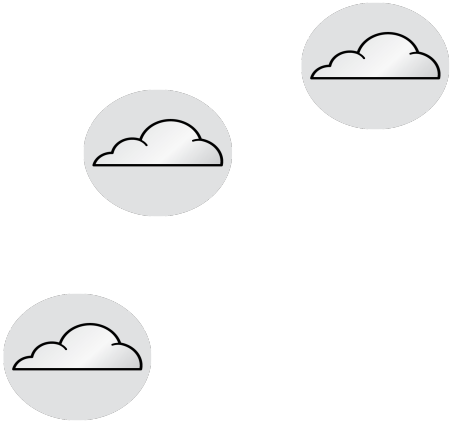
Introduction to Jboss Fuse

Challenges of Integrating the Extended Enterprise

- **Integrating all enterprise assets**
  many business assets are at the edges (outside the data center)
- **Avoiding centralizing all services**
  hub-and-spoke architectures are limiting
- **Getting deployed quickly**
  lengthy development cycles reduce ROI
- **Controlling run-away costs**
  from initial outlay to on-going maintenance
- **Avoiding creating yet-another legacy system**
  want new technology, need to stay agile
- **Creating infrastructure that can scale up...and down**
  elasticity should be easy

How do you integrate everything...



cloud / SaaS apps

partners

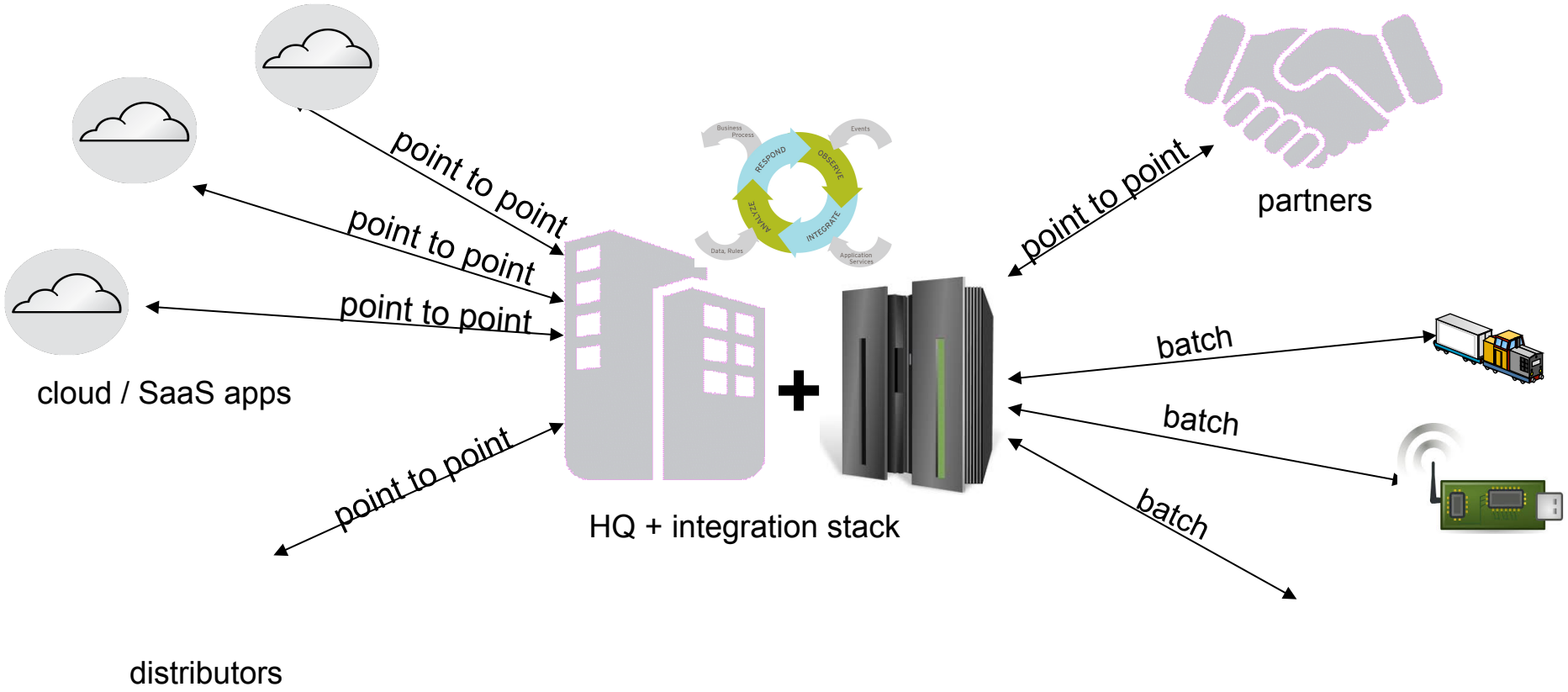
HQ + integration stack

distributors

devices

...when the enterprise extends far beyond the data center?

Traditional integration stack is too expensive, too difficult to manage and maintain...



...and many make do with batch delivery and hub-and-spoke architecture

What JBoss Fuse Brings to Red Hat's Customers:

Integration that extends to the edges of enterprise

Easy to deploy – sophisticated tooling, connectors,small footprint makes it easy to deploy with less hardware and limited IT staffing

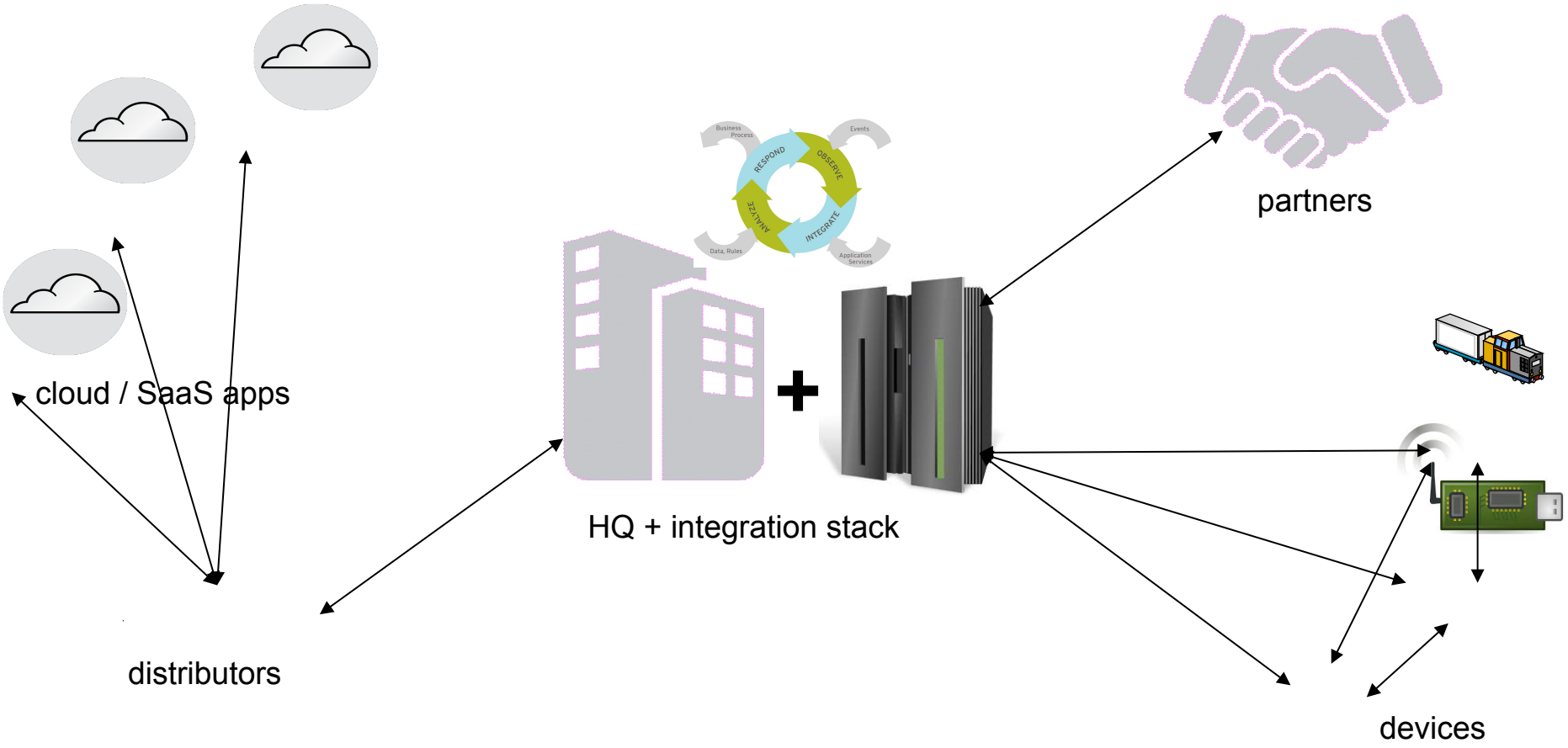
Many deployment options – deploy on-premise or in the cloud in any configuration, and change on the fly with automated provisioning

Standards-based – commitment to industry standards ensures that JBoss infrastructure is easy to modify and maintain

Centralized management – innovative tooling makes it ease to configure, deploy, manage and maintain integration infrastructure

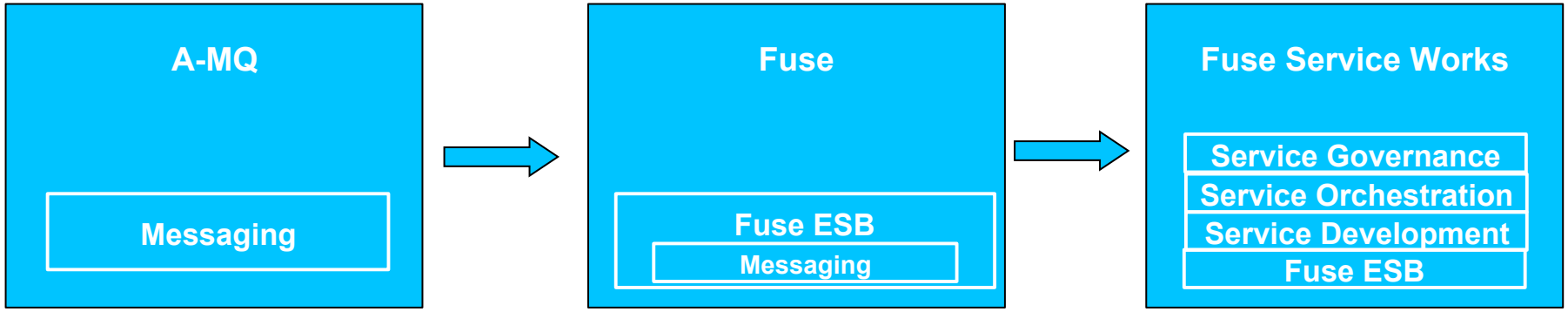
Open source – less expensive and pay as you go to reduce over all costs in all stages of the project

With JBoss Fuse, You Can Integrate Everything...



JBoss Fuse Integration Product line

Additive capabilities to fit different use cases



Messaging Platform

Integrate applications, devices by notification or exchange of data using multiple protocols in any runtime

Integration Platform

Mediate, transform, route and connect between loosely coupled components, services and applications using enterprise integration patterns

Business Services Platform

Develop and choreograph business services, manage lifecycle, define and enforce service policy and monitor service activity

JBoss Fuse – Open Source ESB

Development and tooling

Develop, test, debug, refine, deploy

JBoss Developer Studio

Web services framework
Web services standards, SOAP, XML/ HTTP, RESTful HTTP

Apache CXF

Integration framework
Transformation, mediation, enterprise integration patterns

Apache Camel

Reliable Messaging
JMS/STOMP/NMS/MQTT, publishing-subscribe/point-2-point, store and forward

Apache ActiveMQ

Container
Life cycle management, resource management, dynamic deployment, security and provisioning

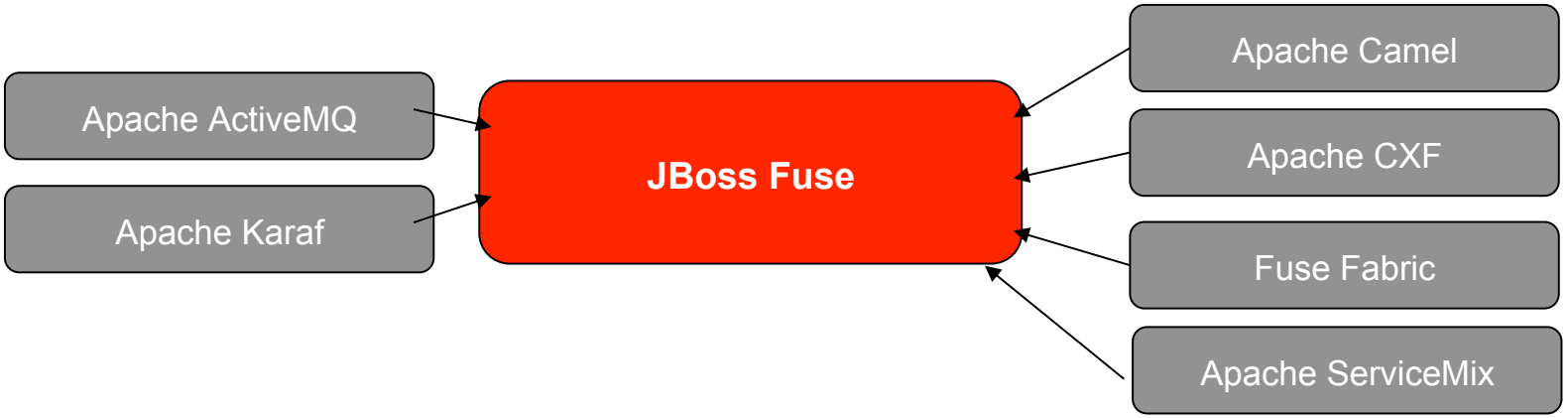
Apache Karaf + Fuse Fabric

Management and monitoring

System and web services metrics, automated discovery, container status, automatic updates

JBoss Operations Network
+
JBoss Fabric Management Console (hawtio)

JBoss Fuse – Open Source Heritage



* Many more OSS projects not listed from: jboss.org, codehaus.org, sourcefourge.net, apache.org

redhat.

Reliable Messaging

Reliable Messaging Included In:

- Red Hat JBoss A-MQ
- Red Hat JBoss Fuse
- Red Hat JBoss Fuse Service Works


ActiveMQ

What is Apache ActiveMQ?

- Top level Apache Software Foundation project
- Wildly popular, high performance, reliable message broker
  - Clustering and Fault Tolerance
  - Supports publish/subscribe, point to point, message groups, out of band messaging and streaming, distributed transactions, …
- Myriad of connectivity options
  - Native Java, C/C++, and .NET
  - AMQP 1.0, MQTT 3.1, STOMP (1.0, 1.1, 1.2), and OpenWire
  - STOMP protocol enables Ruby, JS, Perl, Python, PHP, ActionScript, ...
- Embedded and standalone deployment options
  - Pre-integrated with open source integration and application frameworks
  - Deep integration with Spring Framework, OSGi, and Java EE

Configuring Transport Connectors

- Configured in broker for client connections
- TCP - most used; socket connections using binary OpenWire protocol
- NIO - like TCP, except uses Java NIO to reduce number of threads managing all connections
- SSL - secure TCP connection
- STOMP - text based protocol; facilitates multiple language integration
- MQTT - lightweight publish / subscribe protocol
- VM - enables efficient in-process connections for embedded broker

Examples:

```
<transportConnector uri="tcp://0.0.0.0:61616"/>
<transportConnector uri="nio://0.0.0.0:61616"/>
<transportConnector uri="stomp://0.0.0.0:61617"/>
<transportConnector uri="stomp+nio://0.0.0.0:61617"/>
```

Configuring Client Connections

- Matches configuration for transport connectors in the broker
- Set as broker url on JMS connection factory
- Options can be set in the url as key/value params or directly on the connection factory

Format:

tcp://hostname:port?key=value

Examples:

tcp://myhost:61616?trace=false&soTimeout=60000

Lot more details at:

http://activemq.apache.org/configuring-transports.html

Configuring Client Connections - Wrapper Transports

- Augment / wrap client side connections
- Failover - automatic reconnection from connection failures
- Fanout - simultaneously replicate commands and message to multiple brokers
- Discovery - automatic discovery of brokers

Format:

wrapper:(tcp://hostname:port?key=value)

Examples:

failover:(tcp://master:61616,tcp://slave:61616)

failover:(tcp://virtualIp:61616)

fanout:(static:(tcp://host1:61616,tcp://host2:61616))

discovery:(multicast://default)?initialReconnectDelay=100

Configuring Persistence Adapters

- File system based
  - kahaDB - recommended; improved scalability and quick recovery
  - levelDB - high throughput, quick recovery, better indexing
  - *levelDB replicated - (tech preview only)*
- RDBMS based
  - jdbcPersistenceAdapter - quick and easy to setup
  - journaledJDBC - faster than pure JDBC; file journaling with long term JDBC storage
- Memory based
  - memoryPersistenceAdapter – testing only; same as <broker persistent="false">

Configuring Network Connectors

- Used to connect a broker to other brokers in the network
- Matches configuration for transport connectors in the broker

Format:

tcp://hostname:port?key=value

Examples:

<networkconnector uri="static:(tcp://myhost:61616)"/>

<networkconnector

uri="masterslave:(tcp://master:61616?soTimeout=60000,tcp://slave:61616)"/>

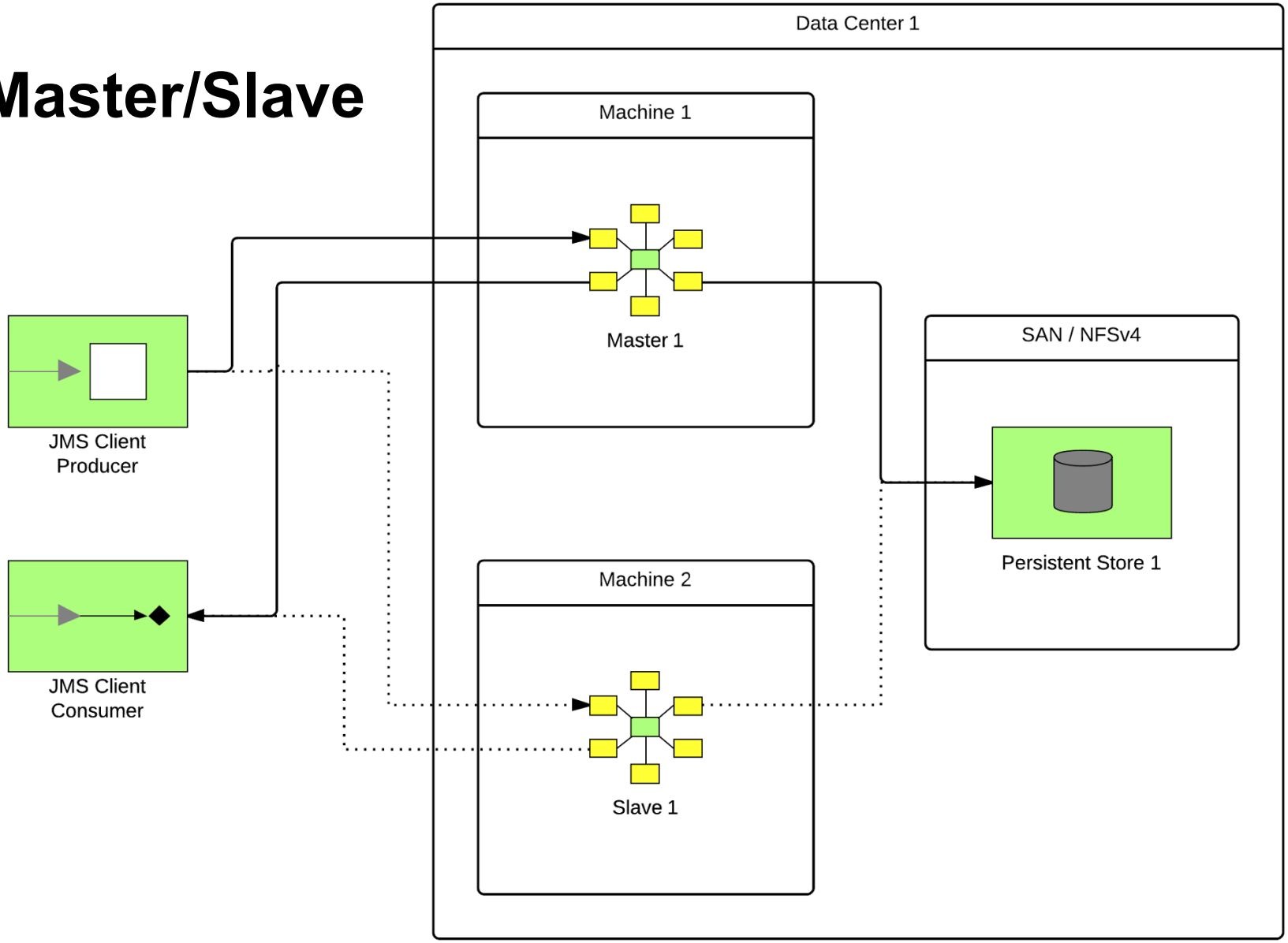
Lot more details at:

http://activemq.apache.org/networks-of-brokers.html

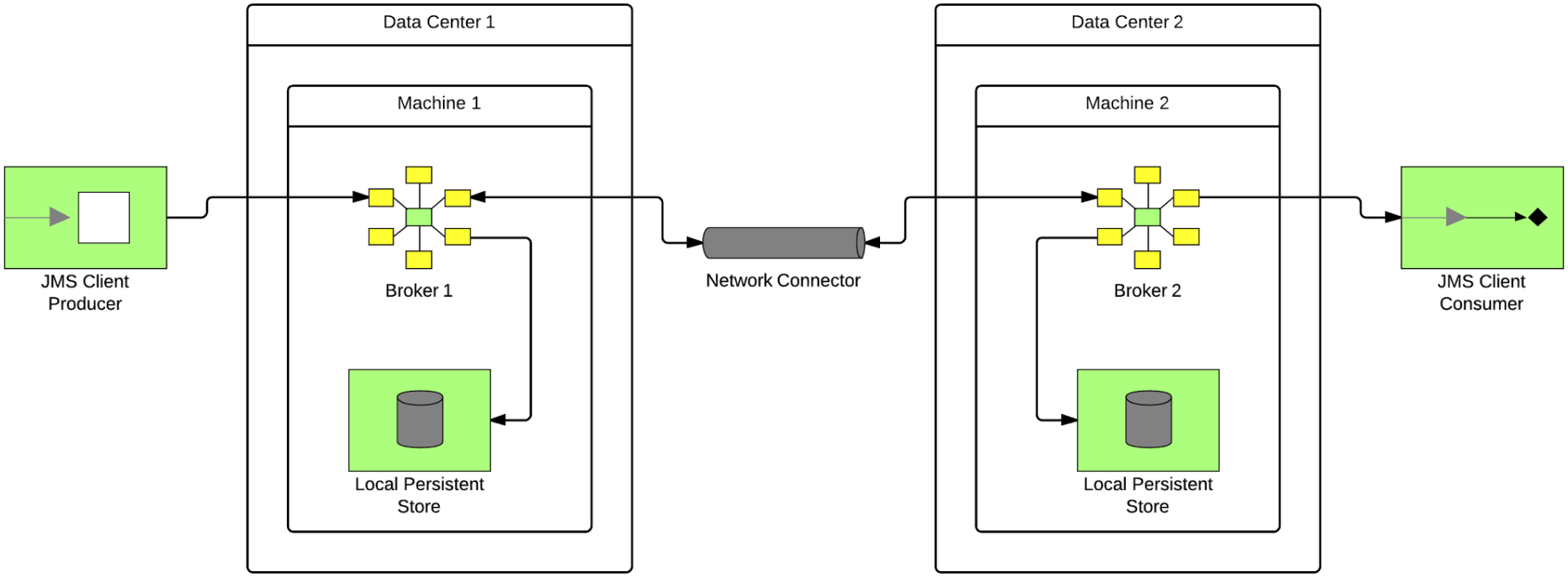
High Availability

- Two complementary approaches:
  - **Master/Slave** - access to persistent messages after broker failure
    - A given message is in one and only one broker (persistence store)
    - If a broker instance fails, all persistent messages are recoverable upon broker restart
    - Master/Slave allows a 2nd broker instance (slave) to be ready to process persistent messages upon master (1st broker) failure
    - Clients should use failover transport to automatically connect to slave
      - failover:(tcp://master:61616,tcp://slave:61616)?randomize=false
  - **Network of Brokers** - scale out message processes
    - Messages can be load balanced to consumers across multiple brokers
    - A message can be forwarded to another broker when a valid consumer is present
    - Brokers can be configured to prioritize local & nearby brokers to reduce network traffic

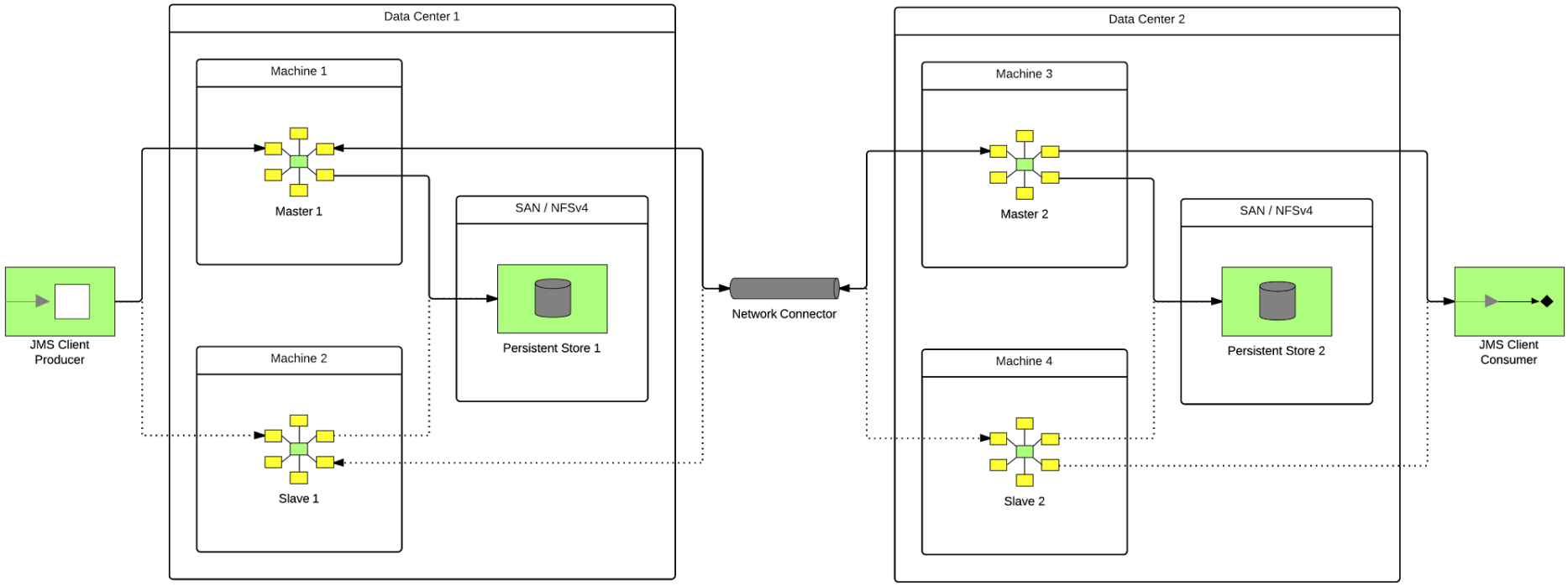
Master/Slave



Network of Brokers



Network of Master/Slave



redhat.

Integration

Integration Included In:

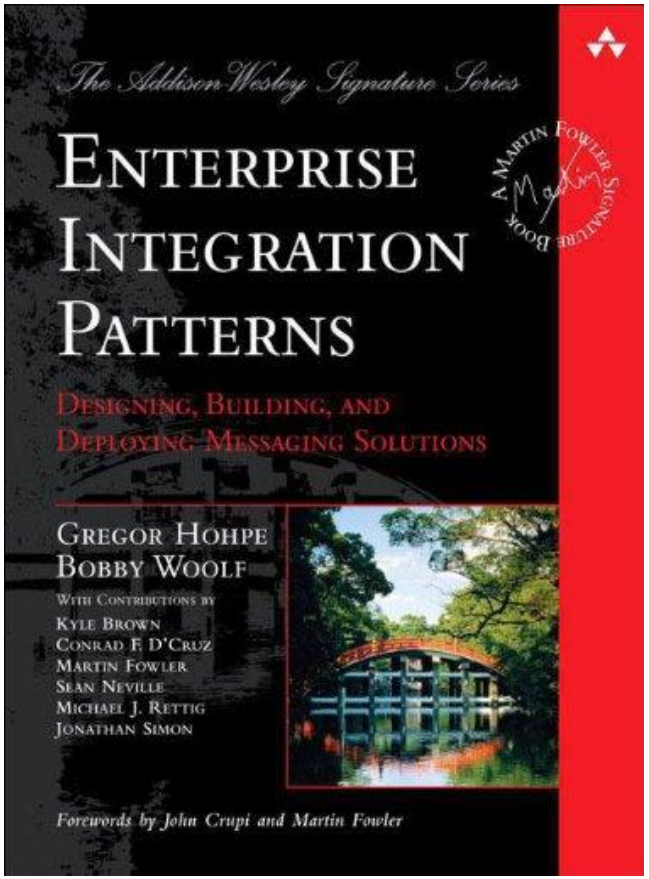
- Red Hat JBoss Fuse
- Red Hat JBoss Fuse Service Works

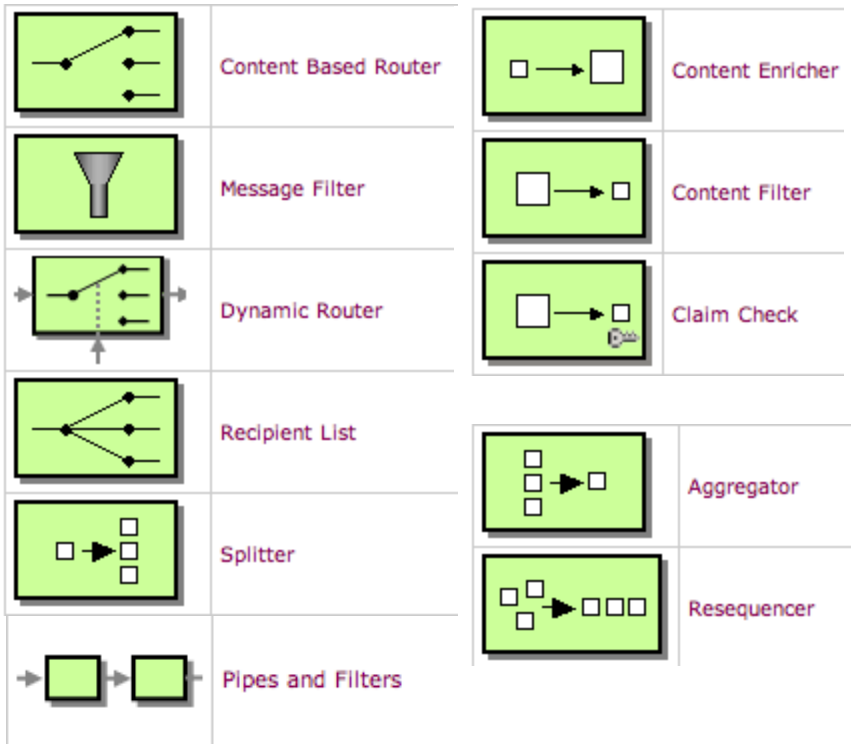

Apache
Camel

What is Apache Camel?

- Implementation framework for Enterprise Integration Patterns (EIP)
- Speeds time to solution and provides multiple connectivity options
- Popular and vibrant community
- Started as a sub-project of Apache ActiveMQ in March 2007
- 80-100k artifact downloads a month
- 120k website views a month
- 1,000+ user mailing list posts per month
- 145+ Components and growing

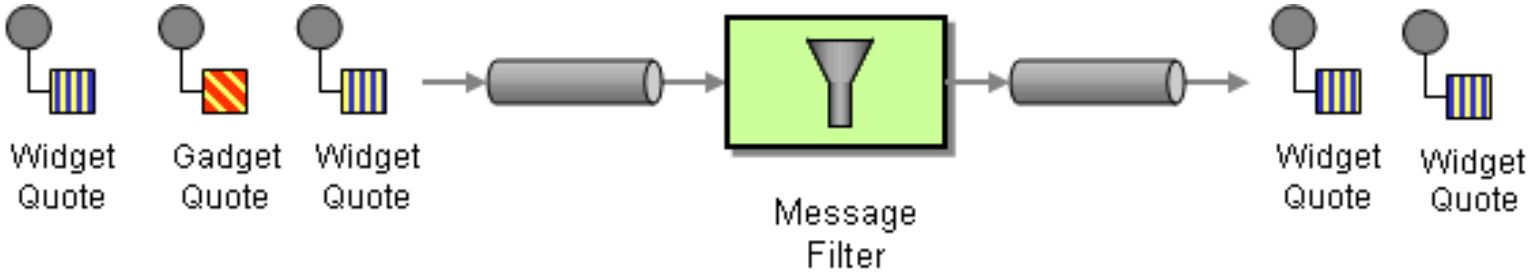
50+ Enterprise Integration Patterns





http://camel.apache.org/eip

Camel Example - Java DSL



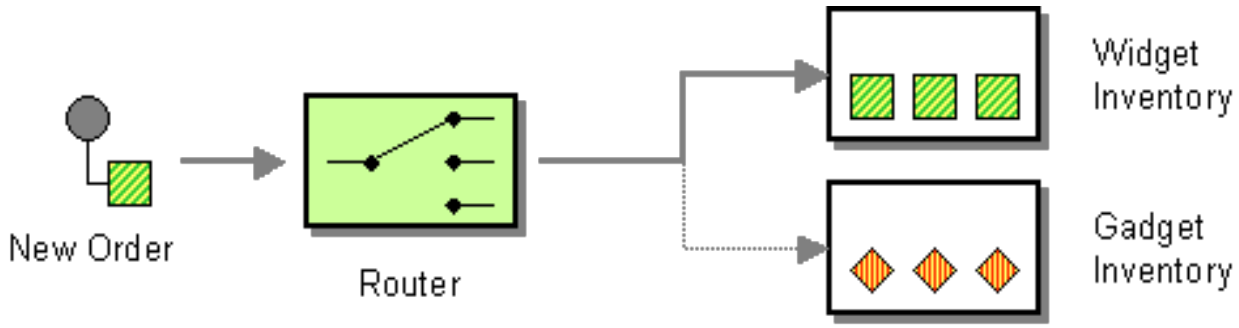

```
import org.apache.camel.builder.RouteBuilder;

public class FilterRoute extends RouteBuilder {

  public void configure() throws Exception {
    Endpoint A = endpoint("activemq:queue:quote");
    Endpoint B = endpoint("mq:quote");
    Predicate isWidget = xpath("/quote/product = 'widget'");

    from(A).filter(isWidget).to(B);
  }
}
```

Camel Example - Spring DSL




```
<camelContext>
  <route>
    <from uri="activemq:New.Orders"/>
    <choice>
      <when>
        <xpath>/order/product = 'widget'</xpath>
        <to uri="activemq:Orders.Widgets"/>
      </when>
      <otherwise>
        <to uri="activemq:Orders.Gadgets"/>
      </otherwise>
    </choice>
  </route>
</camelContext>
```

150+ Endpoint Components

activemq	cxf	flatpack	jasypt
activemq-journal	cxfrs	freemarker	javaspace
amqp	dataset	ftp/ftps/sftp	jbi
atom	db4o	gae	jcr
bean	direct	hdfs	jdbc
bean validation	ejb	hibernate	jetty
browse	esper	hl7	jms
cache	event	http	jmx
cometd	exec	ibatis	jpa
crypto	file	irc	jt/400

150+ Endpoint Components

language	properties	seda	stream
ldap	quartz	servlet	string-template
mail/imap/pop3	quickfix	sip	test
mina	ref	smooks	timer
mock	restlet	smpp	validation
msv	rmi	snmp	velocity
nagios	rnc	spring-integration	vm
netty	rng	spring-security	xmpp
nmr	rss	spring-ws	xquery
printer	scalate	sql	xslt

Endpoint Configuration Example

Programmatic Example:

```
...
FileEndpoint fileEp = new FileEndpoint();
fileEp.setFile(new File("/some/dir"));
fileEp.setDelete(true);
fileEp.setReadLock("changed");

from(fileEp).to(...);
...
```

URI Example:

```
...
from("file:///some/dir?delete=true&readLock=changed").to(...);
...
```

Camel Exchange

Exchange

Exchange Properties

Key	Value
...	...

In Message

Message Headers

Key	Value
...	...

Message Body

Message Attachments x n

Out Message

Message Headers

Key	Value
...	...

Message Body

Message Attachments x n

19 Data Formats

bindy	protobuf
castor	serialization
csv	soap
crypto	syslog
dozer	tidy markup
flatpack	xml beans
gzip	xml security
hl7	xstream
jaxb	zip
json	

Data Format Example

Input XML File:

```
<root>
  <child1>text1</child1>
  <child2>text2</child2>
</root>
```

Output JSON File:

```
{"root": {"child1": "text1",
          "child2": "text2"}}
```

Camel Route:

```
...

from("file:///xmlsourcedir")
  .unmarshal().jaxb()
  .process(...)
  .marshal().json()
  .to("file:///jsondestdir");
...
```

redhat.

Web Services

Web Services Included In:

- Red Hat JBoss Fuse
- Red Hat JBoss Fuse Service Works

Apache CXF

What is Apache CXF?

- Apache CXF is an open source services framework. CXF helps you build and develop services using frontend programming APIs, like JAX-WS and JAX-RS.
- These services can speak a variety of protocols such as SOAP, XML/HTTP, RESTful HTTP, or CORBA and work over a variety of transports such as HTTP, JMS or JBI

Standards Support

JSR Support

- JAX-WS - Java API for XML-Based Web Services (JAX-WS) 2.0 - JSR-224
- Web Services Metadata for the Java Platform - JSR-181
- JAX-RS - The Java API for RESTful Web Services - JSR-311
- SAAJ - SOAP with Attachments API for Java (SAAJ) - JSR-67

WS-* & Related Specifications Support

- Basic support: WS-I Basic Profile 1.1
- Quality of Service: WS-Reliable Messaging
- Metadata: WS-Policy, WSDL 1.1 - Web Service Definition Language
- Communication Security: WS-Security, WS-SecurityPolicy, WS-SecureConversation, WS-Trust (partial support)
- Messaging Support: WS-Addressing, SOAP 1.1, SOAP 1.2, Message Transmission Optimization Mechanism (MTOM)

Transports and Bindings

- **Transports:** HTTP, Servlet, JMS, In-VM and many others via the Camel transport for CXF such as SMTP/POP3, TCP and Jabber
- **Protocol Bindings:** SOAP, REST/HTTP, pure XML
- **Data bindings:** JAXB 2.x, Aegis, Apache XMLBeans, Service Data Objects (SDO), JiBX
- **Formats:** XML Textual, JSON, FastInfoset
- Extensibility API allows additional bindings for CXF, enabling additional message format support such as CORBA/IIOP

redhat.

Red Hat JBoss Fuse Demo

What is Fuse Fabric?

Management for Fuse environments that enable clustering, simplified deployments, configuration management, grouping, dynamic discovery, elasticity of deployments, and cloud-ready deployments

Problems - Deploying & Maintenance

- Installing brokers on multiple hosts
- ssh, untar, set directories and environment
- Setting configuration manually for every broker
- copying xml config, tweaking, testing
- Updating configuration across cluster
- Upgrading brokers

It’s a very tedious and error-prone process

Problems - Clients

- Topology is very “static”
- Clients need to be aware of topology
- Clients need to know broker & service locations
- Changes are not easy as clients need to be updated
- Adding new resources (brokers & services) requires client updates
- Not suitable for “cloud” deployments

Fuse Fabric makes deployments more “elastic”

Key Features

- Support Hybrid deployments - on premise, cloud, both
  - Endpoints can be relocated
  - Endpoints can be load balanced
  - Endpoints can be elastic
  - Endpoints can be highly available
- Distributed Configuration
  - Configuration may be accessed across multiple domains
  - Configuration is highly available
- Distributed Management
  - Easy elastic scaling of services
  - Monitoring and control of resources