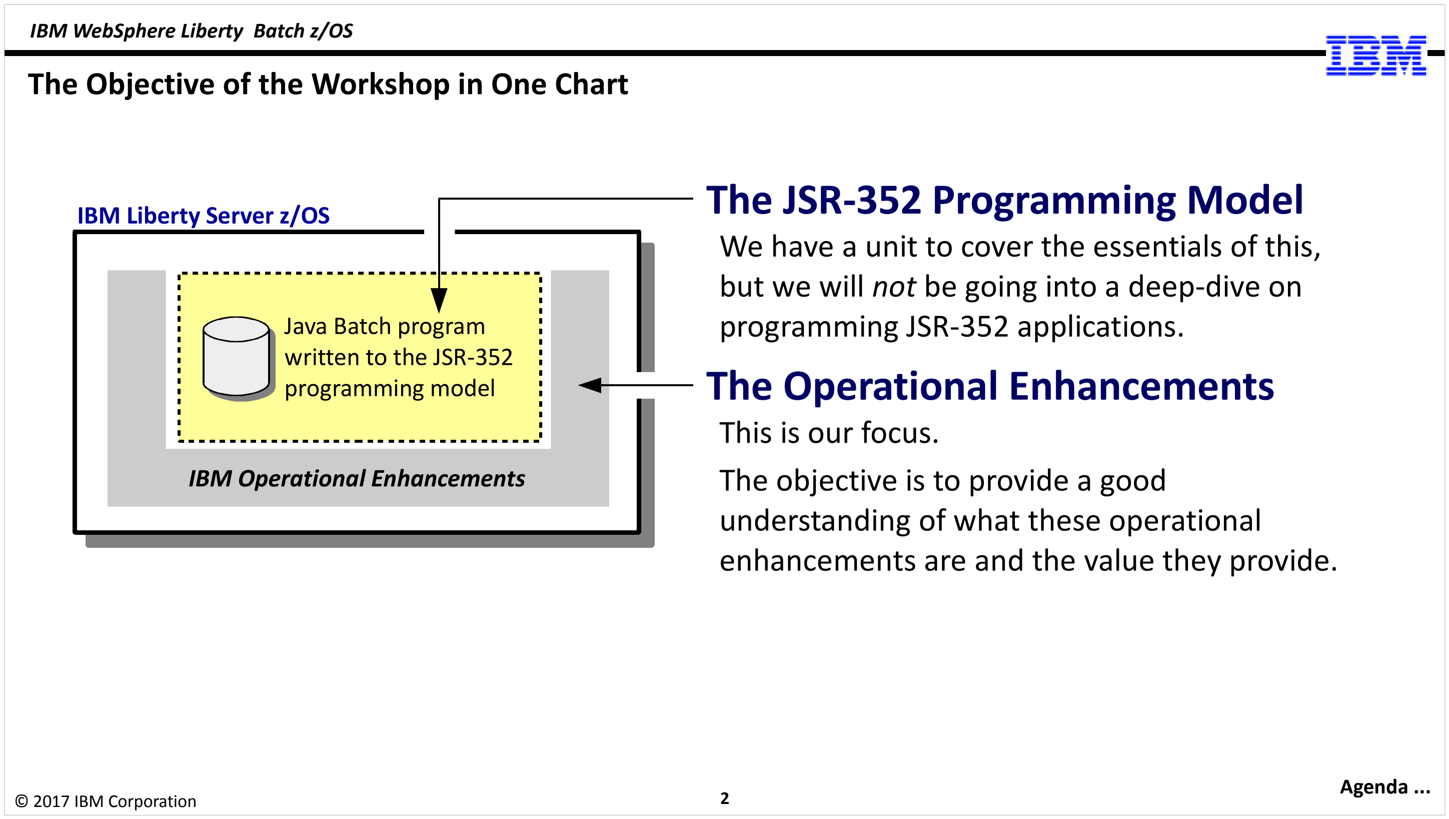
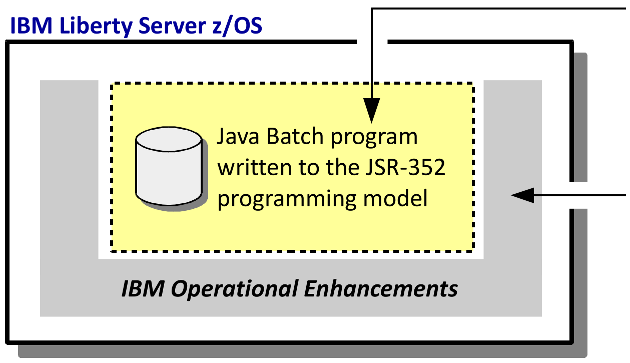
# IBM Liberty Java Batch Workshop



*IBM WebSphere Liberty Batch z/OS*

## The Objective of the Workshop in One Chart


IBM Liberty Server z/OS
Java Batch program written to the JSR-352 programming model
IBM Operational Enhancements

### The JSR-352 Programming Model

We have a unit to cover the essentials of this, but we will *not* be going into a deep-dive on programming JSR-352 applications.

### The Operational Enhancements

This is our focus.

The objective is to provide a good understanding of what these operational enhancements are and the value they provide.

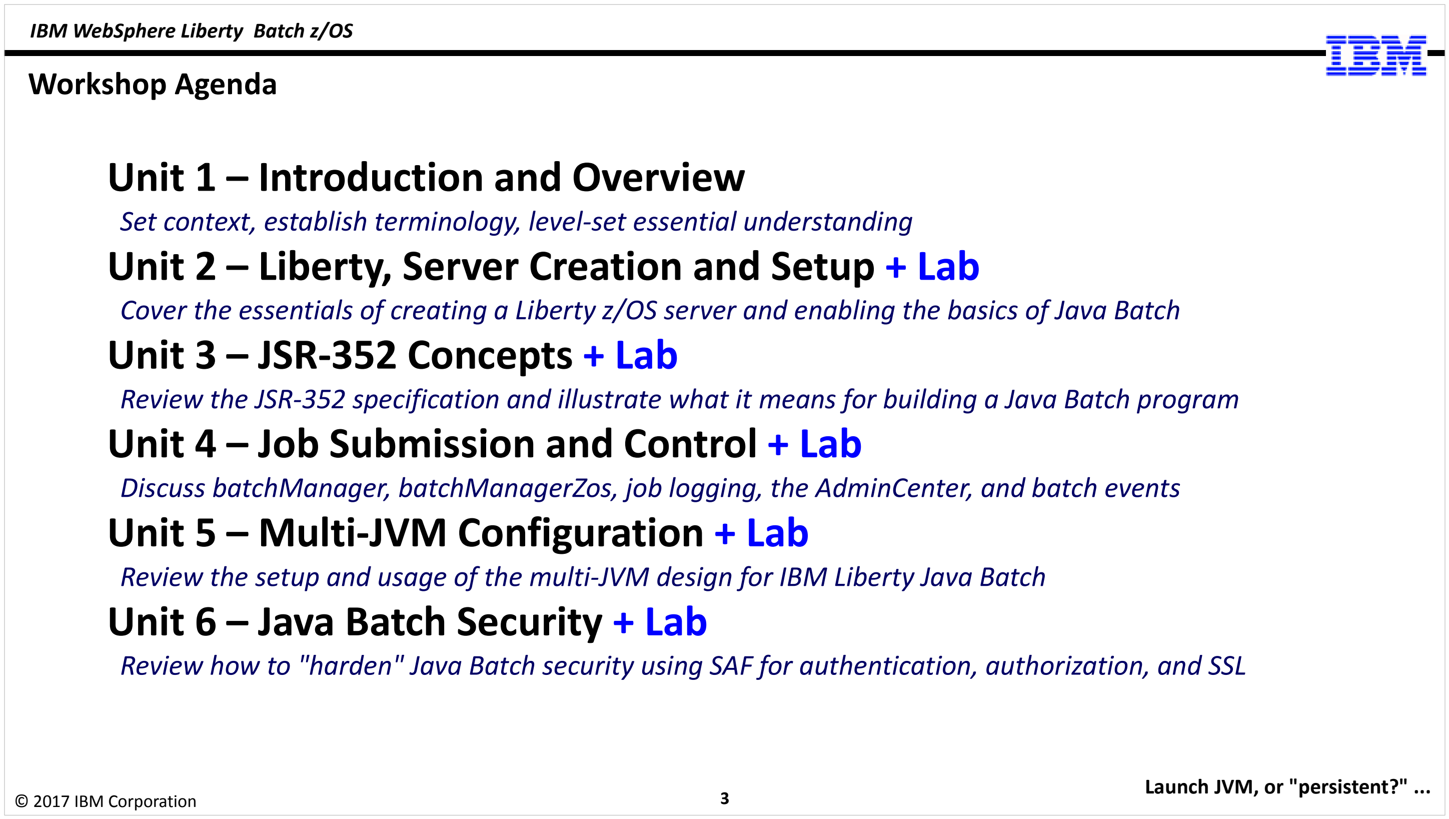
© 2017 IBM Corporation 2 **Agenda ...**

To start this unit, we'll summarize the objectives of the workshop into one picture:

- The JSR-352 specification is the industry's agreed-to standard for a Java batch programming model. In this workshop we will cover some of the *essentials* of this (in Unit 3), but this workshop is not designed to be a deep-dive on programming to the specification. It's an interesting and important topic, but it's not one that *this* workshop will attempt to address.

  **Note:** The WP102706 white paper at ibm.com/support/techdocs provides an approachable narrative on understanding the specification, leading to a deeper appreciation of what it can do. The JSR-352 specification document is also a very good source of information. The URL for that can be found in Unit 3 of this workshop.

- The JSR-352 specification does not attempt to define all the operational aspects of an implementation of the standard. It leaves much of that up to the vendors who implement the standard. IBM is one such vendor, and IBM has added a number of operational enhancements *around* the programming standard to assist with running JSR-352 Java Batch jobs in a production environment. This is the focus of this workshop -- the operational enhancements added by IBM.



*IBM WebSphere Liberty  Batch z/OS*

## Workshop Agenda

**Unit 1 – Introduction and Overview**

*Set context, establish terminology, level-set essential understanding*

**Unit 2 – Liberty, Server Creation and Setup + Lab**

*Cover the essentials of creating a Liberty z/OS server and enabling the basics of Java Batch*

**Unit 3 – JSR-352 Concepts + Lab**

*Review the JSR-352 specification and illustrate what it means for building a Java Batch program*

**Unit 4 – Job Submission and Control + Lab**

*Discuss batchManager, batchManagerZos, job logging, the AdminCenter, and batch events*

**Unit 5 – Multi-JVM Configuration + Lab**

*Review the setup and usage of the multi-JVM design for IBM Liberty Java Batch*

**Unit 6 – Java Batch Security + Lab**

*Review how to "harden" Java Batch security using SAF for authentication, authorization, and SSL*

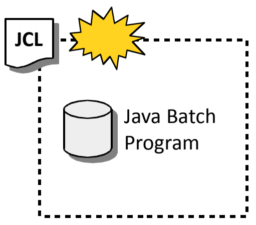
© 2017 IBM Corporation 3

**Launch JVM, or "persistent?" ...**

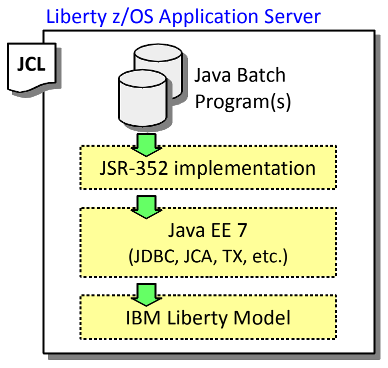
This is the agenda we will follow for this workshop.  We have six units, and the final five units have hands-on labs associated with them.  The agenda is designed to flow systematically from one topic to the next, each building on what was learned in the prior units.

**Note:** we are deferring the topic of security for last for two reasons – (1) it's a complex topic and to wade into that before we've absorbed other key concepts first would be detrimental to learning, and (2) we realize some may not be as interested in the topic as others, so putting this last allows those less interested to finish up the workshop a little early if they need to leave.

## Java Batch: Launch a JVM each time?  Or use a "persistent" JVM model?



- This is what early Java Batch projects used, typically with a Java main() program invoked.
- Downside is you incur the overhead of JVM instantiation and tear-down each time.
- The JSR-352 programming model *can* be used with this model, but not with IBM's implementation of JSR-352.



- This is how IBM implemented the JSR-352 programming model*.
- It is available on all platforms, not just z/OS. z/OS is the focus of this workshop.
- The Liberty server stays active as long as you leave it up.  The batch application is deployed into Liberty like any other application.  It sits ready to be "submitted," at which time it will run and perform batch processing.
- *How* batch jobs are submitted and controlled is a focus of this workshop.

* JSR-352 is part of the Java EE 7 specification, so any Java EE 7 compliant runtime provides the programming interfaces.  Therefore, the recent WAS traditional implementation with Java EE 7 has the JSR-352 compliant programming interfaces, but **not** the operational enhancements we will cover in this workshop.  Only the Liberty implementation has those enhancements.

**Why Java Batch? ...**

When thinking about Java for batch processing, there are two essential ways it can be done:

- **Launch a JVM for each batch job** -- in this model a JVM is launched and a Java batch program is launched.  The Java batch program executes, and when it completes the JVM is taken down.  This model is what early adopters of Java batch (in the days before the open standard) used.  They wrote a Java main() program, launched a JVM and ran the program.  This can be used with the JSR-352 standard as well, but not with IBM's implementation.  The downside of this approach is the overhead associated with the creation of the JVM.  For just a few batch jobs the overhead is small, but when the number of Java batch jobs reaches into the thousands or tens of thousands, then the cummulative overhead becomes an issue.
- **Use a "persistent" JVM model to run batch jobs** -- this is the model IBM uses with its Liberty Java Batch offering.  The "persistent" JVM is that run inside a Liberty server instance.  This is true for Liberty on all supported platforms, not just z/OS.  This model implies the starting of the Liberty server instance, and then the submission of the Java batch jobs at some time after server start.  When the Java batch program ends, the JVM stays active.  Barring some problem with the server, it can stay active for days or weeks at a time, running Java batch jobs over and over again.x

Our focus is the "persistent" model, and more specifically how jobs are submitted and controlled within this environment.

## Why Java Batch?

**People have varying reasons. Here are a few:**

- **On z/OS, zIIP offload from General Processors**
- **Availability of programming skills**
- **Desire to modernize certain batch processing**
  *Example: externalizing business rules to a rules engine*
- **Others?**

Approaches ...

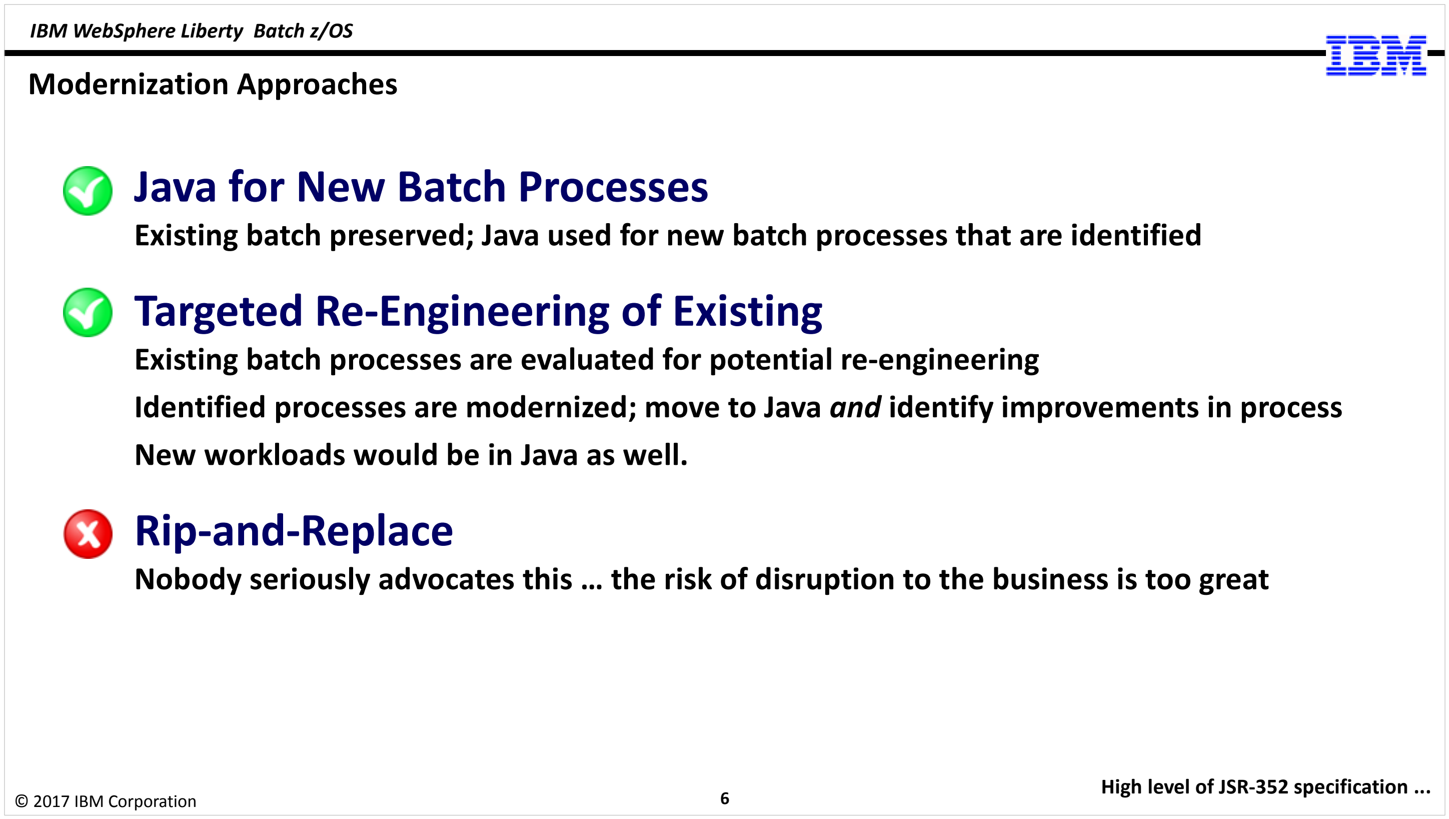
A natural question comes up regarding batch processing and Java: "Why?" There are a number of reasons we've heard over the years, and we'll cover a few of them on this chart:

- **z/OS speciality engine offload** – on z/OS there exists a processing type that is labeled a "specialty engine" (often referred to as "zIIP" engines) that process certain work such as Java. The work that runs on these specialty engines does not count towards the software license charges that are a function of work seen on the general processors. That's a financial incentive to run those designated workloads, including Java, on z/OS. Given that batch processing is a fairly large user of general processor (GP) engines in many locations, there is a desire to reduce the consumption through using Java for batch processing. This is particularly true if the other factors described here are also in play.
- **Skills** – the availability of skills in the "older" programming languages is become less prevalent, while the availability of Java skills is more prevalent. The availability of skills contributes to the cost of acquiring and maintaining those skills, so yet another incentive exists to explore Java as a batch processing language.
- **Opportunity to modernize** – many "legacy" batch processes were written many years ago, and often they still work but do not do *all* the things one would wish them to do. The desire is to crack open those old programs and update them, but then the skills issue comes up, as does the issue of investing more in what is a program asset one hopes oneday to retire. So the desire here is to "modernize" to Java, and – *here's the key* – while re-engineering the batch program in Java take the opportunity to modernize in other ways as well.

  One very interesting opportunity for modernization involves incorporating external rules engines into the design of the batch processing. Often batch programs incorporate business rules hardened in the COBOL coding of the batch program because ... well, that's just what people did back when the program was written. But with the advent of "rules engines" – processes that render decisions based on business rules defined to the process – it's possible to *externalize* the business rules from the batch program. That makes the batch processing much more agile in response to changes to business rules because today's rules engines have very sophisticated business rule editors and repositories. Changes to the business rules are often dynamic, which means if on Tuesday at 2:45pm it's decided to change a business rule (say, a certain discount based on some factor such as prior purchases made), then that change can be made to the business rule and a batch program that uses the rules engine can run at 5:00pm and make use of the new rule. For IBM the rules engine is "IBM Operational Decision Manager." It is a rules engine that runs in either a Java server (such as Liberty), or a standalone started task, or in a CICS region.

You may have other reasons to be looking at Java Batch. There is no "one" reason; there's a set of reasons and you give different weight to each factor than someone else may weight them. In the end, a decision is made to pursue Java for batch.

This workshop is designed to help you better utilize IBM WebSphere Liberty Java Batch.



*IBM WebSphere Liberty Batch z/OS*

## Modernization Approaches

### ✓ Java for New Batch Processes

**Existing batch preserved; Java used for new batch processes that are identified**

### ✓ Targeted Re-Engineering of Existing

**Existing batch processes are evaluated for potential re-engineering**

**Identified processes are modernized; move to Java *and* identify improvements in process**

**New workloads would be in Java as well.**

### ✗ Rip-and-Replace

**Nobody seriously advocates this … the risk of disruption to the business is too great**

© 2017 IBM Corporation 6

**High level of JSR-352 specification ...**

There are a few general approaches to adopting Java as a batch platform for batch modernization. This chart touches on them at a high-level.

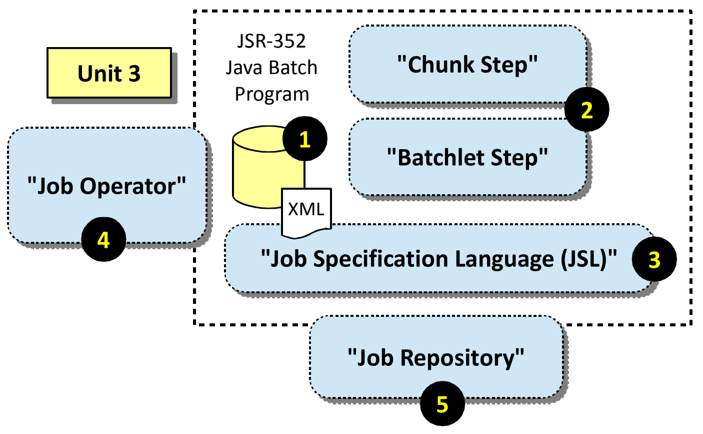
**Note:** there's no formula, as you'd expect. Everyone has a different set of issues, needs, and requirements. What we're showing here is a broad framework that could serve as a starting point for a deeper discussion.

- *New processes* -- the approach here is to draw a line and preserve existing batch processes as they are, but target any new batch processes to adopt Java as the programming language. This would allow you to design for Java from the start on these new processes. This preserves existing processes as they are today, which minimizes any risk of business disruption.
- *Re-engineer some existing* -- this approach involves assessing existing batch processes for re-engineering to Java. Once existing processes are identified as potential candidates, they are evaluated to understand better what the batch processes do, and then a design evaluation begins on how that process can be most effectively modernized.

  **Note:** simply re-writing the existing COBOL processes in Java so the Java merely mimics the COBOL is rarely recommended. When modernizing an existing process there's an opportunity to improve the processing, with things such as multi-row fetch of database records, or, as we mentioned, extracting business rules and externalizing them to rules engines.
- *Rip-and-replace* -- we offer this as an example of what you should *not* do. Nobody approaches batch modernization with a rip-and-replace attitude; the risk to the business would be too great. The approach is always adoptive; either the first (new processes), or the second (targeted re-engineering of existing), or some combination of the two.

*IBM WebSphere Liberty Batch z/OS*

## Very High-Level Overview of the JSR-352 Programming Model



### 1. Java Batch Program

You write this based on the defined JSR-352 requirements. This is packaged as a servlet (WAR) and deployed into Liberty just like any other application would be deployed.

### 2. Job Step Programming Types

Two job step types defined:

**Chunk:** the looping model we most often associate with batch processing. This includes functions such as checkpointing, commits and rollbacks, and job restarts.

**Batchlet:** a simple "invoke and it runs" model. This is useful for non-looping functions such as file FTP steps.

### 3. Job Specification Language (JSL)

An XML specification to describe the batch job: the steps, the Java programs that implement the steps, and the flow of steps within the job.

### 4. Job Operator

This interface defines how to submit and control jobs. This workshop focuses on the IBM enhancements around this job operator definition.

### 5. Job Repository

The specification calls for a repository to track job submissions and results, but leaves it to the vendor to implement. We'll use IBM DB2 z/OS in workshop.

**Job step flow ...**

Here we will offer a very high-level discussion of the JSR-352 programming model specification. Unit 3 of the workshop is designed to provide a bit more detail on some of this.

1. **Java Batch Program** -- this is written to the interface methods defined by the specification. It is packaged as a Web ARchive (WAR) or Enterprise ARchive (EAR) and deployed into the Liberty server, either in the /dropins directory or specified with an <application> element in the server.xml file. The program is "started" by Liberty, but the batch processing does not occur until the program is submitted through the "Job Operator" interface.
2. **JSR-352 Step Type** -- there are two job step types defined: "Chunk" and "Batchlet". A chunk step is the common model of a batch process that loops over data, reading records in, processing the records, and writing them out. A batchlet step is a simpler model in which the step's process() method is called by the batch container, and whatever code you have written in that method will operate. Batchlet steps are useful for non-looping actions, such as FTPing a file.
3. **Job Specification Language** -- or JSL for short, this XML file defines the job, the steps, and the Java class files that implement the job steps. The file also defines the execution flow of steps. If you're familiar with mainframe JCL, then the *concept* of this will be very familiar to you; the difference being the syntax: JCL uses // and JSL is XML with <tags>.
4. **Job Operator** -- this is an interface to the batch container that provides a way to submit and control Java batch jobs in the container. For example, it is through this interface that you would "submit" a job and start it running. It is through this job that could "stop" a job from executing.

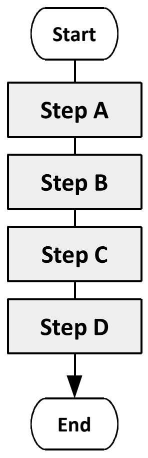
   **Note:** this is the area around which a fair deal of focus will be made in this workshop. IBM has added operational enhancements to the job submission and control process. That is the focus of Unit 4 of this workshop.
5. **Job Repository** -- the JSR-352 specification indicates that a "job respository" is needed, but it does not spell out the details of implementation. IBM has implemented the job repository as either (a) in-memory, which is very simple and great for development activity, but the information will not persist over server restarts; (b) file-based Derby relational database, which addresses the information persistence over server restarts, and is good for development and testing, but is not really robust enough for full production; and (c) vendor relational database products. For this workshop we're going to use IBM DB2 z/OS.

There's more detail to the JSR-352 specification. This provided a high-level overview.

*IBM WebSphere Liberty Batch z/OS*

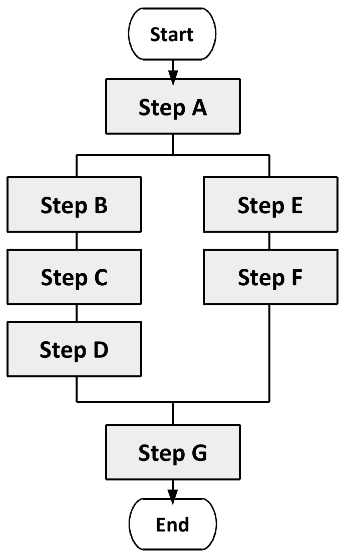
## Job Step Flow Control

A simple sequential processing of steps:



This is what we commonly think of as "batch," and this may be what you do

You can also accomplish more sophisticated flows



The JSR-352 specification has the ability to define simple or sophisticated step processing flows within a job.

The Job Specification Language (JSL) file is what spells all this out.

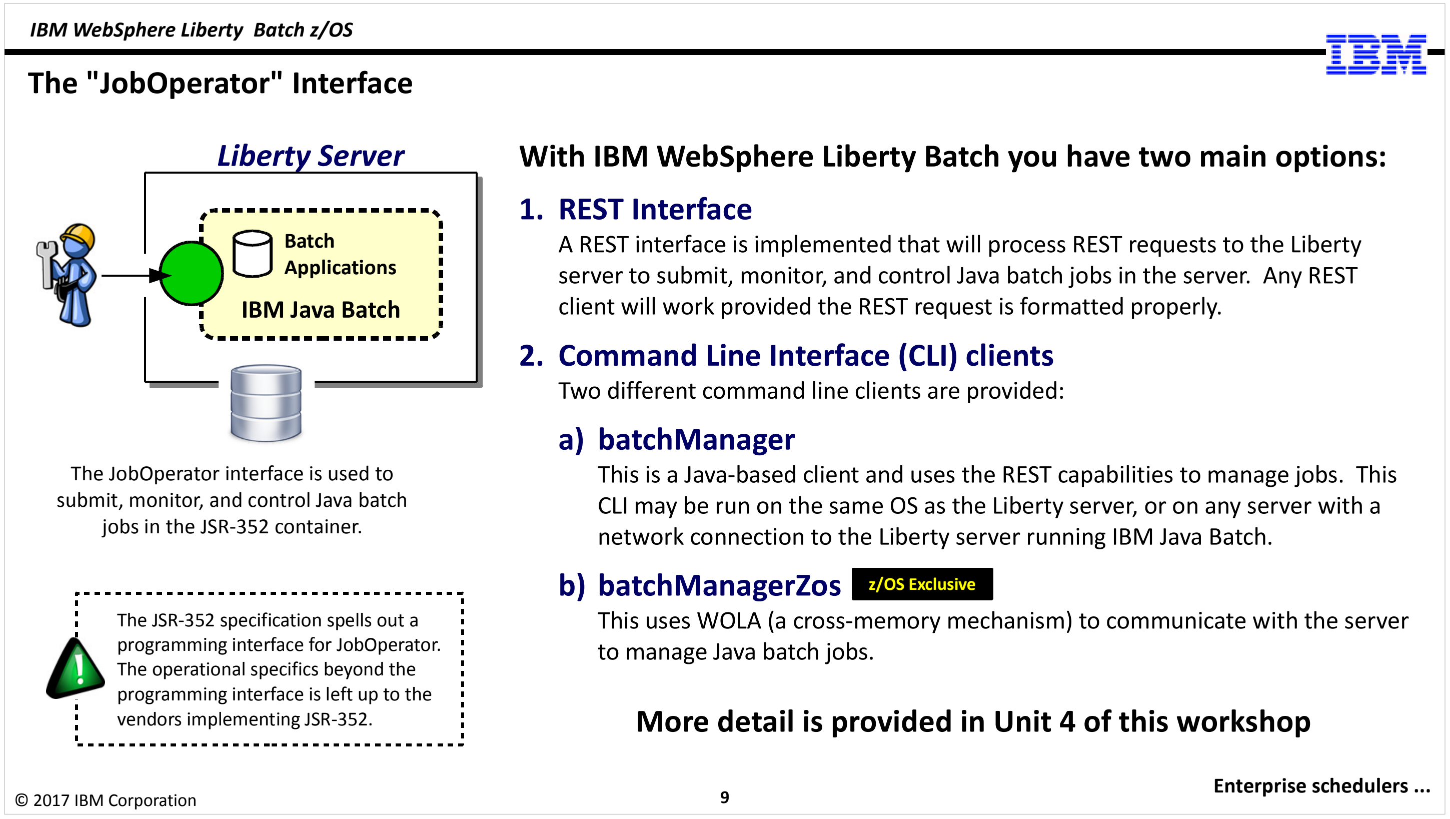
In Unit 3 we will explore the JSL definitions and the tooling that generated the file

Job Operator ...

The previous chart mentioned that JSR-352 jobs can consist of multiple steps, of either "Batchlet," "Chunk," or both types. If a job has more than one step, those steps can be arranged in a simple linear fashion as illustrated on the left side of the chart. Of, if your batch processing supports concurrent operation, then the steps can be arranged into "splits" with paths flowing like what's shown in the center of the chart.

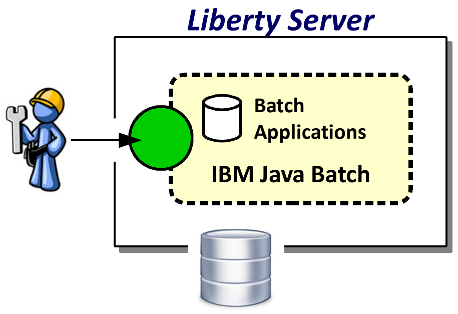
**Note:** you can have even more sophisticated structures than what's shown here – a "split" can go more than two wide, and it's possible to have splits inside of splits. The flexibility is considerable.

All this is spelled out in the JSL file, which has XML syntax to define steps, splits, flows, go-to next, and many other things. We'll explore what the JSL file looks like in more detail in Unit 3, as well as look at the IBM tooling in support of JSR-352 programming and JSL file creation.



*IBM WebSphere Liberty Batch z/OS*

## The "JobOperator" Interface



The JobOperator interface is used to submit, monitor, and control Java batch jobs in the JSR-352 container.

The JSR-352 specification spells out a programming interface for JobOperator. The operational specifics beyond the programming interface is left up to the vendors implementing JSR-352.

**With IBM WebSphere Liberty Batch you have two main options:**

1. **REST Interface**
   A REST interface is implemented that will process REST requests to the Liberty server to submit, monitor, and control Java batch jobs in the server. Any REST client will work provided the REST request is formatted properly.
2. **Command Line Interface (CLI) clients**
   Two different command line clients are provided:
   a) **batchManager**
      This is a Java-based client and uses the REST capabilities to manage jobs. This CLI may be run on the same OS as the Liberty server, or on any server with a network connection to the Liberty server running IBM Java Batch.
   b) **batchManagerZos** z/OS Exclusive
      This uses WOLA (a cross-memory mechanism) to communicate with the server to manage Java batch jobs.

**More detail is provided in Unit 4 of this workshop**

**Enterprise schedulers ...**

© 2017 IBM Corporation 9

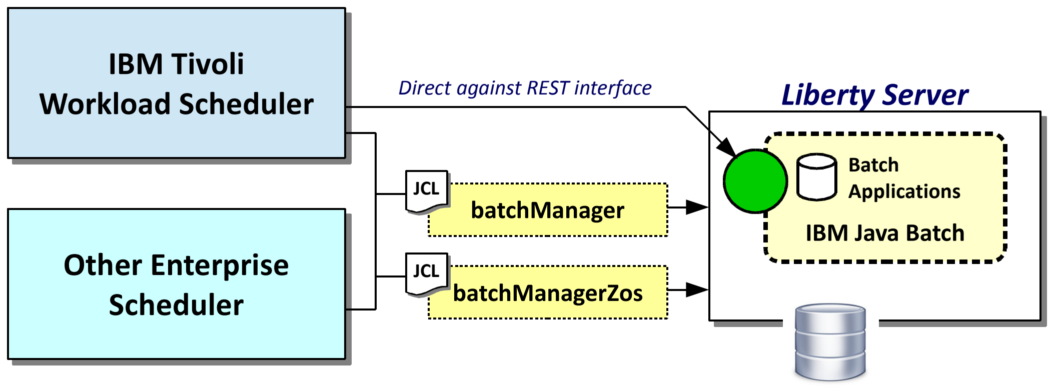
Another element of the JSR-352 specification is something called a "JobOperator," which is the interface to submit and control job execution. The specification very clearly spells out the *programming interfaces* of the JobOperator, but it says nothing about how those programming interfaces are to be invoked. It assumes *something else* is in front of those JobOperator APIs.

That "something else" could be a bit of Java code you write to invoke the JobOperator APIs to control the job. That's exactly what's in the SleepyBatchlet sample we use extensively in this workshop. There's an included servlet you can access with a browser that will start the batch job.

Or that "something else" could be a vendor-supplied function that sits in front of the JobOperator APIs and provides a set of interfaces for submitting and controlling jobs. That's what the IBM WebSphere Liberty Java Batch function provides when you enable the batchManagement-1.0 feature in the Liberty server: it provides a RESTful interface, and two command line clients. The RESTful interface can be invoked with any REST client that undestands the URI patterns and the supported JSON schemas. The two command line clients are (1) batchManager, which is a Java based function that uses the REST interface; that is, it is a REST client with a command line interface on the front; and (2) batchManagerZos, which runs only on z/OS and uses WebSphere Optimized Local Adapters, or WOLA, to access the Liberty z/OS server to submit and control the jobs.

Unit 4 of this workshop covers all this in much greater detail.

## Integrating Enterprise Scheduler Functions


IBM Tivoli Workload Scheduler
Direct against REST interface
Liberty Server
Batch Applications
IBM Java Batch
JCL
batchManager
JCL
batchManagerZos
Other Enterprise Scheduler

### Key things to this:

- **Be able to submit the Java Batch job inside the IBM Liberty Java Batch environment**
- **Hold completion of CLI invocation until Java Batch inside container completes**
- **Be able to retrieve the job log output if desired**

**We explore batchManager and batchManagerZos in Unit 4**

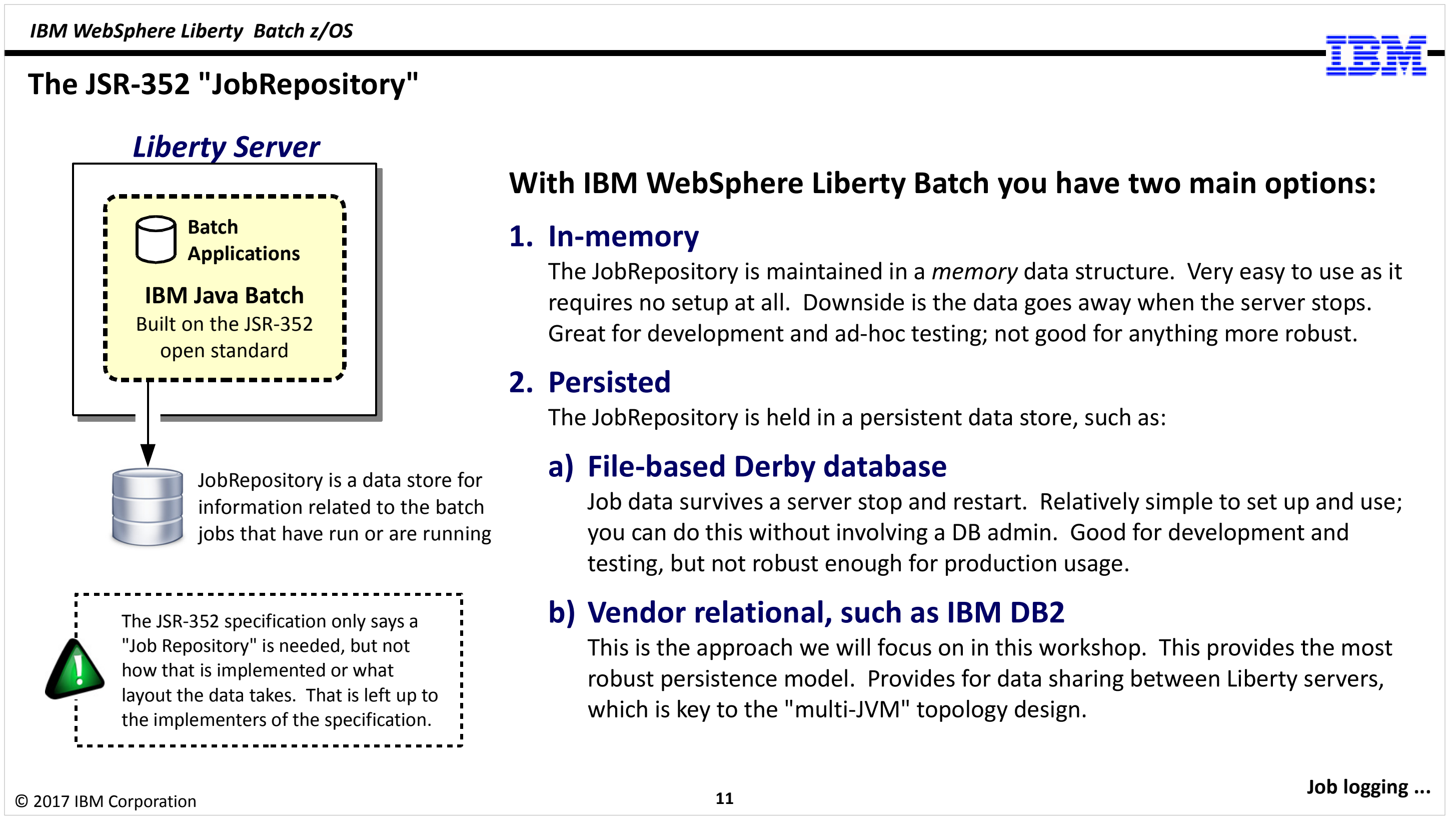
**Job Repository ...**

These command line clients are the key to integration with Enterprise Scheduler functions, such as IBM Tivoli Workload Scheduler, or other vendor products such as CA-7 or Control-M (the latter two products offered by Computer Associates and BMC, respectively).

**Note:** before we get to the use of batchManager or batchManagerZos as the integration point for enterprise schedulers, a brief comment about IBM Tivoli Workload Scheduler: it knows about and has the ability to interact with the IBM WebSphere Liberty Java Batch REST interface directly. That bypasses the need for either batchManager or batchManagerZos. You can still use either batchManager or batchManagerZos with IBM Tivoli Workload Scheduler, or it can be used with the REST interface directly.

Those two command line clients can be submitted using JCL, which means enterprise schedulers can be configured to submit those JCL jobs. But if the command line clients simply started the Java batch jobs and quit, the enterprise scheduler would be left in the dark about the status of the Java batch job. So the client don't just submit-and-quit ... they are capable of staying active until the Java batch job ends, then returning the status to the JCL job and thus the enterprise schedulers. Further, to effectively integrate with enterprise schedulers there must be a way to get the Java batch job log back to the JCL job the enterprise scheduler submits, for that's how batch job archiving is frequently done. The two command line interface utilities permit this (as does the REST interface, by the way).

In a sense the command line client serves as a kind of "proxy" for the Java batch job operating back in the Liberty server. If the batchManager or batchManagerZos client is up, it means the Java batch job is still running. When the command line utility ends, the Java batch job has ended.



*IBM WebSphere Liberty Batch z/OS*

## The JSR-352 "JobRepository"

Liberty Server
Batch Applications
IBM Java Batch
Built on the JSR-352 open standard

JobRepository is a data store for information related to the batch jobs that have run or are running

The JSR-352 specification only says a "Job Repository" is needed, but not how that is implemented or what layout the data takes. That is left up to the implementers of the specification.

**With IBM WebSphere Liberty Batch you have two main options:**

1. **In-memory**
   The JobRepository is maintained in a *memory* data structure. Very easy to use as it requires no setup at all. Downside is the data goes away when the server stops. Great for development and ad-hoc testing; not good for anything more robust.
2. **Persisted**
   The JobRepository is held in a persistent data store, such as:
   a) **File-based Derby database**
   Job data survives a server stop and restart. Relatively simple to set up and use; you can do this without involving a DB admin. Good for development and testing, but not robust enough for production usage.
   b) **Vendor relational, such as IBM DB2**
   This is the approach we will focus on in this workshop. This provides the most robust persistence model. Provides for data sharing between Liberty servers, which is key to the "multi-JVM" topology design.

© 2017 IBM Corporation 11 **Job logging ...**

The JSR-352 specification calls for a "Job Repository," meaning some sort of knowledge base of what jobs have run and are running. The specification does not, however, spell out how such a Job Repository must be implemented. The standard leaves that up to the implementors of JSR-352 Java Batch.

The IBM WebSphere Liberty Java Batch implementation provides two main options for this – (1) an in-memory implementation of the JobRepository, and (2) a persisted model. Under the persisted model are two sub-options: file-based Derby database, or a vendor relational database such as IBM DB2.

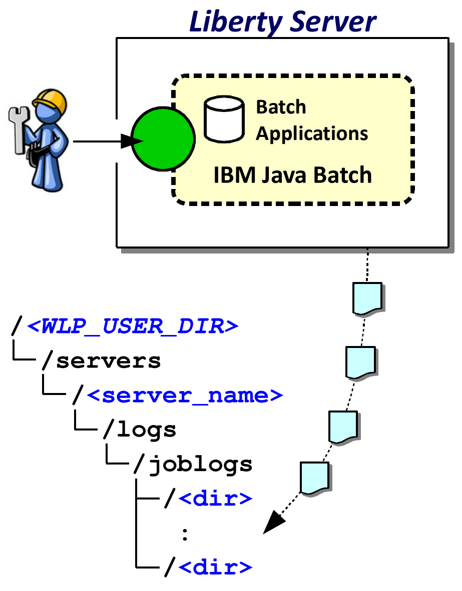
The in-memory model is *great* for development and ad hoc testing. It's the very simplest to set up because there's nothing to set up ... if you don't specify batch persistence, then it defaults to in-memory. There's no tables to define, and no server.xml updates to define JDBC providers and data sources. The "tables" are maintained in memory.

The downside to that, of course, is that if you stop the server those in-memory tables go away. When you start the server again, you start fresh. That may be *exactly* what you want for development and ad hoc testing. But it is likely not what you want for other testing, and certainly not for production.

So we have the persisted model, which uses relational tables that survive from server start to server start. The Derby option is for cases where you want to do testing with a persisted JobRepository, but you do not have the ability to get tables created in a relational product such as IBM DB2. With the Derby option you can create the tables yourself without involving others. Whether this is robust enough for production is open to debate.

True relational database products such as IBM DB2 are what's used for production-level Java Batch. In particular, these products allow for data sharing between servers, and that is key to the use of the "multi-JVM" topology model we'll describe in a bit. For this workshop we'll use IBM DB2 z/OS.

## Job Logging



- **For each job that executes the server will write job log files to separate directories under /logs**
  The files are not all jumbled together; the job log tree is very orderly based on application name, date, and execution numbers
- **This is controllable by server**
  You can turn logging on or off on a server-by-server basis; you may also control log file size (if maximum exceeded it closes file and opens another job log part file).
- **Can be directed to a separate location using the Liberty WLP_OUTPUT_DIR variable**
  This gives you a way to direct output to a location where you can mount a large ZFS to hold verbose output
- **batchManagerZos: capture "as it happens"**
  This is a way to route to JES spool
- **Can retrieve job logs using REST interface**
  Job log parts are returned as JSON objects.
- **Can retrieve job logs using batch events**
  The jobLogPart topic contains a job's log parts
- **Can retrieve directly from the file system**
  They're just files in a directory, so this avenue is always available

**Unit 4**

Batch events ...

The JSR-352 specification mentions job logging as a part of batch processing, but it does not specify any details about it. Therefore, job logging is left up to each implementor as to how it will be done.

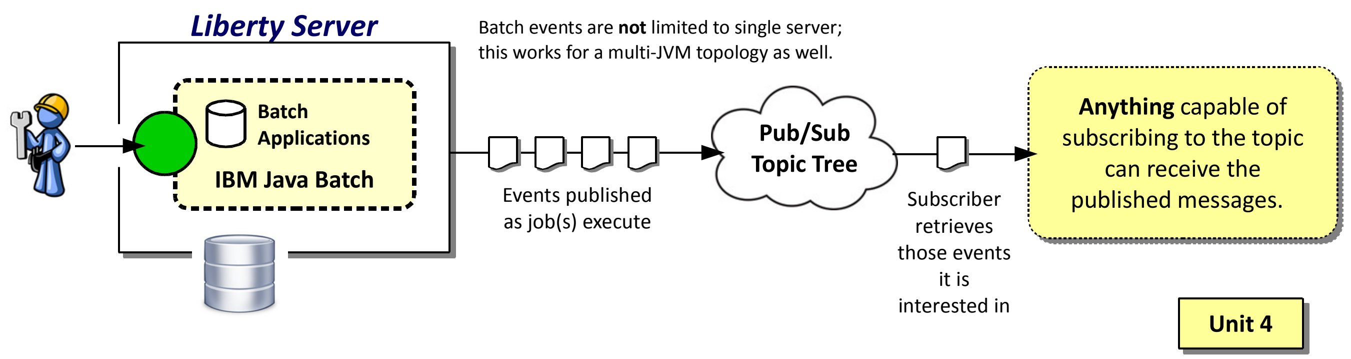
For IBM WebSphere Liberty Java Batch, job logging is enabled with the batchManagement-1.0 feature, and is controllable by server with XML in the server.xml file. Logging can be turned on or off with XML, and the size of the log parts can be deterimined as well.

The logs go to the /jogLogs directory under the /logs directory for the server. By default that will be /logs under the server directory, but the WLP_USER_DIR variable of Liberty can be used to redirect the server output location to any path you wish. The job log files are separated by data, job instance number and job execution number (we'll see what that means in Unit 4).

How can you retrieve the job logs? There are several ways that can be done, including using the REST interface, or the "getJobLog" parameter on the batchManager and batchManagerZos command line, or by going directly to the file location. We cover job logging in more detail in Unit 4.

*IBM WebSphere Liberty Batch z/OS*

## Batch "Events" and Monitoring Job Activity


Liberty Server
Batch events are **not** limited to single server; this works for a multi-JVM topology as well.
Batch Applications
IBM Java Batch
Events published as job(s) execute
Pub/Sub Topic Tree
Subscriber retrieves those events it is interested in
**Anything** capable of subscribing to the topic can receive the published messages.
Unit 4

- **Can be turned on or off on a server by server basis**
  It's a relatively simple `server.xml` update that tells a server to publish events. It does require a properly configured JMS connection to a message queueing mechanism such as IBM MQ
- **Topic tree has events for many state changes in the life cycle of a job**
  Too many to list here, but what you would expect exists: job is queued, starting, executing, stopped, failed, etc.

© 2017 IBM Corporation 13 **Multi-Server ...**

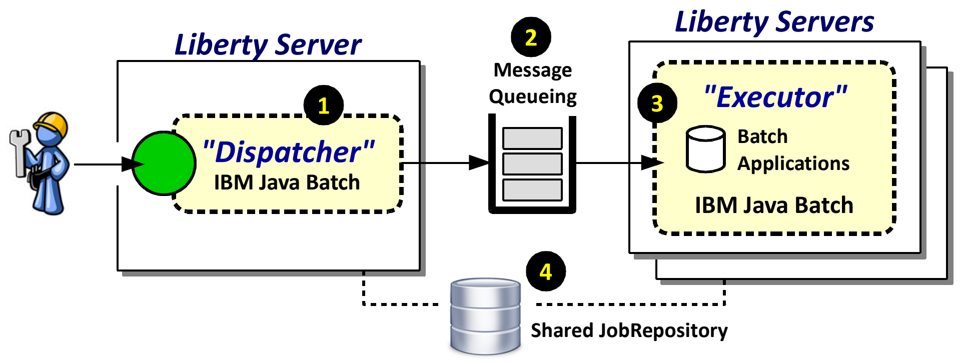
"Batch Events" provides a way to monitor batch job activity by subscribing to a pub/sub topic and watching for messages being published by the servers. If you enable the batch events function, as batch jobs run the server will publish to the topic tree the events for each "state" achieved by the execution of the job. There are many different "states" that are covered – job start, job end, job fail, etc.

The pub/sub topic can be hosted by any messaging function that supports pub/sub and can be accessed from a Liberty server. That includes the default messaging of Liberty itself, as well as a messaging product like IBM MQ.

What can monitor these events? Anything that can subscribe to a topic. You can write your own monitoring software that subscribes to the topic and retrieves those events of interest. For example, a sample monitor program that we'll use in the lab monitors for the job log part, retrieves those and writes them to a file location. The batchManagerZos command line client also has the ability to subscribe and monitor for the status of the job.

*IBM WebSphere Liberty Batch z/OS*

## The "Multi-Server" Topology Model



**The role of dispatcher is separated from executor, and a queueing "pull" model placed between the two**

**Unit 5**

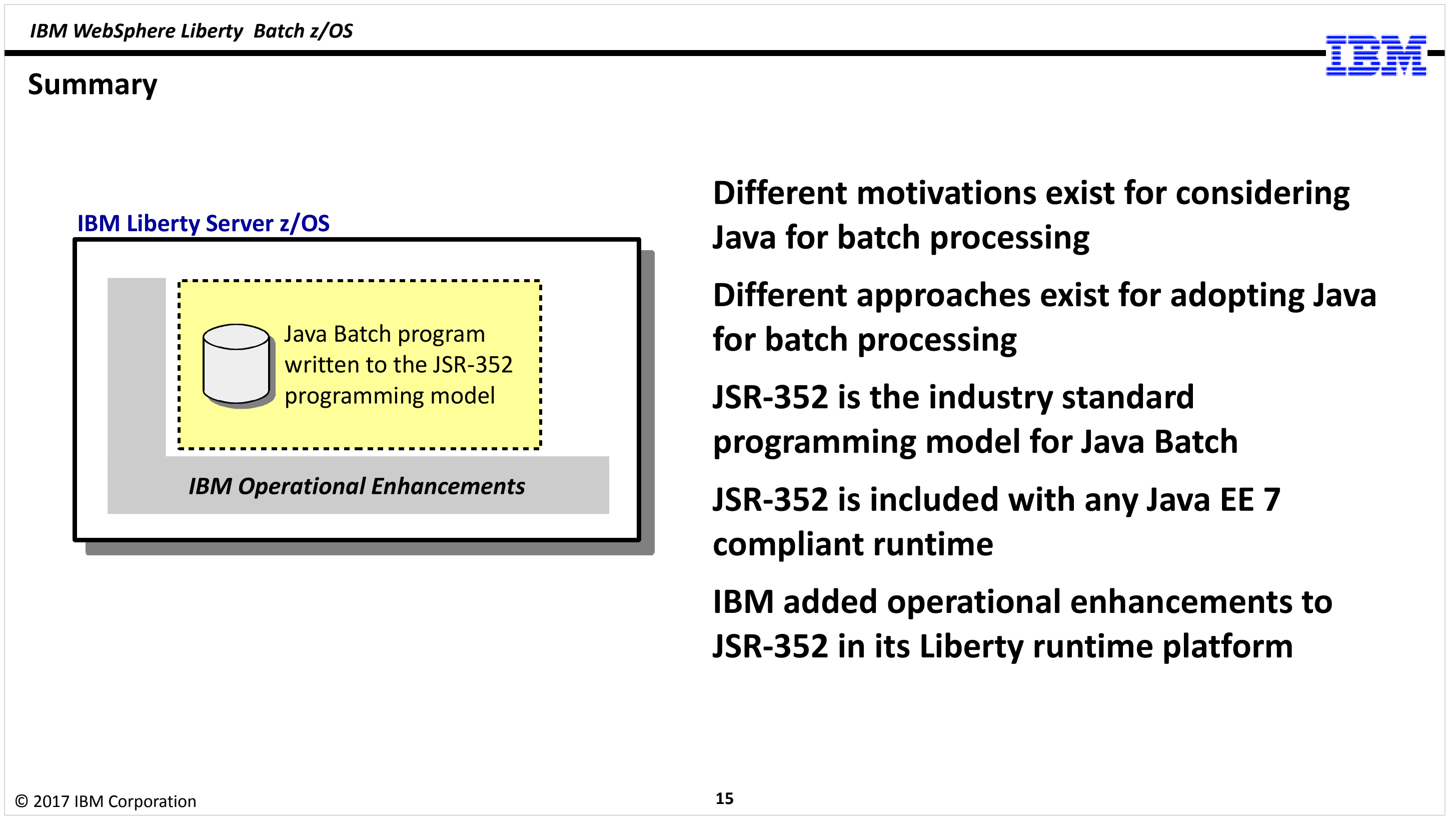
1. **Dispatcher Server**
   This server has no batch applications deployed. It is configured to be a "dispatcher" and will put a job submission message on the configured queue.
2. **Message Queueing**
   This serves as the connection point between the servers. This can be any message queueing function supported by Liberty, including MQ.
3. **Executor Server(s)**
   These servers host the Java batch application. They use JMS activation specs to listen for and pick messages off the queue. They can employ "message selector" strings to pick certain job submission messages but not others.
4. **Shared JobRepository**
   Key requirement: the JobRepository implementation must be shared between servers. DB2 z/OS can do this easily.

14 **Summary ...**

Up to this point we've illustrated the Java batch environment as a single server, but you're not limited to that. The IBM WebSphere Liberty Java Batch function supports a "multi-JVM" topology where a server instance acts as a "Dispatcher," receiving job submission requests, and other servers act as "Executor" servers to run the jobs. Between the two is a queue that serves as the asynchronous integration mechanism. This can be either the default messaging of Liberty, or it can be IBM MQ.

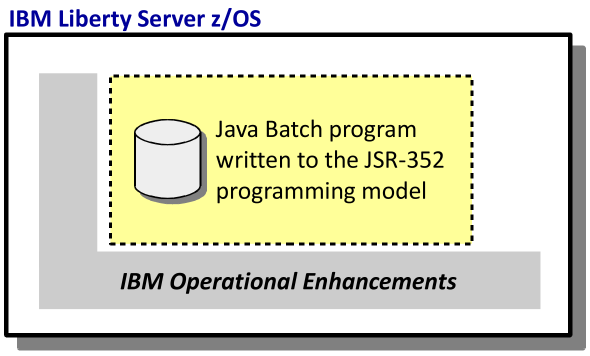
What's the value of this? Several things – (1) it provides a way to have batch jobs run on one of several servers, which provides for both scalability as well as availability; (2) it provides a way to target a job to flow to a server with a higher WLM service class policy if the job is deemed of higher priority (this is done with the message selector definition in the executor, which we'll cover in in Unit 5); and (3) it provides a way to submit jobs at any time, but run them only later when the executor is started.

For this to work the JobRepository must support sharing between servers, which is something the in-memory model does not support but IBM DB2 does.



*IBM WebSphere Liberty Batch z/OS*

## Summary


IBM Liberty Server z/OS
Java Batch program written to the JSR-352 programming model
IBM Operational Enhancements

**Different motivations exist for considering Java for batch processing**

**Different approaches exist for adopting Java for batch processing**

**JSR-352 is the industry standard programming model for Java Batch**

**JSR-352 is included with any Java EE 7 compliant runtime**

**IBM added operational enhancements to JSR-352 in its Liberty runtime platform**

© 2017 IBM Corporation 15

For now that's a good high-level summary of what Java Batch is, what the JSR-352 standard is, and what the IBM operational enhancements to the standard are. That provides us with a good base on which to build with the units that follow, each providing greater detail as well as hands-on labs.

**End of Unit**