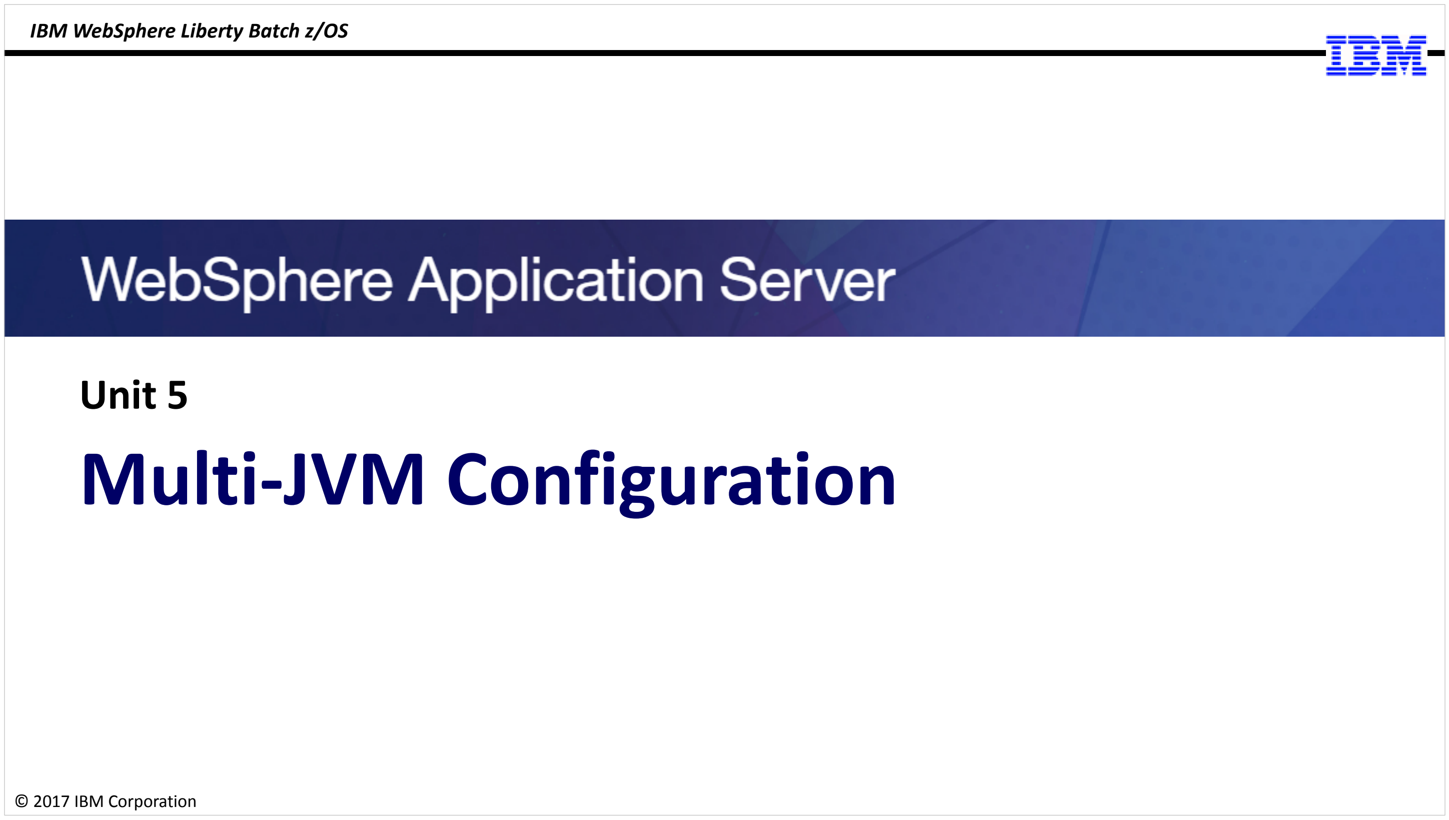
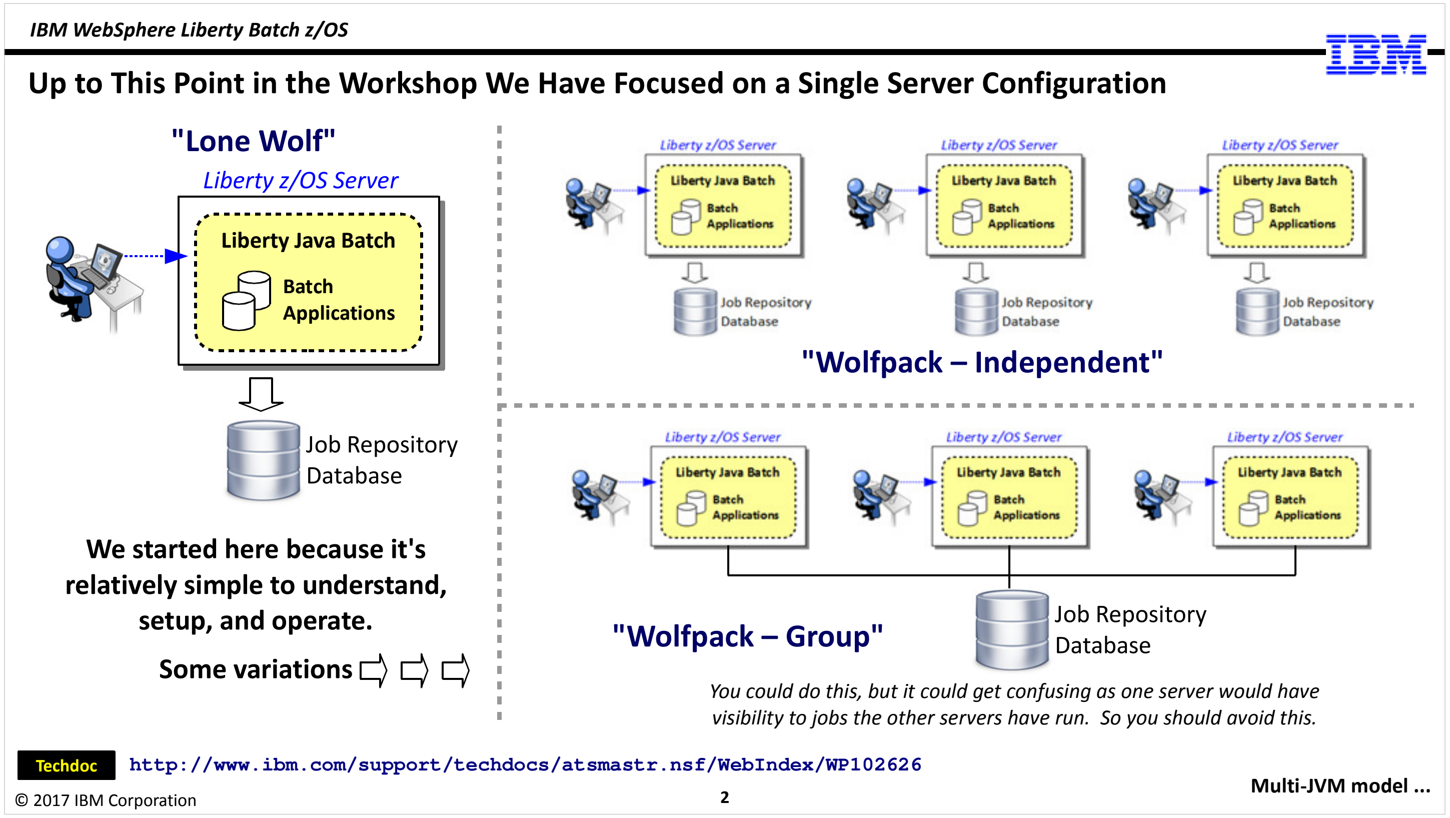


IBM WebSphere Liberty Batch z/OS

WebSphere Application Server

Unit 5

Multi-JVM Configuration



IBM WebSphere Liberty Batch z/OS

Up to This Point in the Workshop We Have Focused on a Single Server Configuration

We started here because it's relatively simple to understand, setup, and operate.

Some variations ⇨ ⇨ ⇨

You could do this, but it could get confusing as one server would have visibility to jobs the other servers have run. So you should avoid this.

Techdoc http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102626

Multi-JVM model ...

© 2017 IBM Corporation 2

We've been working with a single server through the first three labs. We created more servers in the lab to Unit 2, but we didn't use them. Now we're going to make use of them.

In addition to the truly single server configuration we've been working with, you could configure multiple single servers such as what's shown in the upper-right of the chart. In that case each user has their own server, and their own set of DB2 tables. They're not really sharing anything with anyone else.

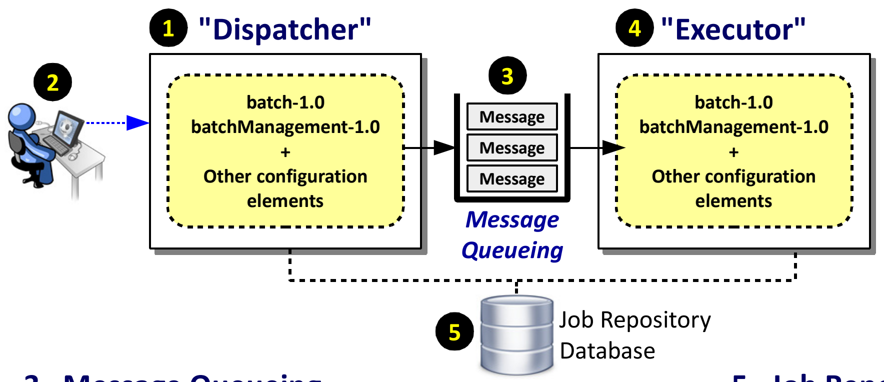
The lower-right portion of the chart shows multiple single servers, but they're all sharing the same DB2 tables. This can be done, but you probably don't want to do this. It could get confusing as each server would be able to see jobs run by the other servers.

If what you're seeking is the ability to effectively configure and manage an environment where multiple servers are running batch jobs, then what you want is what we're going to discuss in this unit.

The WP102626 Techdoc at the URL shown on the chart has a discussion of various topology designs. If you're curious about the different ways things can be done, and the pros/cons of each, take a look at that document.

IBM WebSphere Liberty Batch z/OS

The Multi-JVM Model



1. **Dispatcher Server(s)**
   This server is configured with the Java Batch features as well as other XML elements that define it as a Dispatcher and provide information about the JMS queue to use.
2. **Job Interfaces**
   The same interfaces we've seen earlier apply: batchManager, batchManagerZos, the REST interface, the AdminCenter.
3. **Message Queueing**
   A message queue sits between the Dispatcher and the Executor. The Dispatcher places a job submission message on the queue; the Executor picks it up and executes.
4. **Executor Server(s)**
   This server is configured with the Java Batch features as well as other XML element that define it as an Executor server.
   Configurable JMS activation specs are defined to indicate which messages to pick up from the queue.
5. **Job Repository Database**
   To use the Multi-JVM design you must have employ a relational database capable of sharing data between multiple servers.

This part is key, so let's take a quick look at that before getting into other details ...

3

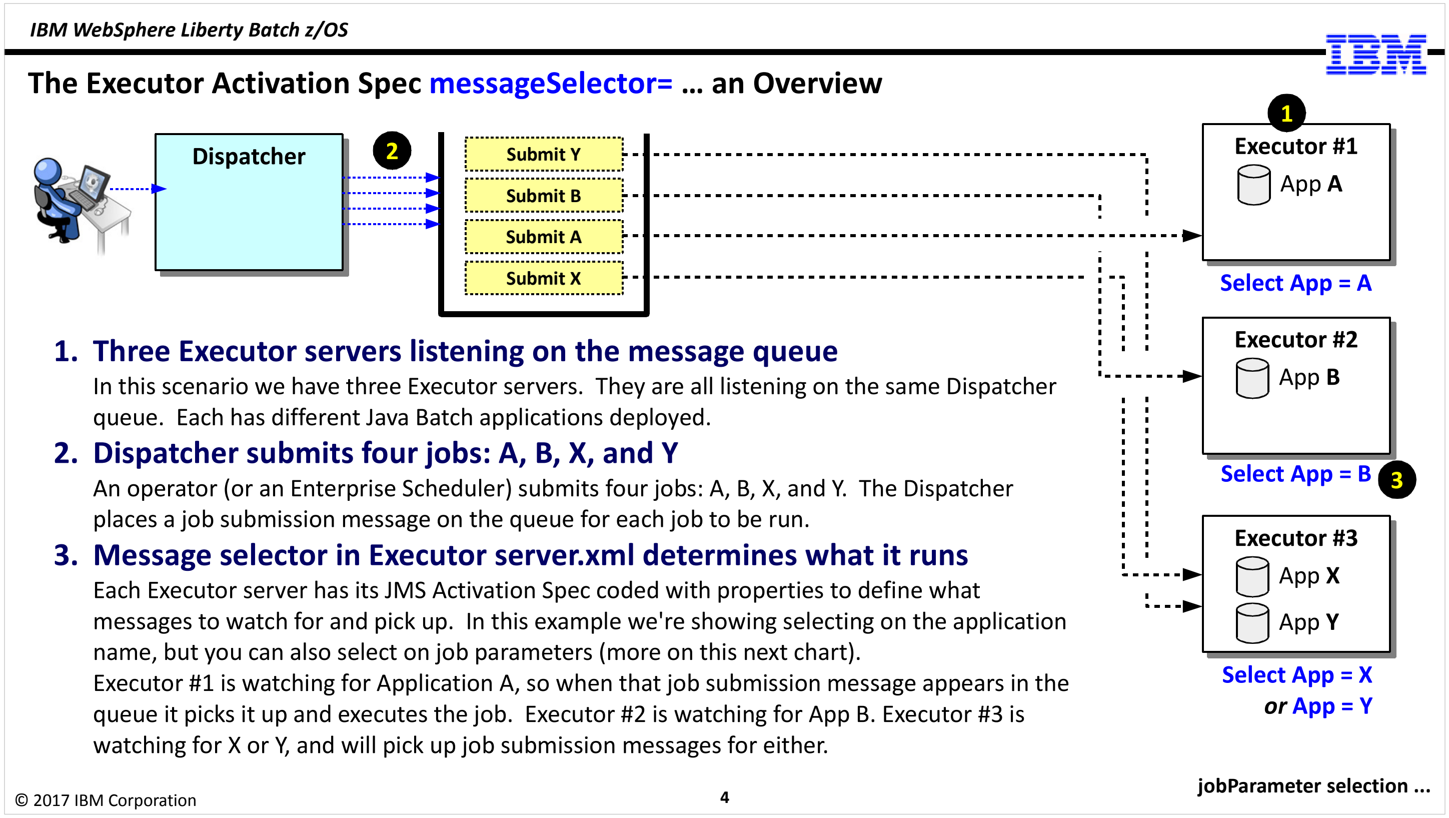
messageSelector overview ...

Let's focus on what we call the "Multi-JVM model," that name referring to the fact that we have at least two servers (and perhaps more) arranged into a configuration where one serves as the interface for job submission, and another serves as the server that runs the batch job. The former is referred to as the "Dispatcher" because it handles the job submission and "dispatches" the job to be run elsewhere; the latter is referred to as the "Executor" because it executes the batch jobs. Let's walk through the numbered circles:

1. The "Dispatcher" server a server that is configured to be a dispatcher (with the <batchJmsDispatcher> element), the batch-1.0 and batchManagement-1.0 features, and JMS definitions that allow it to understand how to put the job submission messages on the queue.
2. The job submission interfaces to the dispatcher are exactly the same as we've seen to this point: batchManager, batchManagerZos, or the REST interface.
3. A queue serves as the intermediary between the dispatcher server and the executor servers. This queue may be hosted in any JMS-compliant messaging engine, such as the default messaging of Liberty itself, or IBM MQ.
4. The "Executor" servers -- you may have one, or you may have many -- are configured to be executors (with the <batchJmsExecutor> element), the batch-1.0 and batchManagement-1.0 features, and the JMS definitions that allow it to listen on the queue and look for job dispatching messages.

   The executor servers use code to retrieve messages from the queue. The "activation spec" -- the code that listens for a message and picks it up on arrival -- can be configured with something called a "message selector," which can filter on the messages it sees and pick up only certain messages. This is a key point, and we will focus in on the message selector over the next several charts.
5. Finally, for this "multi-JVM" topology to work, all the servers that participate must be using the same JobRepository. There is a coordination that goes on between the dispatcher and the executors that relies on a consistent knowledge of the job state and the instance and execution numbers that gets assigned. This is accomplished by sharing the JobRepository. That means the relational system used for the JobRepository must support data sharing. DB2 z/OS can do that easily.

Let's look at the message selector definitions on the activation specs in the executor servers. That's what allows us to configure a server to pick up certain messages. And that opens up some interesting things this "multi-JVM" topology design can be used for.



IBM WebSphere Liberty Batch z/OS

The Executor Activation Spec messageSelector= … an Overview

1. **Three Executor servers listening on the message queue**
   In this scenario we have three Executor servers. They are all listening on the same Dispatcher queue. Each has different Java Batch applications deployed.
2. **Dispatcher submits four jobs: A, B, X, and Y**
   An operator (or an Enterprise Scheduler) submits four jobs: A, B, X, and Y. The Dispatcher places a job submission message on the queue for each job to be run.
3. **Message selector in Executor server.xml determines what it runs**
   Each Executor server has its JMS Activation Spec coded with properties to define what messages to watch for and pick up. In this example we're showing selecting on the application name, but you can also select on job parameters (more on this next chart).
   Executor #1 is watching for Application A, so when that job submission message appears in the queue it picks it up and executes the job. Executor #2 is watching for App B. Executor #3 is watching for X or Y, and will pick up job submission messages for either.

jobParameter selection …

© 2017 IBM Corporation 4

We're going to explain message selectors using a higher-level illustration first, then we'll go deeper and show actual coding examples. For this first illustration assume a single dispatcher and *three* executor servers. Each has a different batch application deployed.

Server 1 has Application A deployed, and its message selector is coded to look for "App = A."

Server 2 has Application B deployed, and its message selector is coded to look for "App = B."

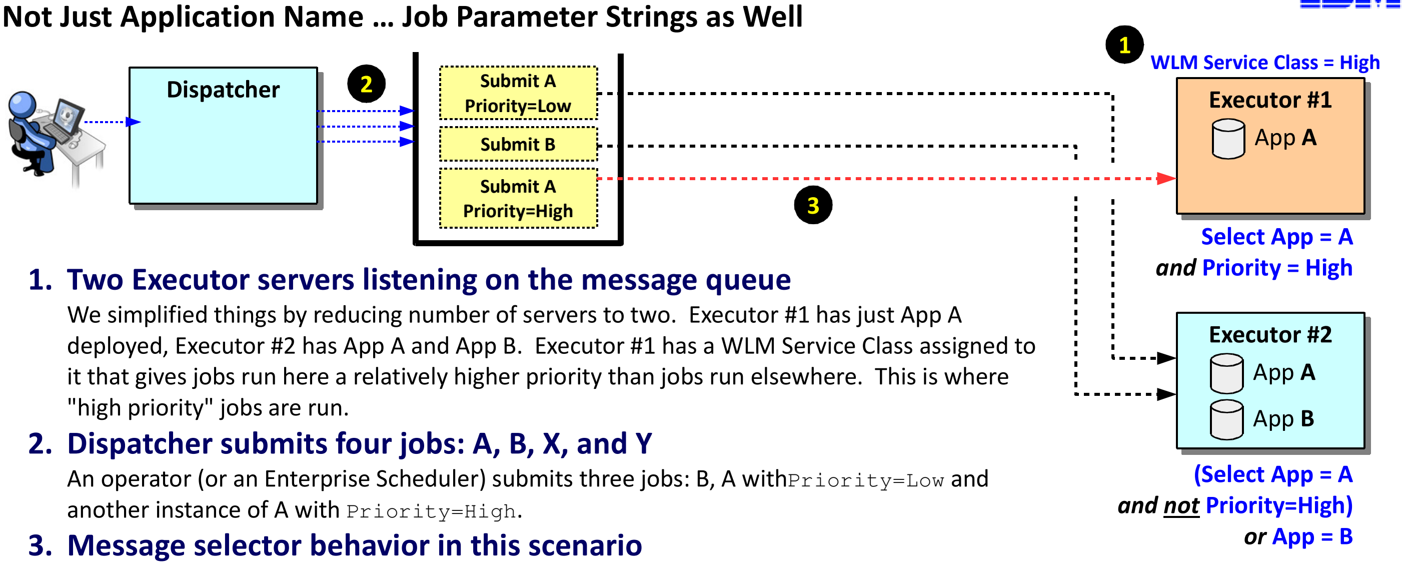
Server 3 has two applications deployed: X and Y. Its message selector is coded to look for "App = X" **or** "App = Y".

The Dispatcher receives four job submission requests and places them all on the queue. The queue now four messages in it: one for App X, one for App A, for for App B, and one for App Y.

Server 1 will see all four messages but pick only the one for App A because its message selector is configured to select only when the application name matches "A". Similarly, server 2 will only pick up the job submission message for application B because its message selector message is configured to select only when the application name matches "B". Server 3 has an "or" logical operator on its message selector, so it will see the job submission messages for X and Y and pick up both.

That's our first scenario, where the message selector is based on the application name. But we can be more creative than that. We can also specify a jobParameter value that can be selected upon as well. And that allows us to do something like what we show on the next chart.

Not Just Application Name … Job Parameter Strings as Well



1. **Two Executor servers listening on the message queue**
   We simplified things by reducing number of servers to two. Executor #1 has just App A deployed, Executor #2 has App A and App B. Executor #1 has a WLM Service Class assigned to it that gives jobs run here a relatively higher priority than jobs run elsewhere. This is where "high priority" jobs are run.
2. **Dispatcher submits four jobs: A, B, X, and Y**
   An operator (or an Enterprise Scheduler) submits three jobs: B, A with`Priority=Low` and another instance of A with `Priority=High`.
3. **Message selector behavior in this scenario**
   Executor #1 is looking for the –JobParameter string of Priority=High. It will only pick up messages that meet that criteria. So it will ignore the Priority=Low.
   Executor #2 is looking for App A with a 'Priority' value of anything other than 'High,' as well as picking up any submission messages for App B.

Value ...

Similar layout, but with two servers. Server 1 has only application A deployed, while server 2 has both applications A and B deployed. This is interesting ... both servers 1 and 2 have application A, so do they just race to see who gets the message first? Well, they could do that, or they could be configured to pick up based on a job parameter in addition to the application name.

Imagine you have the Executor Server #1 configured so it runs as a high-priority started task in z/OS. Anything that runs there gets favored status because z/OS WLM will grant that started task extra system resources. How do we designate a job submission as "high priority" and get it picked up by that server? By submitting the job with a job parameter that will insure it gets picked up by that server.

Imagine Server 1 has a message selector that says, "Select when Application = A **and** Priority = High."

Server 2 has a message selector that says, "Select when (Application = A and **not** Priority = High) **or** Application = B."

Three messages appear in the queue:

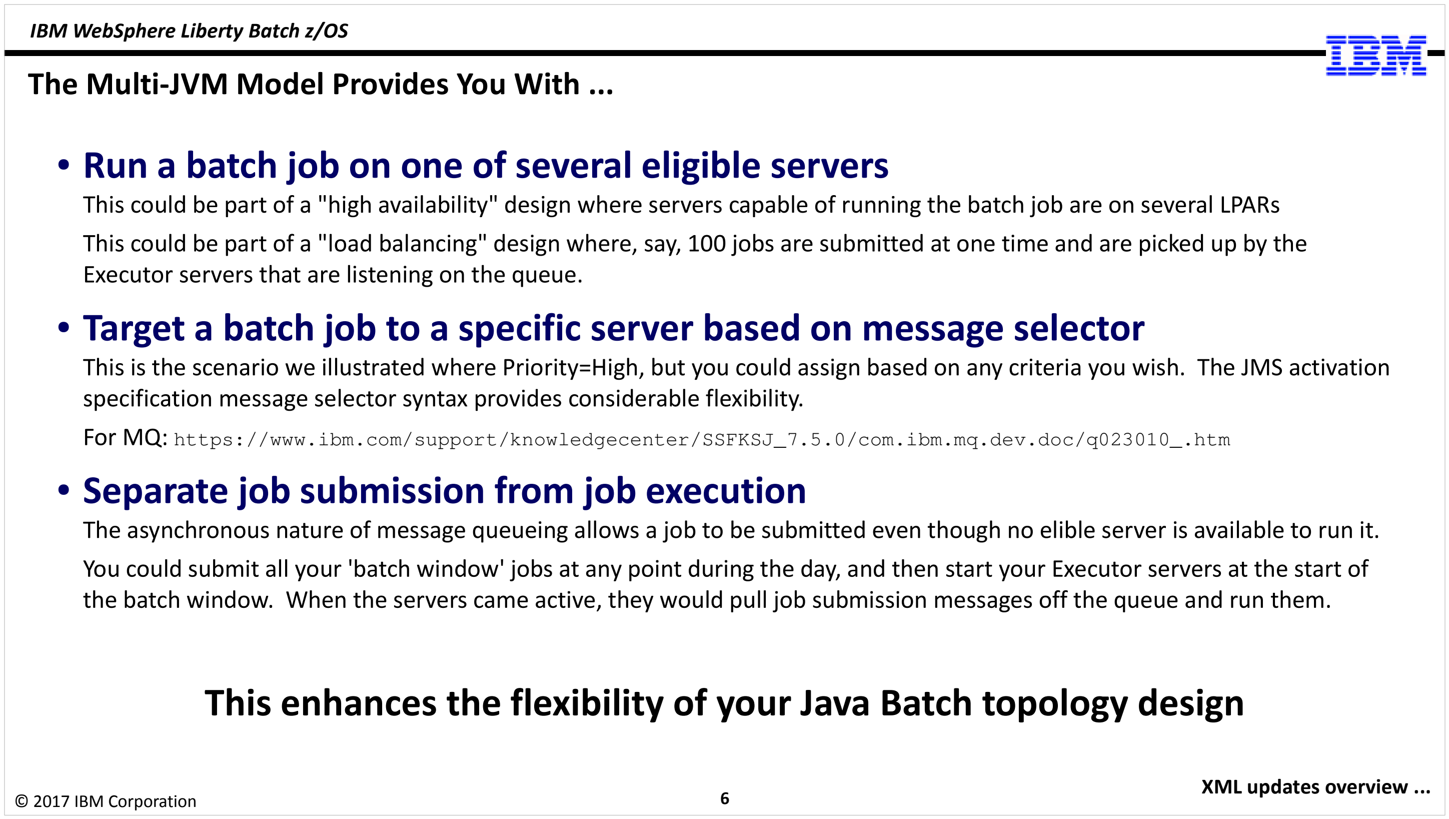
1. Application A + Priority = Low
2. Application B
3. Application A + Priority = High

Executor #1 is looking for A + Priority = High, so it will ignore the first two and select the third message.

Executor #2 will pick up the first message because the first part of its conditional message is satisfied -- the message is for Application A and **not** Priority = High. The priority is coded as "Low," which is not "High," so the condition is met.

Executor #2 is will select the second message (Application B) because the second clause after "or" is satisfied.

The result is the job submission message with the job parameter of Priority = High gets selected by the server with the higher WLM priority status and thus will get more system resources. By coding the message selector you can direct job submissions to one server or another.



IBM WebSphere Liberty Batch z/OS

The Multi-JVM Model Provides You With ...

- **Run a batch job on one of several eligible servers**

  This could be part of a "high availability" design where servers capable of running the batch job are on several LPARs

  This could be part of a "load balancing" design where, say, 100 jobs are submitted at one time and are picked up by the Executor servers that are listening on the queue.

- **Target a batch job to a specific server based on message selector**

  This is the scenario we illustrated where Priority=High, but you could assign based on any criteria you wish. The JMS activation specification message selector syntax provides considerable flexibility.

  For MQ: `https://www.ibm.com/support/knowledgecenter/SSFKSJ_7.5.0/com.ibm.mq.dev.doc/q023010_.htm`

- **Separate job submission from job execution**

  The asynchronous nature of message queueing allows a job to be submitted even though no elible server is available to run it.

  You could submit all your 'batch window' jobs at any point during the day, and then start your Executor servers at the start of the batch window. When the servers came active, they would pull job submission messages off the queue and run them.

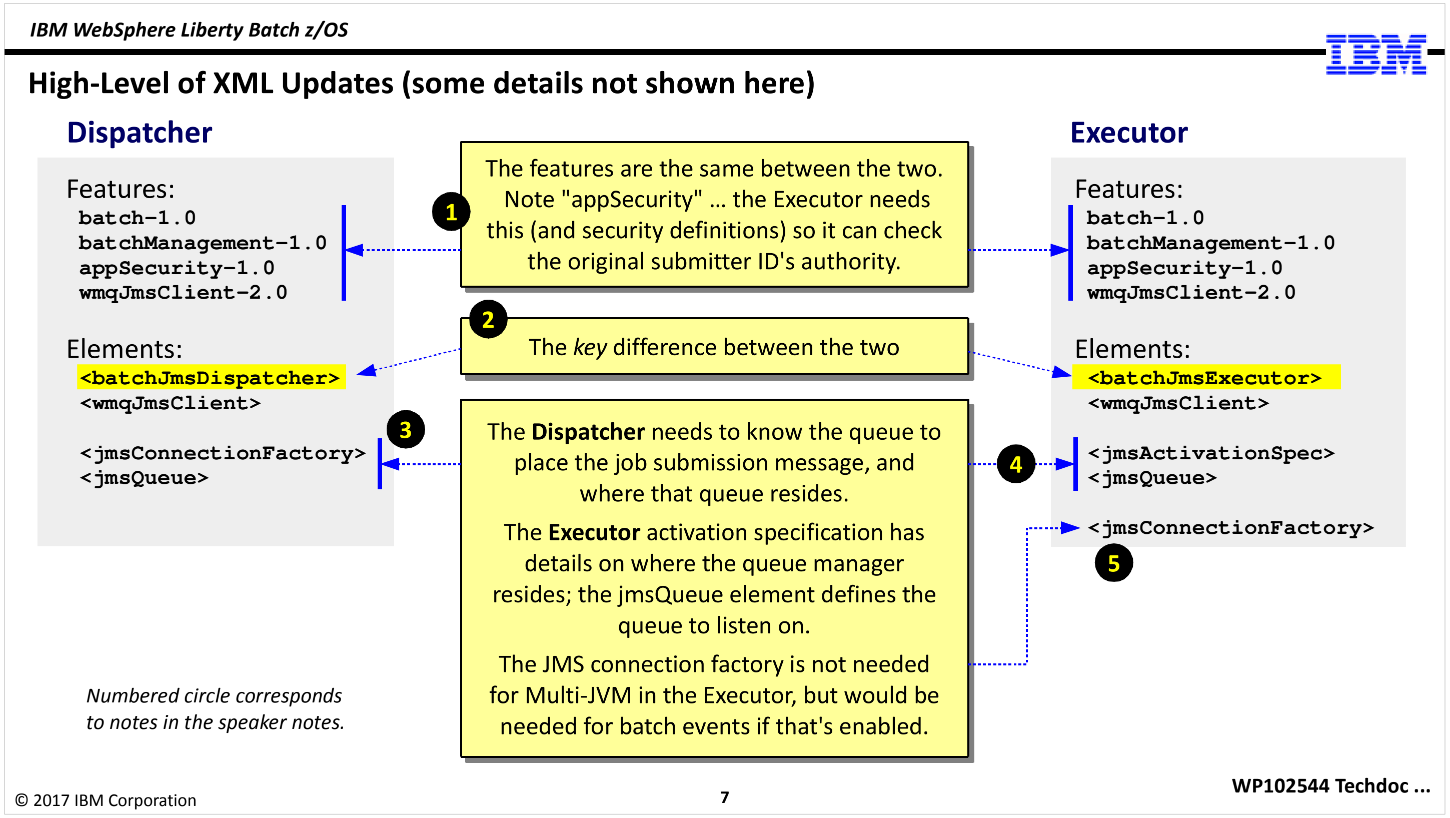
This enhances the flexibility of your Java Batch topology design

© 2017 IBM Corporation 6 **XML updates overview ...**

Let's discuss what value this "Multi-JVM" model brings. It breaks down into three main categories, and perhaps you can see value beyond what's shown on this chart.

- Configure multiple executor servers and have jobs dispatch to one of several eligible servers. This could address a "high availability" requirement where you want jobs to run even if you have an outage of an LPAR. Servers on the surviving LPAR can pick up and run the applications.
- Specifically target a batch job to a server based on your knowledge of the message selectors coded in the server. We illustrated this with the "Priority = High" scenario earlier. But it could be based on any placement criterial you want.
- Submit jobs and have them execute later when the executor servers are started. For instance, you could submit the jobs at 5:00pm but not have them run until 10:00pm when the executors are started at the beginning of your batch window. The job submission messages will just sit in the queue until something picks them up.

This multi-JVM design provides you with considerable flexibility.



IBM WebSphere Liberty Batch z/OS

High-Level of XML Updates (some details not shown here)

1 The features are the same between the two. Note "appSecurity" ... the Executor needs this (and security definitions) so it can check the original submitter ID's authority.

2 The *key* difference between the two

3 4 The **Dispatcher** needs to know the queue to place the job submission message, and where that queue resides.

The **Executor** activation specification has details on where the queue manager resides; the jmsQueue element defines the queue to listen on.

5 The JMS connection factory is not needed for Multi-JVM in the Executor, but would be needed for batch events if that's enabled.

Numbered circle corresponds to notes in the speaker notes.

© 2017 IBM Corporation 7 **WP102544 Techdoc ...**

This chart is comparing the server.xml needed for a Dispatcher compared to that needed for an Executor. Some of the detail has been omitted here, leaving just the high-level element names.

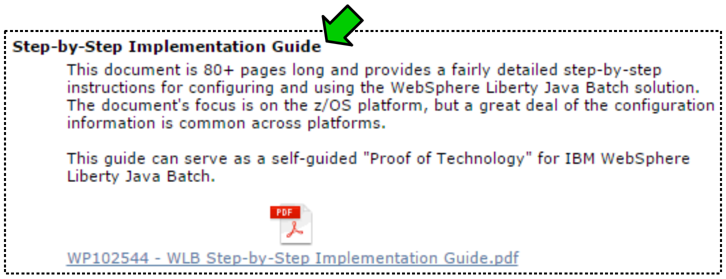
1. The feature names are identical between the two. Both dispatcher and executor need what's shown in the chart. And as the yellow box notes, the executor needs appSecurity-1.0 and the the supporting security to do authentication and authorization. That's because when a job submission message is placed on the queue, the submitter's ID from the Dispatcher server goes with the message. But the Executor server may be on a completely different server box, and it'll have to re-authenticate and re-authorize the submitter ID to do the work in the Exector.
2. The dispatcher is indicated by <batchJmsDispatcher> while the executor is indicated by <batchJmsExecutor>.
3. The dispatcher requires <jmsConnectionFactory> and <jmsQueue> elements to define the queue manager and queue to place the job submission message on.
4. The executor requires <jmsActivationSpec> and <jmsQueue> elements to define which queue manager and queue to listen on. The <jmsActivationSpec> element has many of the same details as the connection factory ... specifying, for example, the queue manager host and port.
5. If the executor servers wish to emit batch events, then you need to provide a <jmsConnectionFactory> definition to tell the server where to publish the batch events.

Seeking details? See the next chart ...

IBM WebSphere Liberty Batch z/OS

WP102544 Techdoc PDF Files

http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102544



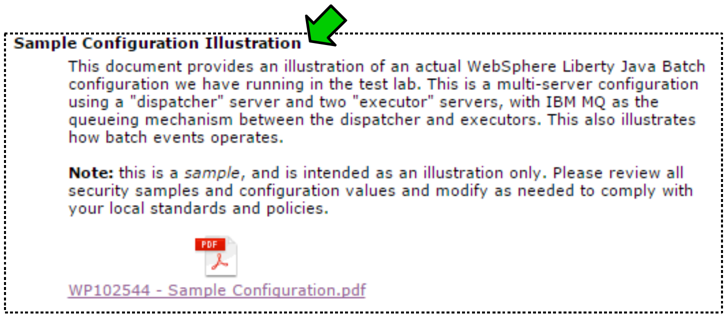
Step-by-Step Implementation Guide

This document is 80+ pages long and provides a fairly detailed step-by-step instructions for configuring and using the WebSphere Liberty Java Batch solution. The document's focus is on the z/OS platform, but a great deal of the configuration information is common across platforms.

This guide can serve as a self-guided "Proof of Technology" for IBM WebSphere Liberty Java Batch.

WP102544 - WLB Step-by-Step Implementation Guide.pdf

This has a detailed illustration of how to enable using the default messaging of Liberty



Sample Configuration Illustration

This document provides an illustration of an actual WebSphere Liberty Java Batch configuration we have running in the test lab. This is a multi-server configuration using a "dispatcher" server and two "executor" servers, with IBM MQ as the queueing mechanism between the dispatcher and executors. This also illustrates how batch events operates.

Note: this is a *sample*, and is intended as an illustration only. Please review all security samples and configuration values and modify as needed to comply with your local standards and policies.

WP102544 - Sample Configuration.pdf

This has a detailed illustration of how to enable using IBM MQ.

The topology illustrated in this PDF is what we'll be doing in the lab as well.

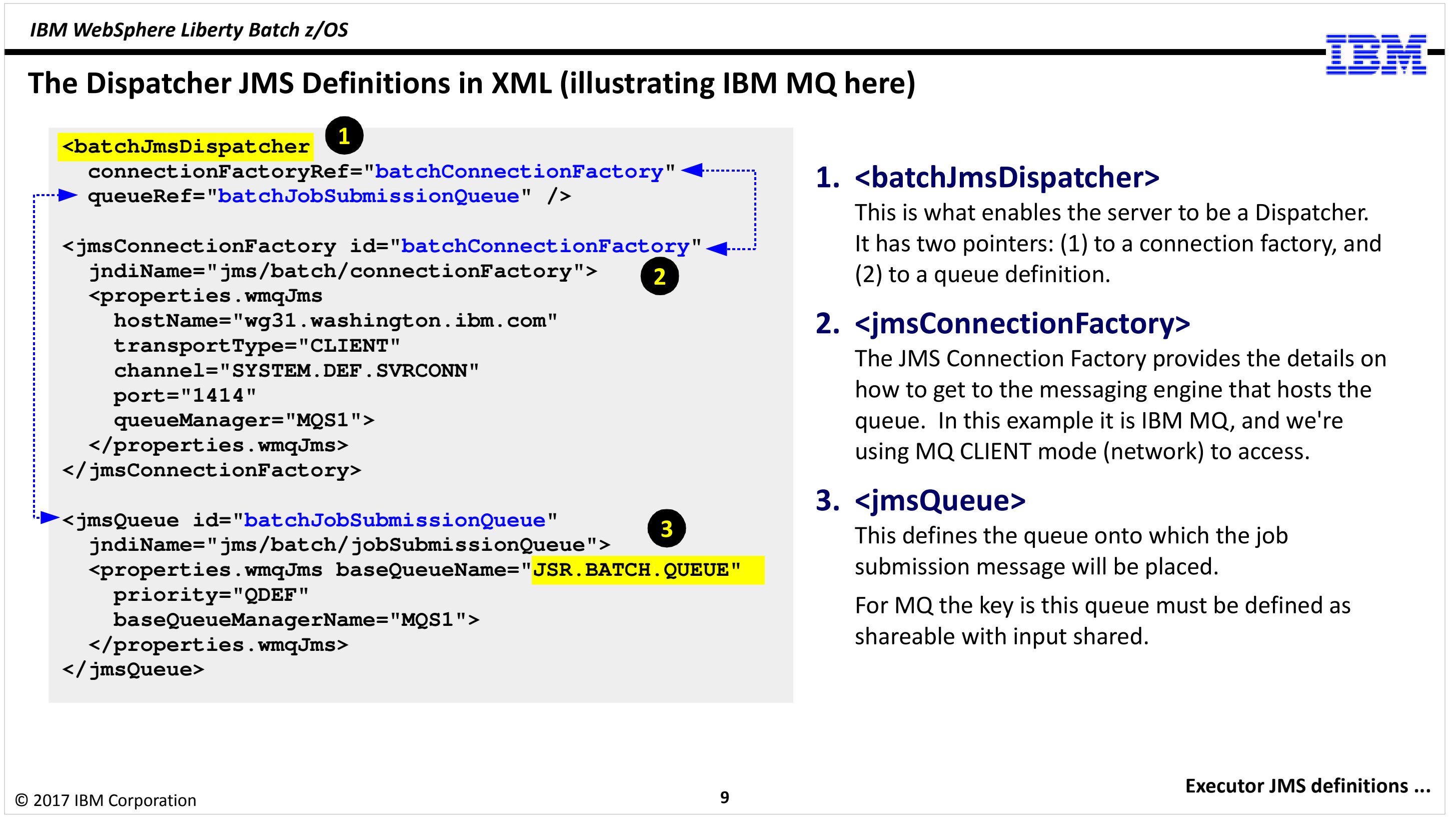
© 2017 IBM Corporation 8

Dispatcher JMS definitions ...

The previous chart was lacking in specific details about how to set up the multi-JVM configuration. We're going to provide you with some of the details in the upcoming charts, but we'd like to draw your attention to the WP102544 Techdoc, which has two documents that provides extensive detail:

- *Step-by-Step Implementation Guide* -- this provides details on setting up the WebSphere Liberty Java Batch environment, and it includes detailed instructions on enabling the multi-JVM support using the default messaging that's built-in to Liberty.
- *Sample Configuration Illustration* -- we built a multi-JVM topology on a real system and captured all the configuration artifacts into this document. The topology in this document is very similar to the topology we're building for this workshop.

So when you leave this workshop you will have access to the details you need to enable all this back home.



IBM WebSphere Liberty Batch z/OS

The Dispatcher JMS Definitions in XML (illustrating IBM MQ here)

```
<batchJmsDispatcher                                              1
  connectionFactoryRef="batchConnectionFactory"
  queueRef="batchJobSubmissionQueue" />

<jmsConnectionFactory id="batchConnectionFactory"
  jndiName="jms/batch/connectionFactory">                        2
  <properties.wmqJms
    hostName="wg31.washington.ibm.com"
    transportType="CLIENT"
    channel="SYSTEM.DEF.SVRCONN"
    port="1414"
    queueManager="MQS1">
  </properties.wmqJms>
</jmsConnectionFactory>

<jmsQueue id="batchJobSubmissionQueue"                           3
  jndiName="jms/batch/jobSubmissionQueue">
  <properties.wmqJms baseQueueName="JSR.BATCH.QUEUE"
    priority="QDEF"
    baseQueueManagerName="MQS1">
  </properties.wmqJms>
</jmsQueue>
```

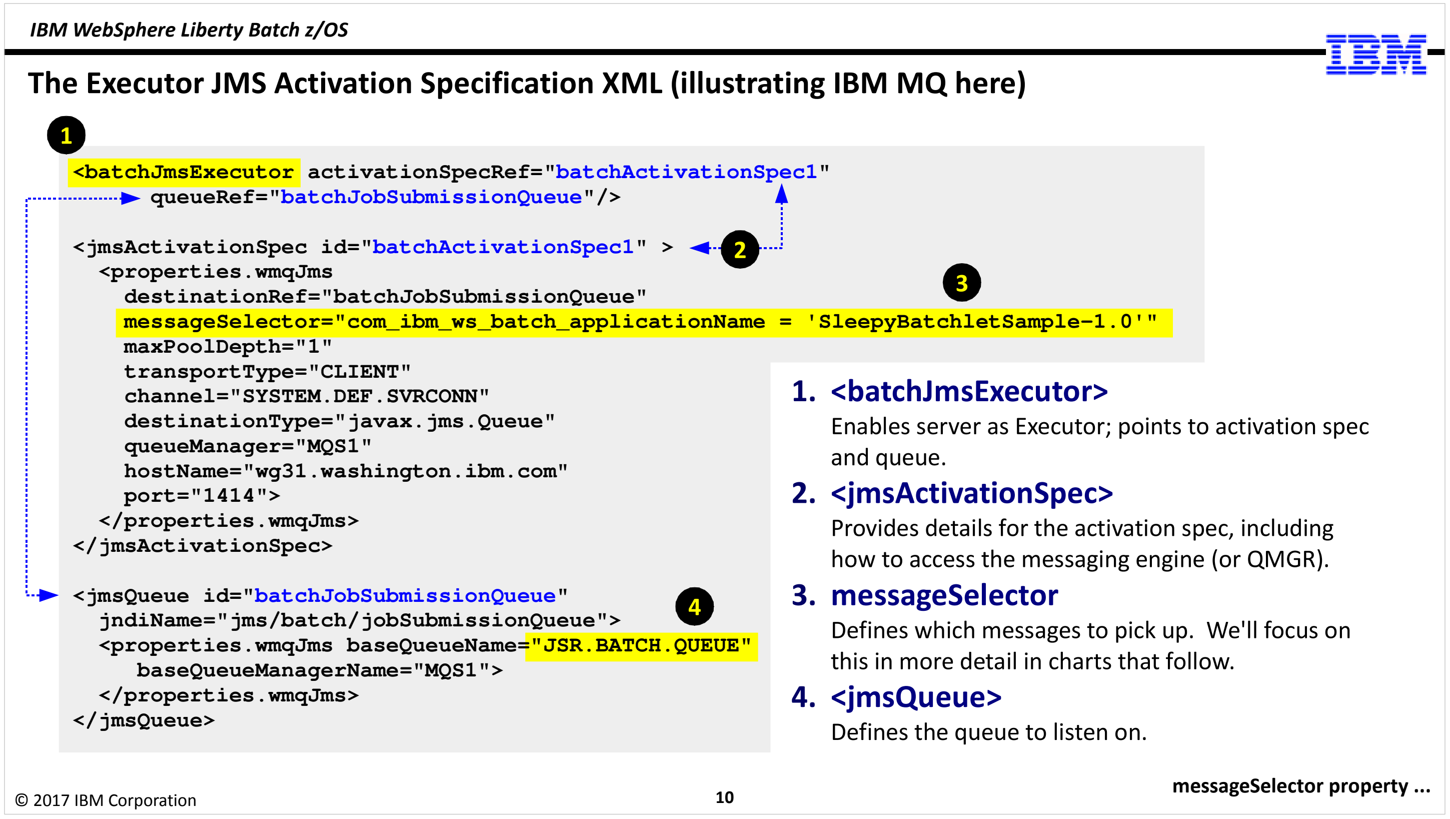
1. **<batchJmsDispatcher>**
   This is what enables the server to be a Dispatcher. It has two pointers: (1) to a connection factory, and (2) to a queue definition.
2. **<jmsConnectionFactory>**
   The JMS Connection Factory provides the details on how to get to the messaging engine that hosts the queue. In this example it is IBM MQ, and we're using MQ CLIENT mode (network) to access.
3. **<jmsQueue>**
   This defines the queue onto which the job submission message will be placed.

   For MQ the key is this queue must be defined as shareable with input shared.

© 2017 IBM Corporation 9 **Executor JMS definitions ...**

Here's a bit of details to show the configuration for the dispatcher and MQ.

1. The Dispatcher server is made a dispatcher with the <batchJmsDispatcher> element defined. This element has two attributes: one that serves as a pointer to the JMS connection factory definition, and another that serves as a pointer to the queue definition. In other words, "I'm a dispatcher, and here are the details to allow me to (a) reach the queue manager, and (b) know what queue to use."
2. The <jmsConnectionFactory> element defines how to access the MQ queue manager. This is pointed to from the <batchJmsDispatcher> element. This has a set of <properties> that define specifics such as the host and port the MQ queue manager is listening on for (in this example) CLIENT connections.
3. The <jmsQueue> element defines the MQ queue to use when issuing a PUT of the job submission message. This queue must be defined as shareable and with input shared.



IBM WebSphere Liberty Batch z/OS

The Executor JMS Activation Specification XML (illustrating IBM MQ here)

```
<batchJmsExecutor activationSpecRef="batchActivationSpec1"
      queueRef="batchJobSubmissionQueue"/>

<jmsActivationSpec id="batchActivationSpec1" >
  <properties.wmqJms
    destinationRef="batchJobSubmissionQueue"
    messageSelector="com_ibm_ws_batch_applicationName = 'SleepyBatchletSample-1.0'"
    maxPoolDepth="1"
    transportType="CLIENT"
    channel="SYSTEM.DEF.SVRCONN"
    destinationType="javax.jms.Queue"
    queueManager="MQS1"
    hostName="wg31.washington.ibm.com"
    port="1414">
  </properties.wmqJms>
</jmsActivationSpec>

<jmsQueue id="batchJobSubmissionQueue"
  jndiName="jms/batch/jobSubmissionQueue">
  <properties.wmqJms baseQueueName="JSR.BATCH.QUEUE"
     baseQueueManagerName="MQS1">
  </properties.wmqJms>
</jmsQueue>
```

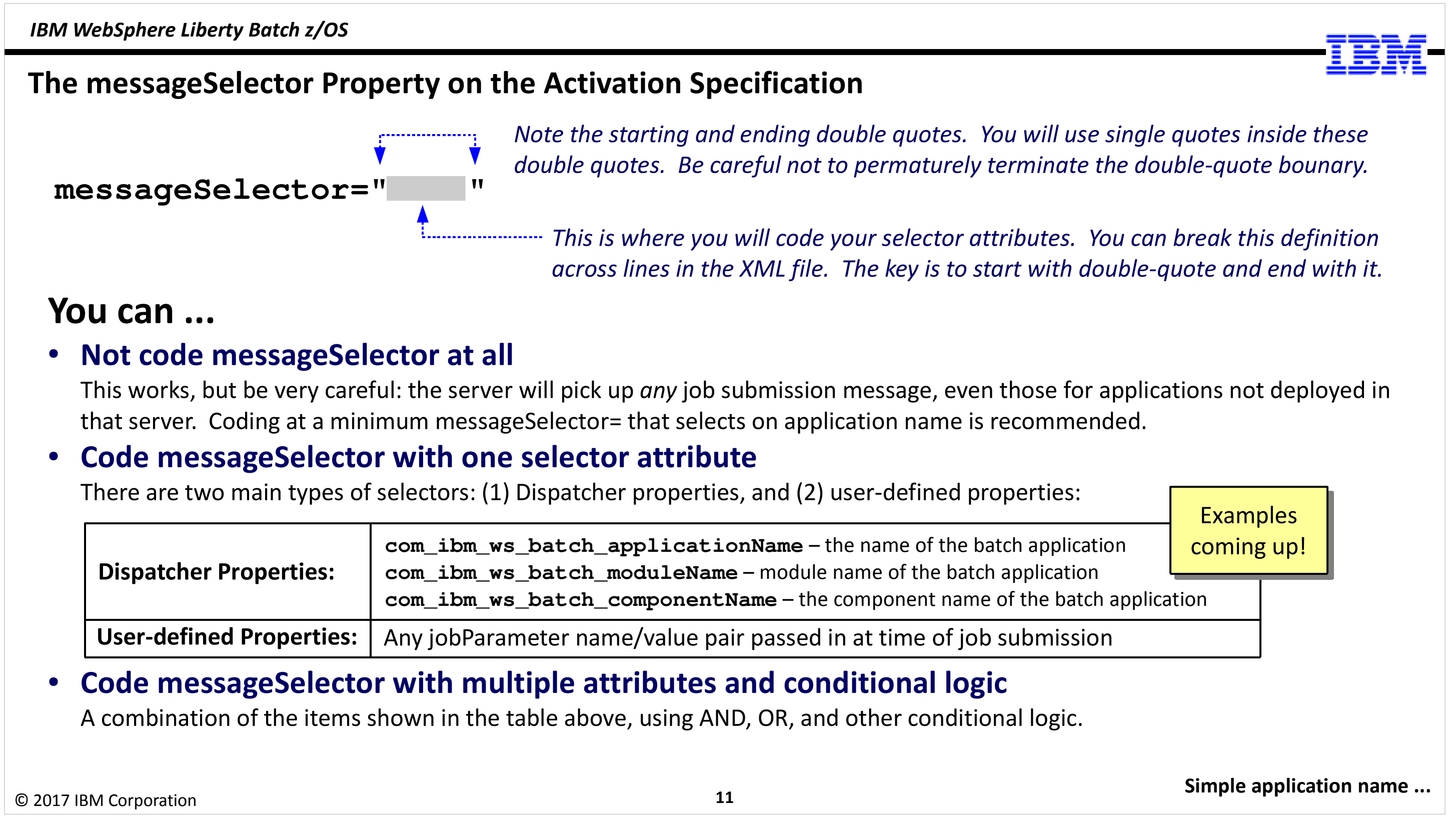
1. **<batchJmsExecutor>**
   Enables server as Executor; points to activation spec and queue.
2. **<jmsActivationSpec>**
   Provides details for the activation spec, including how to access the messaging engine (or QMGR).
3. **messageSelector**
   Defines which messages to pick up. We'll focus on this in more detail in charts that follow.
4. **<jmsQueue>**
   Defines the queue to listen on.

messageSelector property ...

© 2017 IBM Corporation 10

And now the definitions for the executor server:

1. The executor is enabled as an Executor with the <batchJmsExecutor> element. This has two attributes that serve as pointers to other definitions in the XML: the activationSpecRef= attribute points to the <jmsActionSpec> element, which provides details on the JMS activation specification, including the message selector definition we covered earlier; the queueRef= points to the <jmsQueue> element, which defines which queue to listen on.
2. The <jmsActivationSpec> element contains details about the MQ queue manager to make a connection with so the queue can be listened on. The <properties> includes definitions on the MQ queue manager location, such as host and port (and in this way is very similar to the connection factory we saw on the previous page), as well as the messageSelector= definition.
3. The messageSelector= definition is what tells the Executor server what messages to select on. If you remove this property the Executor will attempt to select all messages that arrive on the queue. With this coded, it will select only those messages that match the criteria of the selector. We're going to cover this in a bit more detail over the next several charts.
4. The <jmsQueue> element defines the queue on which to listen. This is the same queue the Dispatcher has defined as it's PUT queue.



IBM WebSphere Liberty Batch z/OS

The messageSelector Property on the Activation Specification

```
messageSelector="        "
```

Note the starting and ending double quotes. You will use single quotes inside these double quotes. Be careful not to permaturely terminate the double-quote bounary.

This is where you will code your selector attributes. You can break this definition across lines in the XML file. The key is to start with double-quote and end with it.

You can ...

- **Not code messageSelector at all**
  This works, but be very careful: the server will pick up *any* job submission message, even those for applications not deployed in that server. Coding at a minimum messageSelector= that selects on application name is recommended.
- **Code messageSelector with one selector attribute**
  There are two main types of selectors: (1) Dispatcher properties, and (2) user-defined properties:

Dispatcher Properties:	**`com_ibm_ws_batch_applicationName`** – the name of the batch application **`com_ibm_ws_batch_moduleName`** – module name of the batch application **`com_ibm_ws_batch_componentName`** – the component name of the batch application
User-defined Properties:	Any jobParameter name/value pair passed in at time of job submission

Examples coming up!

- **Code messageSelector with multiple attributes and conditional logic**
  A combination of the items shown in the table above, using AND, OR, and other conditional logic.

© 2017 IBM Corporation 11

Simple application name ...

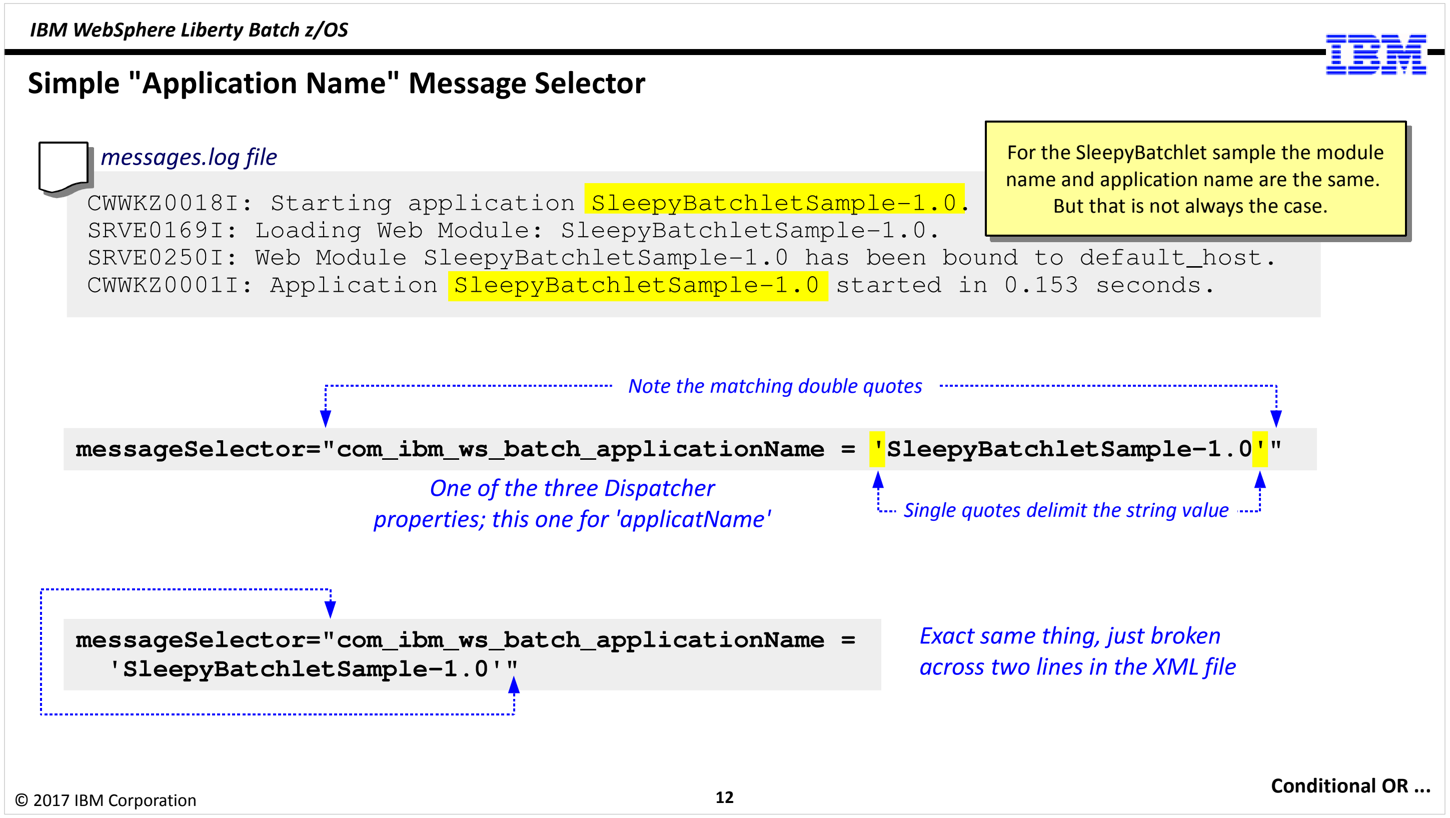
Let's turn our focus to the messageSelector= attribute, which is what we use to refine which messages a given Executor server will pick up from the queue.

The first thing to notice is that the messageSelector= attribute encapsulates its value within double quotes. We draw this to your attention because within those double quotes will be single quotes that delimit other values. So matching all the quotes will be key to avoiding syntax problems. The messageSelector= definition can get very long and you may wish to break it across lines in the server.xml file. That can be done; just be sure to end the messageSelector definition with the double quote.

Options available to you with the message selector:

- You can not code messageSelector= in the <jmsActivationSpec> element. The *absence* of the messageSelector= tells the Executor to pick up any and all messages that hit the queue being listened on. This is potentially problematic because it's possible it'll pick up a message for an application that's not even deployed in the server. It is recommended that at a minimum you have coded a messageSelector that selects on the application name(s) of the applications deployed in the server.
- You can keep things simple and code the messageSelector= with one attribute, such as application name. There are two "classes" of properties you can select on: (1) dispatcher properties, of which there are three: the application name, the module name, or the component name; or (2) any user-defined property passed to the dispatcher at the time of job submission. Earlier we used the example of "Priority=High" ... that was an example of a user-defined property being passed in at time of job submission. That name/value pair flows with the job submission message to the queue and is available to the Executor to select on using the messageSelector= attribute.
- You can get more sophisticated and code multiple selector attributes using conditional logic. This is what we're going to focus on in the next several charts.

The key point here is that an Executor <jmsActivationSpec> can be defined to pick up messages using a variety of degrees of refinement, from "pick up all" to "pick up some" to "pick up a very select few based on a conditional selector clause."



IBM WebSphere Liberty Batch z/OS

Simple "Application Name" Message Selector

messages.log file

```
CWWKZ0018I: Starting application SleepyBatchletSample-1.0.
SRVE0169I: Loading Web Module: SleepyBatchletSample-1.0.
SRVE0250I: Web Module SleepyBatchletSample-1.0 has been bound to default_host.
CWWKZ0001I: Application SleepyBatchletSample-1.0 started in 0.153 seconds.
```

For the SleepyBatchlet sample the module name and application name are the same. But that is not always the case.

Note the matching double quotes

```
messageSelector="com_ibm_ws_batch_applicationName = 'SleepyBatchletSample-1.0'"
```

One of the three Dispatcher properties; this one for 'applicatName'

Single quotes delimit the string value

```
messageSelector="com_ibm_ws_batch_applicationName =
  'SleepyBatchletSample-1.0'"
```

Exact same thing, just broken across two lines in the XML file

© 2017 IBM Corporation 12 **Conditional OR ...**

Let's take a look at the simple example of selecting on the application name. We'll use the SleepyBatchlet sample application.

If you look in the messages.log for a server that has SleepyBatchlet deployed, you will see a series of messages that indicate the application is started, and has been started. In those messages is the *name* of the application as it is known to the server. In the case of the SleepyBatchlet application, this turns out to be the name of the WAR file as well.

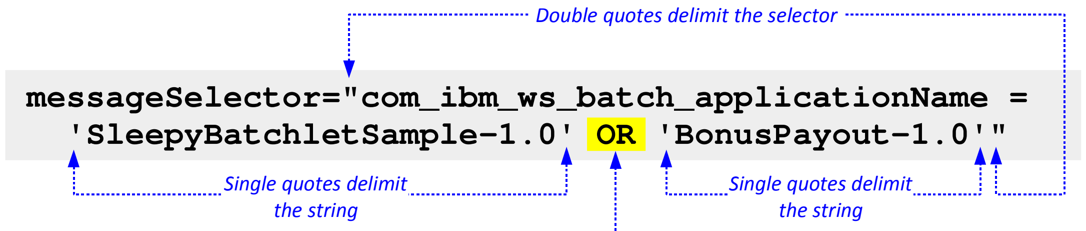
To select on the application name you would code the messageSelector= as illustrated in the middle of the chart:

- Note the double-quotes that start and end the messageSelector= string
- com_ibm_ws_batch_applicationName is one of the pre-defined "dispatcher properties" from the previous chart
- Note the *single*-quotes that delimit the application name

If you wanted to break that across multiple lines in the XML you could do that with the example shown at the bottom of the chart. The key is making sure to match the double and single quotes.

IBM WebSphere Liberty Batch z/OS

Conditional OR on Application Names



```
messageSelector="com_ibm_ws_batch_applicationName =
   'SleepyBatchletSample-1.0' OR 'BonusPayout-1.0'"
```

Double quotes delimit the selector
Single quotes delimit the string
Single quotes delimit the string

The conditional OR, meaning that either application will be selected if seen on the queue

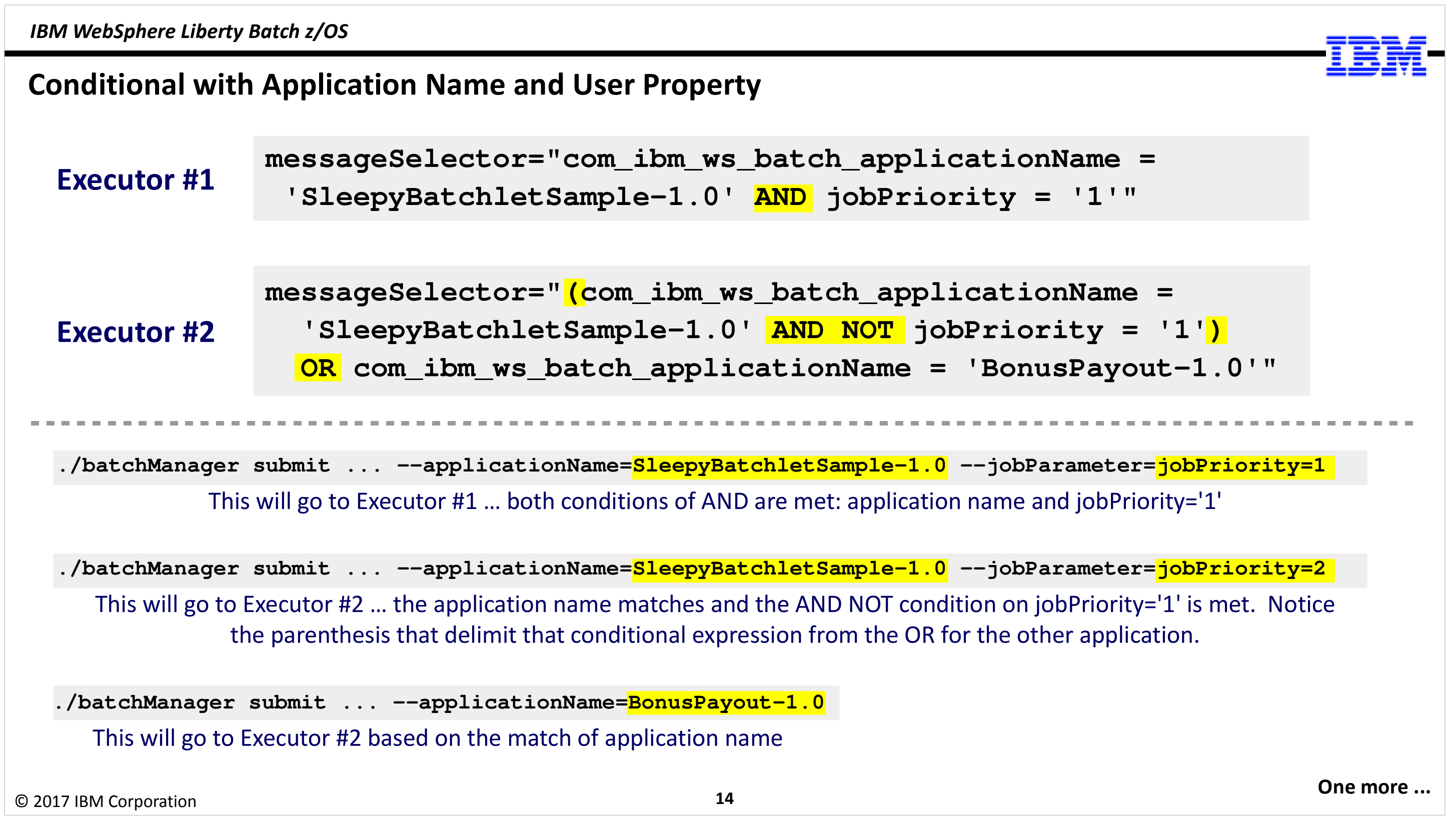
13 **Conditional app and user-defined ...**

Imagine you have both the SleepyBatchlet application and the BonusPayout application deployed in the same server. You want the messageSelector= to pick up either application. You can do that with a conditional "or" expression:

- One messageSelector= is coded, and it has the double-quotes to start and end the string
- The first selector condition is on the dispatcher property com_ibm_ws_batch_applicationName, and that specifies the application name enclosed in single quotes. This is the same as we saw on the previous chart.
- We include a conditional OR operator
- We code the BonusPayout application name.

This will pick up either application job submission message.

Let's get a bit more sophisticated ...



IBM WebSphere Liberty Batch z/OS

Conditional with Application Name and User Property

Executor #1

```
messageSelector="com_ibm_ws_batch_applicationName =
 'SleepyBatchletSample-1.0' AND jobPriority = '1'"
```

Executor #2

```
messageSelector="(com_ibm_ws_batch_applicationName =
  'SleepyBatchletSample-1.0' AND NOT jobPriority = '1')
 OR com_ibm_ws_batch_applicationName = 'BonusPayout-1.0'"
```

```
./batchManager submit ... --applicationName=SleepyBatchletSample-1.0 --jobParameter=jobPriority=1
```

This will go to Executor #1 … both conditions of AND are met: application name and jobPriority='1'

```
./batchManager submit ... --applicationName=SleepyBatchletSample-1.0 --jobParameter=jobPriority=2
```

This will go to Executor #2 … the application name matches and the AND NOT condition on jobPriority='1' is met. Notice the parenthesis that delimit that conditional expression from the OR for the other application.

```
./batchManager submit ... --applicationName=BonusPayout-1.0
```

This will go to Executor #2 based on the match of application name

© 2017 IBM Corporation 14 **One more ...**

Imagine you have two Executor servers: the first has SleepyBatchlet and the second has both SleepyBatchlet and BonusPayout.

You want to designate the first Executor as the one to run SleepyBatchlet in high-priority mode. You want the second Executor to run SleepyBatchlet or BonusPayout in 'normal' priority.

You want to designate priority with a number, with "1" being highest priority. You think maybe sometime in the future you'll have several levels of priority, so today for high priority you'll have a job property of "jobPriority=1" but later you may have 2 and 3. But for now it's just "1" for priority and "not 1" for everything else.

The messageSelector= for each server is coded at the top of the chart:

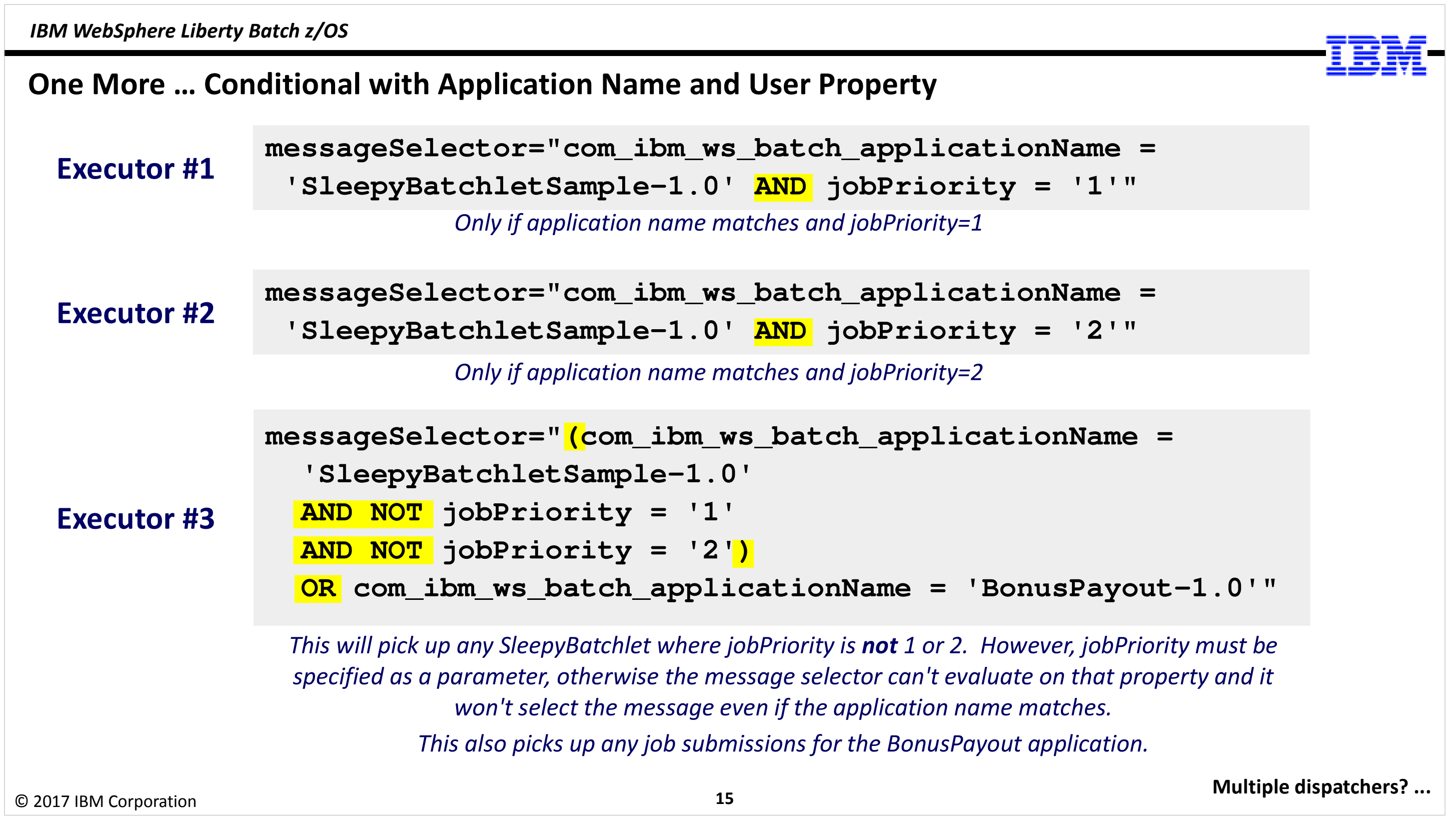
- The messageSelector= for the first (high-priority) server is coded with a selector on the application name AND the user-defined property of jobPriority = '1'. That will only pick up messages that satisfy both of those conditions.
- The messageSelector= for the second (not high-priority) server is coded with a bit more complex set of conditions.

  Note: we can't just code a simple select on SleepyBatchlet OR BonusPayout because that could potentially pick up the jobPriority = 1 message. So we have to explicitely account for *excluding* jobPriority = 1 to make sure we never pick up that.

  So we code this selector to pick up SleepyBatchlet based on the application name AND NOT jobPriority = '1'. We enclose that selector condition in parenthesis. Then we have a conditional OR to pick up BonusPayout whenever it is seen.
- Then we test with a series of job submission examples. The first example will get picked up by Executor #1 because the conditional AND is met with application name and jobPriority = '1'. The second message will go to Executor #2 because the jobPriority = '2' won't be matched in Executor #1, but it is satisfied with AND NOT jobPriority = '1' in Executor #2. The third message goes to Executor #2: the application name doesn't match what's coded in Executor #1, and is met with the conditional OR in the message selector in Executor #2.

Note: one restriction to keep in mind is that a job property that is used on the message selector can't have dots in the name. So in our example jobProperty is acceptable. So too would be job_Property. But job.Property would be unacceptable. If you wondered why the IBM dispatcher properties had underscores rather than dots (the "com_ibm_ws_batch ..." rather than com.ibm.ws.batch), that is why.

Let's look at one more example ...



IBM WebSphere Liberty Batch z/OS

One More ... Conditional with Application Name and User Property

Executor #1

```
messageSelector="com_ibm_ws_batch_applicationName =
 'SleepyBatchletSample-1.0' AND jobPriority = '1'"
```

Only if application name matches and jobPriority=1

Executor #2

```
messageSelector="com_ibm_ws_batch_applicationName =
 'SleepyBatchletSample-1.0' AND jobPriority = '2'"
```

Only if application name matches and jobPriority=2

Executor #3

```
messageSelector="(com_ibm_ws_batch_applicationName =
  'SleepyBatchletSample-1.0'
  AND NOT jobPriority = '1'
  AND NOT jobPriority = '2')
  OR com_ibm_ws_batch_applicationName = 'BonusPayout-1.0'"
```

*This will pick up any SleepyBatchlet where jobPriority is **not** 1 or 2. However, jobPriority must be specified as a parameter, otherwise the message selector can't evaluate on that property and it won't select the message even if the application name matches.*

This also picks up any job submissions for the BonusPayout application.

Multiple dispatchers? ...

© 2017 IBM Corporation

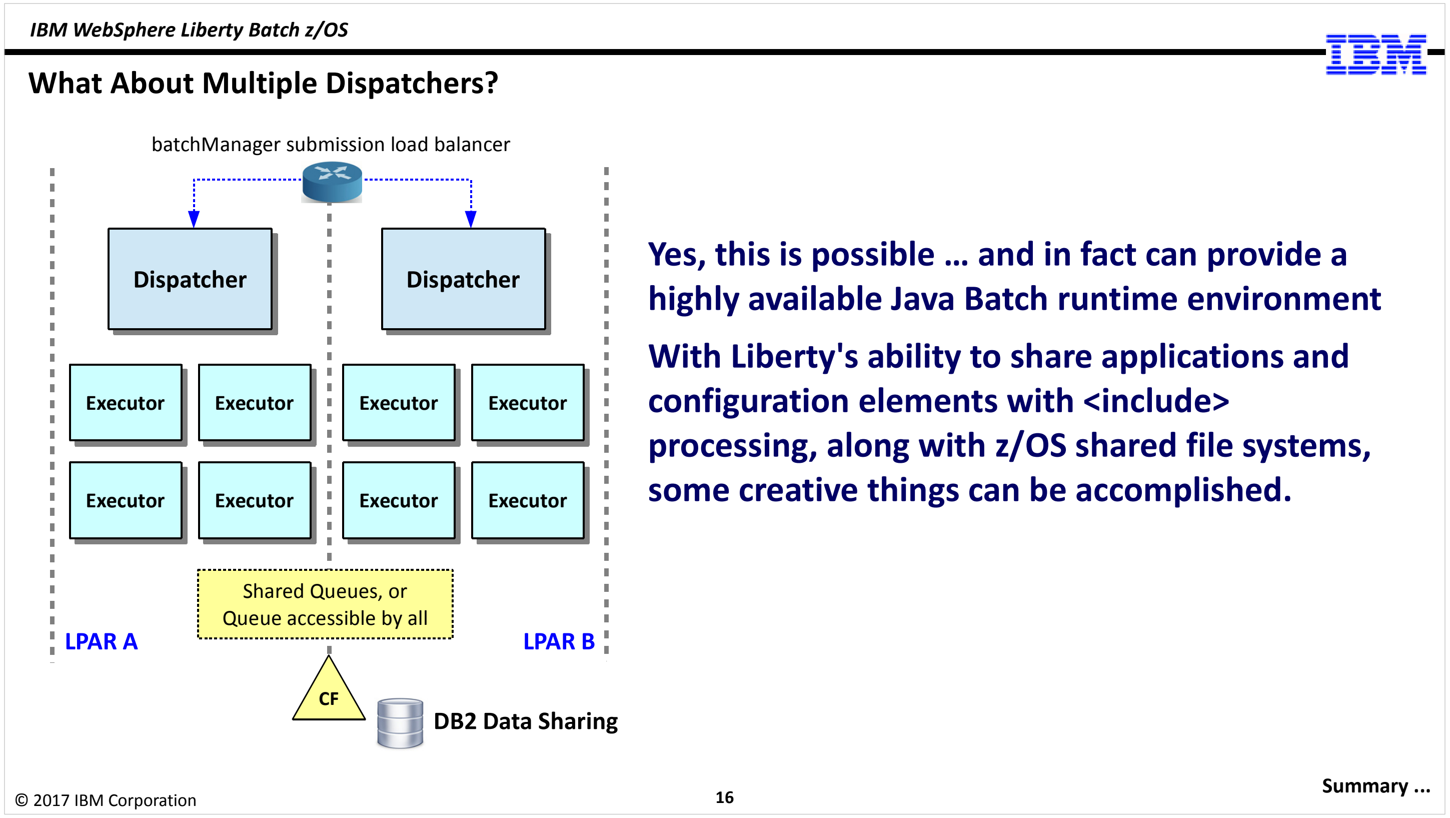
15

Imagine you have three Executor servers. The first is for SleepyBatchlet and jobPriority = '1'; the second is for SleepyBatchlet and jobPriority = '2" and the third is for SleepyBatchlet at any other priority or BonusPayout.

Note: What we're showing on the chart achieves the desired effect. You may find there are other ways to code the selector to achieve the effect as well. We're not saying this is the only way to achieve this; just one way. By all means explore the message selector options and be as simple or sophisticated as you wish.

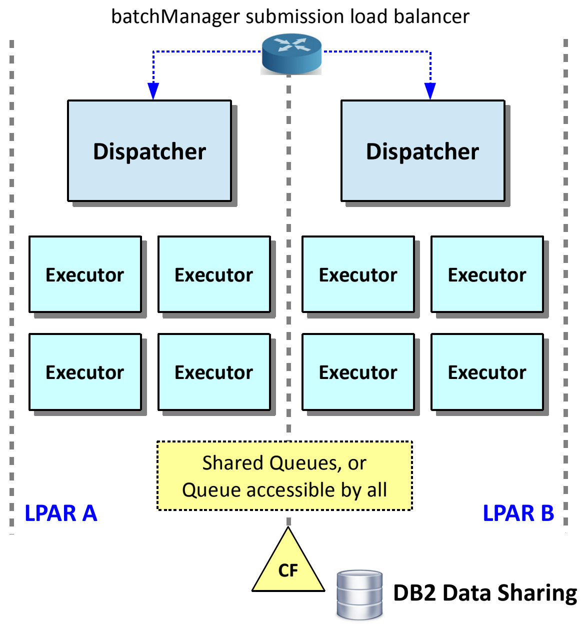
The first two Executor messageSelector= attributes are fairly straight-forward: select on the application name for SleepyBatchlet AND either jobPriority = '1' or '2', dependin on the server.

The third Executor is watching the same queue, but it has to be told to *overlook anything with jobPriority of '1' or '2'*, and to pick up BonusPayout if it sees that. The chart shows a way that can be achieved -- the parenthesis enclose the condition of application name equal to SleepyBatchlet AND NOT jobPriority '1' AND NOT jobPriority '2' ... and the conditional OR picks up BonusPayout. If a message comes on the queue with SleepyBatchlet and jobPriority '3' it'll be picked up by this.



IBM WebSphere Liberty Batch z/OS

What About Multiple Dispatchers?



Yes, this is possible … and in fact can provide a highly available Java Batch runtime environment

With Liberty's ability to share applications and configuration elements with <include> processing, along with z/OS shared file systems, some creative things can be accomplished.

© 2017 IBM Corporation 16 **Summary ...**

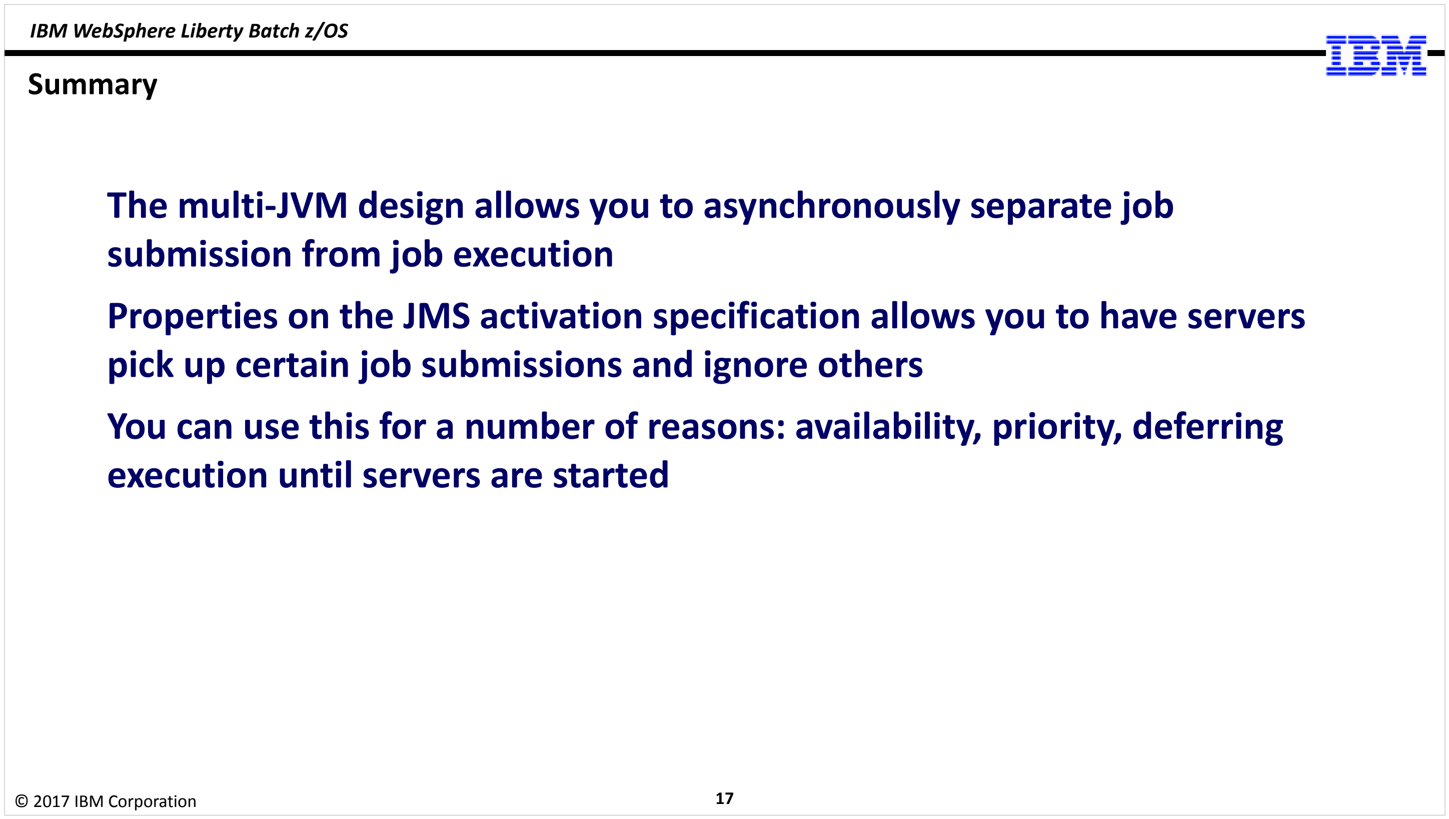
Up to this point we've referenced the case where you have multiple Executors, but one Dispatcher. You may wonder, "What about multiple Dispatchers? Is that possible?" And the answer is "Yes, you can do that as well."

In fact, that becomes the basis for an even more highly-available design. You could have multiple dispatchers feeding multiple executors, with a load-balancing device out front of the dispatchers distributing the batchManager REST-based requests to either LPAR. With shared queues and DB2 data sharing in a Parallel Sysplex environment, this is quite achieveable.

Note: in IBM WebSphere Liberty Java Batch, the job has no affinity to the dispatcher used to submit the job. You can submit the job through a dispatcher, and if that dispatcher crashes the job continues and can be monitored and controlled through a surviving dispatcher. (This was *not* true with Compute Grid, but it's true with the new IBM WebSphere Liberty Java Batch.)

Further, with Liberty's ability to use <include> processing to share common configuration and application elements between servers, you could take advantage of z/OS shared file systems as well.

The lab systems are monoplexes, so we can't have you do this in lab. But there's nothing about this picture that's outside what Parallel Sysplex can do as a normal part of what it's designed to do.



IBM WebSphere Liberty Batch z/OS

Summary

- The multi-JVM design allows you to asynchronously separate job submission from job execution
- Properties on the JMS activation specification allows you to have servers pick up certain job submissions and ignore others
- You can use this for a number of reasons: availability, priority, deferring execution until servers are started

© 2017 IBM Corporation 17

And here's the summary for this unit.

End of Unit