Spring XD

High Availability

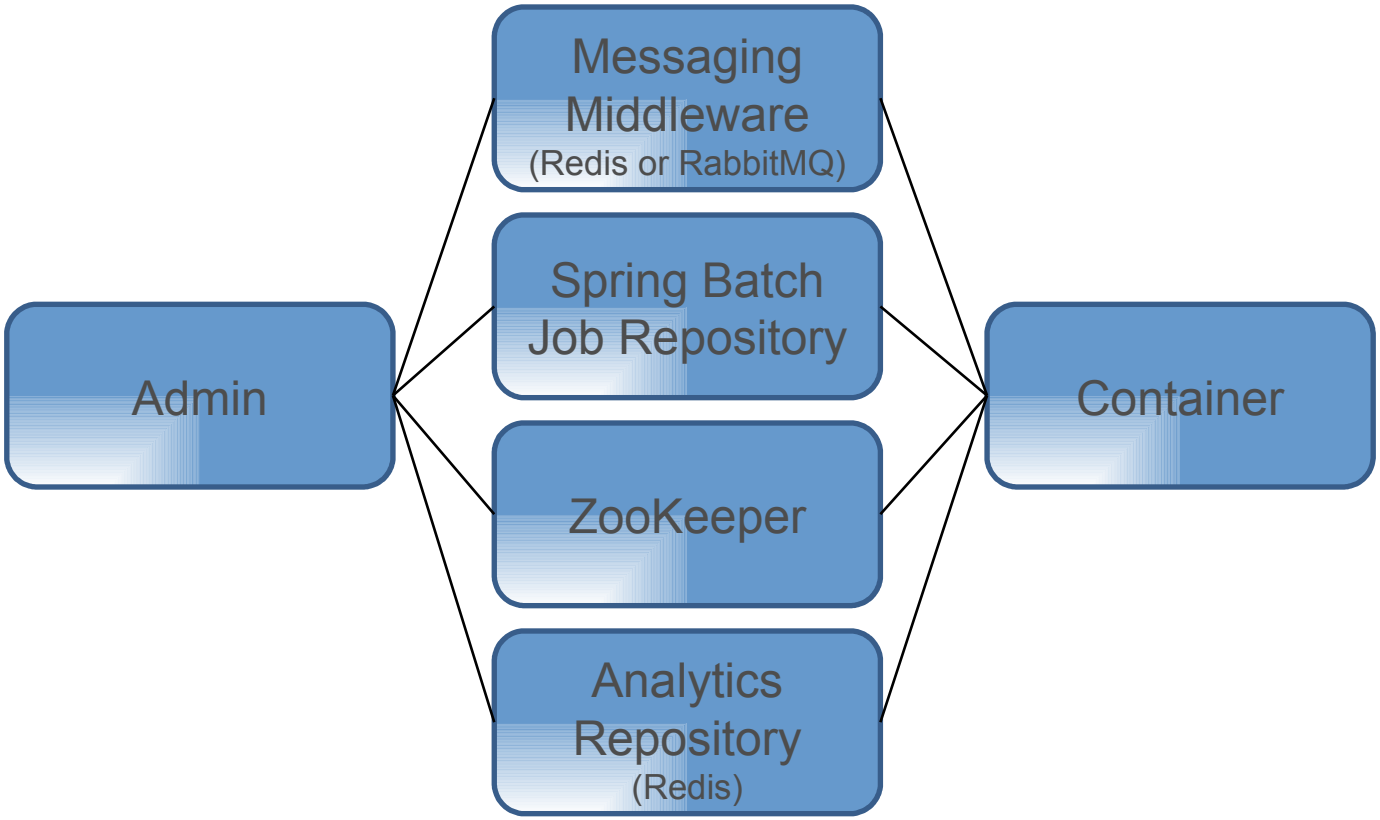
Setup and Configuration

High Availability

- High Availability (HA) and Fault Tolerance (FT) in the Spring XD context
  - ZooKeeper
  - XD Admin
  - Containers
  - Messaging middleware
  - Data stores
- Lab

High Availability

- Each individual component must be highly available



High Availability

- Configuring HA for all external products is beyond the scope of this course
- Focus on the following components:
  - ZooKeeper
  - Admin
  - Container
  - RabbitMQ messaging middleware

ZooKeeper

- Apache project
- Designed for distributed system management and coordination
- Is itself a distributed system
- Allows distributed processes to coordinate with each other through a shared hierarchical name space of data registers called znodes
- Provides clients with high throughput, low latency, highly available, strictly ordered access to the znodes
- Znodes are referenced using paths
- Each znode can have data associated with it

ZooKeeper Spring XD Schema

```
/
  /xd
    /admins
      /<ephemeral admin node>
      /...
    /containers
      /<ephemeral container node>
      /...
    /deployments
      /jobs
      /modules
        /requested
        /allocated
          /<persistent container node>
          /...
       /streams
     /jobs
     /streams
     /taps
```

Definitions (DSL)

ZooKeeper Guarantees

- Sequential Consistency
  - Updates applied in order they were sent
- Atomicity
  - Updates succeed or fail – no partial updates
- Single System Image
  - Clients see a single consistent view regardless of which server they connect to

ZooKeeper Guarantees

- Sequential Consistency
  - Updates applied in order they were sent
- Atomicity
  - Updates succeed or fail – no partial updates
- Single System Image
  - Clients see a single consistent view regardless of which server they connect to

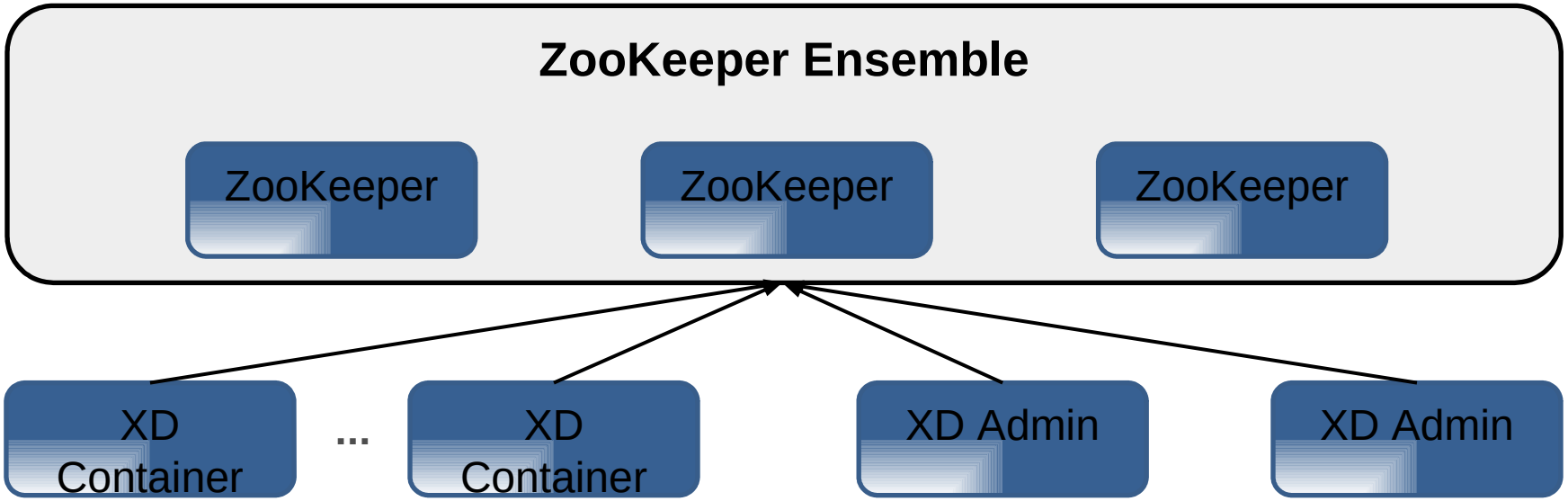
ZooKeeper Guarantees

- Reliability
  - Applied updates will persist until overwritten
- Timeliness
  - Client view is guaranteed to be up-to-date within a specified boundary

ZooKeeper HA

- ZooKeeper HA mechanism is known as an *Ensemble*
- Minimum of 3 server instances running on dedicated hosts
  - Odd number of nodes is recommended
- Spring XD Container and Admin nodes are clients to the ZooKeeper ensemble
  - Specified by **`zd.client.connect`** property
  - Consists of comma-delimited list of one or more <host>:<port> servers

ZooKeeper HA



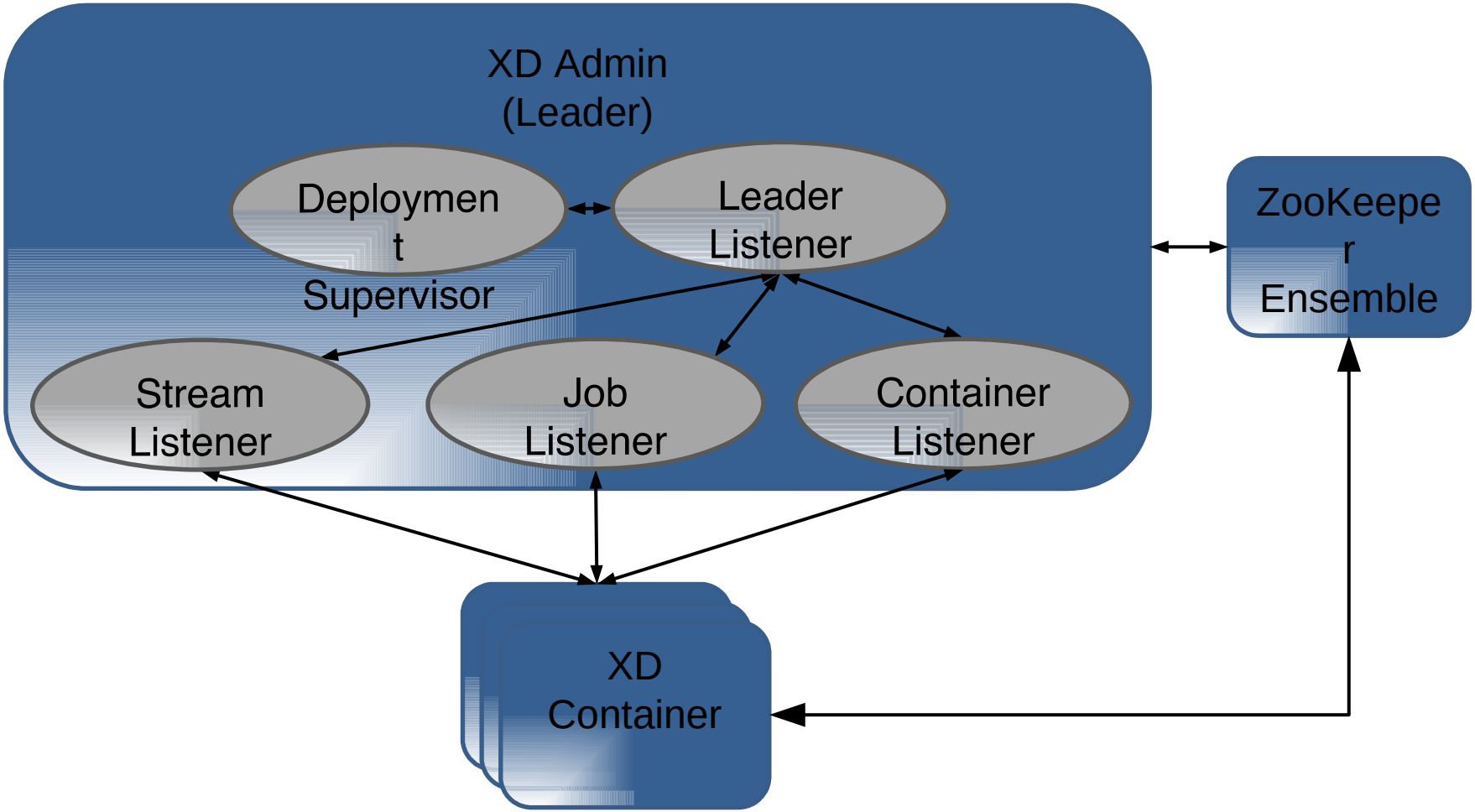
XD Admin HA

- Master-slave architecture, but clients can connect to any instance
- On startup, XD Admin server requests leadership from ZooKeeper
- Only one server designated as “leader”
- If leader goes down, another server will assume leader role
- Failover managed by ZooKeeper
- Curator Framework on ZooKeeper performs Leader Election to determine which node to elect

XD Admin HA

- XD Admin leader creates a Deployment Supervisor to register listeners to handle:
  - Stream/job module deployment requests
  - Addition/removal of containers from cluster
- Listeners listen for changes to node entries in ZooKeeper

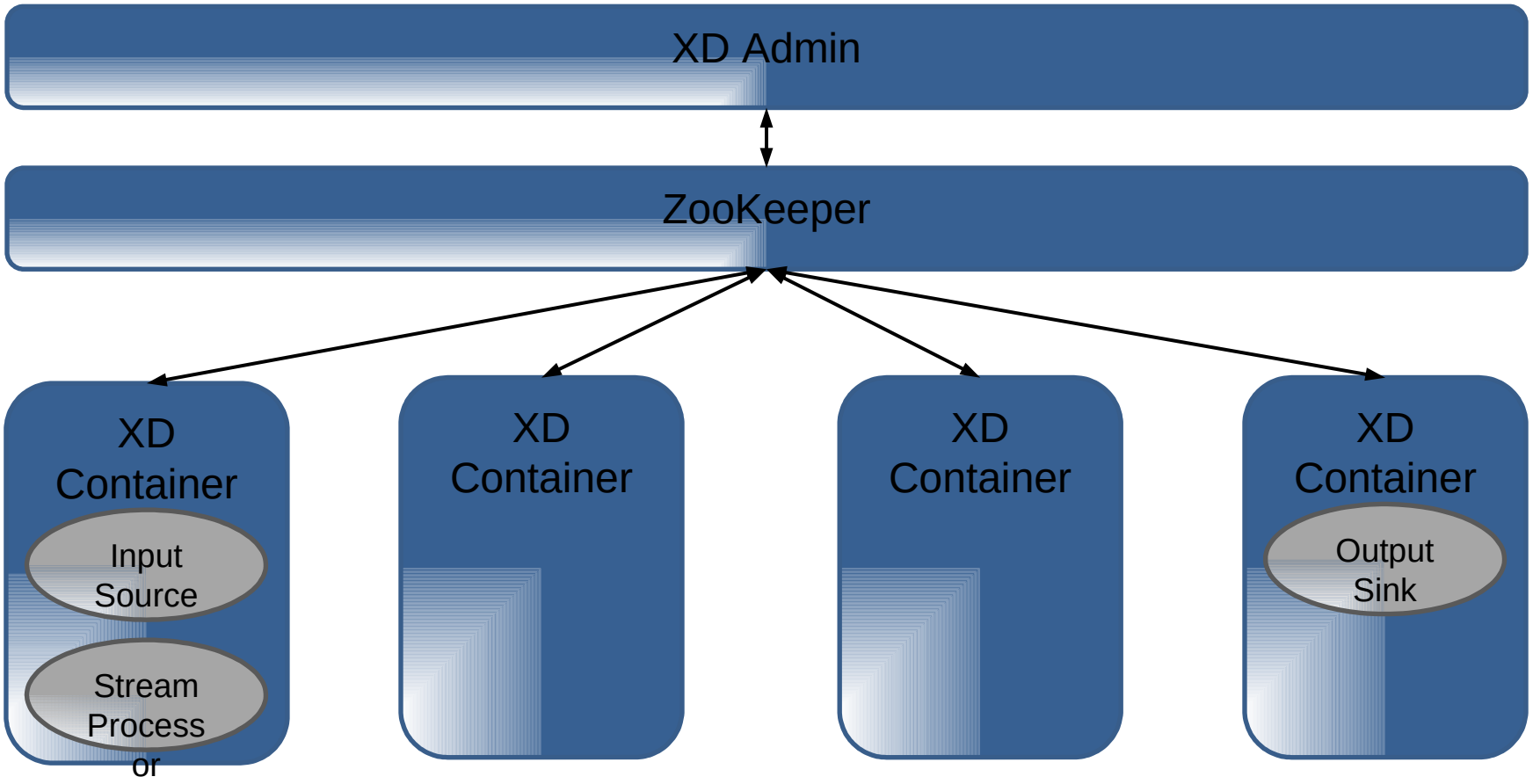
XD Admin Sub-Components



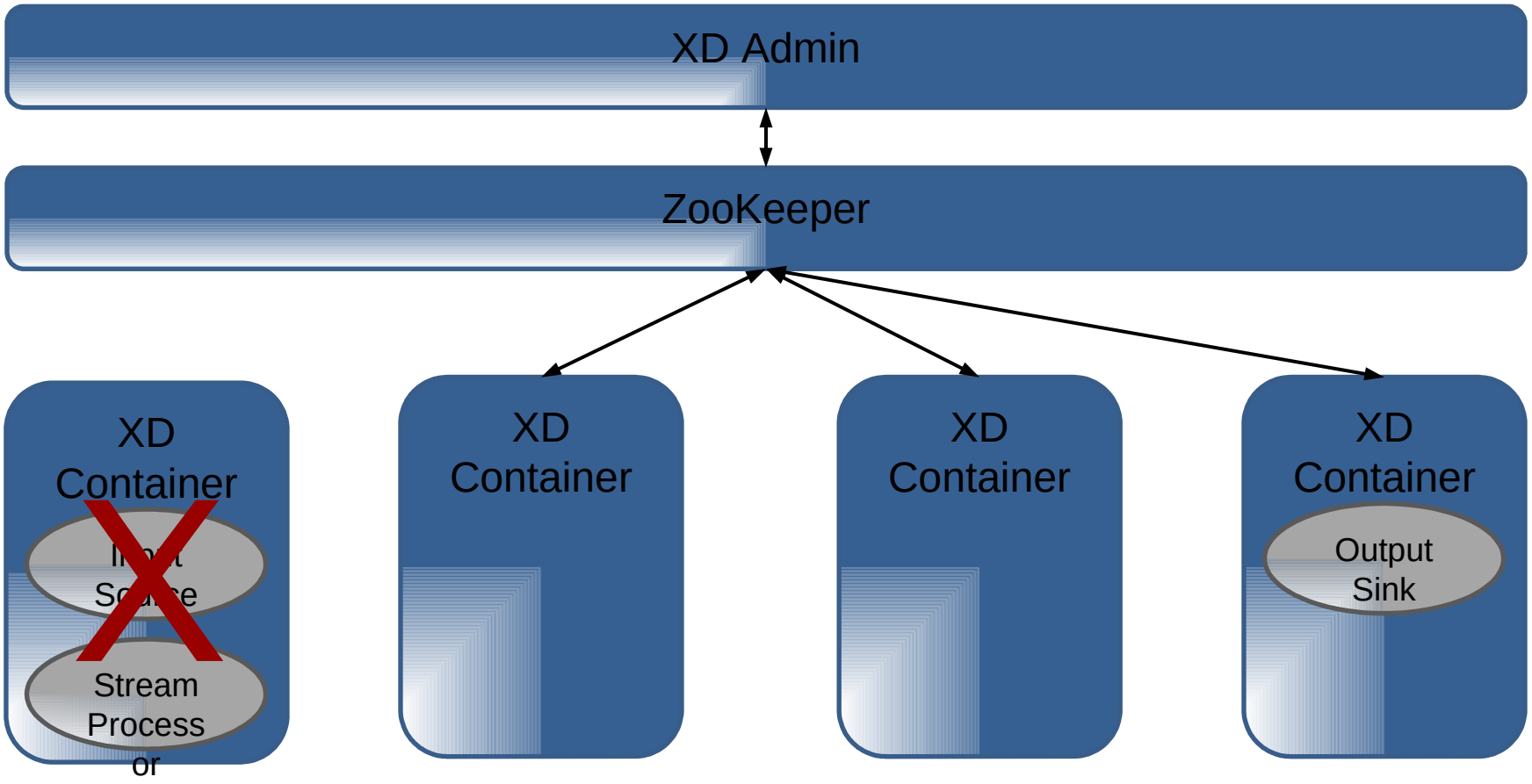
Container HA

- No single point of failure
- Run multiple containers on multiple servers
- Simplest case: all containers are replicas
- If a container goes down, modules are redeployed to other available containers

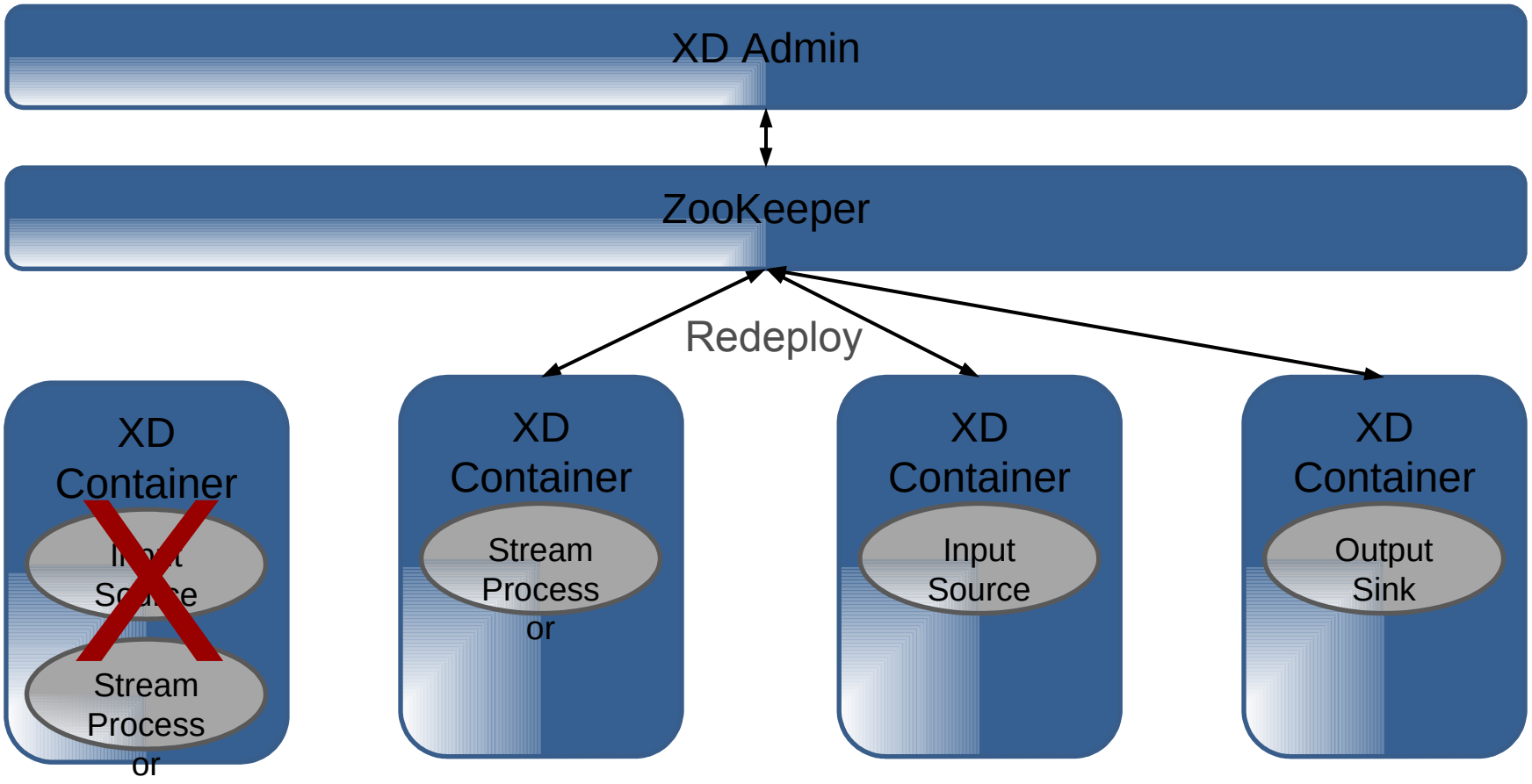
Container HA - Initial State



Container HA – Node Failure



Container HA - Redeployment

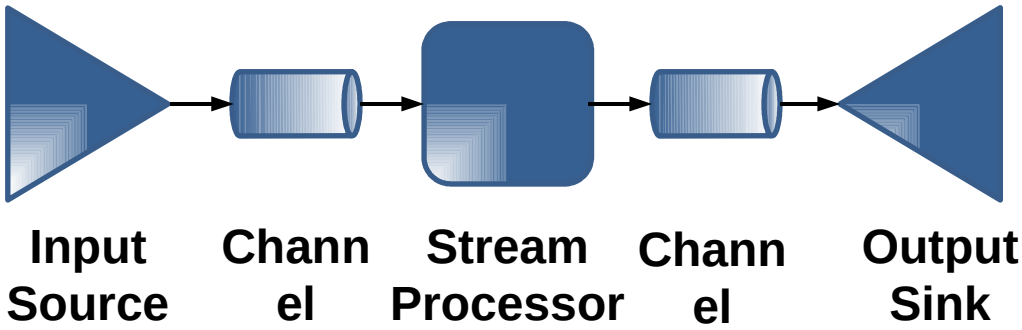


Demo

Container HA Failover

Messaging Middleware HA

- Spring XD uses “channels” to communicate between components
- Channels are backed by messaging middleware
- Redis supported, but RabbitMQ is recommended for HA
- RabbitMQ has it's own mechanism for HA
- Will cover basics, see RabbitMQ product documentation for full details

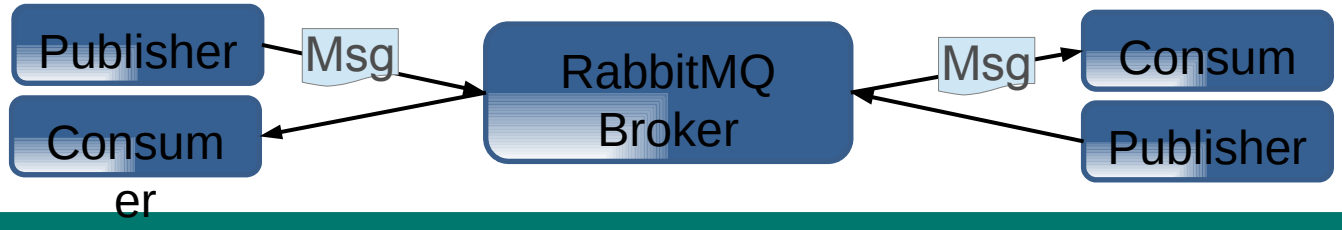


RabbitMQ – Introduction

- RabbitMQ is an Advanced Message Queuing Protocol (AMQP) messaging broker
- Messaging server known as a “broker”
- “Message Oriented Middleware”
- Often used in application integration
- Provides logical and temporal decoupling of applications
- “Hub-and-spoke” architecture

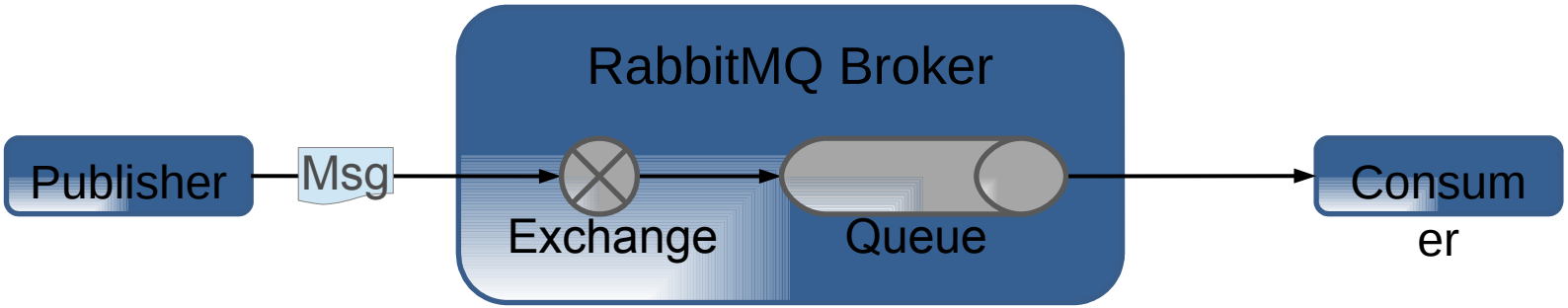
RabbitMQ – Basic Terminology

- Message: Typically consists of a header and payload (or body)
  - Header is key-value pairs
  - Body is a byte array, typically holds JSON or XML data
- Publisher or Producer: A client that sends messages
  - In Spring XD, a source or processor
- Consumer or Subscriber: A client that receives messages
  - In Spring XD, a processor or sink



RabbitMQ – Essential Terminology

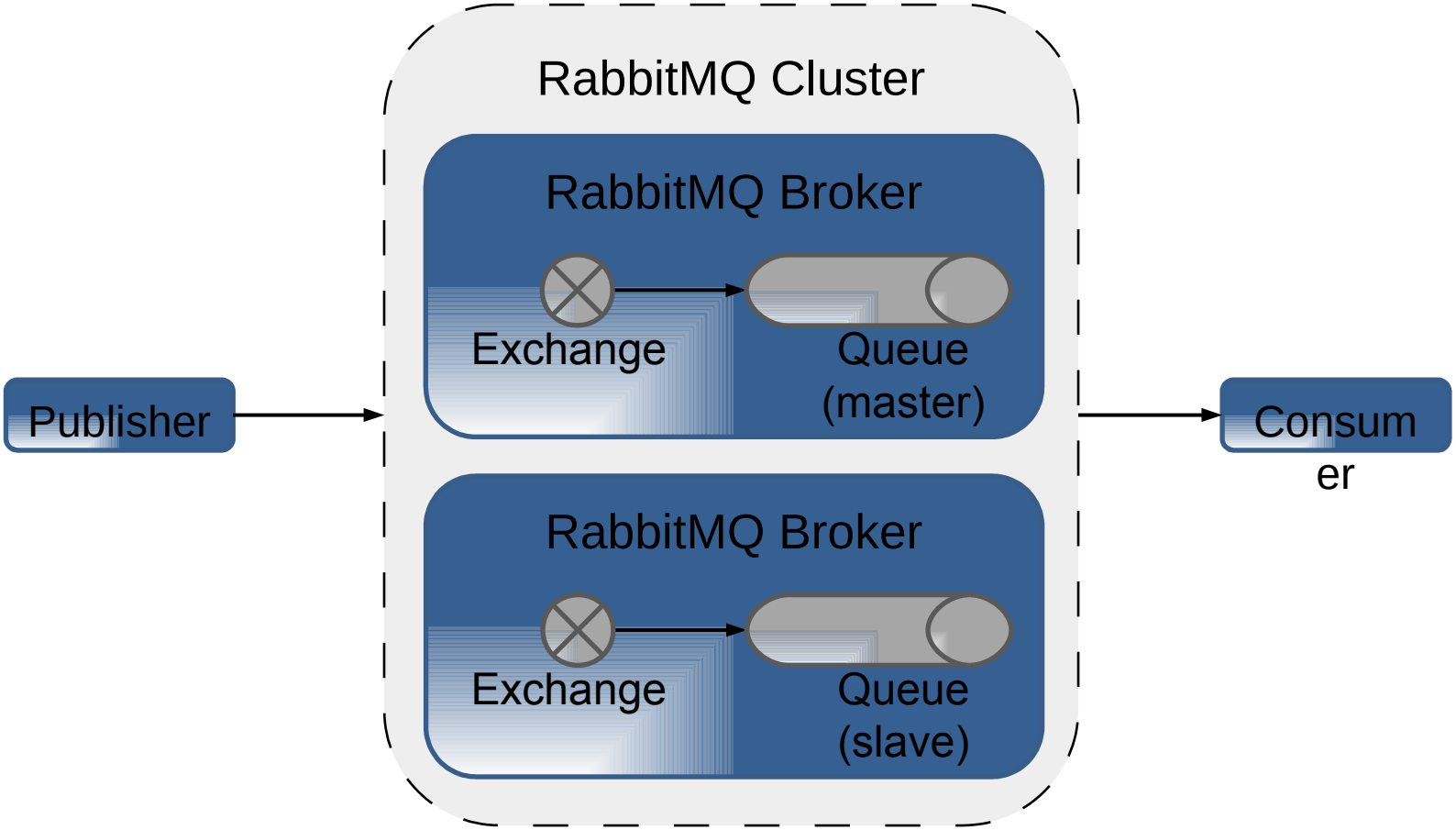
- Exchange: Resides on broker, responsible for filtering and routing messages. Stateless.
- Queue: Resides on broker, responsible for storing messages until the clients can consume them. Stateful.
- Exchanges route and filter messages to queues



RabbitMQ – Clustering and High Availability

- Through clustering and queue mirroring, it is possible to make RabbitMQ highly available
- Each mirrored queue has a designated master, and one or more slave copies
- Messages are kept in sync between the master and slaves
- If the node hosting the master queue goes down, a slave is promoted to master
- Processing continues on clients without message loss

RabbitMQ – HA Queues



RabbitMQ – Clustering and High Availability

- Through clustering and queue mirroring, it is possible to make RabbitMQ highly available
- Each mirrored queue has a designated master, and one or more slave copies
- Messages are kept in sync between the master and slaves
- If the node hosting the master queue goes down, a slave is promoted to master
- Processing continues on clients without message loss

Spring XD RabbitMQ Configuration for HA

- Insert hosts and ports of each RabbitMQ server in cluster, in **`addresses`** property of **`servers.yml`**

```
# RabbitMQ properties
spring:
  rabbitmq:
   addresses: 192.168.0.100:5672,192.168.0.101:5672
   username: guest
   password: guest
   virtual_host: /
   useSSL: false
   sslProperties:
```

Spring XD RabbitMQ Configuration for HA

- All Spring XD queues and exchanges are by default prefixed with **`xdbus.`**
- Create an HA policy in RabbitMQ that will apply to all queues matching the **`xdbus.`** prefix

```
$ rabbitmqctl set_policy ha-xdbus "^xdbus\." '{"ha-mode":"all"}'
```

Spring XD RabbitMQ Error Handling

- If consuming module fails to process a message, it will retry a default of 3 times
- Retries can be configured at the module or message bus level

```
messagebus:
  rabbit:
    default:
      maxAttempts:                    3
```

Spring XD RabbitMQ Error Handling

- After 3 failed attempts, the message is by default discarded
- To prevent discarding messages, enable dead lettering at the bus or module level
- Must enable policy in RabbitMQ

```
$ rabbitmqctl set_policy DLX "^xdbus\..*" '{"dead-
letter-exchange":"xdbus.DLX"}' --apply-to queues
```

Spring XD RabbitMQ Error Handling

- To enable at bus level, in **`servers.yml`**:

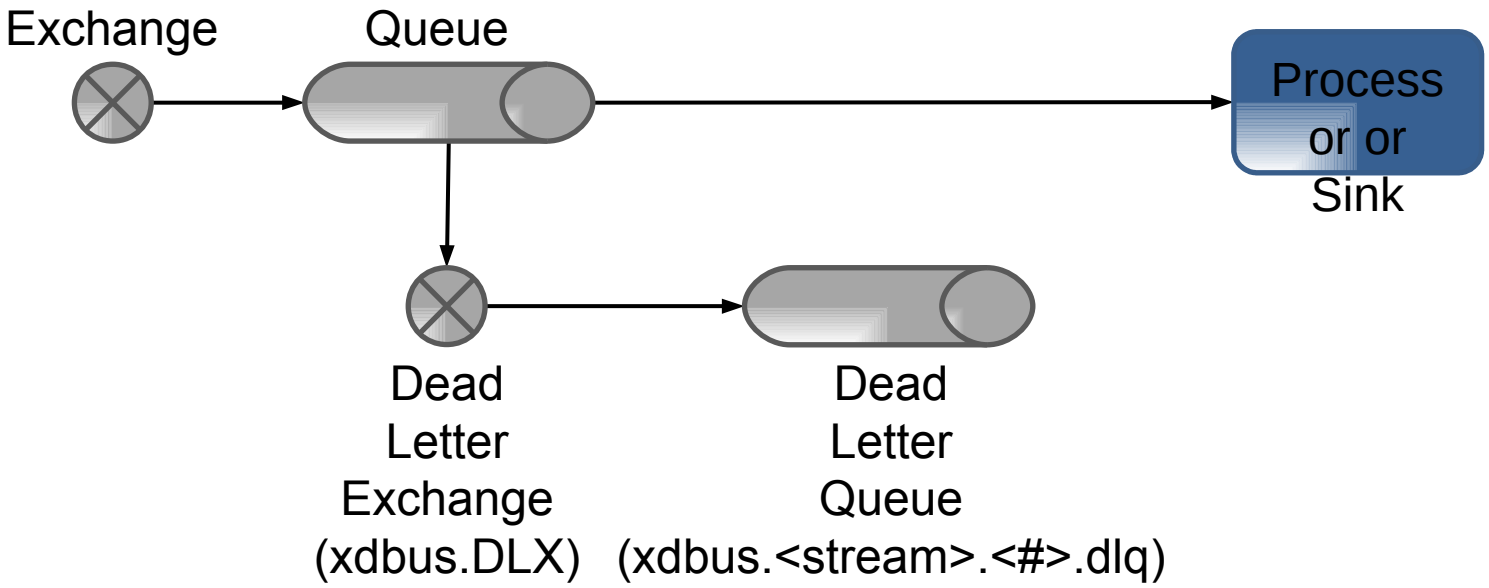
```
messagebus:
  rabbit:
    default:
      AutoBindDLQ:                    true
```

- To enable at module level (for all modules in stream):

```
--properties module.*.consumer.autoBindDLQ=true
```

Spring XD RabbitMQ Dead Lettering

- After maxAttempts have been exhausted, message is routed to Dead Letter Exchange, and onto the Dead Letter Queue



Spring XD RabbitMQ Dead Lettering

- Note that there is no built-in Spring XD mechanism to remove messages from the dead letter queue

Lab

Lab details ...