above; see class slides for background). Take a look in the `<installation-directory>/spring-xd-<version>.RELEASE/xd/logs` directory. You should find a `singlenode.log` log file there. If you don't, verify that you set the `XD_HOME` environment variable, and check the directory permissions. Examine the log file, noting the locations of the configuration files.

1.3.2. Connecting to the Server

1. Let's start up the Spring XD shell client now. Open a separate command prompt, and change to the `<installation-directory>/spring-xd-<version>.RELEASE/shell/bin` directory, and launch the `xd-shell` client. You should again see the Spring XD ASCII-art logo, and the server you connected to.

```
C:\apps\spring-xd\spring-xd-1.1.0.RELEASE\shell\bin>xd-shell
  _____                           __  _______
 /  ___|          (-)             \ \ / /  _  \
 \ `--. _ __  _ __ _ _ __   __ _   \ V /| | | |
  `--. \ '_ \| '__| | '_ \ / _` |  / ^ \| | | |
 /\__/ / |_) | |  | | | | | (_| | / / \ \ |/ /
 \____/| .__/|_|  |_|_| |_|\__, | \/   \/___/
       | |                   __/ |
       |_|                  |___/
eXtreme Data
1.1.0.RELEASE | Admin Server Target: http://localhost:9393
Welcome to the Spring XD shell. For assistance hit TAB or type "help".
xd:>
```

2. From within the XD command shell, examine the list of commands available by typing `help` at the prompt. Note that the shell offers tab-complete, so you can start to type a command and then hit <tab> to see a list of options. Try a couple simple commands, such as `admin config info`, `version`, and `stream list`. What port, by default, does the single-node instance listen on?

```
xd:>admin config info
  -------------  ------------------------------------------
  Result         Successfully targeted http://localhost:9393
  Target         http://localhost:9393
  Timezone used  Eastern Standard Time (UTC -5:00)
  -------------  ------------------------------------------

xd:>version
1.1.0.RELEASE
xd:>stream list
  Stream Name  Stream Definition  Status
  -----------  -----------------  ------

xd:>
```

3. Once you are finished, type `exit` to leave the Spring XD shell.

Chapter 2. Introduction to Streams

2.1. Objective

To explore creating, deploying, and running some simple streams in Spring XD.

2.2. A Simple Time Logger

Let's begin by creating a very simple time logger stream. You should already have a running (standalone) instance of Spring XD.

1. Open a Spring XD shell that is connected to your running server. If you get stuck at any time, remember that you can enter `help`, or hit <tab> in the shell for autocomplete.
2. Create a stream consisting of a time source connected directly to a log sink. Don't forget to give your stream a name. Verify the stream was created by running the `stream list` command. Note the status of the stream.
3. Make sure that you can see the Spring XD server output in the background, and go ahead and deploy your stream. You should observe the time logging in the server window. What is the default frequency of the time source?

   Run the `stream list` command again. What is the status of the stream now?
4. Undeploy and delete the stream by executing the `stream destroy` command. Once again, verify the stream is gone by checking the status.
5. Now, let's demonstrate how to pass in options to the the modules. Create a new stream with the time source and log sink, but this time change the interval from 1 second to 10 seconds. If you don't remember how to do this, go back to the slide notes to check, or check the sources documentation found at http://docs.spring.io/spring-xd/docs/current/reference/html/#time. Deploy and test your new stream, verifying that it logs every 10 seconds now. When you are finished you can destroy it.

2.3. Using Processors

Now that you are familiar with deploying a simple stream, and specifying options for the stream modules, let's add some processing to a stream. This time we will use an HTTP source that will receive a JSON order summary payload, and filter out orders with without a country code of "US". All orders with a "US" countryCode will get written to a file.

1. From the Spring XD shell, create and deploy a stream with an HTTP source, a filter Spring Expression Language (SpEL) processor, and a file sink. You will use the jsonPath SpEL method to extract the `countryCode` field, and discard any order payloads that do not have a country code of US.

The HTTP source should listen on port 9090.

To specify an SpEL on a filter processor, use the `--expression` option. The payload of the HTTP post is represented by the `payload` variable. So the format of the processor module you need to create will be:

```
filter --expression=#jsonPath(payload,'$.countryCode').equals('US')
```

Change the directory of the output file to a temporary location (eg. /tmp), and be sure to append orders to the file. Refer to the online documentation for the file sink found here: http://docs.spring.io/spring-xd/docs/current/reference/html/#file-sink

2. The Spring XD shell includes a utility that will post an HTTP payload to a URL. You can use the following example to post some sample orders to your stream for testing:

```
http post --target http://localhost:9090 --data "{\"id\":\"2773\",\"countryCode\":\"US\",\"orderAmt\":\"100\"}"
```

Post several order payloads, changing the order number and alternating between US and non-US country codes.

Examine the contents of the output file created in the temporary directory. What is the name of the file that was created? You should only see orders with US country codes in the file, if your stream is functioning correctly.

If you have time, feel free to experiment with some of the different options for the modules.

2.4. Solutions

1. Simple Time Logger - Spring XD shell input:

```
xd:>stream create --definition "time | log" --name timer
Created new stream 'timer'
xd:>stream list
  Stream Name  Stream Definition  Status
  -----------  -----------------  ----------
  timer        time | log         undeployed

xd:>stream deploy --name timer
Deployed stream 'timer'
xd:>stream list
  Stream Name  Stream Definition  Status
  -----------  -----------------  --------
  timer        time | log         deployed

xd:>stream undeploy --name timer
Un-deployed stream 'timer'
xd:>stream list
  Stream Name  Stream Definition  Status
```

```
  -----------  -----------------  -----------
  timer        time | log         undeployed

xd:>stream destroy timer
Destroyed stream 'timer'
```

```
xd:>stream create --definition "time --fixedDelay=10 | log" --name timer --deploy
Created and deployed new stream 'timer'
xd:>stream destroy timer
Destroyed stream 'timer'
xd:>
```

Simple Time Logger - Spring XD server log output:

```
16:20:30,866 1.0.3.RELEASE  INFO DeploymentSupervisorCacheListener-0 server.InitialDeploymentListener - Path cache event: pa
16:20:31,006 1.0.3.RELEASE  INFO Deployer server.StreamDeploymentListener - Deploying stream Stream{name='timer'}
16:20:31,032 1.0.3.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Path cache event: path=/deployme
16:20:31,032 1.0.3.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Deploying module 'log' for strea
16:20:31,111 1.0.3.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Deploying module [ModuleDescript
16:20:31,728 1.0.3.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Path cache event: path=/deployme
16:20:31,729 1.0.3.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Deploying module 'time' for stre
16:20:31,852 1.0.3.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Deploying module [ModuleDescript
16:20:32,282 1.0.3.RELEASE  INFO task-scheduler-2 sink.timer - 2015-01-21 16:20:32
16:20:32,293 1.0.3.RELEASE  INFO Deployer server.StreamDeploymentListener - Deployment status for stream 'timer': Deployment
16:20:32,296 1.0.3.RELEASE  INFO Deployer server.StreamDeploymentListener - Stream Stream{name='timer'} deployment attempt c
16:20:33,291 1.0.3.RELEASE  INFO task-scheduler-3 sink.timer - 2015-01-21 16:20:33
16:20:34,298 1.0.3.RELEASE  INFO task-scheduler-5 sink.timer - 2015-01-21 16:20:34
16:20:35,302 1.0.3.RELEASE  INFO task-scheduler-6 sink.timer - 2015-01-21 16:20:35
16:20:36,305 1.0.3.RELEASE  INFO task-scheduler-7 sink.timer - 2015-01-21 16:20:36
.
.
.
16:20:56,401 1.0.3.RELEASE  INFO task-scheduler-8 sink.timer - 2015-01-21 16:20:56
16:20:56,703 1.0.3.RELEASE  INFO main-EventThread server.DeploymentListener - Undeploying module [ModuleDescriptor@364fd7e2
16:20:56,722 1.0.3.RELEASE  INFO main-EventThread server.DeploymentListener - Undeploying module [ModuleDescriptor@5cdf3c40
16:20:56,742 1.0.3.RELEASE  INFO DeploymentSupervisorCacheListener-0 server.InitialDeploymentListener - Path cache event: pa
16:20:56,742 1.0.3.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Path cache event: path=/deployme
16:20:56,745 1.0.3.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Path cache event: path=/deployme
```

```
21:54:21,904 1.0.3.RELEASE  INFO DeploymentSupervisorCacheListener-0 server.InitialDeploymentListener - Path cache event: pa
21:54:22,285 1.0.3.RELEASE  INFO Deployer server.StreamDeploymentListener - Deploying stream Stream{name='timer'}
21:54:22,413 1.0.3.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Path cache event: path=/deployme
21:54:22,429 1.0.3.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Deploying module 'log' for strea
21:54:22,899 1.0.3.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Deploying module [ModuleDescript
21:54:23,943 1.0.3.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Path cache event: path=/deployme
21:54:23,943 1.0.3.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Deploying module 'time' for stre
21:54:24,109 1.0.3.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Deploying module [ModuleDescript
21:54:24,612 1.0.3.RELEASE  INFO Deployer server.StreamDeploymentListener - Deployment status for stream 'timer': Deployment
21:54:24,622 1.0.3.RELEASE  INFO Deployer server.StreamDeploymentListener - Stream Stream{name='timer'} deployment attempt c
21:54:24,628 1.0.3.RELEASE  INFO task-scheduler-3 sink.timer - 2015-01-21 21:54:24
21:54:34,634 1.0.3.RELEASE  INFO task-scheduler-3 sink.timer - 2015-01-21 21:54:34
21:54:44,638 1.0.3.RELEASE  INFO task-scheduler-3 sink.timer - 2015-01-21 21:54:44
21:54:54,682 1.0.3.RELEASE  INFO task-scheduler-2 sink.timer - 2015-01-21 21:54:54
21:55:04,697 1.0.3.RELEASE  INFO task-scheduler-2 sink.timer - 2015-01-21 21:55:04
21:55:05,524 1.0.3.RELEASE  INFO main-EventThread server.DeploymentListener - Undeploying module [ModuleDescriptor@6157824e
21:55:05,550 1.0.3.RELEASE  INFO main-EventThread server.DeploymentListener - Undeploying module [ModuleDescriptor@4fbaa936
21:55:05,571 1.0.3.RELEASE  INFO DeploymentSupervisorCacheListener-0 server.InitialDeploymentListener - Path cache event: pa
21:55:05,571 1.0.3.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Path cache event: path=/deployme
```

```
21:55:05,573 1.0.3.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Path cache event: path=/deployme
```

2. Using Processors - Spring XD shell input:

```
xd:>stream create --name orderFilter --definition "http --port=9090 |
filter --expression=#jsonPath(payload,'$.countryCode').equals('US') |
file --dir=/tmp --mode=APPEND" --deploy
Created and deployed new stream 'orderFilter'
xd:>http post --target http://localhost:9090 --data "{\"id\":\"2773\",\"countryCode\":\"US\",\"orderAmt\":\"100\"}"
> POST (text/plain;Charset=UTF-8) http://localhost:9090 {"id":"2773","countryCode":"US","orderAmt":"100"}
> 200 OK

xd:>http post --target http://localhost:9090 --data "{\"id\":\"2774\",\"countryCode\":\"CA\",\"orderAmt\":\"125\"}"
> POST (text/plain;Charset=UTF-8) http://localhost:9090 {"id":"2774","countryCode":"CA","orderAmt":"125"}
> 200 OK

xd:>http post --target http://localhost:9090 --data "{\"id\":\"2775\",\"countryCode\":\"US\",\"orderAmt\":\"110\"}"
> POST (text/plain;Charset=UTF-8) http://localhost:9090 {"id":"2775","countryCode":"US","orderAmt":"110"}
> 200 OK

xd:>
```

Using Processors - Output:

```
$ more /tmp/orderFilter.out
{"id":"2773","countryCode":"US"}
{"id":"2775","countryCode":"US"}
$
```

Chapter 3. Customizing Streams

3.1. Objective

To install a custom module in Spring XD, create a composite module in the Spring XD shell, and deploy a stream that uses it.

3.2. Installing a Custom Module

In this exercise, we will build and install a custom processor as a custom module into Spring XD.

1. Before installing the custom module, let's modify the location of the custom module registry. Create a subdirectory in your user home directory called `xd-custom-modules`. Open the Spring XD `servers.yml` file (located within XD-HOME/config), uncomment the `xd.customModule.home` section and set the property to the location of the new directory. For Example:

```
xd:
  customModule:
    home: file:///Users/glennrenfro/empty/xd-custom-modules
    -or-
    home: file://c:/project/customModules
```

   Restart the Spring-XD server to pick up the changes.
2. Now, we will build the custom processor. Change to the `payload-conversion` lab project directory, and execute

```
mvn clean package
```

   Verify the project built successfully by checking the `jar` file was created in the `target` subdirectory.
3. Launch the XD shell, and use the `module upload` command to upload the module JAR file into the custom repository.

```
xd:>module upload --name payload-conversion --type processor $LAB_HOME/
labs/payload-conversion/target/payload-conversion-1.0.0.BUILD-SNAPSHOT.jar
Successfully uploaded module 'processor:payload-conversion'
-or-
xd:>module upload --name payload-conversion --type processor c:/$LAB_HOME/
labs/payload-conversion/target/payload-conversion-1.0.0.BUILD-SNAPSHOT.jar
```

4. Verify the module was successfully uploaded by finding it in the output of the `module list`.
5. Take a look at the contents of the custom modules directory, and see that the jar file has been copied there.

6. Now use the XD shell to create a stream using the custom module:

```
xd:>stream create --name test --definition "http | payload-conversion --inputType=application/x-xd-tuple | file" --deploy
Created and deployed new stream 'test'
xd:>
```

7. Test the stream by posting some JSON content to the HTTP source:

```
xd:>http post --data {"symbol":"FAKE","price":75} --contentType "application/json"
```

   Check the output of the stream, but dumping the contents of the output file. On Mac OSX and Linux, enter:

```
xd:>! cat /tmp/xd/output/test.out
```

   On Mac Windows, enter:

```
xd:>! cat /temp/xd/output/test.out
```

   You should observe the following output of the file:

```
{"id":"719f5276-22d2-434d-87dc-39a23a978077","timestamp":1376387174881,"symbol":"FAKE","price":"75"}
```

   Note the tuple conversion has added `id` and `timestamp` fields. What has occurred is that the JSON has been converted to a Java `org.springframework.xd.tuple.DefaultTuple` object, then output as a String using the `toString()` method. When a new `DefaultTuple` object is instantiated, a unique ID (GUID) and timestamp is added to the object. On converting the object back to a `String`, those fields are included as part of the output.
8. Destroy the stream, then delete the custom module using the `module delete --name processor:tupleProcessor` command. Note how when executing this command the xd-shell expects you to prefix the module name with it's type.

   Open the `$LAB_HOME/payload-conversion/src/main/java/org/springframework/xd/samples/conversion/MyTupleProcessor.j` and examine the contents. Observe how the processor expects a tuple as input, and simply returns that tuple from the `process()` method. The file sink simply invokes the `toString()` method on the tuple returned from this method.

Chapter 4. Advanced Streams

4.1. Objective

To explore some of the advanced Spring XD stream capabilities.

4.2. Transform Processor

Let's begin by creating a stream to examine the use of the transform processor. First, we will examine using Spring Expression Language, then we will use a Groovy script.

1. Open a Spring XD shell that is connected to your running server. Create and deploy stream that uses an HTTP source, a transform processor that converts the payload to upper case, and a sink that logs the result to the console.

```
xd:>stream create --name myTransform --definition "http | transform --expression='payload.toUpperCase()' | log" --deploy
Created and deployed new stream 'myTransform'
xd:>
```

2. Test the stream by posting some data to it, verify the payload is transformed correctly.

```
xd:>http post --target "http://localhost:9000" --data "Hello World"
> POST (text/plain;Charset=UTF-8) http://localhost:9000 Hello World
> 200 OK

xd:>
```

   You should observe the following output in the Spring XD server console window:

```
14:48:48,928 1.1.0.RC1  INFO pool-11-thread-4 sink.myTransform - HELLO WORLD
```

3. Now lets's see how to accomplish the same task using a groovy script. Paste the following text into a script called `myUpperCase.groovy`

```
return payload.toUpperCase()
```

   Create a `scripts` subdirectory in your user home directory, and save the script there. (Recall that we could have placed the script into the `${xd.home}/modules/processor/scripts` directory, but remember that directory would disappear during an upgrade of Spring XD and we'd have to copy script into the new

installation.

4. Now, from the Spring XD shell, go ahead and create a new stream that uses the script. Don't forget to change the stream name and the HTTP port.

```
xd:>stream create --name myGroovyTransformer --definition "http --port=9001|
                                transform --script=file:/Users/dbuchko/myUpperCase.groovy | log" --deploy
Created and deployed new stream 'myGroovyTransformer'
xd:>
```

5. Post some data to the stream.

```
xd:>http post --target "http://localhost:9001" --data "Hello World"
> POST (text/plain;Charset=UTF-8) http://localhost:9001 Hello World
> 200 OK

xd:>
```

   Verify the payload was converted to upper case in the Spring XD server console log.

6. When you are finished, delete both streams from the Spring XD server.

4.3. TCP Source

Let's try creating a stream now using the TCP source. The TCP source comes with several decoders to frame messages.

1. First, create a basic stream consisting of a TCP source, that writes to the log sink.

```
xd:> stream create --name tcptest --definition "tcp --port=3000 --outputType=text/plain | log" --deploy
Created and deployed new stream 'tcptest'
xd:>
```

2. Open a new OS command shell (terminal window on OSX, command prompt on Windows, or SSH session in Linux). Launch the telnet utility from the shell, as follows:

```
$ telnet localhost 3000
Trying ::1...
Connected to localhost.
Escape character is '^]'.
```

   Note that on Windows, while you use the same command to launch telnet, your screen will clear and you will be left with a flashing cursor in the upper left corner of the screen.

   Type some characters, and press 'Enter'. Observe the output of the Spring XD server console - you should see the typed characters logged to the console, but only after you've hit 'Enter', which sends CRLF to the TCP source, and causes the buffered characters to be sent to the log sink.

When you have finished experimenting with the stream, type <CTRL-]>, then `close`. On OSX, this will result in the closing of the telnet session. On Windows, you will still have to enter `quit` to close the utility.

Note: if telnet is not installed on your windows machine enter the following from commandline: `pkgmgr /iu:"TelnetClient"` and hit return. (this should install the telnet client)

3. Destroy the stream once you are finished.

4.4. Creating a Composite Module

Composite modules can provide a level of re-usable components in Spring XD. This exercise will demonstrate how to create, register, and use a composite module.

1. Let's create a re-usable composite module, `errorLogger`, that will log streams containing the word 'error', regardless of case. Hint: use the filter processor.

   You should end up with a command similar to:

```
xd:>module compose errorLogger --definition "filter --expression=payload.toUpperCase().contains('ERROR') | log"
Successfully created module 'errorLogger' with type sink
xd:>
```

   Verify the module has been created successfully, by listing all the modules `module list`. Does this module appear as a source, sink, or processor? Is there any indication in the module list output that this is a composite module?

2. Now let's go ahead and demonstrate how to use this module. Create 2 streams, `s1` and `s2`, that receive HTTP posts on different ports and send them to the composite module.

```
xd:>stream create --name s1 --definition "http --port=9090 | errorLogger" --deploy
Created and deployed new stream 's1'
xd:>stream create --name s2 --definition "http --port=9095 | errorLogger" --deploy
Created and deployed new stream 's2'
xd:?
```

3. Test the composite module by posting some data to each of the ports the streams are listening on. Try different combinations, and observe which posts get logged.

```
xd:>http post --target http://localhost:9090 --data "Error"
> POST (text/plain;Charset=UTF-8) http://localhost:9090 Error
> 200 OK

xd:>http post --target http://localhost:9090 --data "Warning"
> POST (text/plain;Charset=UTF-8) http://localhost:9090 Warning
> 200 OK

xd:>http post --target http://localhost:9095 --data "Warning"
> POST (text/plain;Charset=UTF-8) http://localhost:9095 Warning
```

```
> 200 OK

xd:>http post --target http://localhost:9095 --data "error"
> POST (text/plain;Charset=UTF-8) http://localhost:9095 error
> 200 OK

xd:>
```

4. Remove the composite module from Spring XD using the `module delete` command. Did the command complete successfully? Note the results, and take appropriate action to delete the composite module.

Chapter 5. Introduction to Jobs

5.1. Objective

To deploy a simple batch process in Spring XD, without custom coding. This lab exercise is based on the `batch-basic` Github samples repository at https://github.com/spring-projects/spring-xd-samples.

5.2. Installing and Running a Basic Batch Job

This simple example demonstrate how a Spring XD job can read a csv file, replace the commas with spaces, and output the result into a new file.

1. Examine the contents of the `batch-basic/sample.txt` file - note it contains a comma-delimited line of text. Copy the file from the course labs installation directory, into your home directory `$HOME`.
2. Open the `batch-basic/basic_example.xml` file and find the `itemReader` bean. Change the value of the `resource` property to point to the `sample.txt` file that you just copied into your home directory.

```
...
	<bean id="itemReader" class="org.springframework.batch.item.file.FlatFileItemReader">
		<property name="resource" value="file:$HOME/sample.txt" />
		<property name="lineMapper" ref="lineMapper" />
	</bean>
...
```

   Find the `itemWriter` bean, and change the `resource` output file location to your home directory (leaving the filename the same).

```
...
	<bean id="itemWriter" class="org.springframework.batch.item.file.FlatFileItemWriter">
		<property name="resource" value="file:$HOME/sample1out.txt" />
		<property name="lineAggregator" ref="lineAggregator" />
	</bean>
...
```

3. Use the following procedure to upload the module. Note that if this were a JAR file, the `module upload` command could be used.
   a. Under the `XD_HOME` directory, create the `custom-modules/job/basic_example/config` subdirectories.
   b. Copy the `batch-basic/basic_example.xml` file from the course labs installation directory, to the `<XD_HOME>custom_modules/job/basic_example/config` directory.
4. From the Spring XD shell, run the `module list` command and verify the module appears in the list of Job modules.
5. From the XD shell, create a new Spring XD batch job using the `job create` command.

```
xd:>job create --name myBatchJob --definition "basic_example" --deploy
Successfully created and deployed job 'myBatchJob'
xd:>
```

6. Launch the job, using the `job launch` command.

```
xd:>job launch myBatchJob
Successfully submitted launch request for job 'myBatchJob'
xd:>
```

7. Verify the job completed successfully.

   Open the admin UI, navigate to the Job Executions page, and verify the status of the job is "COMPLETED". Click on the "Details" icon (magnifying glass) of your job and examine the information shown such as the duration and steps. Feel free to check out other areas such as the Modules, Definitions, and Deployments tabs.

   Navigate to your `$HOME` directory, and examine the contents of the `sample1out.txt` file. You should see that the commas have been replaced with spaces.

Chapter 6. Customizing Jobs

6.1. Objective

To create and install a custom Spring XD job.

6.2. Creating a Custom Spring XD Job

In order to create a custom Spring XD job, it is necessary to create a tasklet, create a custom context XML file, build and package it into a single jar file, then install it into the module registry. It is then available to be deployed as a new job.

1. Take a look at the `HelloSpringXDTasklet.java` source code in the `batch-simple` project. Note that it implements the Tasklet and StepExecutionListener interfaces. Also, examine how the job parameters are extracted for use. The code essentially just iterates over all the parameters passed in, and logs them to the console.

   Open the `spring-module.xml` context configuration file, found in the `src/main/resources/config` subdirectory. Note how the tasklet in the job steps reference the bean, which is an instance of the custom HelloSpringXDTasklet class.

   Change to the `springxd-batch-simple` lab project directory, and execute

```
mvn clean package
```

   Verify the project built successfully by checking the `jar` file was created in the `target` subdirectory.
2. Launch the XD shell, and use the `module upload` command to upload the module into the custom repository. Verify the module was successfully uploaded by finding it in the output of the `module list`. Also take a look at the contents of the custom modules directory, and see that the jar file has been copied there.
3. From the XD shell, create the job using the custom module as the definition. Verify by listing the jobs using the shell, or opening the administration UI web-page. Go ahead and deploy the job.
4. Launch the job, and specify some parameters for it. Any number of parameters can be added to the job - this would actually depend on the exact implementation of your tasklet. Experiment with different numbers and types of parameters. Specify a parameter "throwError", if it is set to "true" then the tasklet will succeed every 4th execution, otherwise it will throw an exception. Observe the output of the Spring XD server log.

```
xd:>job launch helloSpringXD --params '{"throwError":"true"}'
Successfully submitted launch request for job 'helloSpringXD'
```

```
xd:>
```

Chapter 7. Advanced Jobs

7.1. Objective

To configure Spring XD to connect to an Hadoop instance, and create, deploy, and run a simple job that ingests data into HDFS.

7.2. Setting Up Hadoop

This section will cover Hadoop configuration, and how to set up Spring XD to connect to your Hadoop instance.

1. This lab assumes you are have a copy of the Pivotal Hadoop single-node environment VM, which is available at https://network.gopivotal.com/products/pivotal-hd. Follow the instructions at https://github.com/spring-projects/spring-xd/wiki/Using-Hadoop-VMs-with-Spring-XD#pivotal-hd-single-node-vm, which involves editing the `/etc/hosts` file to bind the host name to the proper IP associated with the external facing NIC (as opposed to the loopback interface). You should also modify your client `/etc/hosts` file to resolve the `pivhdsne` hostname. Finally, if you cannot connect, make sure that you do not have any firewall rules blocking the connection to your hadoop server.

   You can also use the Apache Hadoop distribution, and configure it to run on a single node as per the instructions at http://hadoop.apache.org/docs/r2.5.1/hadoop-project-dist/hadoop-common/SingleCluster.html. There are also a few ready-to-use Hadoop VM's from other vendors that will work. Keep in mind if you do use a different distribution, you will have to provide the appropriate `--hadoopDistro` command line option in Spring XD.

   You will also need to create the Spring XD HDFS directory. This is typically done by executing the following commands on Hadoop, after you've started up the server:

```
sudo -u hdfs hdfs dfs -mkdir /xd
sudo -u hdfs hdfs dfs -chmod 777 /xd
```

   It will also be necessary to enable append in HDFS, to run some of the built-in Spring XD jobs. This is typically done by updating the Hadoop `hdfs-site.xml` file with the following settings:

```
    <property>
        <name>dfs.support.append</name>
        <value>true</value>
    </property>
```

Again, depending on the particular version of Hadoop you are using, these instructions may differ slightly. Refer to the following pages for additional help if necessary:

- https://github.com/spring-projects/spring-xd/wiki/Using-Hadoop-VMs-with-Spring-XD
- https://github.com/spring-projects/spring-xd/wiki/Hadoop-Installation

2. Open the `servers.yml` configuration file in a text editor, and find the "Hadoop properties" section. Set the host name and port of the `spring.hadoop.fsUrl` property to the namenode of your Hadoop distribution. For Pivotal HD, the entry is as follows:

```
spring:
  hadoop:
   fsUri: hdfs://pivhdsne:8020
```

3. Start the Spring XD single-node instance. This time, however, add the appropriate `--hadoopDistro` command line option. If using Pivotal Hadoop, the command line will be as follows:

```
$ bin/xd-singlenode --hadoopDistro phd21
```

You should see a corresponding entry in the Spring XD startup log showing the Hadoop version.

```
11:49:59,026 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - Hadoop Distro: phd21
11:49:59,029 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - Hadoop version detected from classpath: 2.2.0-gphd-3
```

4. Once your Spring XD server is up and running, let's go ahead and get the shell ready to create some HDFS streams. Start by launching the shell with the same `--hadoopDistro` command line option you used to start the server. Don't forget to adjust the parameter accordingly for the Hadoop distribution you are using.

```
$ bin/xd-shell --hadoopDistro phd21
```

5. Before the shell can work with Hadoop, we have to tell it where to find the Hadoop namenode. We do that be executing the following command:

```
xd:>hadoop config fs --namenode hdfs://pivhdsne:8020
```

Verify your connection by executing the `hadoop fs ls /` command. You should see the root HDFS directory listing. Verify that the `xd` subdirectory exists and is writeable. Note how it is possible to execute Hadoop commands from the Spring XD shell by prefixing them with the `hadoop` keyword.

```
xd:>hadoop fs ls /
Hadoop configuration changed, re-initializing shell...
Found 8 items
drwxr-xr-x   - hdfs    hadoop          0 2014-08-23 20:54 /apps
```

```
drwxr-xr-x   - gpadmin hadoop          0 2014-08-23 21:02 /hawq_data
drwxr-xr-x   - hdfs    hadoop          0 2014-08-23 20:56 /hive
drwxr-xr-x   - mapred  hadoop          0 2014-08-23 20:55 /mapred
drwxrwxrwx   - hdfs    hadoop          0 2014-08-23 20:55 /tmp
drwxrwxrwx   - hdfs    hadoop          0 2014-08-23 21:10 /user
drwxrwxrwx   - hdfs    hadoop          0 2015-01-22 12:03 /xd
drwxr-xr-x   - hdfs    hadoop          0 2014-08-23 20:56 /yarn
xd:>
```

At this point, you have a functioning Hadoop instance you can connect to from Spring XD. Let's now see how we can ingest some data into it.

7.3. Ingesting CSV Files into HDFS

Now you will learn how to use the pre-defined `filepollhdfs` job that copies CSV files into HDFS. The job included is triggered by a stream, and does a straight file copy. Later we will see how to customize it to make it more realistic for actual use cases.

1. First, create and save the CSV files that you want to import. You can use the following contents, or create your own. (Just pay close attention to the field names in the file that you use.)

   `orders-20150107.csv`

```
orderid,customerid,orderitemcount,ordertotal
4434,84661,4,123.88
4435,11465,2,75.33
4436,88424,10,422.85
```

   `orders-20150108.csv`

```
orderid,customerid,orderitemcount,ordertotal
4782,77546,2,69.48
4783,34464,7,310.22
4784,51114,1,22.87
4785,15746,4,148.55
```

   `orders-20150109.csv`

```
orderid,customerid,orderitemcount,ordertotal
4933,24841,5,105.55
4934,88464,2,40.21
4935,84874,4,77.97
4936,94414,6,118.22
```

2. The pre-defined `filepollhdfs` job requires you explicitly specify the names of fields in your CSV file, as it

generates tuples that can be used later in the job processing steps. The option required is `--names`, followed by a comma-delimited list of fields.

If you are sharing an HDFS instance with other students, each of you must use a unique job name in order to not conflict with each other. Spring XD uses the job name as the subdirectory name in HDFS, so each student will have their own subdirectory under `/xd` that contains their copied CSV files.

Go ahead and create and deploy your job from the XD shell.

```
xd:>job create orders --definition "filepollhdfs --names=orderid,customerid,orderitemcount,ordertotal" --deploy
Successfully created and deployed job 'ordersjob1'
```

3. Now you need something to trigger the job, so lets go ahead and create a stream to do this.

   The Spring XD Domain Specific Language (DSL) allows you to specify a job to act as a sink for a stream definition. So you need to create a stream that uses the file source, and sends the name of the file to the input queue that triggers the job we just created. The directory and indicator to send just the file reference (and not the contents itself) must be specified as an option to the file source using the `--dir` and `--ref` directives. In addition, you can also specify the pattern (`--pattern`) so that only CSV files are picked up. Don't forget to deploy the job.

```
xd:>stream create ordersJobTrigger --definition "file --ref=true --dir=/tmp/orders --pattern=*.csv > queue:job:ordersjob1" -
Created and deployed new stream 'ordersJobTrigger'
```

   Take a look in the Admin UI and see that your stream and job have been deployed.
4. Trigger the job execution by copying one of CSV files into the source directory you specified when you created the stream. Go to the "Executions" section of the "Jobs" tab in the Admin UI, you should see that your job has executed successfully. From the XD shell, use the `job execution list` command to observe the same results. Use the `hadoop fs ls` command to see that the file was successfully copied to HDFS.

```
xd:>hadoop fs ls /xd/ordersjob1
Found 1 items
-rw-r--r--   3 student hadoop        105 2015-01-26 14:33 /xd/ordersjob1/ordersjob1-0.csv
xd:>
```

5. Go ahead and copy the other 2 files into the source directory, and examine the results in the Admin UI. Try submitting a badly formatted file, and check the results. See if you can find the stack trace in the UI.

That's it, you've created your first Spring XD batch job!

Chapter 8. Installing Spring XD in Distributed Mode

8.1. Objective

To have a functioning version of Spring XD installed and running in distributed mode on your workstation.

8.2. Installation

As noted previously, Spring XD consists of several components that satisfy different functions. The standalone configuration runs all components in a single container. In this lab exercise, you will separate the components and run them independently in distributed mode, which is how you would run Spring XD in a production environment.

The following components will be configured to run independently:

- HSQLDB
- Redis
- ZooKeeper
- Spring XD Admin Server
- Spring XD Container Server

Before starting, make a backup copy of the `$XD_HOME/config/servers.yml` file.

8.2.1. Configuring HSQLDB

Spring XD supports any JDBC-compliant SQL database, but for demonstration purposes we will use the standalone HSQLDB instance included with the distribution.

Change to the `hsqldb` directory in the Spring XD installation, and start the HSQLDB server by executing `bin/hsqldb-server`.

```
Daniels-Mac:hsqldb dbuchko$ bin/hsqldb-server
[2015-01-08 15:26:52.003] boot - 62533  INFO [main] --- HsqlServerApplication: Starting HsqlServerApplication v1.0.3.RELEASE
[2015-01-08 15:26:52.611] boot - 62533  INFO [main] --- AnnotationConfigApplicationContext: Refreshing org.springframework.c
[2015-01-08 15:26:54.677] boot - 62533  INFO [main] --- XmlBeanDefinitionReader: Loading XML bean definitions from class pat
[2015-01-08 15:26:56.269] boot - 62533  INFO [main] --- PropertiesFactoryBean: Loading properties file from URL [jar:file:/U
[2015-01-08 15:26:56.286] boot - 62533  INFO [main] --- IntegrationRegistrar: No bean named 'integrationHeaderChannelRegistr
[2015-01-08 15:26:56.856] boot - 62533  INFO [main] --- DefaultConfiguringBeanFactoryPostProcessor: No bean named 'errorChan
...
[2015-01-08 15:27:09.561] boot - 62533  INFO [main] --- HsqlServerApplication: Started HsqlServerApplication in 3.221 second
```

8.2.2. Configuring Redis

Spring XD comes with the Redis source code, and an installation script that compiles the code into a binary executable. Note that if you are completing these lab exercises on Windows, you should download and install the pre-built version of Redis for Windows that can be found at https://github.com/MSOpenTech/redis/releases. After installing, proceed to starting the Redis server.

1. Change to the `redis/bin` subdirectory under the Spring XD installation, and execute the `install-redis` script to extract and build the Redis server.

```
Daniels-Mac:bin dbuchko$ ./install-redis
mac64 platform detected.
cd src && make all
rm -rf redis-server redis-sentinel redis-cli redis-benchmark redis-check-dump redis-check-aof *.o *.gcda *.gcno *.gcov redis
(cd ../deps && make distclean)
(cd hiredis && make clean) > /dev/null || true
(cd linenoise && make clean) > /dev/null || true
(cd lua && make clean) > /dev/null || true
(cd jemalloc && [ -f Makefile ] && make distclean) > /dev/null || true
...
Hint: To run 'make test' is a good idea ;)

installation script completed.
Selecting a non-default memory allocator when building Redis is done by setting
the MALLOC environment variable. Please refer to /Users/dbuchko/Applications/spring-xd-1.0.3.RELEASE/redis/redis-2.8.3/READM
To start redis server, please run: /Users/dbuchko/Applications/spring-xd-1.0.3.RELEASE/redis/bin/redis-server
Daniels-Mac:bin dbuchko$
```

   Validate the build by changing to the Redis directory, and running the `make test` target. Look for a message at the end of the tests stating `All tests passed without errors!`.

```
Daniels-Mac:redis-2.8.3 dbuchko$ make test
cd src && make test
Cleanup: may take some time... OK
Starting test server at port 11111
[ready]: 63626
...
\o/ All tests passed without errors!

Cleanup: may take some time... OK
Daniels-Mac:redis-2.8.3 dbuchko$
```

2. From the root directory of the Redis installation, start the server by executing the `../bin/redis-server redis.conf` binary. You should observe the server start up, with the ASCII-art appearing and indicating the server is ready to accept connections.

```
Daniels-Mac:redis-2.8.3 dbuchko$ ../bin/redis-server redis.conf
[65055] 08 Jan 17:38:47.901 * Max number of open files set to 10032
                _._
           _.-``__ ''-._
      _.-``    `.  `_.  ''-._           Redis 2.8.3 (00000000/0) 64 bit
  .-`` .-```.  ```\/    _.,_ ''-._
 (    '      ,       .-`  | `,    )     Running in stand alone mode
```

```
 |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6379
 |    `-._   `._    /     _.-'    |     PID: 65055
  `-._    `-._  `-./  _.-'    _.-'
 |`-._`-._    `-.__.-'    _.-'_.-'|
 |    `-._`-._        _.-'_.-'    |           http://redis.io
  `-._    `-._`-.__.-'_.-'    _.-'
 |`-._`-._    `-.__.-'    _.-'_.-'|
 |    `-._`-._        _.-'_.-'    |
  `-._    `-._`-.__.-'_.-'    _.-'
      `-._    `-.__.-'    _.-'
          `-._        _.-'
              `-.__.-'

[65055] 08 Jan 17:38:47.903 # Server started, Redis version 2.8.3
[65055] 08 Jan 17:38:47.903 * The server is now ready to accept connections on port 6379
```

8.2.3. Manual Installation of ZooKeeper

1. Download a copy of ZooKeeper from the links on this page: http://zookeeper.apache.org/releases.html. Extract the archive.
2. In the ZooKeeper `bin` directory, you will find several shell scripts for starting the server and command-line client. Spring XD contains a sample ZooKeeper configuration file located in the `zookeeper/conf/zoo.cfg` that is required for starting the server. To start the server (on OSX or Linux): `bin/zkServer.sh start-foreground ../spring-xd-1.0.3.RELEASE/zookeeper/conf/zoo.cfg`. (You can also start the server as a background process by passing in `start` instead of `start-foregound`.)

```
Daniels-Mac:zookeeper-3.4.6 dbuchko$ bin/zkServer.sh start-foreground ../spring-xd-1.0.3.RELEASE/zookeeper/conf/zoo.cfg
JMX enabled by default
Using config: ../spring-xd-1.0.3.RELEASE/zookeeper/conf/zoo.cfg
2015-01-09 14:56:35,778 [myid:] - INFO  [main:QuorumPeerConfig@103] - Reading configuration from: ../spring-xd-1.0.3.RELEASE
2015-01-09 14:56:35,784 [myid:] - INFO  [main:DatadirCleanupManager@78] - autopurge.snapRetainCount set to 3
2015-01-09 14:56:35,784 [myid:] - INFO  [main:DatadirCleanupManager@79] - autopurge.purgeInterval set to 0
2015-01-09 14:56:35,784 [myid:] - INFO  [main:DatadirCleanupManager@101] - Purge task is not scheduled.
2015-01-09 14:56:35,784 [myid:] - WARN  [main:QuorumPeerMain@113] - Either no config or no quorum defined in config, running
2015-01-09 14:56:35,799 [myid:] - INFO  [main:QuorumPeerConfig@103] - Reading configuration from: ../spring-xd-1.0.3.RELEASE
2015-01-09 14:56:35,799 [myid:] - INFO  [main:ZooKeeperServerMain@95] - Starting server
...
2015-01-09 16:44:18,531 [myid:] - INFO  [main:ZooKeeperServer@764] - minSessionTimeout set to -1
2015-01-09 16:44:18,531 [myid:] - INFO  [main:ZooKeeperServer@773] - maxSessionTimeout set to -1
2015-01-09 16:44:18,546 [myid:] - INFO  [main:NIOServerCnxnFactory@94] - binding to port 0.0.0.0/0.0.0.0:2181
```

8.2.4. Spring XD Configuration

Now that the external components are installed and running, it is necessary to configure Spring XD to connect and use them. Make a backup copy of and open the `$XD_HOME/config/servers.yml` in your favourite editor.

1. First, change the transport from Rabbit to Redis by uncommenting and modifying the xd.transport parameter as shown:

```
#XD data transport (default is redis for distributed, local for single node)
xd:
  transport: redis
```

2. Configure the HSQLDB connection parameters, in this case the included default values will suffice - it is necessary to just uncomment the parameters.

```
#Config for use with HSQLDB
#
#Change the database host, port and name
hsql:
  server:
    host: localhost
    port: 9101
    dbname: xdjob
#Change database username and password
spring:
  datasource:
    url: jdbc:hsqldb:hsql://${hsql.server.host:localhost}:${hsql.server.port:9101}/${hsql.server.dbname:xdjob}
    username: sa
    password:
    driverClassName: org.hsqldb.jdbc.JDBCDriver
    validationQuery: select 1 from INFORMATION_SCHEMA.SYSTEM_USERS
```

3. Let's also enable the ability to shut down containers from the Admin UI. Follow the instructions in the configuration file, and uncomment the parameters.

```
# Config to enable the shutdown button on the UI

#---
spring:
  profiles: container
management:
  port: 0
```

4. Setup the Redis connection parameters, using the default host and port.

```
# Redis properties
spring:
  redis:
   port: 6379
   host: localhost
```

5. Finally, setup the ZooKeeper connection properties.

```
# Zookeeper properties
# namespace is the path under the root where XD's top level nodes will be created
# client connect string: host1:port1,host2:port2,...,hostN:portN
zk:
  namespace: xd
  client:
     connect: localhost:2181
```

Save the configuration file.

8.2.5. Starting Spring XD

Now that you've modified the configuration, you can start up the Spring XD Admin and Container servers.

1. Open a new shell, and start the Admin server first by executing the `$XD_HOME/bin/xd-admin` shell script.

```
aniels-Mac:xd dbuchko$ bin/xd-admin

  _____                           __  _______
 /  ___|          (-)            \ \ / /  _  \
 \ `--. _ __  _ __ _ _ __   __ _   \ V /| | | |
  `--. \ '_ \| '__| | '_ \ / _` |  / ^ \| | | |
 /\__/ / |_) | |  | | | | | (_| | / / \ \ |/ /
 \____/| .__/|_|  |_|_| |_|\__, | \/   \/___/
       | |                  __/ |
       |_|                 |___/
1.0.3.RELEASE                    eXtreme Data


Started : AdminServerApplication
Documentation: https://github.com/spring-projects/spring-xd/wiki

17:36:07,038 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - XD Home: /Users/dbuchko/Applications/spring-xd-1.0.3
17:36:07,039 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - Transport: redis
17:36:07,039 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - XD config location: file:/Users/dbuchko/Applications
17:36:07,039 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - XD config names: servers,application
17:36:07,040 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - XD module config location: file:/Users/dbuchko/Appli
17:36:07,041 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - XD module config name: modules
17:36:07,041 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - Admin web UI: http://Daniels-Mac.local:9393/admin-ui
17:36:07,042 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - Zookeeper at: localhost:2181
17:36:07,042 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - Zookeeper namespace: xd
17:36:07,042 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - Analytics: redis
17:36:07,207 1.0.3.RELEASE  INFO main server.AdminServerApplication - Started AdminServerApplication in 13.173 seconds (JVM
17:36:07,207 1.0.3.RELEASE  INFO LeaderSelector-0 server.DeploymentSupervisor - Leader Admin 192.168.73.128:9393 is watching
17:36:07,347 1.0.3.RELEASE  INFO DeploymentSupervisorCacheListener-0 server.ContainerListener - Path cache event: type=INITI
```

   Open your browser and verify you can navigate to the Spring XD admin page, at `http://localhost:9393/admin-ui`.
2. Now start up the container server in another shell, by executing `$XD_HOME/bin/xd-container`.

```
Daniels-Mac:xd dbuchko$ bin/xd-container

  _____                           __  _______
 /  ___|          (-)            \ \ / /  _  \
 \ `--. _ __  _ __ _ _ __   __ _   \ V /| | | |
  `--. \ '_ \| '__| | '_ \ / _` |  / ^ \| | | |
 /\__/ / |_) | |  | | | | | (_| | / / \ \ |/ /
 \____/| .__/|_|  |_|_| |_|\__, | \/   \/___/
       | |                  __/ |
       |_|                 |___/
1.0.3.RELEASE                    eXtreme Data


Started : ContainerServerApplication
Documentation: https://github.com/spring-projects/spring-xd/wiki
```

```
17:41:48,516 1.0.3.RELEASE  INFO main server.ContainerServerApplication - Starting ContainerServerApplication v1.0.3.RELEASE
17:41:49,549 1.0.3.RELEASE  INFO main server.ContainerServerApplication - Started ContainerServerApplication in 2.773 second
[2015-01-09 17:42:05.165] boot - 66411  INFO [main-SendThread(localhost:2181)] --- ClientCnxn: Opening socket connection to
[2015-01-09 17:42:05.209] boot - 66411  INFO [main-SendThread(localhost:2181)] --- ClientCnxn: Socket connection established
17:42:15,773 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - XD Home: /Users/dbuchko/Applications/spring-xd-1.0.3
17:42:15,807 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - Transport: redis
17:42:15,815 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - XD config location: file:/Users/dbuchko/Applications
17:42:15,816 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - XD config names: servers,application
17:42:15,816 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - XD module config location: file:/Users/dbuchko/Appli
17:42:15,817 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - XD module config name: modules
17:42:15,817 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - Hadoop Distro: hadoop22
17:42:15,868 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - Hadoop version detected from classpath: 2.2.0
17:42:15,868 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - Zookeeper at: localhost:2181
17:42:15,870 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - Zookeeper namespace: xd
17:42:15,871 1.0.3.RELEASE  INFO main util.XdConfigLoggingInitializer - Analytics: redis
17:42:15,999 1.0.3.RELEASE  INFO DeploymentsPathChildrenCache-0 server.DeploymentListener - Path cache event: type=INITIALIZ
17:42:16,004 1.0.3.RELEASE  INFO main server.ContainerRegistrar - Container {groups=, host=Daniels-Mac.local, id=0ecbb231-f1
17:42:16,562 1.0.3.RELEASE  INFO main server.ContainerServerApplication - Started ContainerServerApplication in 7.507 second
```

Navigate to the Containers page on the Admin UI, and verify the container appears up and running in the list. You'll also note that the "Shutdown" button is now enabled. Try it, and observe that your container shuts down in the shell. Restart it.

3. For fun, open another shell window and start another container server. Verify the additional container is running in the Admin UI. Deploy a simple stream, and observe in the UI how the source and the sink get deployed to different containers.

   Note if you run into issues, check the log files in the `$XD_HOME/logs` directory.
4. Once you've verified your installation is running, shut down all the components.