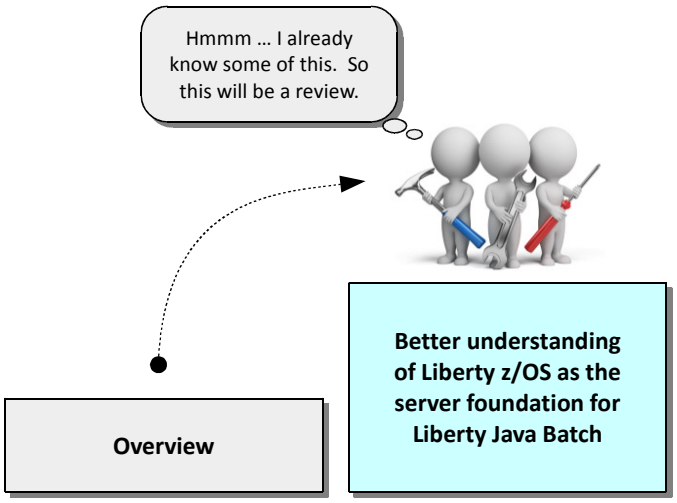
WebSphere Application Server

Unit 2

Liberty, Server Creation and Setup

© 2017 IBM Corporation

Objective of this Unit in the Workshop



- **The Java Batch function operates within the context of a Liberty server**
- **Understanding a few key things about Liberty will help with using Java Batch**
- **Liberty z/OS has a few things unique that allow it to operate on z/OS**
- **Purpose of this unit is to give you an understanding of the Liberty runtime so we can then move on to Java Batch specific things**

© 2017 IBM Corporation

Liberty is a fairly sophisticated application server platform, and to cover it all would be impossible within the time we've set for this workshop. So our objective for this unit is much more limited: to cover some key things about Liberty, and Liberty z/OS, that are fundamental to better understanding how the Java Batch function works inside Liberty.

The IBM WebSphere Liberty Java Batch function runs in Liberty on all platforms supported by Liberty. You can't run this function outside of Liberty. So understanding how Liberty works will help you understand how Java Batch works inside of Liberty.

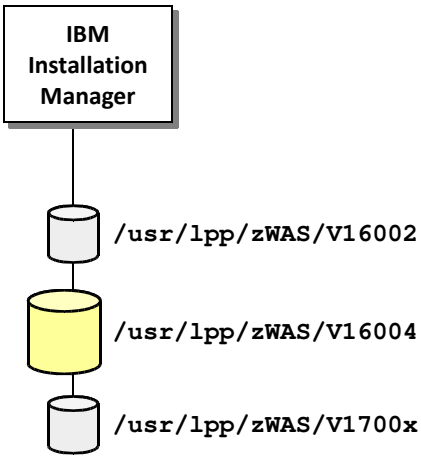
Liberty z/OS has a few things that are unique to z/OS, and the Java Batch function has the ability to take advantage of this. For instance, the batchManagerZos command line client is exclusive to the z/OS platform, and it uses a cross-memory mechanism (WOLA) to communicate with the Liberty server to submit and control jobs. Also, z/OS has the SAF security interface that can be used for security constructs such as the user registry, application role authorization, and SSL digital certificates.

So we'll take a relatively high-level tour of the Liberty server and then we'll go into the first hands-on lab.

© 2017, IBM Corporation

IBM WebSphere Liberty Batch z/OS

Liberty z/OS After the Installation Manager Work Has Been Done



Liberty z/OS requires the use of IBM Installation Manager (IM) to perform the installation

The IM topic is beyond the scope of this workshop; we assume you have IM working

The result of an IM install of a Liberty is a ZFS file system at a mount point

That file system may then be copied, DUMP/RESTORE'd just like any other ZFS

You may have multiple levels of Liberty installed at one time. They are just mounted ZFS file systems.

In fact, we encourage you to install new levels into separate ZFS file systems (rather than "in place" updated of the same ZFS). That makes it very easy to test new levels and to fall back if there's an issue with a new release.

© 2017 IBM Corporation

We'll start with a brief discussion of how Liberty z/OS is installed. It is done using an installation utility called "IBM Installation Manager," or IM for short.

Note: the Installation Manager topic is too big to cover with any detail in this workshop. If you'd like more information, see the WP102544 Techdoc at ibm.com/support/techdocs. That provides information on acquiring IM (it's a no-charge function), installing it, and using it.

What IM produces is a file system at a mount point you specify. That's really all there is to a Liberty z/OS installation: a ZFS file system. It's about 200MB or so in size, and you may have multiple levels mounted at the same time so some servers can operate at one level and other servers at another. The "no migration" design of Liberty – meaning, you can move up a level or fall back a level without requiring any changes to the configuration structure – makes having multiple levels concurrently appealing. We do not recommend the "in place" update of IM where the one level is deleted and a new level installed into the same ZFS; it works, but it takes longer, and the flexibility of having multiple levels present outweighs the additional DASD consumed.

The ZFS file systems are capable of being copied and sent to other LPARs as well, which means you do **not** have to have IM on every LPAR where Liberty z/OS operates. You can – and we encourage this as well – operate a "service zone" for IM work, then copy the file system and move it to whereever it will be used.

© 2017, IBM Corporation

Creating a Liberty z/OS Server

UNIX shell … Telnet, SSH, OMVS

```
> cd /usr/lpp/zWAS/V16004/bin 1
> export JAVA_HOME=/shared/java/J8.0_64 2
> export WLP_USER_DIR=/JavaBatch 3
>./server create JBSRV01 4
```

```
/JavaBatch
└─/servers
  └─/JBSRV01
    ├─/apps
    ├─/dropins
    └─/logs
      server.xml
```

1. **Change to the /bin directory of the Liberty install**
   That is where the 'server' shell script is located. That shell script is used to create the servers.
2. **Export JAVA_HOME**
   Or have that value specified in .profile … key point is the location of a valid 64-bit Java must be specified to the environment with the JAVA_HOME variable.
3. **Export WLP_USER_DIR**
   The WLP_USER_DIR variable specifies where the server will be created. This may be any location you wish. The ID you use to create the server must have WRITE to this directory location.
4. **Create the server**
   The server shell script 'create' verb will cause the server to be created with the name you specify on the create command. The shell script then goes to the specified WLP_USER_DIR location and creates the server.

© 2017 IBM Corporation

Start PROC ...

Creating a Liberty z/OS server involves opening a UNIX shell environment -- which can be a Telnet session, an SSH session, or even an OMVS screen – setting two environment variables, and then issuing the 'server create' command.

The two variables are:

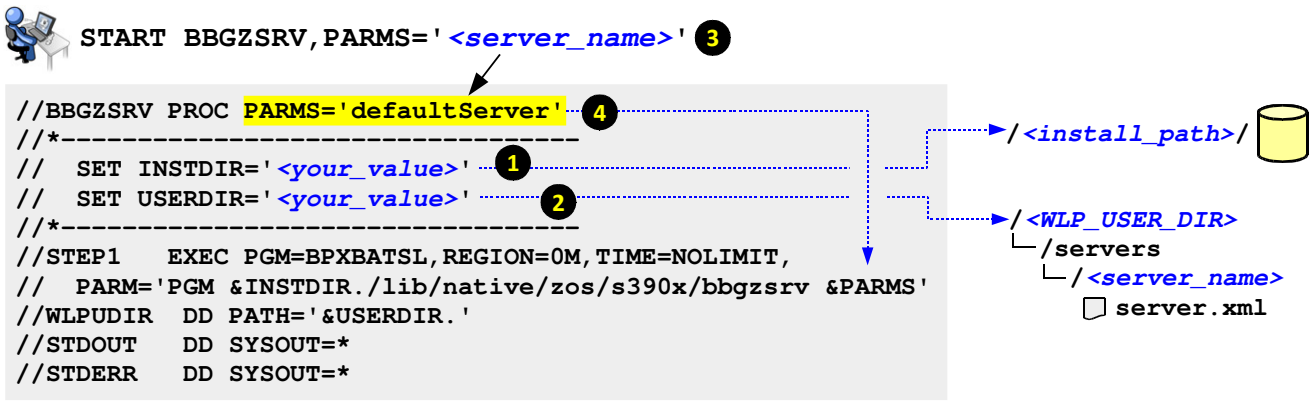
- **JAVA_HOME** – Liberty does not come with its own Java; you must have a valid 64-bit Java installed somewhere on the system that can be pointed to with this variable. Then it's just a matter of exporting JAVA_HOME to the environment so that when the 'server create' is run it can access and use Java.
- **WLP_USER_DIR** – this is where the server will be created. It can be anywhere you wish. The 'server' shell script will read the value of WLP_USER_DIR in the shell environment and attempt to create the server directories there. Provided the ID under which you execute the 'server' command has write permissions to that location, your server will be created.

The 'server' shell script resides in the /bin directory of the Liberty install. So you change directories to that location and issue the command 'server create <server_name>'. The server name you provide ends up being a directory under the WLP_USER_DIR location. It then creates some other directories and copies in a default server.xml file. Your server is created.

© 2017, IBM Corporation

IBM WebSphere Liberty Batch z/OS

The Supplied Sample BBGZSRV JCL Start Procedure … and How it Works



1. **INSTDIR=**
   This points to the location where Liberty z/OS is installed.
2. **USERDIR=**
   This points to the WLP_USER_DIR under which the server you wish to start resides.
3. **START command PARMS=**
   By default the START command includes a PARMS= that names the server.
4. **PARMS= resolution on EXEC statement**
   The PARMS= on the START command overrides the PARMS= on the PROC statement, and that resolves the &PARMS variable on the EXEC statement

© 2017 IBM Corporation

Once the server is created we can start it either as a UNIX process, or as a z/OS started task. Starting it as a UNIX process involves using the same 'server' shell script you used to create the server, but with a different verb: 'server start' rather than 'server create". Our interest in this workshop is starting it as a z/OS started task. To do that you use the supplied sample JCL, which is found in the install file system at this location: /*<install_mount_point>*/templates/zos/procs/. Copy the procs to your system proclib and customize.

The sample JCL looks like what's shown in this chart (we've cut out some comment lines). Follow the numbered circles:

1. The INSTDIR= variable points to the installation location for the level of Liberty z/OS you wish to use. This value then resolves down to the &INSTDIR. substitution variable on the EXEC statement to complete the path to the 'bbgzsrv' module.
2. The USERDIR= variable points to the WLP_USER_DIR under which the server you wish to start resides.
3. By default (we're going to show you some customization options on the next chart) the START command includes a PARMS= in which you name the server you wish to start.
4. The PARMS= you supply on the START command overrides what's coded on the PROC statement, and that then resolves down to the &PARMS substitution variable on the EXEC statement.

It's not that complex a JCL start proc. And it's open to customization if you wish, which we explore on the next chart.

© 2017, IBM Corporation

Some Customized Variations on the Sample JCL Start Procedure

Hard-code the server name on the PROC statement

```
//JBSRV01 PROC PARMS='JBSRV01'
//*-------------------------------------------------
 :
 :
//STEP1   EXEC PGM=BPXBATSL,REGION=0M,TIME=NOLIMIT,
//  PARM='PGM &INSTDIR./lib/native/zos/s390x/bbgzsrv &PARMS'
//WLPUDIR  DD PATH='&USERDIR.'
```

Use this when you want each server to have its own JCL proc

START *<proc>*

Pass in Liberty z/OS version value as a parameter

START *<proc>*,**VERSION='V16004'**

```
//JBSRV01 PROC VERSION='',PARMS='JBSRV01'
//*-------------------------------------------------
//  SET INSTDIR='/zLiberty'
 :
 :
//STEP1   EXEC PGM=BPXBATSL,REGION=0M,TIME=NOLIMIT,
//  PARM='PGM &INSTDIR./&VERSION./lib/native/zos/s390x/bbgzsrv &PARMS'
//WLPUDIR  DD PATH='&USERDIR.'
```

Append Version

/zLiberty/V16002
/zLiberty/V16003
/zLiberty/V16004

This would work well for test environments where you're going between versions frequently

Java ...

© 2017 IBM Corporation

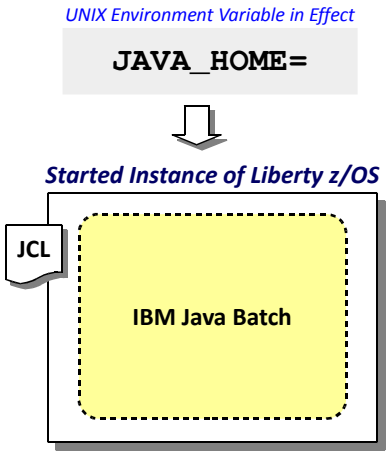
Here we illustrate two customizations to the default JCL start procedure. As mentioned on the previous chart, this JCL is not that complex, and you are free to customize it to fit your specific needs.

- **Hard-code server name on PROC statement** – the objective here is to simplify the START command to just START <proc>. This implies each server has its own JCL start procedure (which potentially impacts the SAF STARTED profiles used to assign the ID to the started task, but we'll defer that conversation to Unit 6 where we discuss security issues). If each server has its own dedicated JCL start procedure, then you may wish to consider renaming the JCL to equal the server name (which is why, as a general "good practice" we recommend server names be 8 characters or less and upper-case so you can name z/OS artifacts like JCL start procs equal to the server name).
- **Pass in other parameters** –for example, passing in the Liberty z/OS version level to use when starting the server. This implies you have multiple levels of Liberty z/OS installed, and the mount point name for each lends itself to this practice, such as we're illustrating here. In this example we're adding VERSION='' to the PROC statement, and we append /&VERSION. to the EXEC statement. Then when the START is issued with VERSION='V16004' for example, that value gets passed and it resolves down to complete the path to the designated version number you wish to use.

These are just examples. You don't have to do either of these. But it shows how the JCL start procedure is a relatively simple one and can be customized fairly easily.

© 2017, IBM Corporation

The Level of Java the Liberty Started Task Will Use



There are several ways to inject JAVA_HOME into the environment. A relatively simple way is to have the server.env file present and have it specify JAVA_HOME=

```
/<WLP_USER_DIR>
└─/servers
  └─/<server_name>
     server.xml
     server.env  ◄── ****** ********************************
                     000001 JAVA_HOME=/shared/java/J8.0_64
                     ****** ********************************
```

When the server is started the environment variable is read and the Java at that location is used.

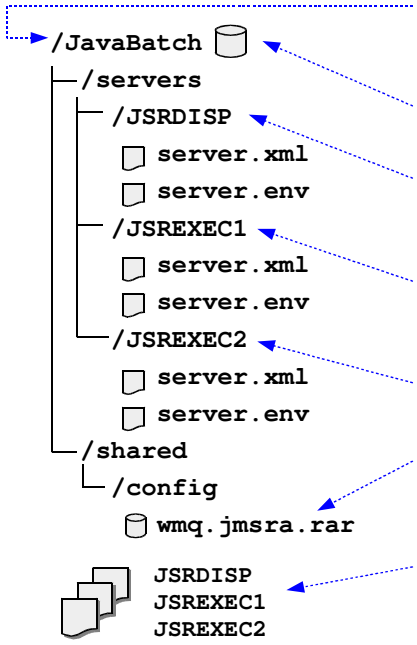
© 2017 IBM Corporation

What about the Java location that the server will use? We do that by making the JAVA_HOME variable available to the environment in which the server starts. There are several ways to accomplish this, but one of the easiest to understand is simply coding a server.env file alongside your server.xml file, and populating that file with JAVA_HOME= and pointing to the 64-bit Java you wish to use.

© 2017, IBM Corporation

IBM WebSphere Liberty Batch z/OS

What We Will Construct for the Lab Environment



The WLP_USER_DIR

For this workshop the value `/JavaBatch` is arbitrary. You may choose another more meaningful value for your location.

File system allocated and mounted at WLP_USER DIR

Not strictly required, but a good thing to do.

The "Dispatcher" server

Our aim is to create a "multi-JVM" environment. Initially we'll configure this server to be used as a single-JVM Java Batch server, then later we'll switch it so it acts as the Dispatcher.

The first of two "Executor" servers

We are going to create two Executor servers so we can illustrate the "message selector" on the JMS activation specification. This is the first of two Executor servers.

The second of two "Executor" servers

This is the second Executor server.

Sharing the MQ RAR file from /shared/config

We could have placed the MQ RAR file each server's root directory, but we thought showing how you can share configuration elements under this shared location would be interesting.

Each server will have its own JCL start procedure

That was not required; we could have shared one JCL proc among the three. But having a separate JCL for each makes the START command very simple.

server.xml ...

© 2017 IBM Corporation

Let's step back and think about the server environment we'll use for this workshop labs. The chart shows the server structure we intend to have you build as part of the hands-on lab.

Note: the names we're using here for WLP_USER_DIR and the server names are very generic. They don't represent a very well-formed naming convention. For this workshop it's fine because we're trying to get across some key concepts, and using simple-to-remember names helps. But in your setup back home you will want to think through thenaming convention a bit more carefully.

We're going to have you build three servers even though two of them will go mostly unused until we get the "multi-JVM" lab after Unit 5. We thought if you were in creating one server it would be easy enough for you to create two more. Let's walk through the chart from top to bottom:

- The WLP_USER_DIR location will be /JavaBatch, and will have a file system allocated and mounted at that location.
- The "Dispatcher" server will be called JSRDISP in upper-case. For this lab, and the Unit 3 and Unit 4 labs this server will be the server we'll use. It's not until the Unit 5 lab that it really becomes a "multi-JVM dispatcher," but that's okay ... it's just a name.
- We'll create two executor servers – JSREXEC1 and JSREXEC2 – that will go unused until the Unit 5 lab. We're creating two executor servers because we want to show you some things related to MDB message selectors, and to do that we need at least two servers to best illustrate job submission messages on the dispatching queue being selectively picked up.
- To use MQ for the batch events (Unit 4) and "multi-JVM" configuration (Unit 5), we need to make the MQ resource adapter available to each server. We're going to show you how you can make use of the /shared directory to share common things between servers under the same WLP_USER_DIR location.
- Each server is going to have its own JCL start procedure. We do this because it simplifies the START command, and it avoids the problem of case mis-matches on PARMS= that is supply on the START command.

This topology is more advanced than a single server, but less sophisticated than what's possible with Liberty. This will give you a very good sense for server creation and starting servers when multiple servers exist under a single WLP_USER_DIR location.

© 2017, IBM Corporation

The Primary Configuration File for a Server – server.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

    <featureManager>
        <feature>servlet-3.1</feature>
        <feature>batch-1.0</feature>
        <feature>batchManagement-1.0</feature>
    </featureManager>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="25080"
                  httpsPort="25443" />

</server>
```

"Features"

This is what tells the server what functions to load. This is what makes Liberty "composable." For Java Batch, the two key features are shown.

Other Configuration Elements

This where we'll place a great deal of other XML to configure things such as JDBC for DB2, and JMS for access to MQ.

HTTP ports

Liberty Java Batch makes use of REST, which is based on HTTP, which means we need to open ports for that. This illustrates how that's defined.

On z/OS that file is "tagged ASCII," which means system editors such as OEDIT* can read and edit even though the file is in ASCII. It autoconverts because it understands the tagging.

* When environment variable _BPXK_AUTOCVT=ON is set

© 2017 IBM Corporation

The heart of a server's configuration is the server.xml file. This is the primary configuration file for a server. (Other files exist: we saw the server.env for environment variables earlier, and later we'll see the bootstraps. properties file. But the server.xml is the primary configuration file.)

What does it look like? The picture above shows you an example: at the top is a "feature list" ... this is where you "compose" the Liberty server to load the features you want, based on what you want to do with the server. For this workshop the key features are batch-1.0 (enables the JSR-352 support) and batchManagement-1.0 (enables the operational extensions IBM has produced). You'll see more features as the workshop goes on, but those are the two key features of interest for Java Batch.

At the bottom is the httpEndpoint definitions, which is where we define the HTTP and HTTPS ports for the server.

In the middle is where we're going to put all sorts of XML to do various things – define the Java Batch persistence, provide the JDBC definitions to get to DB2 z/OS, some basic security settings we'll use until we get to Unit 6 (when we'll remove them and add others to use SAF for security), and in Unit 5 we'll add various JMS definitions to use MQ.

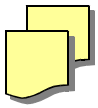
The best way to learn about the elements to use in the server.xml is to see examples, then study the specific elements references in the Knowledge Center, which you can Google fairly easily. The WP102544 Techdoc at ibm.com/support/techdocs has a "Samples" document that illustrates a three-server environment very much like what we're doing in this workshop.

The server.xml file for Liberty z/OS is "tagged ASCII," which means it's an ASCII file, but an extended attribute on the file tells the z/OS system that the file can be "auto-converted" in editors so you can edit using system editors such as OEDIT.

© 2017, IBM Corporation

In This Workshop We'll Update XML by Copying in Pre-Built Versions

XML we built ahead of time and stored at a defined location



```
/JavaBatch
  /servers
    /JSRDISP
      server.xml
      server.env
    /JSREXEC1
      server.xml
      server.env
    /JSREXEC2
      server.xml
      server.env
  /shared
    /config
      wmq.jmsra.rar
```

We do this for several reasons:

- Some configuration updates imply a lot of XML
- Nobody likes typing a lot of XML
- Typos in XML can lead to lost time debugging syntax errors
- Even copy/paste can be an issue when XML is very long

By all means, please do look at the XML we're having you copy in. We'll explain in lecture what the XML is and what it's doing. We just want to avoid the tedious exercise of typing it all in.

© 2017 IBM Corporation

Key pieces ...

Let's talk about editing the server.xml file for this workshop ...

We're going to supply pre-built server.xml files you simply copy from one location over to your server location. That will replace the file and the Liberty server will dynamically update with the new information.

Why are we doing this?

Because there's a lot of XML to add for things like JDBC and JMS configurations. In other workshops we tried providing a copy-and-paste file where you could insert rows in server.xml and paste, but we found that sometimes lines would get overlaid if not enough lines were inserted, and then XML was lost and the server had problems.

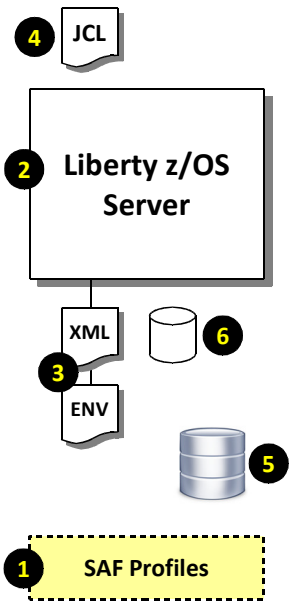
The objective of this workshop is not to make you typists, nor is it to have you inserting lines and scrolling up and down pasting in large chunks of XML. We want you to get to success without having to fight through problems that result from some little error caused by a typo, or an overlaid line, or whatever. So we'll supply the full XML files and you'll issue a UNIX 'cp' command to copy the file from our pre-built location to your server location.

But by all means look the files! We absolutely want you to understand what each copy is bringing in. We'll help with that by illustrating in the hands-on lab guide what new XML is coming in with each cp command.

© 2017, IBM Corporation

IBM WebSphere Liberty Batch z/OS

The Key Pieces to Setting Up Your First Liberty Java Batch Server



1. SAF Profiles

We need to setup up a few things ahead of time: notably, the ID and group the servers will operate under, and the STARTED profiles so the JCL start procs will assign that ID to the started tasks.

2. Create Server

With the ID created we can go into a UNIX shell under that ID and create the server.

3. Configuration Files

The act of creating the server will copy in a default template. As mentioned, we created configuration files ahead of time and for the upcoming lab you'll copy them in.

4. JCL Start Procs

Sample JCL is supplied with the Liberty install. It's a simple matter to copy them from the file system to your proclib and customize. We did that ahead of time.

5. Java Batch Job Repository

We could start with in-memory, but we'd rather go with the JobRepository in DB2 z/OS.

We will have you generate the DDL using the genDDL shell script. But then we'll have you submit a JCL job we created that uses that same DDL to batch-create the tables in DB2. In the "real world" you'd review the generated DDL with your DB Admin and create according to your DB2 procedures.

6. Sample Java Batch Application

For the initial validation we're going to use the "SleepyBatchlet" sample available out on Git. This is easy to use as an IVP because it requires no other data input or output sources.

© 2017 IBM Corporation

What's in XML ...

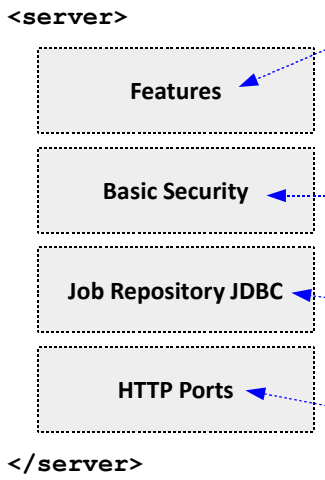
Let's step back and take a high-level view of the key components that go into creating and running your first Java Batch server, and how we'll accomplish some of those things in this workshop. Follow the numbered circles:

1. There's a few SAF profiles that must be in place initially – the server ID and group, as well as the STARTED profiles that will assign the ID when the START command is issues. We'll have a pre-built JCL job you submit to do that.
2. You'll create the server by opening a Telnet session and issuing the 'server create' command.
3. The configuration files we'll have you copy in from our pre-built directory, as we discussed on the previous chart.
4. The JCL start procs will be copied into the SYS1.PROCLIB data set ahead of time, but you'll customize them.
5. The JobRepository consists of a set of DB2 tables. We could create those ahead of time, but we wanted you to experience running the genDDL shell script to generate the DDL. You'll do that, but then we'll have you run a pre-built JCL job that has the same DDL to create the tables. That will spare you the time of getting the generated the DDL into a 80-column JCL format.
6. There's a sample Java Batch program we'll have you use that's relatively simple. We have it placed in a directory where you can simply copy it into your server's /dropins locations.

Yes, some of this is simply you submitting a job we spent time building ahead of time. If we had you run through all the steps we'd never fit this into two days. You're not missing anything important provided you understand the key things going on. The work we did ahead of time was mostly just mundane stuff to save you time.

© 2017, IBM Corporation

The server.xml in Support of the *Initial** Java Batch Server



Feature Specification

We have a few key features to make sure are in place – `batch-1.0`, `batchManagement-1.0`, and `appSecurity-2.0`. The first enables the JSR-352 batch function; the second enables the IBM operational enhancements; and the third is because the REST interface is going to impose security requirements.

Basic Security Definitions

In the "Security" unit we'll talk about how to get security definitions into SAF, but to start out we can keep things simple by coding them in the `server.xml` file.

Job Repository and JDBC Definitions

To use the DB2 JobRepository tables we have to tell the Liberty z/OS server how to get to DB2 z/OS and a few other things.

HTTP Ports

The REST interface we'll use to submit the job requires HTTP and HTTPS ports to be open.

* We will add more to this as we build out the configuration to suppor the multi-JVM topology

© 2017 IBM Corporation

This is a very abstract representation of the server.xml structure that you'll see for this workhop. What we're trying to do here is convey the idea that information in the server.xml file is organized into groups of XML, each with a particular purpose. You've already seen a little of the XML earlier: with the feature list and the HTTP ports. Those two things are represented by the top and bottom boxes in this picture.

In adddition we'll add a section for "basic security." This will satisfy the security requirement of IBM WebSphere Liberty Java Batch without having to get too complicated with security too early. Another block of XML will be associated with the JDBC definitions needed to access the DB2 tables for the JobRepository. (And later, in Unit 4 and 5, we'll add more to represent JMS for both batch events and the multi-JVM design.)

© 2017, IBM Corporation

IBM WebSphere Liberty Batch z/OS

Batch Persistence and JDBC Definition

```
<batchPersistence jobStoreRef="BatchDatabaseStore" />

<databaseStore id="BatchDatabaseStore"
  createTables="false"
  dataSourceRef="batchDB" schema="JBATCH" tablePrefix="" />

<jdbcDriver id="DB2T4" libraryRef="DB2T4LibRef" />

<library id="DB2T4LibRef">
  <fileset dir="/shared/db21010/jdbc/classes/"
  includes="db2jcc4.jar db2jcc_license_cisuz.jar sqlj4.zip" />
</library>

<authData id="batchAlias" user="xxxxx" password="xxxxx" />

<dataSource id="batchDB"
  containerAuthDataRef="batchAlias"
  type="javax.sql.XADataSource"
  jdbcDriverRef="DB2T4">
 <properties.db2.jcc
   serverName="wg31.washington.ibm.com"
   portNumber="9446"
   databaseName="WG31DB2"
   driverType="4" />
</dataSource>
```

1. **batchPersistence**
   This turns on batch persistence. Absent this it would be in-memory JobRepository
2. **dataBaseStore**
   This points to the dataSource to use, and it also specifies the DB2 z/OS table schema for the Java Batch tables.
3. **library**
   Points to where the DB2 JDBC drivers are
4. **authData**
   We're showing JDBC T4, and that requires an authentication alias.
5. **dataSource**
   Provides the specifics of the connection to DB2

This is why we provide pre-built XML to copy in, rather than having you type this by hand. ☺

© 2017 IBM Corporation

Here we start showing some of the detail of the XML ... specifically, the JDBC definitions to access DB2 where the JobRepository tables are maintained. The numbered circles correspond to the notes here:

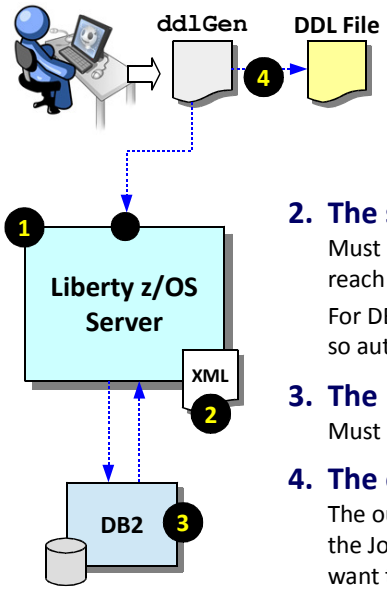
1. The <batchPersistence> element provides a pointer ("jobStoreRef=") to more XML that begins to define the batch persistence information.
2. The <databaseStore> element is pointed to by <batchPersistence>, and it provides more details: (a) it will not attempt to auto-create the tables in the database engine; (b) the data source is pointed to with the ID "batchDB"; (c) the tables will use a schema value of JBATCH; and (d) no table prefix is used for the tables.
3. The <library> element is used to provide information about where the JDBC drivers are located.
4. The <authData> element is used to specify the authentication alias for the JDBC Type 4 connection we're using to DB2.

   Note: JDBC Type 2 (cross-memory z/OS) is possible, but we're choosing not to do that in this workshop because it requires RRS, which is a z/OS authorized service and would mean the Liberty Angel process would be required. Later we're going to require the Angel for the batchManagerZos command line client, which uses WOLA, which is a z/OS authorized service as well. But for now we're "keeping it simple" with JDBC T4, which does not require the Angel process.
5. The <dataSource> element defines the final set of details needed to access DB2, including the host and port where DB2 is listening.

That's a lot of detail, but it's necessary detail to understand how and where to get to DB2 to access the JobRepository tables.

© 2017, IBM Corporation

Generating the JobRepository DDL using the genDDL Utility



The ddlGen utility will create the DDL for the table definitions based on the relational system defined to the running server:

1. **The Liberty server**
   The server must be up and running for ddlGen to work
2. **The server.xml file**
   Must be configured with valid `<batchPersistence>` and JDBC definitions so it knows how to reach the DB2 system. Also, the `localConnector-1.0` and `batch-1.0` features must be defined.
   For DB2 on z/OS we advise setting `createTables="false"` on the `<dataBaseStore>` element so automatic generation of tables in DB2 is *not* attempted.
3. **The DB2 system**
   Must be started and running with the Liberty server able to connect to it.
4. **The output file**
   The output from the ddlGen utility is a file with the DDL statements to create the tables used for the JobRepository. This DDL does not have database and STOGROUP definitions, so you would want to review this DDL with your DB2 Admin so it can be customized to your local DB2 policies.

© 2017 IBM Corporation

WebSphere Liberty Java Batch comes with a utility called "ddlGen" which will generate a set of Data Definition Language (DDL) statements to create the tables needed for the JobRepository. The general process is this: you generate the DDL, you review with your DB Admin, and then you implement the tables.

Note: by default this utility will try to auto-create the tables as well. That works great for something like Derby, and maybe DB2 on the Linux/Unix/Windows platforms, but is probably *not* something you'd do with DB2 z/OS. There's a way to stop it from trying to auto-create the tables, and that's with the createTables="false" tag on the server.xml <dataBaseStore> element.

This utility has a few requirements for it to work:

- Your Liberty server must be up and running, and it must have valid JDBC definitions in the server.xml to allow it to connect to the database system you intend to use. The utility will connect to the running server and have the server connect to the database system.
- The server.xml must also have the localconnector-1.0 feature defined. This is what provides JMX support. The ddlGen utility will use JMX and connect to the running server.
- The database system – DB2 z/OS in our workshop – must be started. The Liberty server will connect to the DB2 instance using the JDBC definitions in the server.xml.

With all that in place the ddlGen utility will produce an output file that contains the DDL for the tables. This file has very long lines, one for each table creation or index creation. It is definitely not formatted for FB 80, so you'd need to do that to get it into JCL for table creation. Also, the ddlGen utility does not create any STOGROUP, TABLESPACE, or DATABASE statements. It just creates TABLE and INDEX statements. So for DB2 z/OS you should definitely have your DB2 administrator take a look at the statements and assist with the creation of the SQL to properly format the tables for your system.

© 2017, IBM Corporation

The Liberty Java Batch Features

There are two levels we can turn on: basic JSR-352 support, and IBM operational extensions

```
<featureManager>
    <feature>servlet-3.1</feature>
    <feature>batch-1.0</feature>
    <feature>batchManagement-1.0</feature>
</featureManager>
```

This enables the JSR-352 Java Batch support. If you coded *just* this, you could run JSR-352 batch jobs, but you would not have the IBM extensions such as the REST interface, the batchManager command line client, the batchManager Zos command line client, or the multi-JVM support.

This enables the IBM operational extensions to the JSR-352 support found in Java EE 7. If you code just this, then batch-1.0 would be enabled automatically. Coding both (as shown) does no harm. For this workshop, we intend to illustrate the IBM operational extensions.

© 2017 IBM Corporation

There are two features that are key to the use of the WebSphere Liberty Java Batch function – batch-1.0 and batchManagement-1.0.

batch-1.0 is what enables the JSR-352 support. But coding *only* this would mean you could run JSR-352 applications, but you would not have the operational enhancement functions IBM has provided. Those are enabled with the batchManagement-1.0 feature.

As a general rule, you will code both of these. Yes, you *could* use just batch-1.0, but there's little reason not to use the operational enhancements that come with batchManagement-1.0. So ... code them both.

© 2017, IBM Corporation

Basic Security

The final unit of this workshop focuses on security … specifically, the use of SAF to "harden" security constructs. *Initially* we can satisfy security requirements with "basic" security:

```
<keyStore id="defaultKeyStore"
  password="Liberty"/>

<basicRegistry id="basic1" realm="jbatch">
  <user name="Fred" password="fredpwd" />
</basicRegistry>

<authorization-roles id="com.ibm.ws.batch">
  <security-role name="batchAdmin">
    <user name="Fred" />
  </security-role>
</authorization-roles>
```

Liberty-generate key/trust store

With this line, Liberty will generate a file keystore with a self-signed certificate for SSL. You'd never use this for production, but it's "good enough" to start with.

User Registry

If we're required to log in, we'll need a registry of user identities. Normally this is LDAP or SAF, but to start we'll code it here in the server.xml file. "Fred" is our user, and his password is "fredpwd" (case sensitive).

Application Role

The `batchManagement-1.0` function REST interface requires the authenticated user to be granted access to one of the defined roles. Here we're granting Fred access to the "batchAdmin" role, which allows Fred administrator rights.

This is simply a way to achieve the minimum security requirements quickly and easily for initial validation and usage. See the security unit for more on SAF security implementation.

© 2017 IBM Corporation

Try as we might, we can't avoid the subject of security. ☺

When batchManagement-1.0 is enabled, the REST interface (and the batchManager command line client which uses the REST interface) imposes a set of security requirements that must be satisfied. Those requirements are:

- **Encryption** – the network connection with the REST interface is marked protected, which means it will be redirected to the HTTPS port of the server. That will trigger the establishment of an SSL connection, which means a server digital certificate is required. Normally that is done by generating a certificate and having a well-known "Certificate Authority" sign it. But this early in the workshop we're going to take a shortcut and use a built-in Liberty function that generates a certificate and places it in a file. That's enough to satisfy the SSL requirement. (We'll look at SAF-based SSL and keyrings in Unit 6.)
- **Authentication** – before entry is permitted the user presenting themselves need to be authenticated. That means Liberty must compare the ID and password against a registry of users and their passwords. Normally that's done with LDAP or SAF, but again, we're taking a shortcut early in this workhop and we'll code the registry right in the server.xml. (More on SAF registry in Unit 6.)
- **Authorization** – a user can be authenticated, but they may not be permitted to use the Java Batch function. Authority is checked against defined "application roles," which determine how much authority a user has within the function. Normally that would be done with either LDAP or SAF, but here we'll use roles defined in the server.xml.

As the chart indicates, this is all just a way to simplify the security requirement so we can focus on the other functions of WebSphere Liberty Java Batch without having to jump through too many security hoops too early. But you'd never operate your production environment with what we show you in the chart. It's great for development and ad hoc testing, but for anything more than that better security is needed. We have the entire Unit 6 dedicated to this topic.

© 2017, IBM Corporation

IBM WebSphere Liberty Batch z/OS

The SleepyBatchlet Sample and Application Deployment

`https://github.com/WASdev/sample.batch.sleepybatchlet`

SleepyBatchletSample-1.0.war

For this workshop we did the download. The WAR file is on the z/OS system ready to copy into your server's /dropins directory

The sample, when submitted, loops for a specified number of seconds (default 15) and then ends.

It has no data input or output requirements.

It's an ideal IVP because it has no dependencies other than an operational Java Batch runtime

```
/JavaBatch
└─/servers
  └─/JBSRV01
    ├─/apps
    ├─/dropins
    │  SleepyBatchletSample-1.0.war
    └─/logs
      server.xml
```

With Liberty you can deploy an application either by using dynamic file monitoring and the /dropins directory, or by statically defining the application in server.xml

For this workshop we're going to use the /dropins directory

© 2017 IBM Corporation

Job submission ...

To validate the infrastructure we're going to have you use a sample Java Batch application called "SleepyBatchlet," which is available for download on Git at the URL shown on the chart.

The application is good for initial verification because it requires no other data input or output definitions. It's a relatively simple JSR-352 application that is structured as a one-step job, and the step is a "batchlet." The batchlet simply loops and "sleeps" for a specified number of seconds, then ends. The number of seconds it "sleeps" default to 15, and that value can be passed in as a job parameter at time of job submission (we'll see how that works in Unit 3).

The sample application is packaged as a WAR file (Web ARchive), and can be deployed by simply placing the file in the /dropins directory of the server. The server will detect the new file and dynamically pick it up and load the application.

Java Batch applications may be "started" in the Liberty server, but that does not mean the batch function is processing. That's done by submitting the job through the JobOperator interface. For IBM WebSphere Liberty Java Batch, IBM has interfaced the JobOperator with a REST interface, and has provided two command line interfaces: batchManager and batchManagerZos. For the lab coming up after this unit we'll use the batchManager client because it's relatively easy to use initially. (The other command line client, batchManagerZos, requires WOLA, and that means the Angel needs to be in place as well as some SAF SERVER profiles. We'll do that up in Unit 4. For now we'll stay simple and use batchManager.)

© 2017, IBM Corporation

IBM WebSphere Liberty Batch z/OS

Submitting the SleepyBatchlet Job

UNIX shell ... Telnet, SSH, OMVS

```
> cd /usr/lpp/zWAS/V16004/bin
> export JAVA_HOME=/shared/java/J8.0_64
> (see command below)
```

One long command line ...

```
./batchManager submit --batchManager=localhost:25443 (1)
   --user=Fred --password=fredpwd (2)
   --applicationName=SleepyBatchletSample-1.0 (3)
   --jobXMLName=sleepy-batchlet.xml (4)
   --trustSslCertificates (5) --wait (6)
```

1. **Point to server host and HTTPS port**
   Here we're invoking on the same LPAR, so we use "localhost".
2. **Provide authentication information**
   This matches what we defined in the "basic" security in server.xml
3. **Name the application to submit**
   The server may have many different applications deployed, so this names the batch job to submit
4. **Specify the JSL XML file**
   This is part of the application WAR file. Naming this tells the batch container what JSL to use.
5. **Trust the SSL certificate**
   This tells batchManager to automatically trust the SSL certificate without performing any verification. When using the "basic" security this is necessary because the "basic" self-signed certificate is not trusted, and an SSL handshake error would occur.
6. **Wait for job completion to return to prompt**
   This tells batchManager to hold return to prompt until the job completes.

© 2017 IBM Corporation

Techdoc ...

This chart illustrates how the SleepyBatchlet job is submitted using the batchManager command line client. The batchManager utility is located in the /bin directory of the install path. You invoke it from a shell environment (Telnet, SSH, OMVS), or with JCL and BPXBATCH (which creates a shell as well, but one under the control of the JCL). You have to have JAVA_HOME exported to the shell environment for this to work. Then you issue the command.

The "verb" is 'submit' ... there are other verbs on the batchManager command line interface, and we'll explore those up in Unit 4. For now, we'll focus on 'submit.' The submit verb is followed by a series of parameters. Follow the numbered circles:

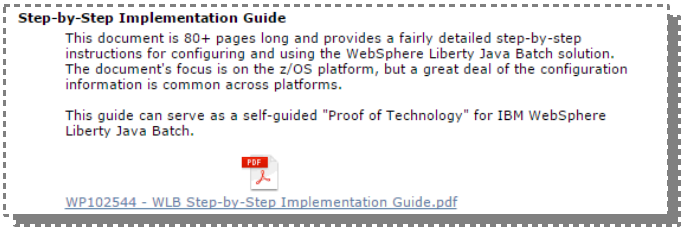
1. The --batchManager= parameter specifies the host and port of the Liberty server that we'll communicate with to run the Java batch job. Here we're showing "localhost" because we'll be invoking on the same LPAR as the server. But you could as easily specify a real DNS host name. Which tells us something about batchManager – you can run this utility anywhere; it does not need to be run on the same LPAR as the Liberty z/OS server. The other required bit of information is the port number. This is the HTTPS (secure) port.
2. The userid and password of the ID that will attempt access. Do you recall our chart with talk of authentication and user registries? This is why we need it ... to submit a job we have to authenticate. That means the ID we're passing in (Fred in this case) must be in the registry. If you recall our basic registry example, Fred is coded in the registry section of server.xml.
3. We name the Java Batch application so the Liberty server knows which one to invoke. You may have many Java Batch applications deployed in a server, so we have to indicate which one we're interested in.
4. The --jobXMLName refers to the "Job Specification Language" (JSL) file to read in and use for the job. The SleepyBatchlet sample has the JSL file packaged with the application. So we only need refer to the JSL name. There is an option to use a JSL file that's outside the application package. That's called "inline JSL," and we cover that in Unit 4.
5. The --trustSslCertificates tells the batchManager client to overlook issues with server certificates that are not valid for some reason. For this lab we're using the "basic" security with the self-generated certificate. That certificate is a good certificate, it's just not signed by a certificate authority. It is "self-signed," and SSL clients in the world are programmed to be wary of such things. This parameter allows us to work past that. (The alternative would be to create a JVM argument that provides access to that self-signed certificate so batchManager could verify what it has received matches what's in its "trust store." Using trustSslCertificates is easier. But it's important to understand you should not use this for anything but development and testing where you know the environment well and can trust the server.
6. The --wait parameter tells batchManager to wait for the job to complete. It periodically polls the server to find out the status of the job. When the job finishes, the batchManager invocation ends. That's an important piece of the "enterprise scheduler integration" puzzle, so remember this --wait thing.

When you issue that command it will build an SSL connection to the target server, submit the job, and wait for it to complete. You will get back a message indicating the job completion status.

© 2017, IBM Corporation

WP102544 Techdoc

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



A detailed step-by-step implementation guide

Links to other Techdocs related to the Java Batch topic

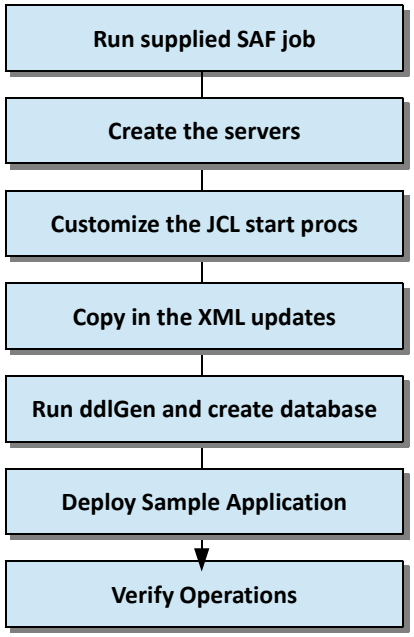
© 2017 IBM Corporation

Lab ...

We have a very detailed Techdoc that steps through implementing the WebSphere Liberty Java Batch function. So if you're concerned about the things we did ahead of time for this workshop, you can rest assured the details are available. That Techdoc also has a set of links at the bottom that point off to other Techdocs related to Java Batch.

© 2017, IBM Corporation

The Hands-On Lab



Objective: get hands-on with Liberty; setup servers for Java Batch

Lab instructions are fairly specific

Use the supplied copy-and-paste file for commands

Steady pace … don't rush

Off to lab!

© 2017 IBM Corporation

We're ready to go into the first lab. This lab will have you do the things that are outlined in the boxes down the left side of the chart.

The objective is to create the server environment and set it up for Java Batch.

The lab instructions are fairly specific – submit this, do that, etc. The lab instructions early in this workshop will be very specific, but will back off as we get deeper in the later labs. By then you'll be familiar with things and won't need all the details.

We have said we don't want this to be a typing exercise, so we're also supply a text file that contains the long commands to enter to do things. Please make use of that copy-and-paste file ... it'll save time, and it'll help you avoid problems related to typo errors.

The key to these labs is a steady pace. You don't need to rush. A good steady pace will do it.

End of Unit

© 2017, IBM Corporation