Spring XD

Architecture

Streams, Jobs and Taps

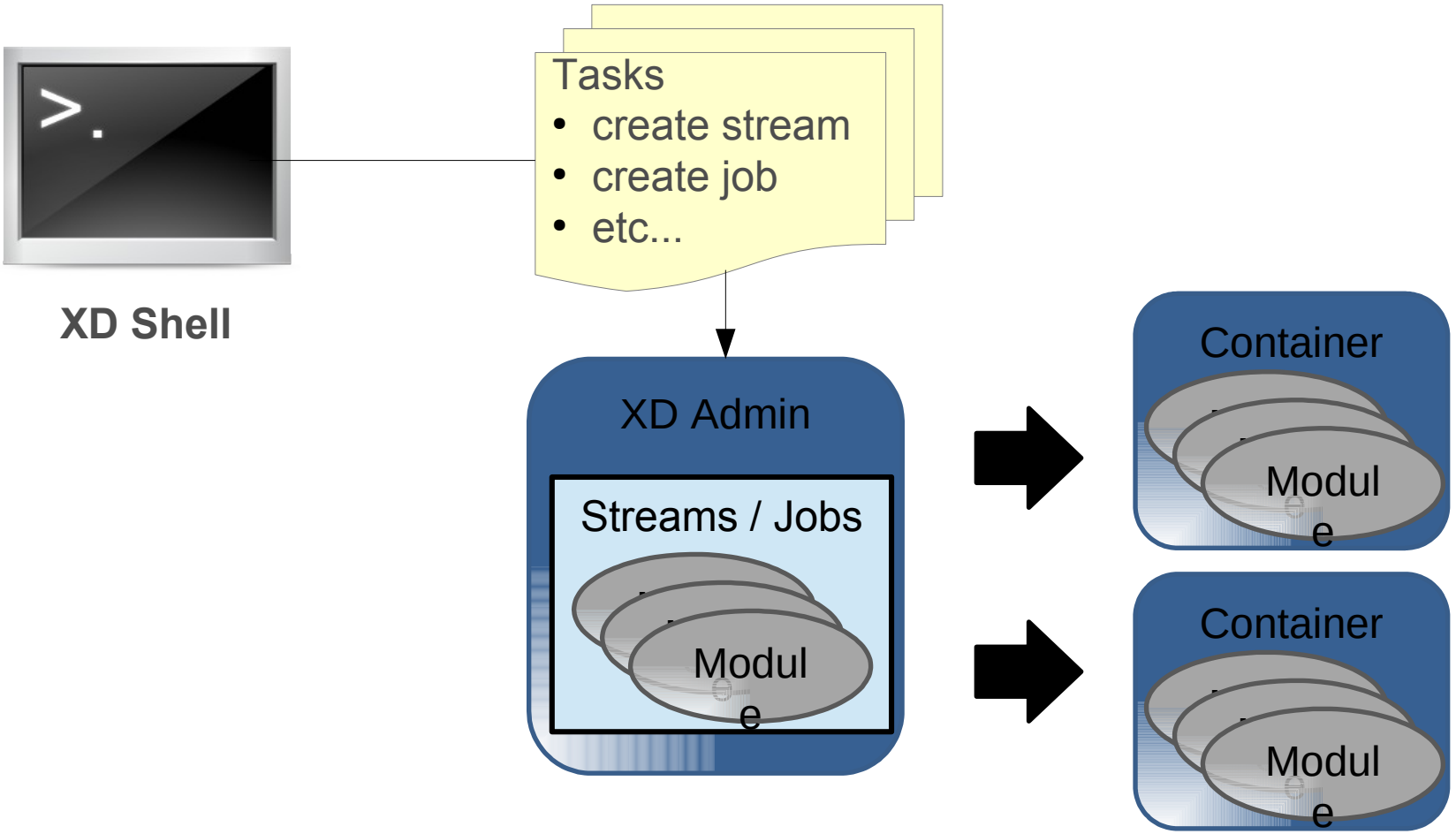
Architecture

- Overview
- Streams
- Jobs
- Taps

Overview

- User interacts with the XD Admin server using the XD Shell
- Submits tasks
- XD Admin server is responsible for executing tasks to manage streams and jobs
  - Creating, deploying, destroying, etc.
- Streams and jobs consist of modules
- Modules run in containers

Overview



Overview

- Tasks posted to XD Admin server
- XD Admin server maps tasks to modules
- Modules are Spring Application Contexts
- XD Admin server distributes modules to XD Container servers using ZooKeeper
- XD Container servers can run multiple modules

Modes of Operation

- Recall 2 modes of operation
- Standalone - single node
  - XD Admin and all modules run in a single XD Container
- Distributed Integration Runtime (DIRT)
  - Modules spread across multiple XD Containers
  - Messaging middleware (eg. RabbitMQ, Redis) used for inter-module communication

XD Admin Server

- Embedded servlet container
- Exposes REST endpoints for
  - Creating, deploying, undeploying, and destroying streams and jobs
  - Querying runtime state, analytics, and the like
- Monitors runtime state using Apache ZooKeeper

XD Container Server

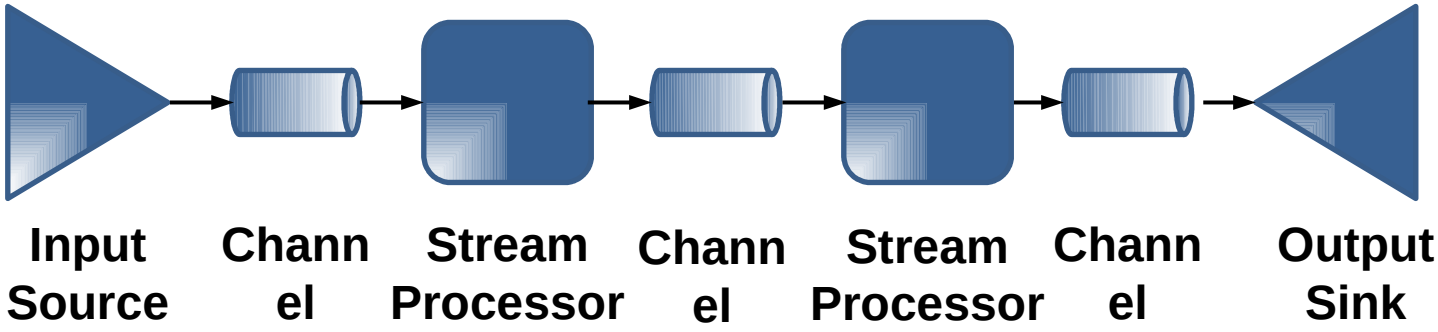
- Workhorse of Spring XD
- Containers implement and execute streams, jobs, and taps
- Can deploy multiple container servers for scaling

Streams

- Streams
  - Continuous "feed" of data
  - Often real time
  - Eg: Tweets, Log Messages, Stock tickers

Stream Processing

- Based on Enterprise Integration components from Spring Integration
- Consists of
  - Input Source Module
  - Processing Modules (optional)
  - Output Sink Module
  - Channels

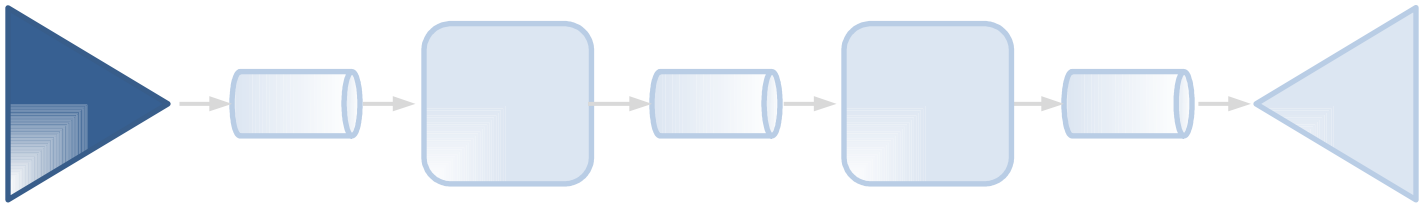


Modules

- Definitions stored in Module Registry
  - In **`<xd-install-directory>/modules`**
- Consist of
  - Spring XML configuration file
  - Classes for validation and options
  - Dependent JAR's
- Options
  - Placeholders that correspond to Domain Specific Language (DSL) parameters
  - Facilitates module customization eg. "--port=8100" would change listening port of HTTP source

Input Source Modules

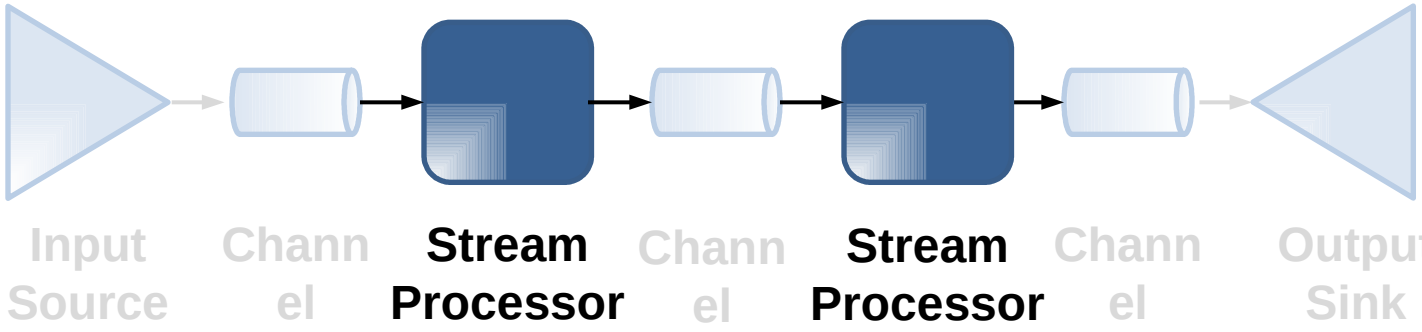
- Always first module in a stream
- Produce internal XD messages from an external source
- Output Spring message
  - Key/value headers
  - Payload
- Various source modules supported, such as HTTP, Syslog, etc.



Processing Modules

- Consume an input message, perform some processing, emit an output message.
- Several built-in processors available for simple cases
  - Filters, Transformers, Script executors, etc.
- Example:
  - (Transformer Using Spring Expression Language (SpEL)):

```
http | transformer --expression=payload.toUpperCase() | file
```



Custom Processing Modules

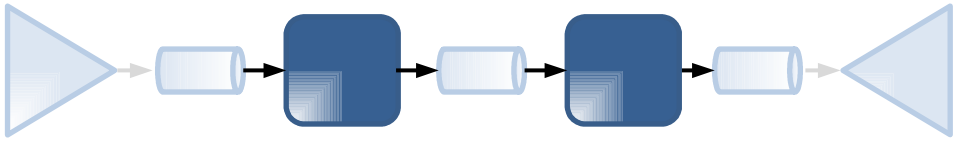
- If not using SpEL or Groovy, then can write custom processors in Java
- Central concept is Message Handler class
  - Relies on coding conventions to map inbound messages to processing methods

```
public class SimpleProcessor {

  public String process(String payload) {
    return payload.toUpperCase();
  }
}
```

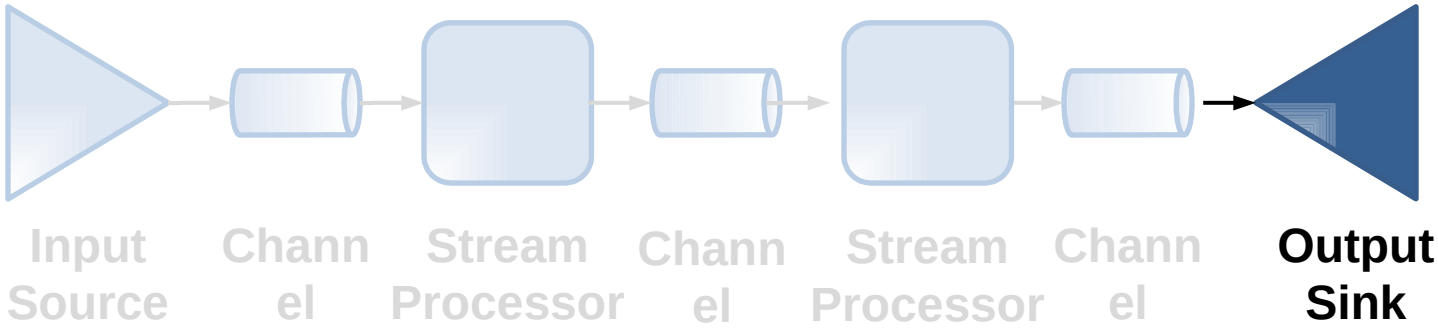
Input

Output



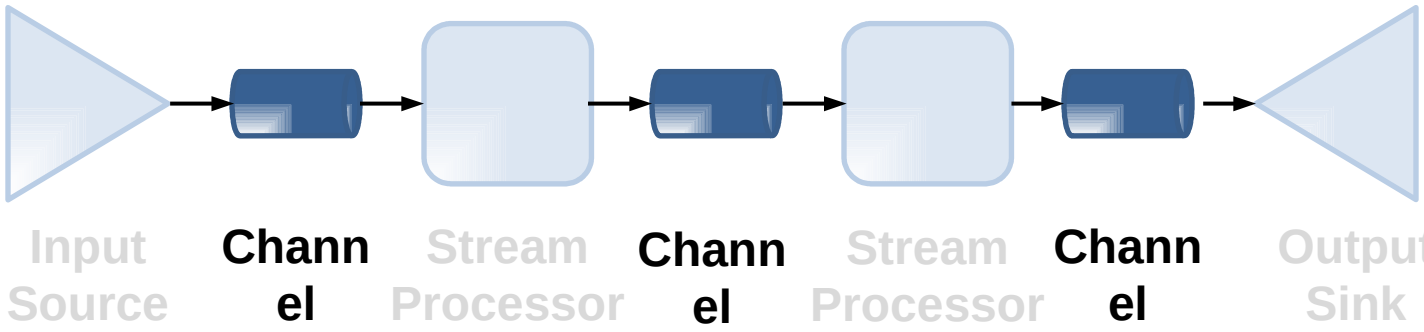
Output Sink Modules

- Always last module in a stream
- Receives internal XD messages from processor or source, and sends to external target
- Input Spring message
  - Key/value headers
  - Payload
- Various sink modules supported, such as JDBC, HDFS, etc.



Channels

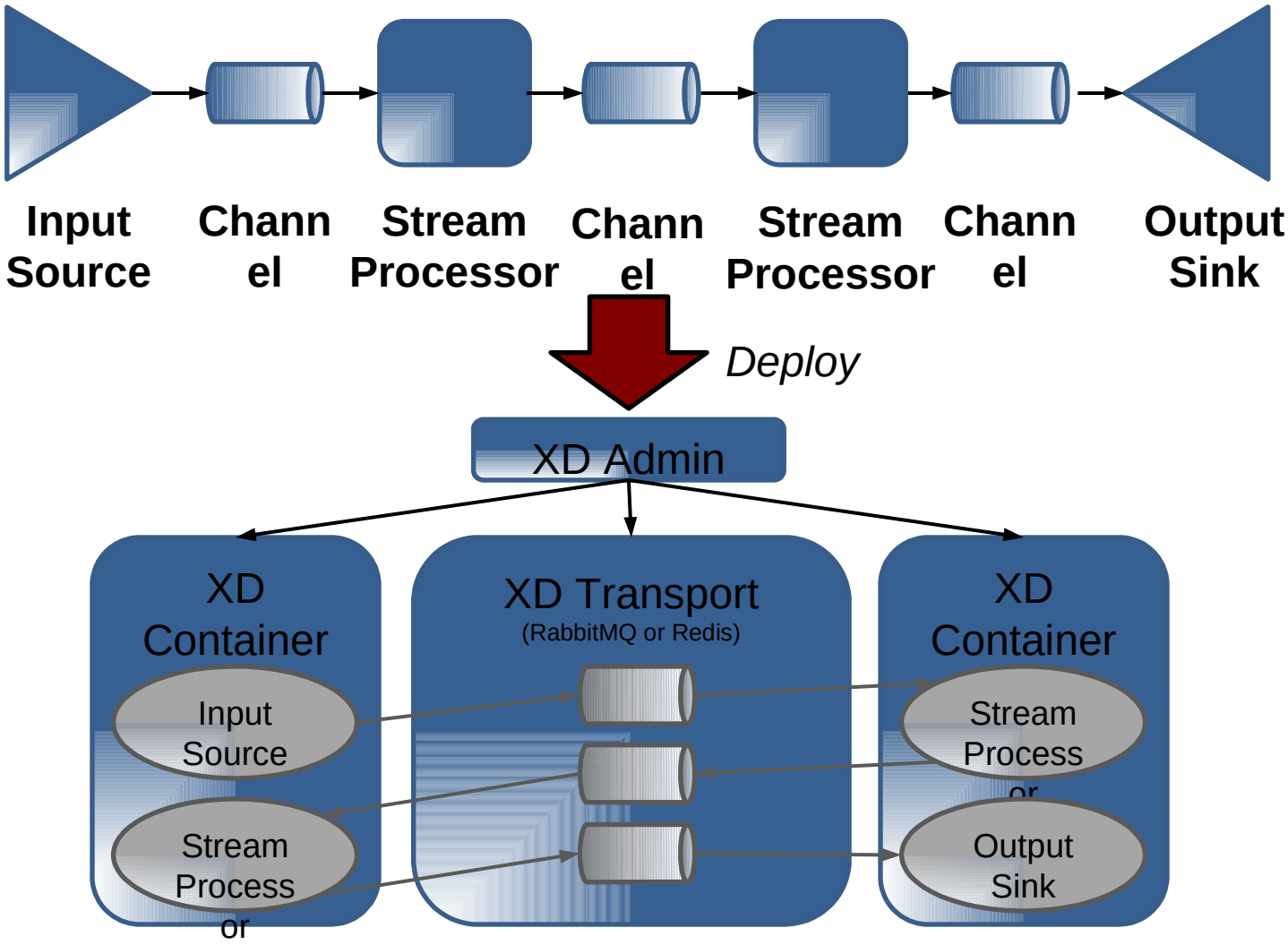
- Channel implementation is pluggable
- Local, RabbitMQ queues, Kafka and Redis currently supported
- Future releases to support other implementations, eg. JMS
- Single node deployment uses local channels by default



Stream Deployment

- Streams are divided into modules
- Modules are deployed to one or more containers
- Container servers listen for deployment events from the admin server
- On receiving deployment event, container connects input and output channels to modules

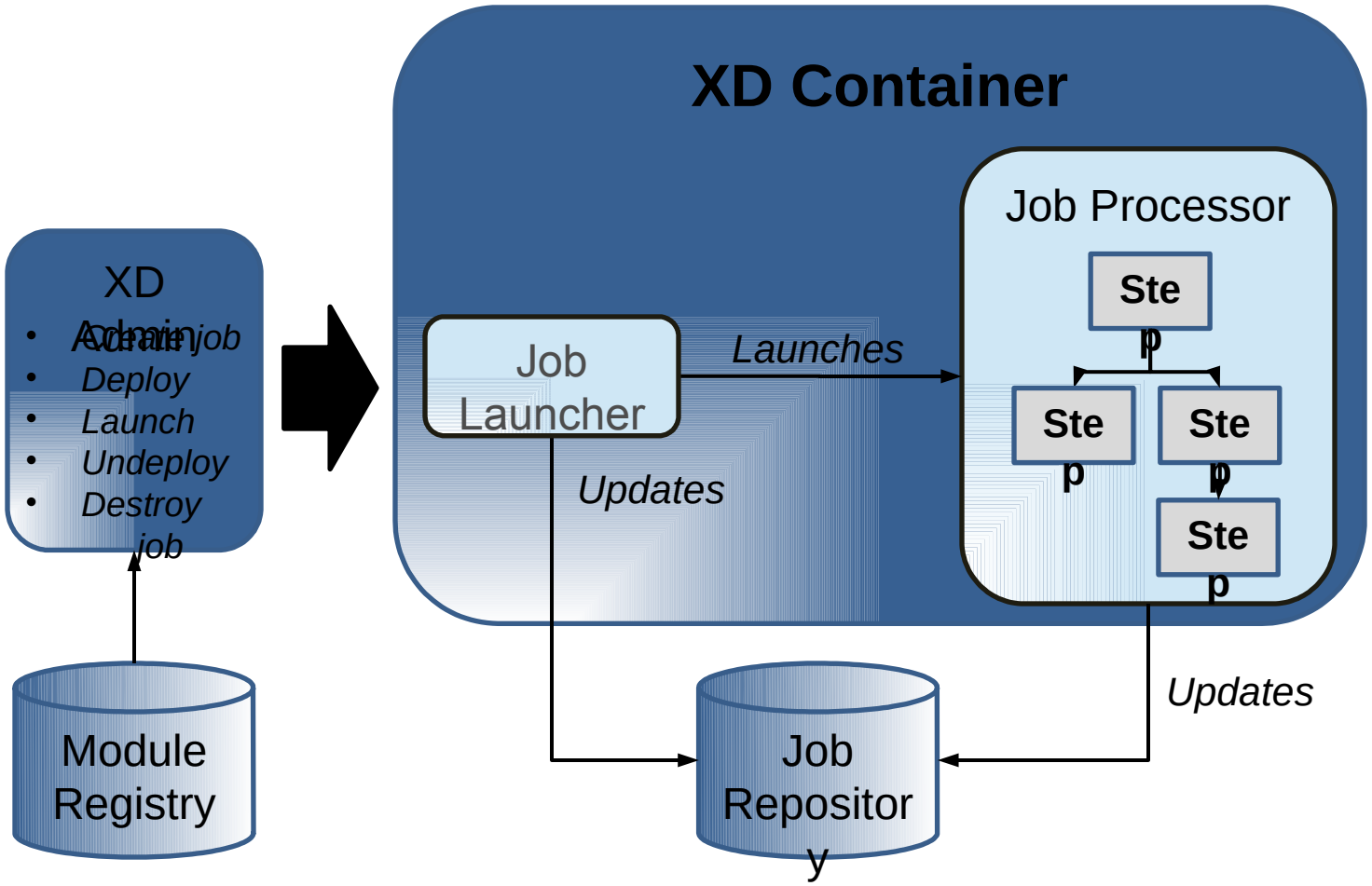
Stream Deployment



Jobs

- Batch processing based on Spring Batch
- Consist of one or more workflow steps
  - Executed sequentially or in parallel
- Can be started, stopped, restarted
- Job state is persisted
- Eg: File loaded into HDFS

Jobs

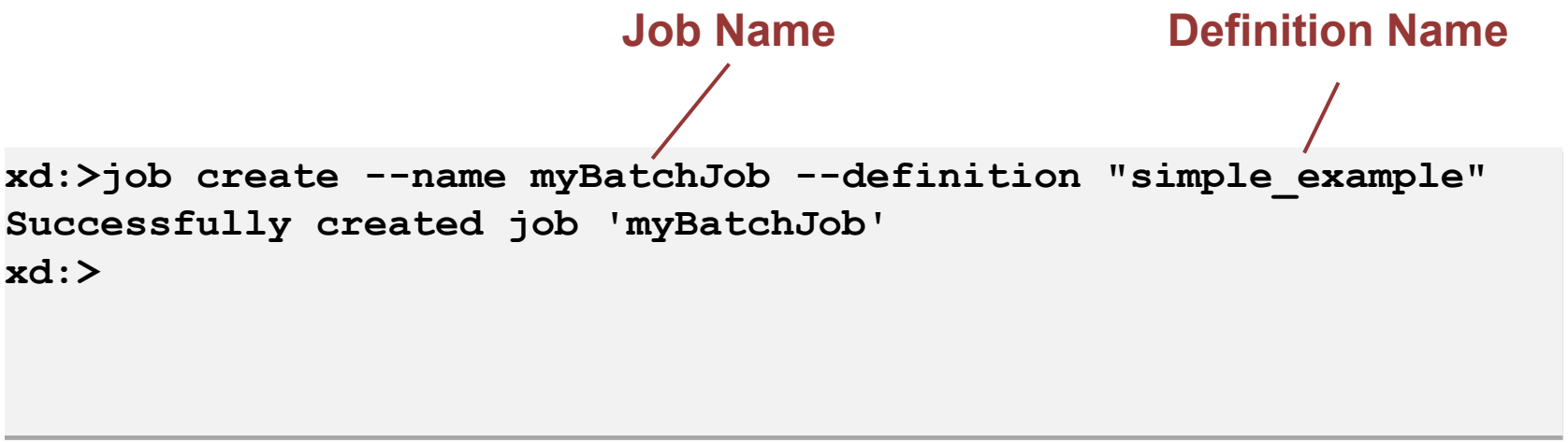


Job Lifecycle

- Register job module
- Create job definition
- Deploy job to container
- Launch job
- Job execution
- Un-deploy job
- Destroy job definition

Create Job Definition

- From the XD shell, specify Job Module by providing:
  - Definition name
  - Job instance properties



Deploy Job

- Deploy job definition to one or more Spring XD containers

Launch Job

- Send message containing job parameters to job queue
- Several mechanisms:
  - Ad-hoc from the shell
  - Cron-Trigger
  - Sink
- Creates a job instance

Job Execution

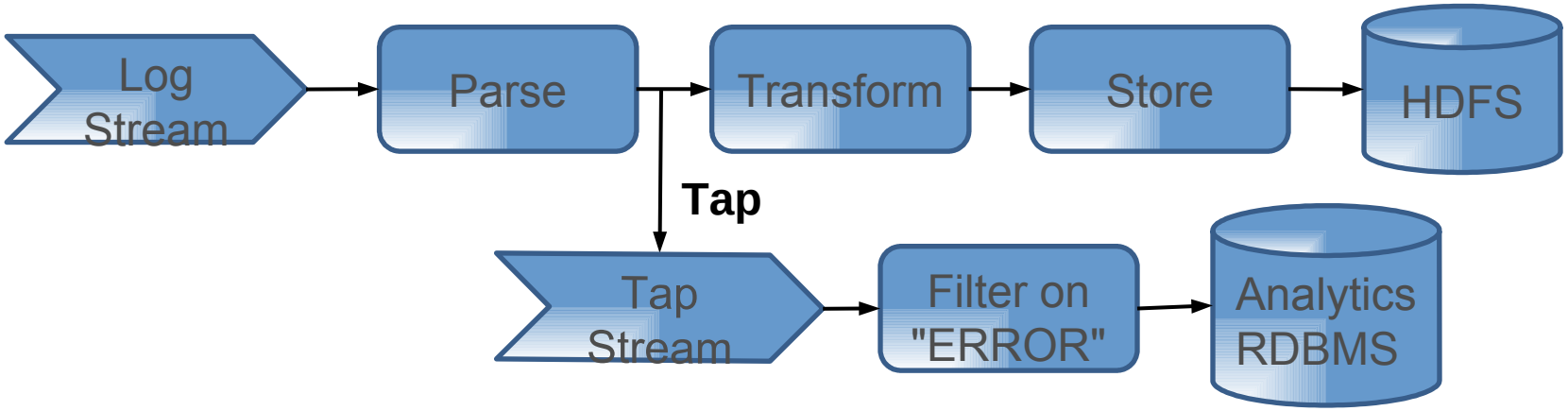
- Job execution object is created
- Captures success or failure of job
- Can query job executions associated with a given job name

Taps

- A special type of stream
- Used to intercept stream or job data non-invasively
- Can be inserted at any point in stream processing pipeline
- Does not affect original stream behavior

Taps

- Eg: Gather real-time analytics on number of "ERROR" type messages seen in a logging stream



Taps

- Creation and deployment similar to streams
- Use **`stream create`** from shell
- Specify location to place tap in existing stream

Tap Example

- First, create a **`timelogger`** stream that logs the time every 10 seconds.
  - Filter sends output to log when last digit of time is 0.

```
xd:>stream create --name timeLogger --definition "time | filter
--expression=payload.endsWith('0') | log" --deploy
Created and deployed new stream 'timeLogger'
xd:>
```

```
23:30:50,478 1.1.0.RC1 INFO…sink.timeLogger - 2015-02-10 23:30:50
23:31:00,498 1.1.0.RC1 INFO…sink.timeLogger - 2015-02-10 23:31:00
23:31:10,515 1.1.0.RC1 INFO…sink.timeLogger - 2015-02-10 23:31:10
23:31:20,538 1.1.0.RC1 INFO…sink.timeLogger - 2015-02-10 23:31:20
```

Tap Example (continued)

- Create a tap **`timetap`** after the **`time`** source
  - Specified by **`tap:stream:timeLogger.time`**
  - Redirect the output of the tap to the XD log sink

```
xd:>stream create --name timeTap --definition
"tap:stream:timeLogger.time > log" --deploy
Created and deployed new stream 'timeTap'
xd:>
```

```
23:30:50,478 1.1.0.RC1 INFO…sink.timeLogger - 2015-02-10 23:30:50
23:31:00,498 1.1.0.RC1 INFO…sink.timeLogger - 2015-02-10 23:31:00
23:31:10,515 1.1.0.RC1 INFO…sink.timeLogger - 2015-02-10 23:31:10
23:31:20,538 1.1.0.RC1 INFO…sink.timeLogger - 2015-02-10 23:31:20
```

Tap Example (continued)

```
...
23:31:26,564 1.1.0.RC1  INFO…sink.timeTap - 2015-02-10 23:31:26
23:31:27,565 1.1.0.RC1  INFO…sink.timeTap - 2015-02-10 23:31:27
23:31:28,567 1.1.0.RC1  INFO…sink.timeTap - 2015-02-10 23:31:28
23:31:29,568 1.1.0.RC1  INFO…sink.timeTap - 2015-02-10 23:31:29
23:31:30,570 1.1.0.RC1  INFO…sink.timeLogger - 2015-02-10 23:31:30
23:31:30,571 1.1.0.RC1  INFO…sink.timeTap - 2015-02-10 23:31:30
23:31:31,574 1.1.0.RC1  INFO…sink.timeTap - 2015-02-10 23:31:31
23:31:32,577 1.1.0.RC1  INFO…sink.timeTap - 2015-02-10 23:31:32
23:31:33,579 1.1.0.RC1  INFO…sink.timeTap - 2015-02-10 23:31:33
23:31:34,589 1.1.0.RC1  INFO…sink.timeTap - 2015-02-10 23:31:34
23:31:35,593 1.1.0.RC1  INFO…sink.timeTap - 2015-02-10 23:31:35
23:31:36,597 1.1.0.RC1  INFO…sink.timeTap - 2015-02-10 23:31:36
23:31:37,599 1.1.0.RC1  INFO…sink.timeTap - 2015-02-10 23:31:37
23:31:38,603 1.1.0.RC1  INFO…sink.timeTap - 2015-02-10 23:31:38
23:31:39,603 1.1.0.RC1  INFO…sink.timeTap - 2015-02-10 23:31:39
23:31:40,605 1.1.0.RC1  INFO…sink.timeTap - 2015-02-10 23:31:40
23:31:40,606 1.1.0.RC1  INFO…sink.timeLogger - 2015-02-10 23:31:40
...
```

Summary

- Spring XD Admin server distributes work to containers
- Containers run streams and jobs
- Streams consist of input source and output sink modules, with optional processing modules
- Jobs are created based on job modules, then deployed and executed
- Taps can be used to non-invasively intercept stream or job data