POLE STAR
www.polestarglobal.com

Purplefinder Enterprise Platform Messagng with ActiveMQ

Peter Potts

13th October 2010

Resources

- Manning Book: ActiveMQ in Action
- Apache Documentation & download: http://activemq.apache.org/
- 8 example applications in PEP: http://repository.enterprise.purplefinder.com
- Available on public Maven repositories.

Java Message Service (JMS)

- The JMS API is a MOM API for sending messages between two or more clients.
- JMS is part of Java EE.
- Apache ActiveMQ is an implementation of JMS API.

JMS Models

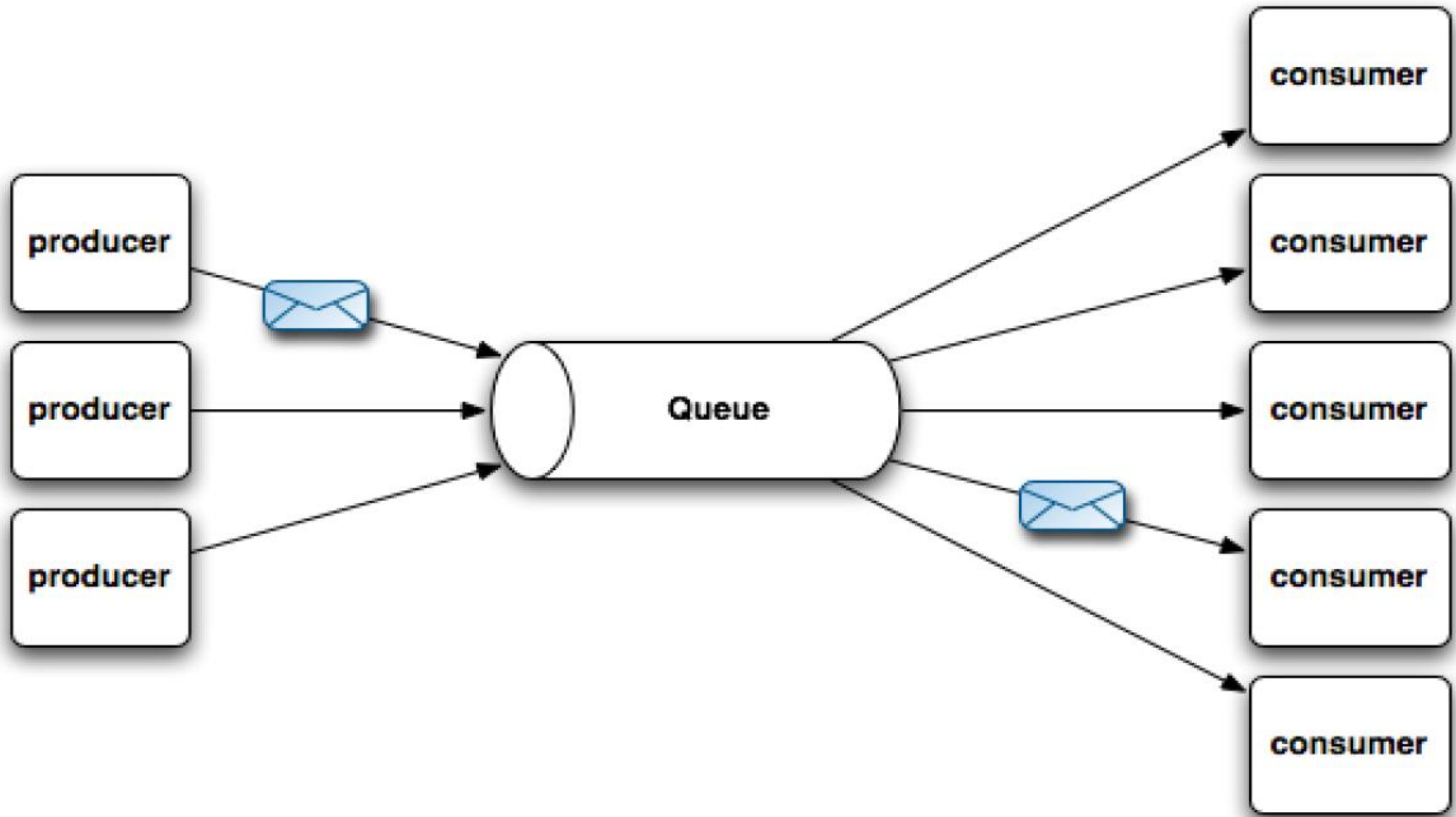
- **Queue** = Point-to-point model.
- **Topic** = Publish and subscribe model.

JMS Queue

- Point-to-point model.
- Producers and consumers.
- A sender posts messages to a particular queue.
- A receiver reads messages from that queue.
- Only one consumer gets the message.
- The producer and consumer **do not** have to be running at the same time.

POLE STAR
www.polestarglobal.com

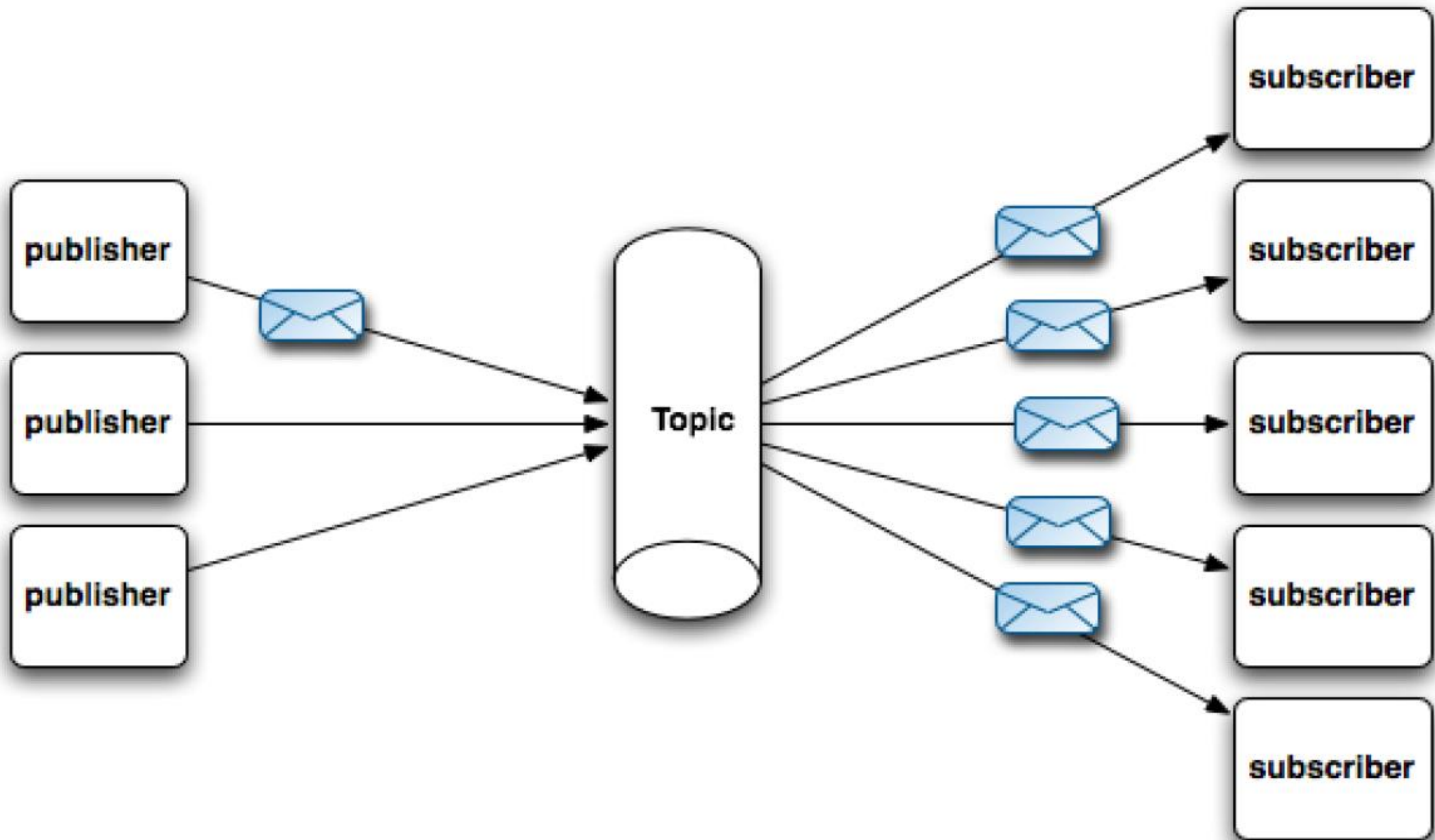
JMS Queue



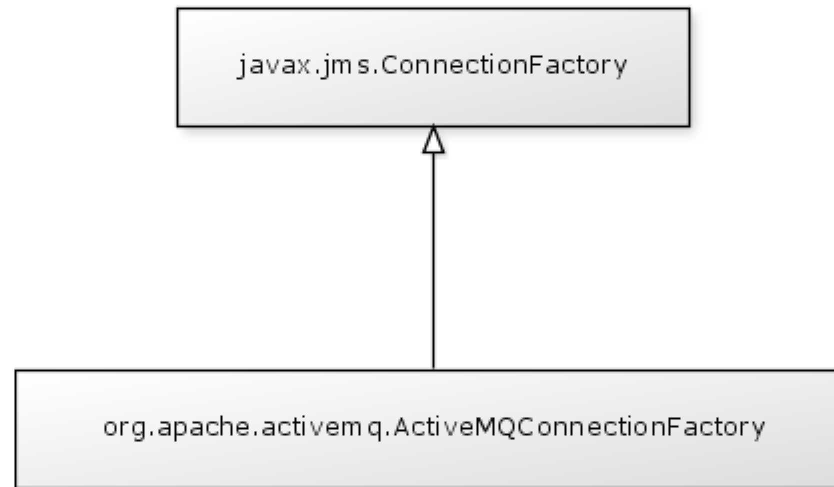
JMS Topic

- Publish and subscribe model.
- Publishers and subscribers.
- A message is published to a particular topic.
- Subscribers may register interest in receiving messages on a particular topic.
- Zero or more consumers get the message.
- The producer and consumer **do** have to be running at the same time.
- Except for **durable** subscriptions.

JMS Topic



Where does ActiveMQ meet JMS?



- Inject implementation of JMS connection factory.
- Use JMS API defined in javax.jms package.
- But this is only the client side of the story.

ActiveMQ Broker Service

org.apache.activemq.broker.BrokerService

- The JMS client application must connect to an ActiveMQ broker.
- The broker can be embedded or standalone.
- There can be a network of brokers.

POLE STAR
www.polestarglobal.com

How to send a text message?

```
val cf = new ActiveMQConnectionFactory("vm://brokerName")
val c = cf.createConnection
c.start
 val s = c.createSession(false, AUTO_ACKNOWLEDGE)
  val q = s.createQueue(“queueName”)
  val p = s.createProducer(q)
   val m = s.createTextMessage(“Hello World”)
   p.send(m)
  p.close
 s.close
c.stop
c.close
```

POLE STAR
www.polestarglobal.com

How to receive a text message?

```
val cf = new ActiveMQConnectionFactory("vm://brokerName")
val c = cf.createConnection
c.start
  val s = c.createSession(false, AUTO_ACKNOWLEDGE)
    val q = s.createQueue("queueName")
    val c = s.createConsumer(q)
      val m = c.receive
      println(m.getText)
    c.close
  s.close
c.stop
c.close
```

Synchronous versus Asynchronous

- Message listener can be used to receive messages asynchronously.
- Asynchronous to get better performance.
- Synchronous to avoid shared data issues.
- Mixing messaging passing and shared data is hard.
- Can always have multiple consumers to scale too.

Message types and payloads

- Message – No payload.
- TextMessage – String payload.
- MapMessage – Name/value pairs as payload.
- BytesMessage – Byte array payload.
- StreamMessage – Stream of primitive types.
- ObjectMessage – Serialized Java object.

What about loose coupling?

Request / Reply Messaging

- JMS does not formally define this.
- But their are headers and convenience classes.
- Probably best avoided.
- Apache Camel can help here.
- Use Apache Camel for this.

POLE STAR
www.polestarglobal.com

How do I started an embedded broker?

```
val b = new BrokerService {
  setBrokerName(“brokerName”)
  setUseJmx(false)
  setPersistent(false)
}
b.start
  printf("Press enter to quit: ")
  readLine
b.stop
```

What about persistence?

- AMQ – Overall default but superceded by KahaDB.
- Memory– No persistence default.
- KahaDB – Ultra fast and recommended.
- JDBC – Slow but makes transactions simpler.

KahaDB

- File based message store.
- Transactional journal for durability.
- Highly tuned for messaging.
- Scalable to 10,000 active connections per broker.

```
val b = new BrokerService {
  setBrokerName(“brokerName”)
  setTmpDataDirectory(new File(new File(SAN, brokerName), "transient"))
  setPersistenceAdapter(new KahaDBStore {
    setDirectory(new File(new File(SAN, brokerName), "persistent"))
  })
}
```

Connectivity

- Many protocols supported such as HTTP, HTTPS, IP multicast, SSL, Stomp, TCP, UDP, XMPP and NIO.
- OpenWire is the default protocol.
- OpenWire over TCP is optimal in general.
- Use OpenWire over NIO to scale massively.
- Suggest OpenWire not used as public API.

POLE STAR
www.polestarglobal.com

Basic Transport Connectors

- Client-to-broker communication.
- Client connects to embedded broker:

  vm://brokerName

- Client connects to standalone broker:

  tcp://host:port

e.g.

```
val cf = new ActiveMQConnectionFactory("tcp://bs:61616")
val bs = new BrokerService { addConnector("tcp://localhost:61616") }
```

Other Transport Connectors

- Client connects to dynamic network:

  discovery:(multicast://address:port?group=name)

  gives flexibility and failover
- Client connects to static network:

  failover:(tcp://host1:port1,tcp://host2:port2)

  gives failover only

POLE STAR
www.polestarglobal.com

Basic Network Connectors

- Broker-to-broker communication
- Broker connects to standalone broker

tcp://host:port

e.g.

val fbs = new BrokerService { addNetworkConnector("tcp://tbs:61616") }

val tbs = new BrokerService { addConnector("tcp://localhost:61616") }

POLE STAR
www.polestarglobal.com

Other Network Connectors

- Embedded broker connects to dynamic network

  multicast://address:port?group=name

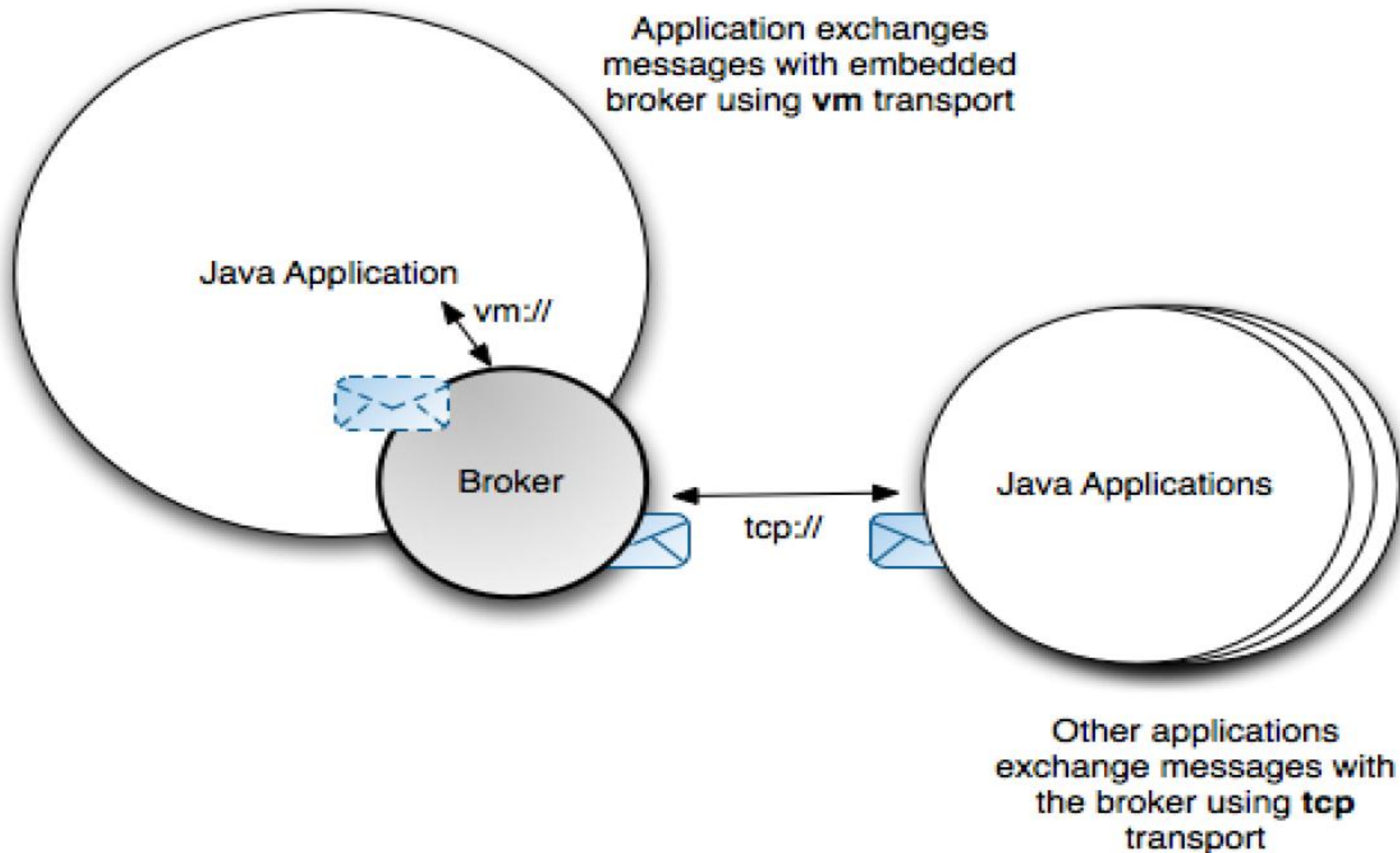
  gives flexibility, load balancing and failover
- Embedded broker connects to static network

  static:(tcp://host1:port1,tcp://host2:port2)

  gives load balancing and failover

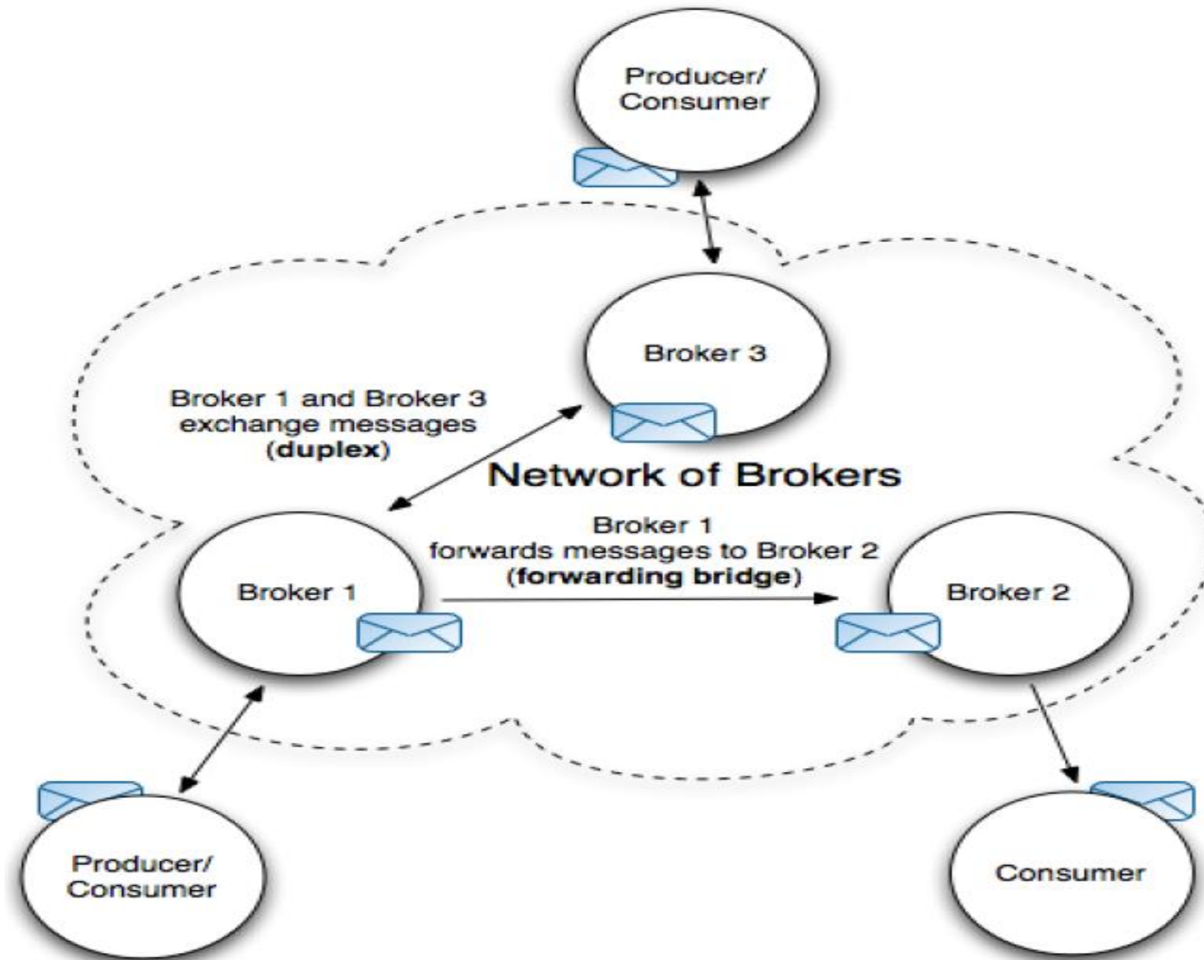
POLE STAR
www.polestarglobal.com

Simple Network Topology

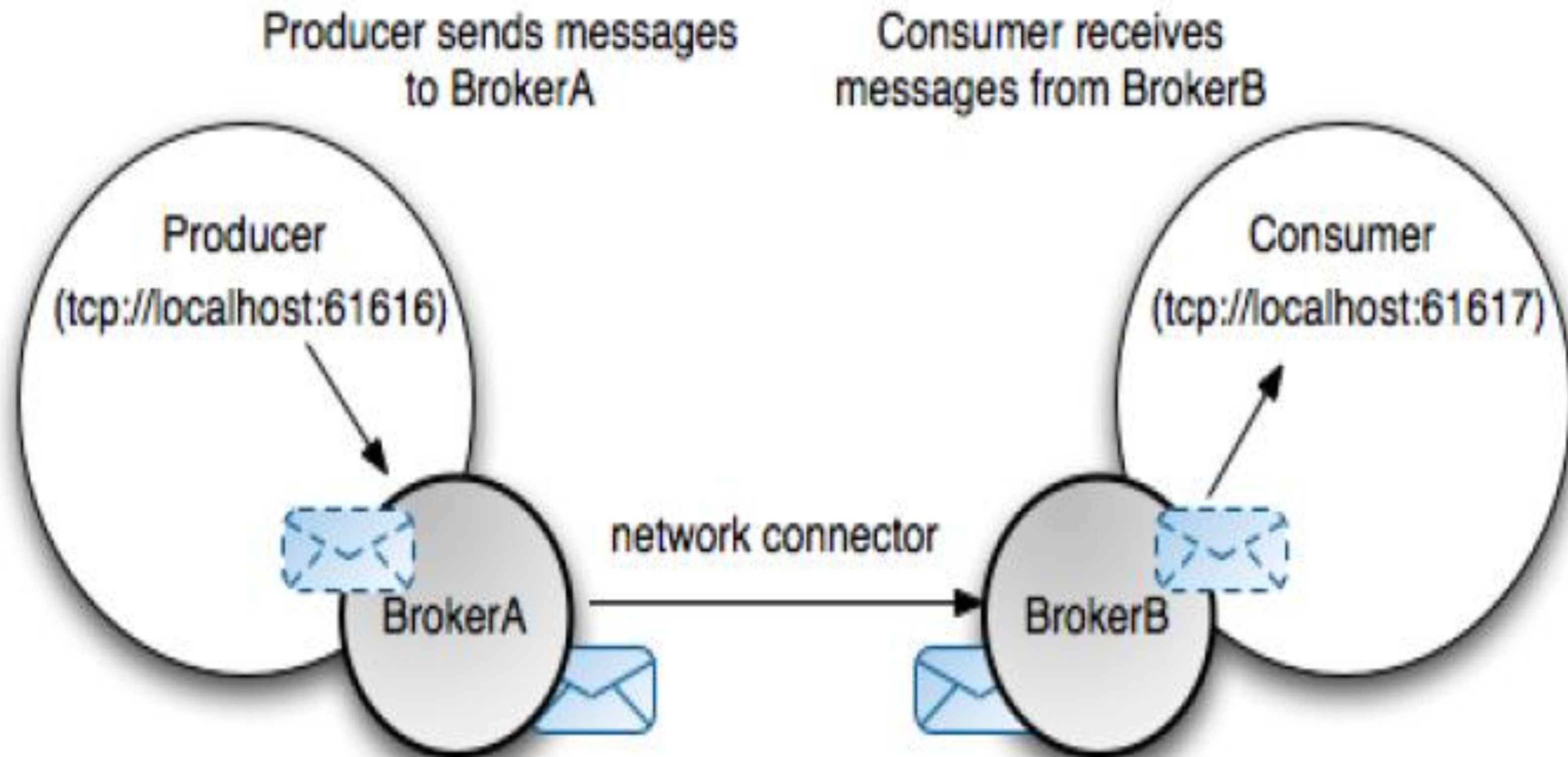


POLE STAR
www.polestarglobal.com

Complex Network Topology



Guaranteed Delivery Topology



POLE STAR
www.polestarglobal.com

Acknowledge is not a transaction

- Auto-acknowledge.

  `val s = c.createSession(false, Session.AUTO_ACKNOWLEDGE)`

  No protection after message has been received.

- Client acknowlege.

  `val s = c.createSession(false, Session.CLIENT_ACKNOWLEDGE)`

  Call `message.acknowledge` to commit a bunch.

  Call `session.recover` to rollback a bunch.

  As good as transaction for idempotent consumer.

Transactions

```
val s = c.createSession(true, Session.SESSION_TRANSACTED)
```

- For when you just can’t lose a message.
- Be sure you need it.
- It is actually hard to lose even one message using acknowledgements with persistence.
- Use Camel to achieve distributed transactions.

  Call `session.commit` to commit a bunch.

  Call `session.rollback` to rollback a bunch.

Redelivery Policy

```
val cf = new ActiveMQConnectionFactory {
  setBrokerURL("vm://brokerName")
  setRedeliveryPolicy(new RedeliveryPolicy {
    setInitialRedeliveryDelay(500L)
    setUseExponentialBackOff(true)
    setBackOffMultiplier(2.0)
    setMaximumRedeliveries(4)
  })
```

Transient and Permanent Errors

- Redeliver messages after transient error.
  i.e. Recover or rollback.
- Do not redeliver messages after a permanent error.
  i.e. Acknowledge or commit as if ok.
- Message can be setup to expire too.
- Apache Camel can help here.
- An undelivered message is forwarded to the Dead Letter Queue (DLQ).
- Default DLQ is a queue called “ActiveMQ.DLQ”.

Java Management Extensions (JMX)

- JMX is enabled by delfault for a broker.
- Allows status of the broker to interrogated.
- Allows queues to be cleared.

Advisory topics

- Subscribe to topics at “ActiveMQ.Advisory.>”.
- Event driven status.
- Populate a Comet driven Web dashboard.
- E.g. DLQ advisory messages on topic “ActiveMQ.Advisory.MessageDLQd.>”.
- E.g. Slow consumer advisory messages on topic “ActiveMQ.Advisory.SlowConsumer.>”.

Statistics Broker Plugin

- Add this plugin to the broker.
- Send an empty message to "ActiveMQ.Statistics.Broker" for example.
- Receive a message of name/value pairs by a transient reply queue.

Security

- Client-side:

```
val cf = new ActiveMQConnectionFactory(“vm://brokerName")
val c = cf.createConnection(userName, password)
```

- Server-side:

```
val bs = new BrokerService {
  setBrokerName("brokerName")
  setPlugins(??? authentication & authorization ???)
}
```

- Send, receive & admin rights of queues and topics controlled by users and groups.
- Wait for Security using LDAP talk.

POLE STAR
www.polestarglobal.com

XML and Spring

- The documentation on ActiveMQ frequently suggests using XML configuration files and also Spring with even more XML configuration files.
- Why pretend your XML is confuration when it is actually code?
- Anyway, the IDE helps with auto-completion and Scala makes it look good anyway.
- Finally, use Guice to inject when you can and Spring JavaConfig when you really must.

Clustering

- Queue consumer clusters
- Broker clusters
- Discovery of brokers
- Network of brokers
- Master slave
- Replicated message stores

Apache Camel

- Integrates well with Apache Camel.
- Wait for Enterprise Interation Patterns with Camel talk.