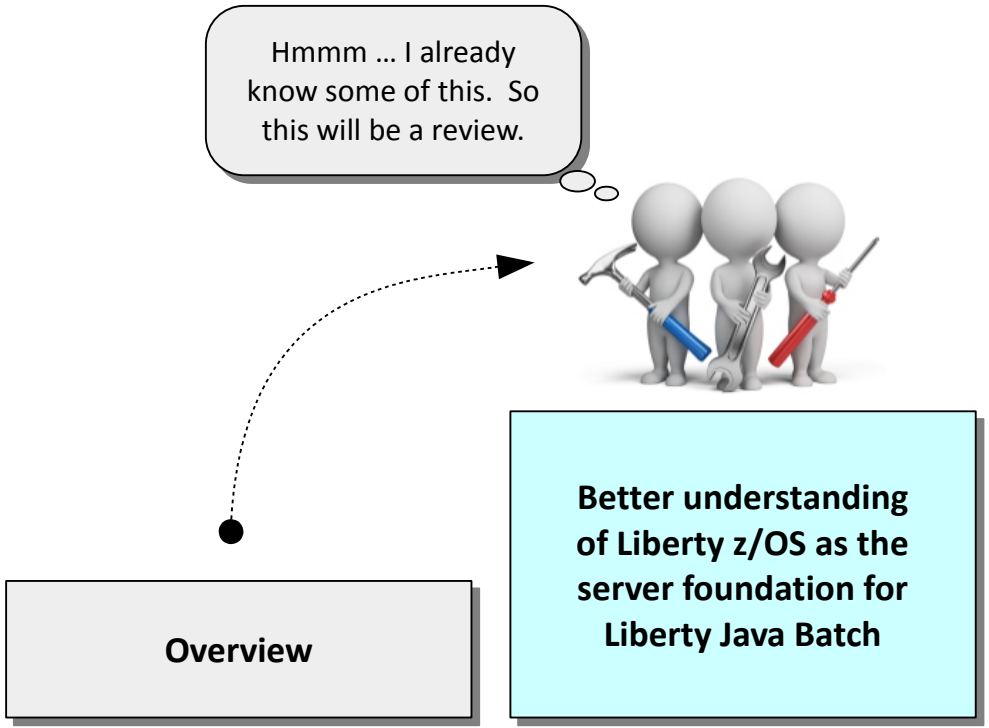
WebSphere Application Server

Unit 2

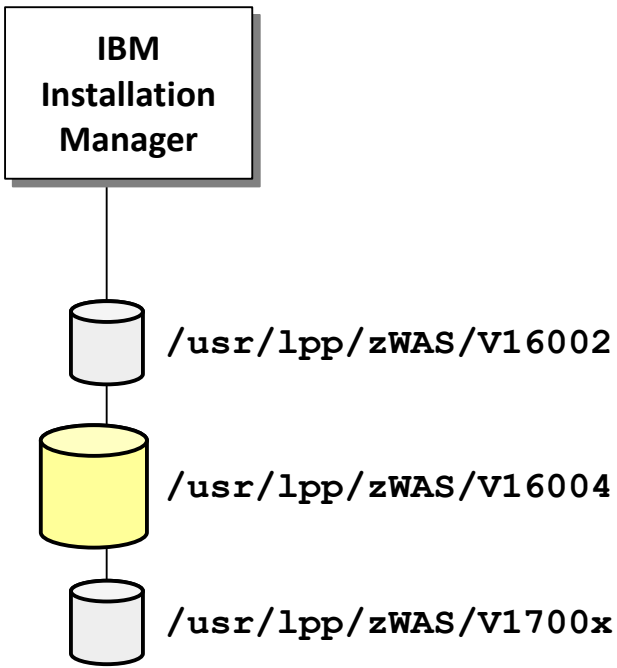
Liberty, Server Creation and Setup

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**

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.

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.

Start PROC ...

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

```
START BBGZSRV,PARMS='<server_name>'  3
```

```
//BBGZSRV PROC PARMS='defaultServer'  4
//*-----------------------------------
//  SET INSTDIR='<your_value>'  1
//  SET USERDIR='<your_value>'  2
//*-----------------------------------
//STEP1   EXEC PGM=BPXBATSL,REGION=0M,TIME=NOLIMIT,
//  PARM='PGM &INSTDIR./lib/native/zos/s390x/bbgzsrv &PARMS'
//WLPUDIR  DD PATH='&USERDIR.'
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
```

```
/<install_path>/
/<WLP_USER_DIR>
  └─/servers
    └─/<server_name>
      server.xml
```

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

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'`

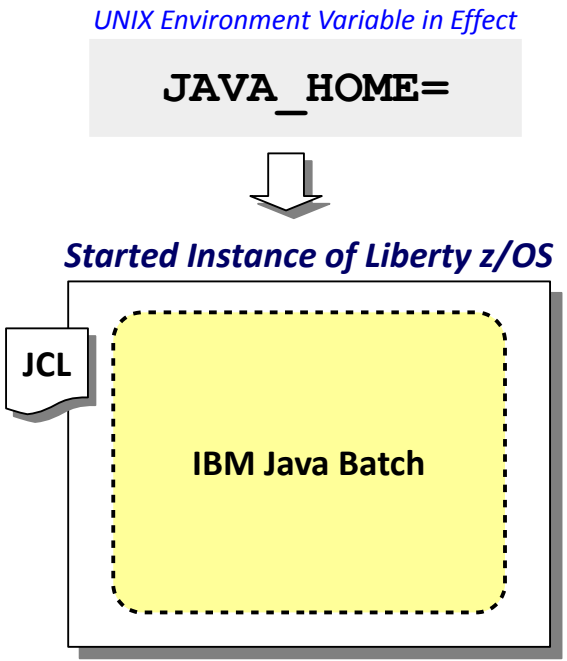
```
//JBSRV01 PROC VERSION='',PARMS='JBSRV01'
//*-------------------------------------------------
//  SET INSTDIR='/zLiberty'
 :
 :
//STEP1   EXEC PGM=BPXBATSL,REGION=0M,TIME=NOLIMIT,
//  PARM='PGM &INSTDIR./&VERSION./lib/native/zos/s390x/bbgzsrv &PARMS'
//WLPUDIR  DD PATH='&USERDIR.'
```

Append Version

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

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

The Level of Java the Liberty Started Task Will Use



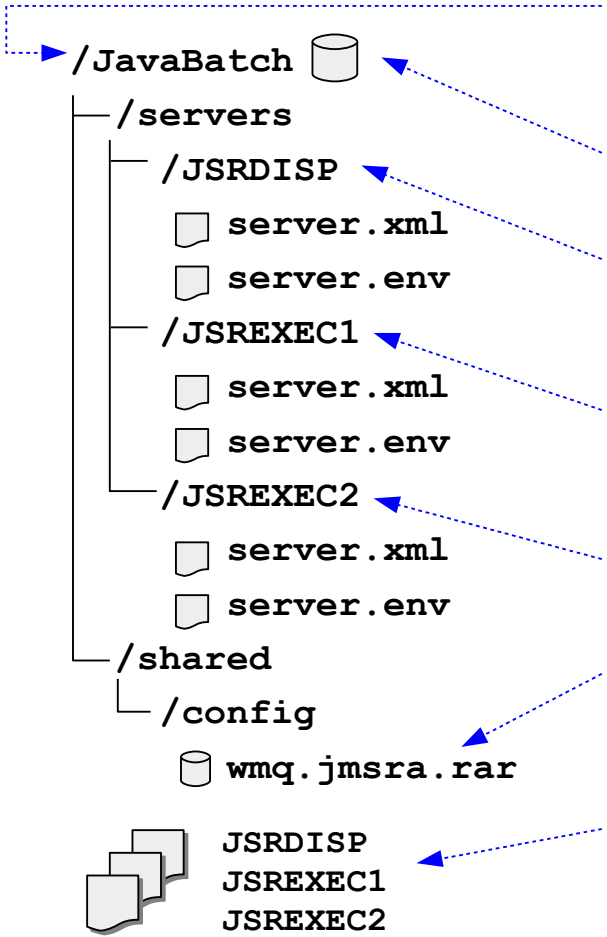
There are several ways to inject JAVA_HOME into the environment. A relatively simple way is to have the server.env file present and have it specify JAVA_HOME=

```
/<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.

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.

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

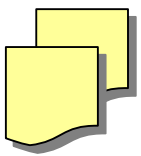
Liberty Java Batch makes use of REST, which is based on HTTP, which means we need to open ports for that. This illustrates how that's defined.

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

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
```

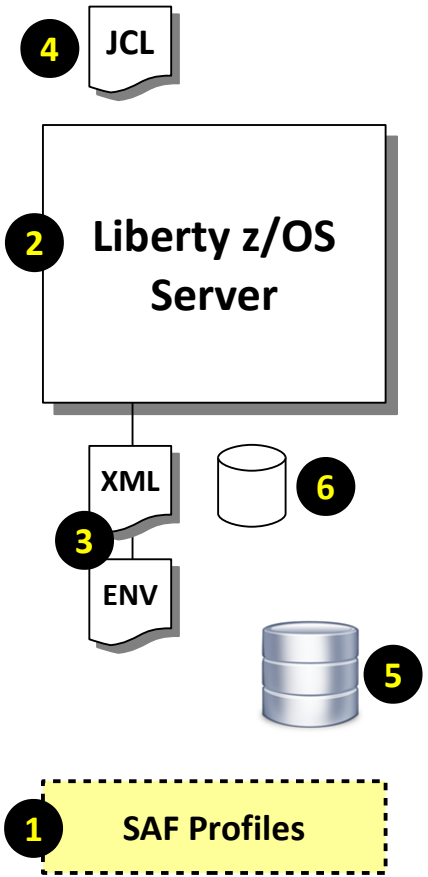
We do this for several reasons:

- 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.

Key pieces ...

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

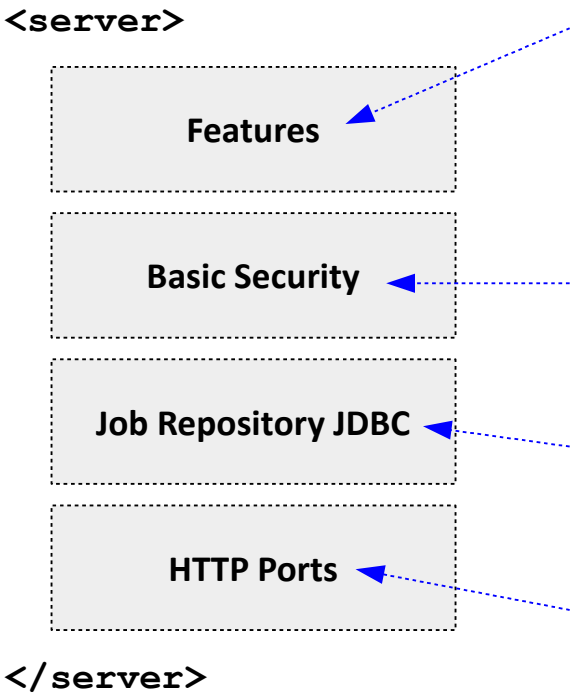
We could start with in-memory, but we'd rather go with the JobRepository in DB2 z/OS.

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.

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

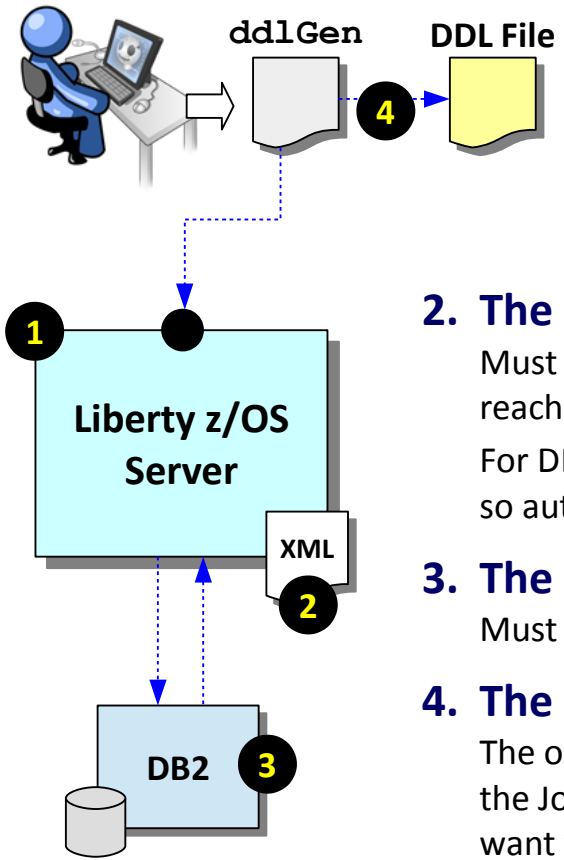
Batch Persistence and JDBC Definition

```
<batchPersistence jobStoreRef="BatchDatabaseStore" /> 1

<databaseStore id="BatchDatabaseStore"
  createTables="false"                                2
  dataSourceRef="batchDB" schema="JBATCH" tablePrefix="" />

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

<library id="DB2T4LibRef">
  <fileset dir="/shared/db21010/jdbc/classes/"        3
  includes="db2jcc4.jar db2jcc_license_cisuz.jar sqlj4.zip" />
</library>

<authData id="batchAlias" user="xxxxx" password="xxxxx" /> 4

<dataSource id="batchDB"
  containerAuthDataRef="batchAlias"                   5
  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. ☺

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.

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.

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.

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

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)
```

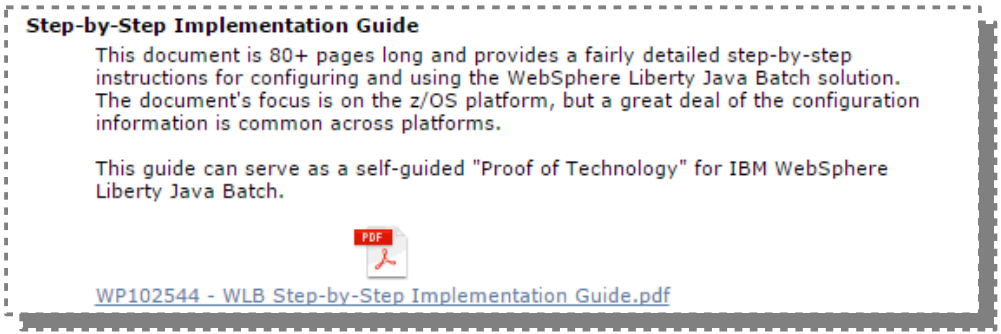
One long command line …

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

WP102544 Techdoc

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

A detailed step-by-step implementation guide

Other WebSphere Liberty Batch Techdoc Pages

Job Classification: http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102600

Batch Events: http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102603

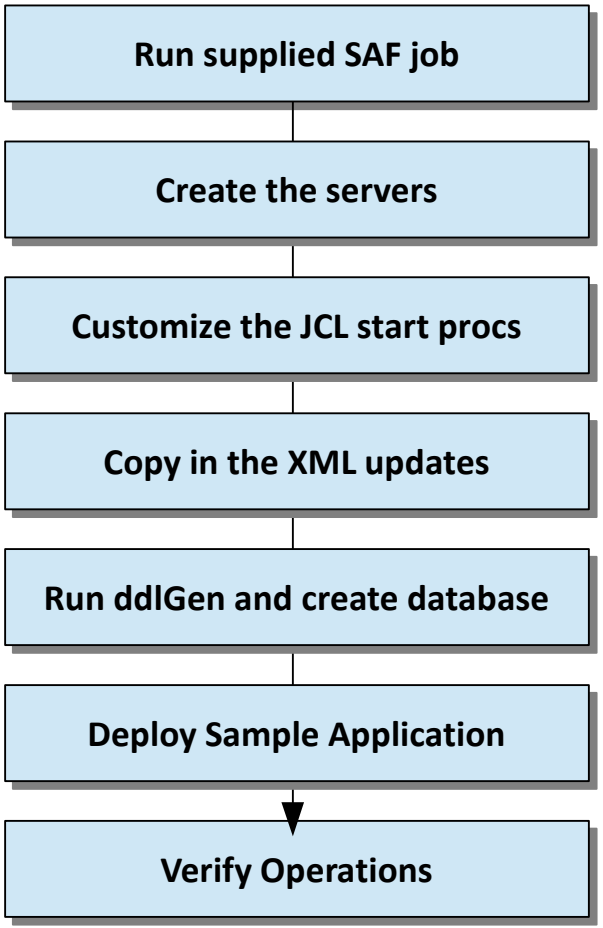
Batch SMF Record: http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102668

Java Batch in CICS:

https://developer.ibm.com/cics/2016/10/04/java-batch-in-cics-concepts/
https://developer.ibm.com/cics/2016/10/04/java-batch-in-cics-tutorial/

Links to other Techdocs related to the Java Batch topic

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!