Spring XD

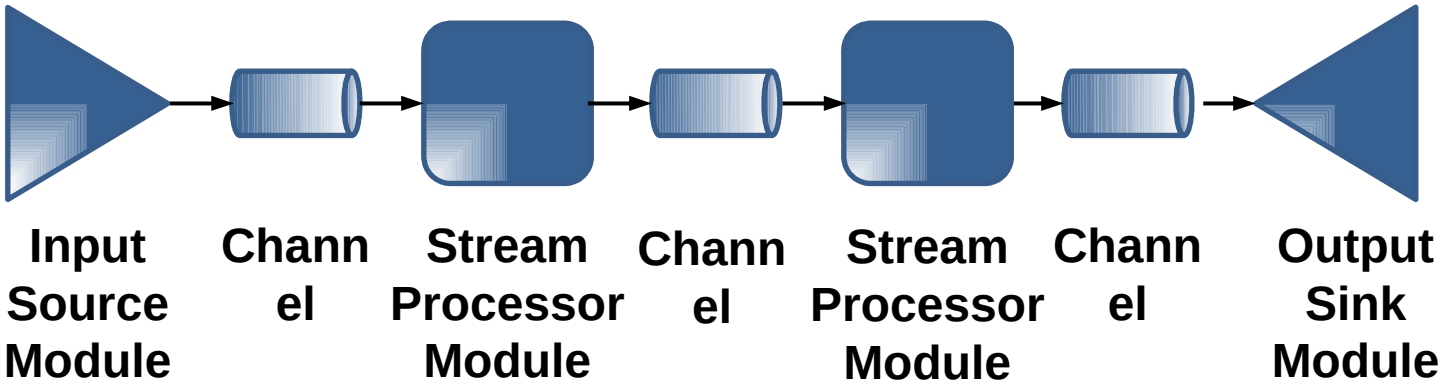
Introduction to Streams

Introduction to Streams

- Overview
- Defining streams
- Sources
- Processors
- Sinks
- Examples
- Taps
- Lab

Overview

- “A basic stream defines a flow of event driven data from a source to a sink via any number of processors.”
- A stream is composed of modules and channels
- Deployed to containers by the XD Admin server



Stream Modules

- Three types of modules: sources, processors and sinks.
- Source: digests some form of input and emits a "message"
- Processor: receives a message, performs some kind of processing, emits another message
- Sink: receives a message, writes to some kind of output
- Under the covers, Sources, Processors, and Sinks are actually stand-alone Spring Integration contexts with a specific purpose

Defining Streams

- XD Domain Specific Language (DSL) mimics Unix pipes and filters syntax
- New streams created by posting stream definitions from Spring XD shell
- Use **`stream create –name <streamname> –definition <streamdef>`** command

```
xd:> stream create --name ticktock --definition "time | log"
```

Streams need a name

Streams need a definition

Definitions need a source and a sink (minimum)

Checking Stream Status

- Use **`stream list`** command
- Displays definition and status

```
xd:>stream list
  Stream Name   Stream Definition   Status
  -----------   -----------------   -----------
  ticktock      time | log          undeployed

xd:>
```

Module Options

- Module options (properties), such as port numbers or directories, can be prefixed with "--"
  - Provided when creating the stream
  - Options are dependent on module implementations
    - Refer to module documentation
- Modules can also be *labeled* by adding a prefix
  - Analogous to an alias
  - Useful for making unique within a stream, or to reference a module from a tap

```
http --port=8091 | myAlias:file --dir=/tmp/httpdata/
```

Options

Label

Deploying Streams

- Streams must be explicitly *deployed*
- Use **`stream deploy --name <streamname>`** command, or **`--deploy`** option on **`stream create`** command
- Copies stream to container(s)
- “Starts” the stream running

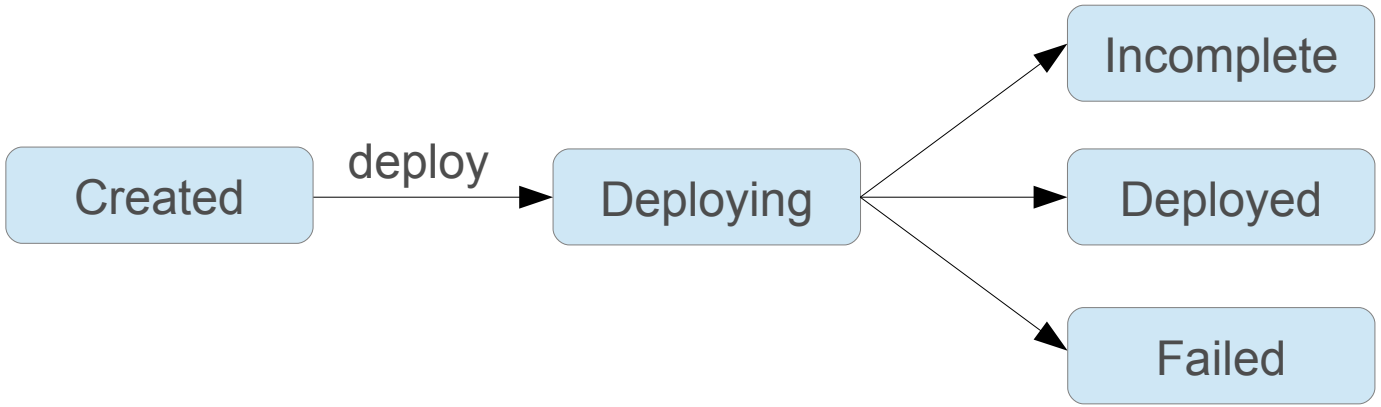
```
xd:> stream deploy --name ticktock
```

Deploy a stream previously created

```
xd:> stream create --name ticktock
      --definition "time | log" --deploy
```

Create *and* deploy a stream

Deployment States



- Created: Stream created but not deployed
- Deploying: Deployment in progress
- Incomplete: At least one instance of each module was deployed successfully
- Deployed: All modules deployed successfully
- Failed: At least one module was not deployed

Stream Deployment Properties

- **`stream deploy`** command accepts **`properties`** parameter
- Contains comma-delimited list of key/value pairs to control
  - The number of module instances
  - A target server or server group
  - MessageBus attributes required for a specific module
  - Stream Partitioning
  - Direct Binding
  - History Tracking

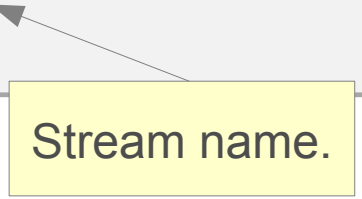
Undeploying Streams

- To stop a stream from running

  `stream undeploy --name <streamName>`

- Keeps stream definition intact, but stops and removes running streams from containers

```
xd:> stream undeploy --name ticktock
```



Stream Deployment Properties

- Properties key format is:

  `module.<modulename>.<keyName>`

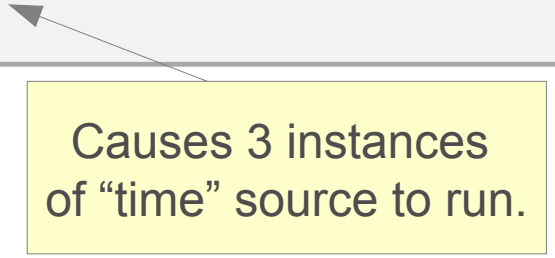
- **`<modulename>`**
  - The label or name of the module the property should apply to, eg. **`http`**
  - '**`*`**' to apply the property to all modules in the stream
- **`<keyname>`**
  - Numerous properties available, such as **`count`** and **`criteria`**

See: Deployment Properties
http://docs.spring.io/spring-xd/docs/current/reference/html/#deployment-properties

Stream Deployment Properties Example

- Control the number of module instances

```
xd:> stream deploy --name ticktock
        --properties "module.time.count=3"
```



Deleting Streams

- Use **`stream destroy`** command
- Destroying a deployed (i.e. running) stream will also undeploy it

```
xd:> stream destroy --name ticktock
```

Sources

- Numerous input sources available, based on Spring Integration Adapters

HTTP	*Gemfire Source*	*RabbitMQ*
SFTP	*Gemfire CQ*	*Time*
Tail	*Syslog*	*MQTT*
File	*TCP*	*Stdout Capture*
Mail	*TCP Client*	*Kafka*
Twitter Search	*Reactor IP*	*JDBC*
Twitter Stream	*JMS*	

Processors

- Numerous processors available
- Many based on Spring Integration Endpoints

Filter	*Splitter*	*Shell Command*
Transform	*Aggregator*	*JSON to Tuple*
Script	*HTTP Client*	*Object to JSON*

Sinks

- Numerous sinks available, based on Spring Integration Adapters

Log	*Shell Command*	*MQTT*
File	*Mongo*	*Dynamic Router*
HDFS	*Mail*	*Null Sink*
HDFS Dataset	*RabbitMQ*	*Redis*
JDBC	*GemFire Server*	*Kafka*
TCP	*Splunk Server*	

Simple Time Source Example

- Periodically outputs a String with current time
- Options:
  - Frequency in seconds (**`--fixedDelay`**)
  - String indicating output format, using SimpleDateFormat convention (**`--format`**)

See: Spring XD Reference - Time
http://docs.spring.io/spring-xd/docs/current/reference/html/#time

Simple Time Source Example

```
xd:>stream create --name clock --definition
"time --fixedDelay=5 --format=HH:mm:ss | file" --deploy

Created and deployed new stream 'clock'
```

```
$ more /tmp/xd/output/clock.out
16:45:50
16:45:55
16:46:00
```

Default file directory and name
Override with --dir and --name

Simple HTTP Source Example

- Instantiates an HTTP listener
- Options:
  - Flag for HTTPS (**--https**)
  - Port to listen on (**--port**)
  - Location of SSL properties containing pkcs12 keyStore and pass phrase (**--sslPropertiesLocation**)

See: Spring XD Reference - HTTP
http://docs.spring.io/spring-xd/docs/current/reference/html/#http

Simple HTTP Source Example

```
xd:>stream create --name httpdemo --definition
"http --port=9090 | file"  --deploy
Created and deployed new stream 'httpdemo'
```

```
xd:>http post
    --target http://localhost:9090
    --data "Hello Spring XD!"
```

```
xd:>! more  /tmp/xd/output/httpdemo.out
command is:more /tmp/xd/output/httpdemo.out
Hello Spring XD!
```

Simple File Source Example

- Outputs either:
  - Contents of a file as a byte array
  - The file reference
    - Useful when file contents are large
- Options:
  - Source directory (**`--dir`**), default is **`/tmp/xd/input`**
  - Polling interval (**`--fixedDelay`**)
  - “Ant” style pattern for filtering (**`--pattern`**)
  - Flag to prevent duplicate processing (**`--preventDuplicates`**)
  - Flag to output reference instead of contents (**`--ref`**)

See: Spring XD Reference - File
http://docs.spring.io/spring-xd/docs/current/reference/html/#file

Simple File Source Example

```
xd:>stream create --name poller --definition
"file --dir=/tmp --outputType=text/plain --fixedDelay=5
--pattern=*.csv --preventDuplicates=true
| outfile:file"  --deploy

Created and deployed new stream 'poller'
```

Label sink to make unique

```
$ more /tmp/xd/output/poller.out
Hello world
```

Default file sink output directory

JSON File Filtering Example

- File source (containing JSON)
- JSON filter
  - Only ingest files with country code of “CA”
- File output sink

```
xd:>stream create --name poller --definition
"file --dir=/tmp --pattern=*.json --outputType=text/plain
| filter --expression=
#jsonPath(payload,'$.origin.country').equals('CA')
| outfile:file --dir=/tmp --mode=APPEND"
--deploy
```

See: Spring XD Reference - Filter
http://docs.spring.io/spring-xd/docs/current/reference/html/#filter

JSON File Filtering Example – Input Files

```
{
  "origin": {
    "id": 3372,
    "country": "CA"
  }
}
```

countries.json

```
{
  "origin": {
    "id": 3373,
    "country": "US"
  }
}
```

countries-2.json

```
{
  "origin": {
    "id": 3374,
    "country": "CA"
  }
}
```

countries-1.json

JSON File Filtering Example – Output File

```
{
    "origin": {
        "id": 3372,
        "country": "CA"
    }
}
{
    "origin": {
        "id": 3374,
        "country": "CA"
    }
}
```

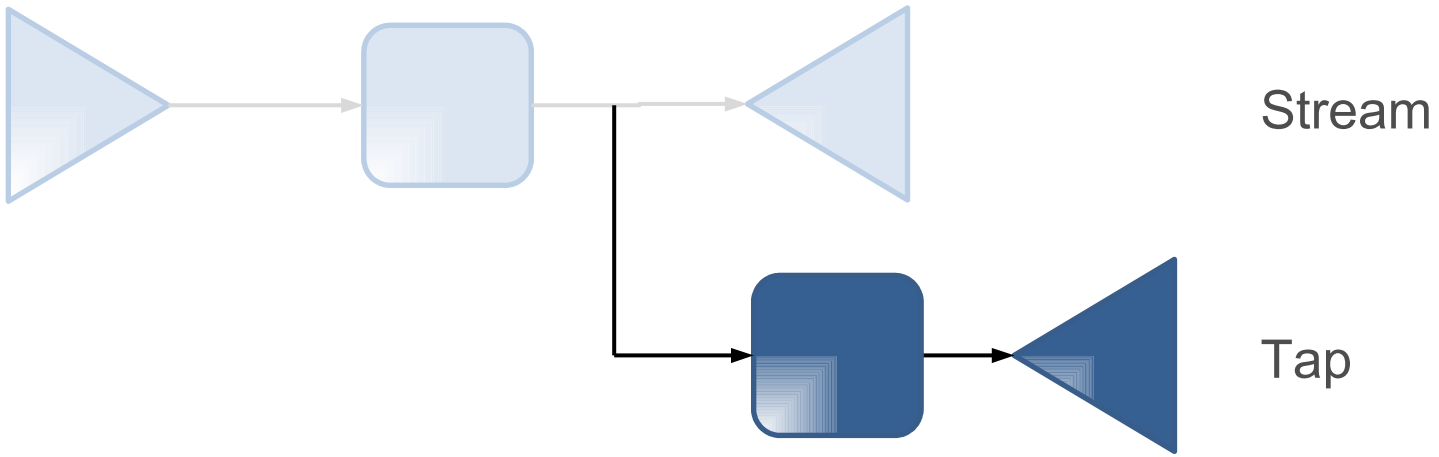
/tmp/poller.out

Additional Sources, Processors, and Sinks

- Refer to official Spring XD reference documentation for details on other modules
- http://docs.spring.io/spring-xd/docs/current/reference/html

Taps

- Taps are used to intercept data within a stream or job
- “Listen”, without modifying
  - Analogous to a phone wiretap
- A stream that uses a point in another stream as a source



Taps

- To specify point in stream to tap, use <stream_name>.<module_name>
  - If <module_name> is omitted then stream is tapped at source
  - Can also use an alias for <module_name>
- Tap is not deleted when stream is destroyed
  - Tap is actually a separate stream
- Can also tap other XD targets such as jobs

See: Spring XD Reference - Taps
http://docs.spring.io/spring-xd/docs/current/reference/html/#taps

Taps

- First create the stream

```
xd:>stream create
--name timer
--definition "time | t1:filter
--expression=payload.endsWith('0') | log"
--deploy
```

Label

- Then create the tap

```
xd:>stream create
--name timertap
--definition "tap:stream:timer.t1 > log"
--deploy
```

Note redirection symbol instead of pipe

Summary

- Streams are composed of source, processor, and sink modules
- Spring XD comes with several pre-defined modules
- Each module in a stream can be configured by specifying options at the time of creation
- The XD shell is used to manage the stream lifecycle
- Taps are special streams used to eavesdrop on data as it passes through streams or jobs

Lab

Introduction to Streams

If you finish early add 2 taps to the stream you created

For project 2.3 (Using Processors).

Let's say you name your stream filterme

- Add a tap at the source
  - `stream create tap1 --definition "tap:stream:filterme.http >log" -deploy`
- Add a tap to the filter
  - `stream create tap2 --definition "tap:stream:filterme.filter >log" -deploy`
- Follow instructions in step 2 in 2.3(Using Processors) and view the xd console

Taps

- First create the stream

```
xd:>stream create
--name timer
--definition "time | t1:filter
--expression=payload.endsWith('0') | log"
--deploy
```

Label

- Then create the tap

```
xd:>stream create
--name timertap
--definition "tap:stream:timer.t1 > log"
--deploy
```

Note redirection symbol instead of pipe