Spring XD

Advanced Streams

Advanced Streams

- **Monitoring**
- Hadoop (HDFS)
- Transform/Script processors
- TCP/UDP sources
- Composite Modules
- Lab

Monitoring Deployed Streams

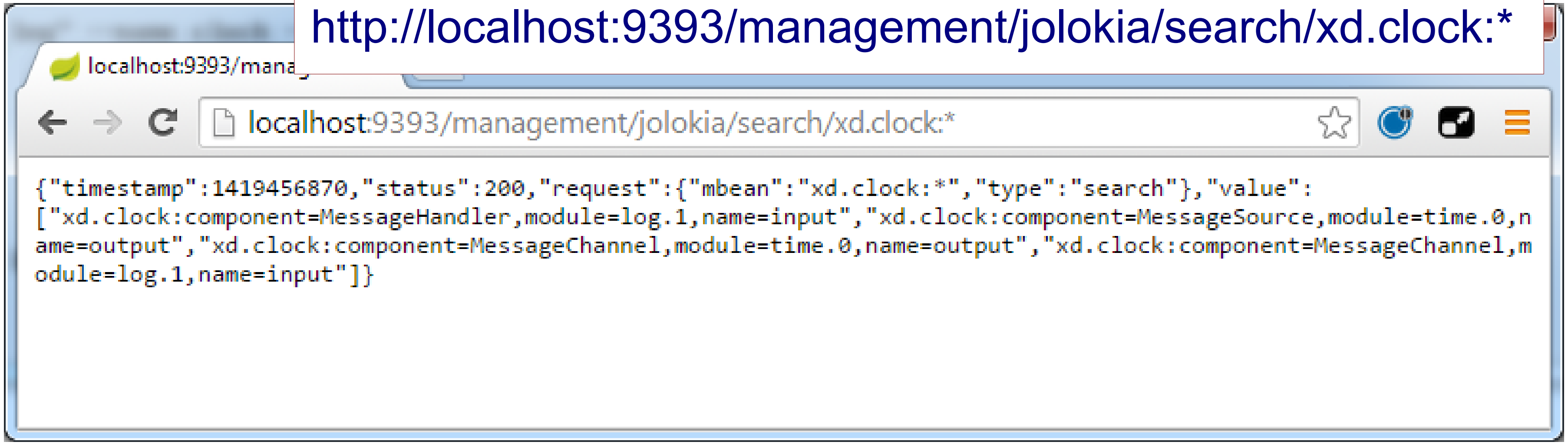
- Example: Fetch all modules for deployed stream “clock”

```
xd:>stream create --definition "time | log" --name clock
Created new stream 'clock'
xd:>stream deploy --name clock
Deployed stream 'clock'
```

- Creates Mbeans with domain name xd.clock
- Creates objects time.0 and log.1

Monitoring Deployed Streams

http://localhost:9393/management/jolokia/search/xd.clock:*


localhost:9393/mana

localhost:9393/management/jolokia/search/xd.clock:*

```
{"timestamp":1419456870,"status":200,"request":{"mbean":"xd.clock:*","type":"search"},"value":
["xd.clock:component=MessageHandler,module=log.1,name=input","xd.clock:component=MessageSource,module=time.0,n
ame=output","xd.clock:component=MessageChannel,module=time.0,name=output","xd.clock:component=MessageChannel,m
odule=log.1,name=input"]}
```

```
{"timestamp":1419456870,"status":200,"request":
{"mbean":"xd.clock:*","type":"search"},"value":
["xd.clock:component=MessageHandler,module=log.1,name=input",
"xd.clock:component=MessageSource,module=time.0,name=output",
"xd.clock:component=MessageChannel,module=time.0,name=output"
,"xd.clock:component=MessageChannel,module=log.1,name=input"]
}
```

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

Hadoop Support

- Major Hadoop distributions are supported
- Needed to be specified during startup of containers and shell using **`--hadoopDistro`** argument
- See next slide for supported values or execute "**`xd-shell --hadoopDistro`**"

```
$ xd-container --hadoopDistro phd21
$ xd-shell --hadoopDistro phd21
$ xd-singlenode --hadoopDistro phd21
```

Supported Hadoop Distributions

- Supported values for –hadoopDistro arg:
  - hadoop25 - Apache Hadoop 2.5.2
  - hadoop26 - Apache Hadoop 2.6.0 (default)
  - phd1 - Pivotal HD 1.1
  - phd21 - Pivotal HD 2.1
  - phd21 - Pivotal HD 2.1
  - cdh5 - Cloudera CDH 5.3
  - hdp22 - Hortonworks Data Platform 2.2

List varies with Spring XD versions

Hadoop Setup

- Hadoop client not required
  - No core-site.xml
  - Nameserver config in xd/config/servers.yml (default localhost:8020)

```
# Hadoop properties
spring:
  hadoop:
   fsUri: hdfs://192.168.218.140:8020
```

  - Additional config in xd/config/hadoop.properties

HDFS Sink

- Writes raw data to one or more HDFS files
  - Simple example:

```
stream create
   --name myTimeHdfs
   --definition "time | hdfs"
   --deploy
```

- Creates the file /xd/myTimeHdfs/myTimeHdfs-0.txt in HDFS.
- But, file might not be closed for long time
  - hence, not readable from other processes

HDFS Open Files

- Spring XD adds a prefix/suffix to open files
  - Default suffix: .tmp
  - Can be changed with **`--inUseSuffix`** and **`--inUsePrefix`** options

```
stream create
   --name myTimeHdfs
   --definition
      "time | hdfs  --inUseSuffix='' --inUsePrefix='tmp-'"
   --deploy
```

HDFS Rollover

- Default rollover is 1GB
  - can be changed with the rollover option:

```
stream create
   --name myTimeHdfs
   --definition "time | hdfs --rollover=256K"
   --deploy
```

- Value is treated as bytes by default.
  - Other examples: 64K, 128M, 512G, 1T
- Forces early closing of file, results in file names like:
  - myTimeHdfs-0.txt, myTimeHdfs-1.txt, myTimeHdfs-2.txt

HDFS Idle Time

- Rollover only good for events with continuous high rate
- Not so good for low frequency burst events
  - E.g. from HTTP Source. No consistent state in closed files on HDFS.
- Use idleTimeout instead (milliseconds)

```
stream create
   --name myTimeHdfs
   --definition "http | hdfs --idleTimeout=3000"
   --deploy
```

- Can be combined with rollover

HDFS Naming Strategies

- Default file location is
  - */xd/<stream name>/<stream name>-<rolling part>*.txt
- Can be changed using
  - `--directory`, `--fileName` and `--fileExtension`

```
stream create
   --name myTimeHdfs
   --definition
      "time | hdfs  ---directory=/my --fileName=data"
   --deploy
```

- Creates files in /my/data-0.txt, /my/data-1.txt, ...

XD Shell HDFS commands

- Spring XD shell supports some useful HDFS commands
  - No need to install Hadoop client!
- Needs to be configured first:

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

- Supports basic HDFS operations, like:

```
xd:> hadoop fs ls /xd
xd:> hadoop fs cat /xd/myTimeHdfs/myTimeHdfs-0.txt
xd:> hadoop fs rm /xd/myTimeHdfs/myTimeHdfs-0.txt
xd:> hadoop fs rm /xd --recursive
```

Advanced Streams

- Hadoop (HDFS)
- **Transform/Script processors**
- TCP/UDP sources
- Composite Modules
- Lab

Transform Processor

- Generic way to convert messages using a Spring Integration transformer
- Supports Spring Expression Language (SpEL) expressions and Groovy scripts

```
xd:> stream create --name myTransform --definition
"http | transform --expression='payload.toUpperCase()'
| log" --deploy
Created and deployed new stream 'myTransform'
xd:>http post --data "Transform me!"
> POST (text/plain;Charset=UTF-8) http://localhost:9000
Transform me!
> 200 OK
```

SpEL

```
13:30:34,535 1.1.0.RELEASE  INFO pool-21-thread-4
sink.myTransform - TRANSFORM ME!
```

Groovy Script Locations

- Groovy scripts need to be placed in ${xd.home}/modules/processor/scripts
- Or specify alternate location using **`--script`** option and **`file:`** prefix
  - Recommend using a shared NFS location

```
xd:> stream create --name myTransform --definition
"http
| transform --script=file:/myscripts/myUpperCase.groovy
| log" --deploy
```

Transform Processor

- Example with Groovy script
- Script *must* return a value

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

```
xd:> stream create --name myTransform  --definition
"http
| transform --script=myUpperCase.groovy
| log" --deploy
```

```
$ curl -d "Test" http://localhost:9000

19:07:47,105 1.0.2.RELEASE  INFO pool-31-thread-4
sink.myTransform - TEST
```

Script Processor

- Generic way to process messages using a Spring Integration Service Activator
- Script does *not* have to return a value

```
return null
```

```
xd:> stream create --name myTransform --definition
"http
| script --script=myUpperCase.groovy
| log" -deploy
Created and deployed new stream 'myTransform'

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

Difference Between Filter/Transform/Script

- All of them can execute Groovy scripts
- Filter: Groovy scripts decides if message gets dropped
  - Return false to drop message
- Transform: Groovy script needs to return something, can only convert message
  - Throws an exception if no value is returned
- Script: Groovy script can convert message or drop
  - Return null to drop message

- Conclusion: Script is the most generic one

Advanced Streams

- Hadoop (HDFS)
- Transform/Script processors
- **TCP/UDP sources**
- Composite Modules
- Lab

TCP Source

- Acts as a raw TCP server
- Several decoders available to frame a message out of the TCP stream
  - CRLF (default), LF, NULL, STXEX, ..
- Produces byte[], can be changed with **`--outputType=text/plain`**

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

TCP Source Decoders (Framing)

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

Default
CRLF Decoder

```
$ echo -en 'foobar\r\n' | netcat localhost 3000
```

Send <CRLF> terminated
string to localhost:3000

```
xd:> stream create --name tcptest --definition
    "tcp --port=3000 --outputType=text/plain --decoder=NULL
     | log" --deploy
```

null Decoder

```
$ echo -en 'foobar\x00' | netcat localhost 3000
```

Send null terminated
string to localhost:3000

Using Spring XD to Capture Stdout

- No stdin source provided
- Use TCP source instead:

```
xd:> stream create --name tcpforstdout --definition
    "tcp --port=3000 --outputType=text/plain --decoder=LF
     | log" --deploy
```

```
$ cat mylog.txt | netcat localhost 3000
```

Advanced Streams

- Hadoop (HDFS)
- Transform/Script processors
- TCP/UDP sources
- **Composite Modules**
- Lab

Composite Modules

- Possible to combine 2 or more modules into a composite module
- Promotes re-use
- Improves performance
  - Composite module is deployed as a single unit
  - Local in-memory channel will be used instead of messaging middleware
  - Fewer network hops

Composite Modules

- Given the following streams:

Identical

```
stream1 = http | filter
      --expression=payload.contains('foo') | file
stream2 = file | filter
      --expression=payload.contains('foo') | file
```

- Create a composite module from the common components

```
xd:> module compose foo
  --definition "filter
  --expression=payload.contains('foo') | file"
```

Composite Modules

- Composite modules will appear in the **`module list`** with a **`(c)`** beside them

- Create the streams using the composite component

```
xd:> stream create httpfoo --definition
     "http | foo" -deploy

xd:> stream create filefoo --definition "file
     --outputType=text/plain | foo"  --deploy
```

Reactor IP Source

- Comparable to TCP Source, but based on Reactor project
  - Much higher message throughput
  - Also supports UDP (`--transport=udp`)
  - Produces Strings as default (can be changed with `--codec`)
- Only supports linefeed (default) and length framing
  - Called decoders in TCP source

```
xd:> stream create --name tcptest --definition
    "reactor-ip --port=3000 | log" --deploy
```

```
$ echo -en 'foobar\n' | netcat localhost 3000
```

Composite Modules

- A composite module that contains:
  - a source and one or more processors is considered a <u>source</u> module
  - only processors is considered a <u>processor</u> module
  - a sink and one or more processors is considered a <u>sink</u> module
- In the example below we are creating the foo sink

```
xd:> module compose foo --definition "filter
  --expression=payload.contains('foo') | file"
```

- To delete the module you would have to refer to it as:

```
xd:> module delete –name sink:foo
```

spring Pivotal™

Lab

Experiment with Transformers and Composed Modules