Liberty z/OS Java Batch Sample Configuration

An illustration of a multi-JVM topology

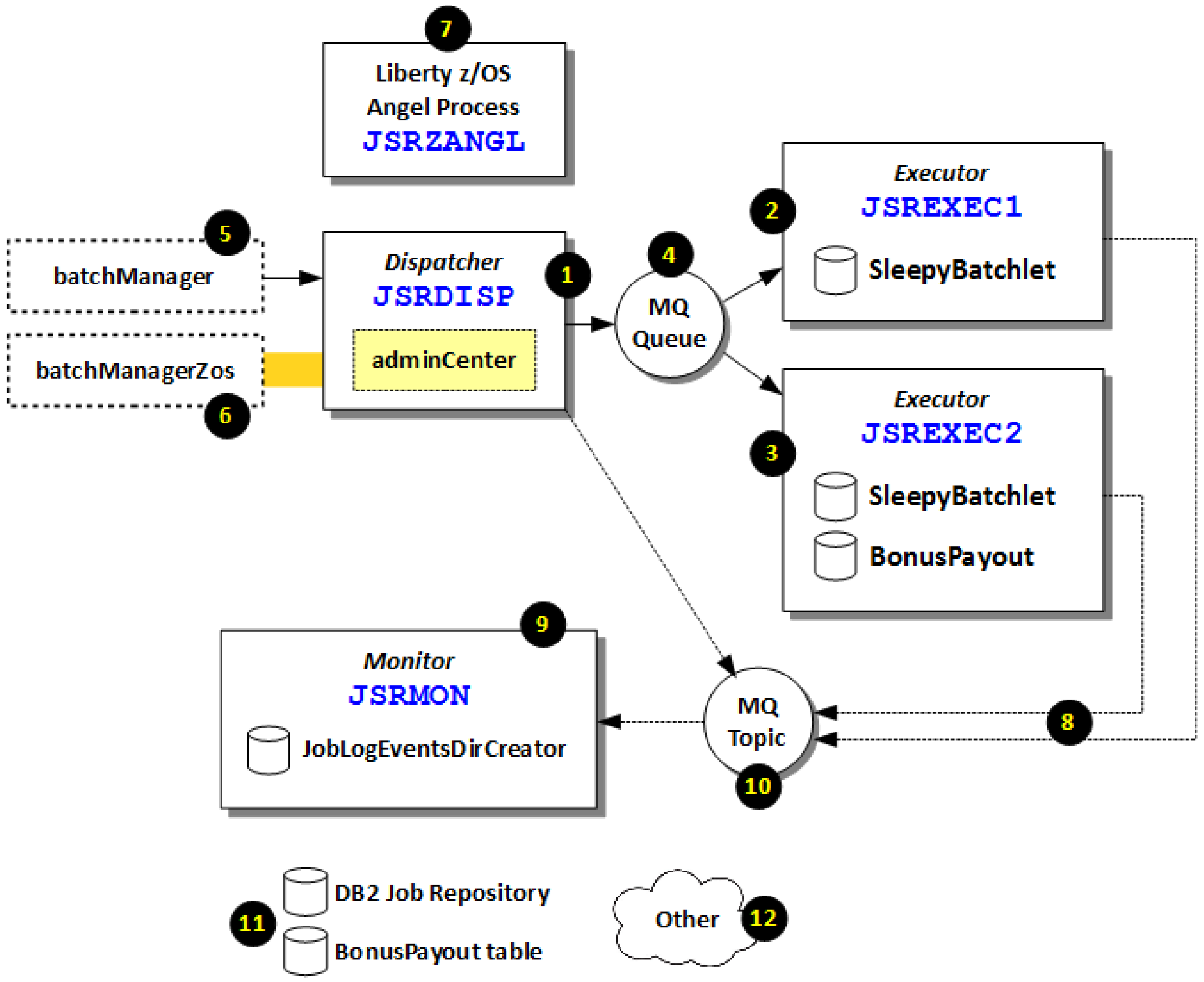
Version Date: June 6, 2017

See "Document Change History" on page 34 for a description of the changes in this version of the document

Table of Contents

Introduction

The purpose of this document is to illustrate and document the following IBM Liberty z/OS Java Batch configuration topology:



We have that topology working in a real z/OS environment, and in this document we will provide the configuration files, RACF commands, and other things involved with making this work.

The numbered circles in the picture correspond to the following notes:

1. **Dispatcher** – the topology we are illustrating separates the job submission and dispatching Java Virtual Machine (JVM) from the job execution JVM. This server (`JSRDISP`) acts as the *dispatcher* in this multi-JVM topology. We also have the Liberty `adminCenter-1.0` feature configured to illustrate the Java batch tool that provides.
2. **Executor** – this is the first of two *executor* servers. In this server we had the *SleepyBatchlet* sample application deployed. That application is a simple batchlet that loops for the specified amount of time, then ends.
3. **Executor** – this is the second of two executor servers. In this server we had both the *SleepyBatchlet* and the *BonusPayout* applications deployed[1]. BonusPayout is a "chunk" application that illustrates iterative processing with checkpoints.
4. **MQ queue** – the integration between Dispatcher and Executor is a simple queue. In this sample we are illustrating the use of IBM MQ as the queueing mechanism.

1 We did this to illustrate the use of "or" in the message selector definition on the JMS activation specification. This server will pick up and run a job submission message for either application.

5. **batchManager** – this is a command line interface client that uses REST to communicate with the Dispatcher server.
6. **batchManagerZos** – this is a command line interface client that uses WebSphere Optimized Local Adapters (WOLA) to communicate with the Dispatcher server.
7. **Angel Process** – the Angel Process is required when a Liberty z/OS server intends to use a z/OS authorized service. WOLA is a z/OS authorized service. Therefore, the Angel Process is required, along with SAF profiles to allow the Dispatcher access to WOLA[2].
8. **Batch events** – we configured the servers to emit batch events for the purpose of monitoring the status of jobs.
9. **Events monitor** – this server runs the sample JobLogEventsDirCreator application to capture the job log events emitted by the servers and write the output to a directory and file. This validates the batch events configuration is working as designed.
10. **MQ topic** – the batch events function is a publication/subscribe model, and we used IBM MQ as the mechanism for hosting the topic and allowing the monitor server to subscribe.
11. **DB2 artifacts** – underlying this configuration topology is the set of tables for the "JobRepository" function, as well as a single table for the BonusPayout application.
12. **Other artifacts** – here we collect up everything else: SAF profiles, JCL start procedures, configuration files, application files. We will illustrate all those as well.

That picture *looks* complicated, but as you'll see the configuration is not *that* complex. That's the reason for this document: to reveal the actual configuration elements so you can see how things map to this picture.

What this sample does not include

We are intentionally keeping the *security* model of this configuration as simple as we can make it. This configuration makes use of Liberty "basic" security. You will see these "basic" security definitions in the `server.xml` configuration files.

Why are we doing this? Because we want the focus on the key functional things illustrated in the picture above. The moment we introduce such things as SAF keyrings, digital certificates, and EJBROLE definitions we start to take the focus away from the key points.

WebSphere Liberty Batch is not limited to this "basic" security. It *can* use SAF for the user registry, for SSL digital certificates, and for application role enforcement. But that is outside of the scope of *this* document. Here we want you to stay focused on the multi-JVM model, the use of MQ as the integration mechanism between Dispatcher and Executors, and the batch events mechanism for monitoring.

We are not showing step-by-step in this document

In this document we are showing you the final result, along with some of the commands used to create the final result. But we are not going through each step we performed to accomplish this final result. That is provided at:

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

Under the "Step-by-Step Implementation Guide" section header found there.

Not necessarily "best practices"

What we're showing here may not be exactly what you would do. For example, the naming convention we use may not map to what you require. Or our use of JDBC Type 4, or MQ client mode may not be what you would do. That's okay: use this as an *example*. Tailor your actual environment based on the specific things you know are right for you.

2 If the `batchManagerZos` command line client were removed from this picture, then WOLA would not be present and the Angel Process would not be required. The Angel process may come *back* into the picture if other z/OS authorized services were used, such as SAF authorization. But for this sample illustration we need the Angel Process just for WOLA.

Configuration Illustration

Liberty z/OS install location

In our environment, the 16.0.0.4 level of Liberty z/OS was installed at:

```
/shared/zWebSphere/Liberty/V16004
```

RACF job to create foundational profiles

The following was the SYSTSIN DD for the JCL we ran to create the RACF profiles for this sample configuration.

Important! Security profiles should be reviewed by your security administrator prior to being created and used. These are what we used, but your security policies may indicate different values.

```
ADDGROUP JSRGRP OMVS(AUTOGID) OWNER(SYS1)

ADDUSER JSRANGL DFLTGRP(JSRGRP) OMVS(AUTOUID HOME(/shared/jsrhome/) -
  PROGRAM(/bin/sh)) NAME('LIBERTY ANGEL')  NOPASSWORD NOOIDCARD

ADDUSER JSRSERV DFLTGRP(JSRGRP) OMVS(AUTOUID HOME(/shared/jsrhome) -
  PROGRAM(/bin/sh)) NAME('LIBERTY SERVER')
ALTUSER JSRSERV PASSWORD(JSRSERV) NOEXPIRED

RDEFINE STARTED JSRZANGL.* UACC(NONE) -
  STDATA(USER(JSRANGL) GROUP(JSRGRP) -
  PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
RDEFINE STARTED JSRDISP.* UACC(NONE) -
 STDATA(USER(JSRSERV) GROUP(JSRGRP) -
 PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
RDEFINE STARTED JSREXEC1.* UACC(NONE)-
 STDATA(USER(JSRSERV) GROUP(JSRGRP) -
 PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
RDEFINE STARTED JSREXEC2.* UACC(NONE)-
 STDATA(USER(JSRSERV) GROUP(JSRGRP) -
 PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
RDEFINE STARTED JSRMON.* UACC(NONE)-
 STDATA(USER(JSRSERV) GROUP(JSRGRP) -
 PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
SETROPTS RACLIST(STARTED) REFRESH

RDEFINE SERVER BBG.ANGEL UACC(NONE) OWNER(SYS1)
PERMIT  BBG.ANGEL CLASS(SERVER) ACCESS(READ) ID(JSRSERV)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE) OWNER(SYS1)
PERMIT  BBG.AUTHMOD.BBGZSAFM -
        CLASS(SERVER) ACCESS(READ) ID(JSRSERV)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.SAFCRED UACC(NONE)
PERMIT  BBG.AUTHMOD.BBGZSAFM.SAFCRED -
        CLASS(SERVER) ACCESS(READ) ID(JSRSERV)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSWLM UACC(NONE)
PERMIT  BBG.AUTHMOD.BBGZSAFM.ZOSWLM  -
        CLASS(SERVER) ACCESS(READ) ID(JSRSERV)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.TXRRS UACC(NONE)
PERMIT  BBG.AUTHMOD.BBGZSAFM.TXRRS   -
        CLASS(SERVER) ACCESS(READ) ID(JSRSERV)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSDUMP UACC(NONE)
PERMIT  BBG.AUTHMOD.BBGZSAFM.ZOSDUMP  -
        CLASS(SERVER) ACCESS(READ) ID(JSRSERV)
RDEFINE SERVER BBG.SECPFX.BBGZDFLT UACC(NONE)
PERMIT  BBG.SECPFX.BBGZDFLT  -
        CLASS(SERVER) ACCESS(READ) ID(JSRSERV)
```

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.WOLA UACC(NONE) OWNER(SYS1)
PERMIT  BBG.AUTHMOD.BBGZSAFM.WOLA -
        CLASS(SERVER) ACCESS(READ) ID(JSRSERV)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.LOCALCOM UACC(NONE) OWNER(SYS1)
PERMIT  BBG.AUTHMOD.BBGZSAFM.LOCALCOM -
        CLASS(SERVER) ACCESS(READ) ID(JSRSERV)
RDEFINE SERVER BBG.AUTHMOD.BBGZSCFM UACC(NONE) OWNER(SYS1)
PERMIT  BBG.AUTHMOD.BBGZSCFM -
        CLASS(SERVER) ACCESS(READ) ID(JSRSERV)
RDEFINE SERVER BBG.AUTHMOD.BBGZSCFM.WOLA UACC(NONE) OWNER(SYS1)
PERMIT  BBG.AUTHMOD.BBGZSCFM.WOLA -
        CLASS(SERVER) ACCESS(READ) ID(JSRSERV)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.PRODMGR UACC(NONE) OWNER(SYS1)
PERMIT  BBG.AUTHMOD.BBGZSAFM.PRODMGR -
        CLASS(SERVER) ACCESS(READ) ID(JSRSERV)
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSAIO UACC(NONE) OWNER(SYS1)
PERMIT  BBG.AUTHMOD.BBGZSAFM.ZOSAIO -
        CLASS(SERVER) ACCESS(READ) ID(JSRSERV)
SETROPTS RACLIST(SERVER) REFRESH

RDEFINE CBIND BBG.WOLA.LIBERTY.BATCH.MANAGER UACC(NONE) OWNER(SYS1)
PERMIT BBG.WOLA.LIBERTY.BATCH.MANAGER CLASS(CBIND) ACCESS(READ) -
  ID(USER1)
SETROPTS RACLIST(CBIND) REFRESH
```

Notes:

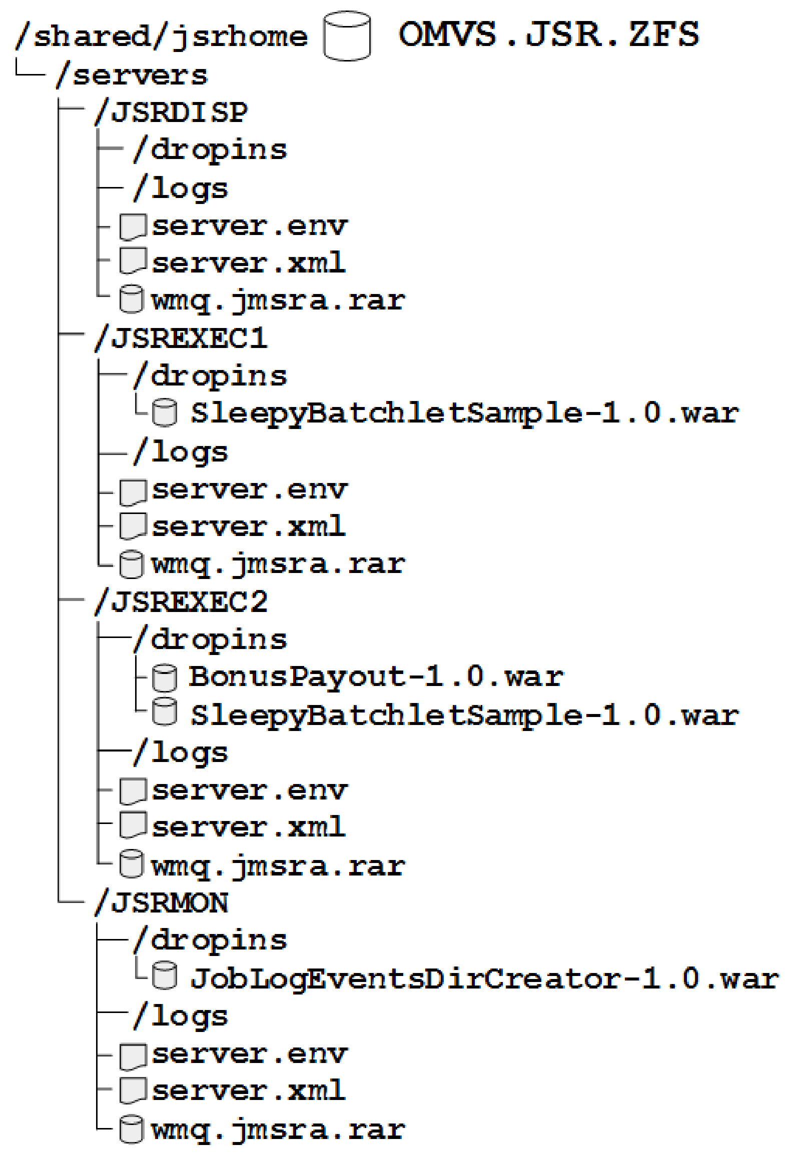
- The `JSRSERV` ID is the STC ID, and here we show it being assigned a password. STC IDs should not have passwords as a general rule, but in this case we gave it one so we could easily log in using that ID and create the server. At a minimum we would modify this ID to have `NOPASSWORD` after the server was created.
- The `JSRSERV` ID is the both the configuration file owning ID as well as the STC ID. You may not wish to do this in a production environment. Your security policies may require the STC ID not have write to the configuration file. For a test environment such as this, having the STC ID own the configuration files is simpler.
- The `STARTED` profiles assume we have a separate JCL start procedure for each server. As you will see, we did just that: we copied the sample JCL start proc and named it equal to each server name.
- We created every possible `SERVER` profile and granted the STC ID `JSRSERV` read access to all of them. This was more than was needed. The `batchManagerZos` use of WOLA implied just a few were *actually* needed. We provided this to illustrate what all the `SERVER` profiles are, and to show you the messages in the `messages.log` file that illustrate each authorized function being available to the server.
- The CBIND profile is what allows the user – `USER1` in this example – the ability to run `batchManagerZos` and establish a WOLA connection into the Dispatcher server. This example assumes a WOLA "three part name" of `LIBERTY+BATCH+MANAGER`. You'll see that reflected in the `server.xml` for the Dispatcher server.

If you are interested in a better understanding of the options available for the file system ownership and STC ID for Liberty z/OS, see:

`http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102687`

File system, directories and file structure

The following diagram illustrates the file system structure[3] for the sample environment:



```
/shared/jsrhome  OMVS.JSR.ZFS
└─/servers
   ├─/JSRDISP
   │  ├─/dropins
   │  ├─/logs
   │  ├─server.env
   │  ├─server.xml
   │  └─wmq.jmsra.rar
   ├─/JSREXEC1
   │  ├─/dropins
   │  │  └─SleepyBatchletSample-1.0.war
   │  ├─/logs
   │  ├─server.env
   │  ├─server.xml
   │  └─wmq.jmsra.rar
   ├─/JSREXEC2
   │  ├─/dropins
   │  │  ├─BonusPayout-1.0.war
   │  │  └─SleepyBatchletSample-1.0.war
   │  ├─/logs
   │  ├─server.env
   │  ├─server.xml
   │  └─wmq.jmsra.rar
   └─/JSRMON
      ├─/dropins
      │  └─JobLogEventsDirCreator-1.0.war
      ├─/logs
      ├─server.env
      ├─server.xml
      └─wmq.jmsra.rar
```

Notes:

- The `WLP_USER_DIR` in this case was `/shared/jsrhome`. We mounted the `OMVS.JSR.ZFS` at that location, then created the servers with the Liberty server shell script.
- We created the servers while logged in as `JSRSERV`. All the files are owned by `JSRSERV`, with group ownership of `JSRGRP`.
- We could have made use of shared directories and Liberty `<include>` processing, but chose to keep things intuitively simple, even if that meant some duplication existed.

Job repository DB2 DDL

The following DDL was produced by the `ddlGen` shell script which comes with Liberty. That shell script produced output to a file. The DDL was modified to fit into 80 colums and placed in a SPUFI input member and run. See the notes that follow.

3 Not all directories are shown. Some were omitted to provide focus on the key directories and files.

```
CREATE TABLE JBATCH.JOBINSTANCE
 (JOBINSTANCEID BIGINT GENERATED ALWAYS AS IDENTITY NOT NULL,
  AMCNAME VARCHAR(512), BATCHSTATUS INTEGER NOT NULL, CREATETIME
  TIMESTAMP NOT NULL, EXITSTATUS VARCHAR(512),
  INSTANCESTATE INTEGER NOT NULL, JOBNAME VARCHAR(256),
  JOBXMLNAME VARCHAR(128), JOBXML BLOB(64000),
  NUMEXECS INTEGER NOT NULL, RESTARTON VARCHAR(128),
  SUBMITTER VARCHAR(256), UPDATETIME TIMESTAMP,
  PRIMARY KEY (JOBINSTANCEID))  CCSID UNICODE;
CREATE TABLE JBATCH.STEPTHREADINSTANCE
 (PARTNUM INTEGER NOT NULL, STEPNAME VARCHAR(128) NOT NULL,
  THREADTYPE VARCHAR(31), CHECKPOINTDATA BLOB(64000),
  FK_JOBINSTANCEID BIGINT NOT NULL, FK_LATEST_STEPEXECID
  BIGINT NOT NULL, PARTITIONED SMALLINT DEFAULT 0 NOT NULL,
  PARTITIONPLANSIZE INTEGER, STARTCOUNT INTEGER,
  PRIMARY KEY (PARTNUM, STEPNAME, FK_JOBINSTANCEID))
  CCSID UNICODE;
CREATE INDEX JBATCH.STI_FKINSTANCEID_IX ON
  JBATCH.STEPTHREADINSTANCE (FK_JOBINSTANCEID);
CREATE INDEX JBATCH.STI_FKLATEST_SEI_IX ON
  JBATCH.STEPTHREADINSTANCE (FK_LATEST_STEPEXECID);
CREATE TABLE JBATCH.JOBEXECUTION
 (JOBEXECID BIGINT GENERATED ALWAYS AS IDENTITY NOT NULL,
  BATCHSTATUS INTEGER NOT NULL, CREATETIME TIMESTAMP NOT NULL,
  ENDTIME TIMESTAMP, EXECNUM INTEGER NOT NULL,
  EXITSTATUS VARCHAR(512), JOBPARAMETERS BLOB(64000),
  UPDATETIME TIMESTAMP, LOGPATH VARCHAR(512),
  RESTURL VARCHAR(512), SERVERID VARCHAR(256),
  STARTTIME TIMESTAMP, FK_JOBINSTANCEID BIGINT
  NOT NULL, PRIMARY KEY (JOBEXECID))  CCSID UNICODE;
CREATE INDEX JBATCH.JE_FKINSTANCEID_IX ON
  JBATCH.JOBEXECUTION (FK_JOBINSTANCEID);
CREATE TABLE JBATCH.STEPTHREADEXECUTION
 (STEPEXECID BIGINT GENERATED ALWAYS AS IDENTITY NOT NULL,
  THREADTYPE VARCHAR(31), BATCHSTATUS INTEGER NOT NULL,
  M_COMMIT BIGINT NOT NULL, ENDTIME TIMESTAMP, EXITSTATUS
  VARCHAR(512), M_FILTER BIGINT NOT NULL, INTERNALSTATUS
  INTEGER NOT NULL, PARTNUM INTEGER NOT NULL,
  USERDATA BLOB(64000), M_PROCESSSKIP BIGINT NOT NULL,
  M_READ BIGINT NOT NULL, M_READSKIP BIGINT NOT NULL,
  M_ROLLBACK BIGINT NOT NULL, STARTTIME TIMESTAMP,
  STEPNAME VARCHAR(128) NOT NULL, M_WRITE BIGINT NOT NULL,
  M_WRITESKIP BIGINT NOT NULL, FK_JOBEXECID BIGINT NOT NULL,
  FK_TOPLVL_STEPEXECID BIGINT, ISPARTITIONEDSTEP SMALLINT
  DEFAULT 0, PRIMARY KEY (STEPEXECID))  CCSID UNICODE;
CREATE INDEX JBATCH.STE_FKJOBEXECID_IX ON
  JBATCH.STEPTHREADEXECUTION (FK_JOBEXECID);
CREATE INDEX JBATCH.STE_FKTLSTEPEID_IX ON
  JBATCH.STEPTHREADEXECUTION (FK_TOPLVL_STEPEXECID);
CREATE TABLE JBATCH.JOBPARAMETER (NAME VARCHAR(255),
  VALUE VARCHAR(255), FK_JOBEXECID BIGINT)  CCSID UNICODE;
CREATE INDEX JBATCH.JP_FKJOBEXECID_IX ON
  JBATCH.JOBPARAMETER (FK_JOBEXECID);
ALTER TABLE JBATCH.STEPTHREADEXECUTION ADD CONSTRAINT
  STPTHRADEXECUTION0 UNIQUE (FK_JOBEXECID, STEPNAME, PARTNUM);
ALTER TABLE JBATCH.STEPTHREADINSTANCE ADD CONSTRAINT
  STPTHRFKLTSTSTPXCD FOREIGN KEY (FK_LATEST_STEPEXECID)
  REFERENCES JBATCH.STEPTHREADEXECUTION (STEPEXECID);
ALTER TABLE JBATCH.STEPTHREADINSTANCE ADD CONSTRAINT
```

```
  STPTHRDNFKJBNSTNCD FOREIGN KEY (FK_JOBINSTANCEID)
  REFERENCES JBATCH.JOBINSTANCE (JOBINSTANCEID);
ALTER TABLE JBATCH.JOBEXECUTION ADD CONSTRAINT
  JBXCTNFKJBNSTNCEID FOREIGN KEY (FK_JOBINSTANCEID)
  REFERENCES JBATCH.JOBINSTANCE (JOBINSTANCEID);
ALTER TABLE JBATCH.STEPTHREADEXECUTION ADD CONSTRAINT
  STPTHFKTPLVLSTPXCD FOREIGN KEY (FK_TOPLVL_STEPEXECID)
  REFERENCES JBATCH.STEPTHREADEXECUTION (STEPEXECID)
  ON DELETE NO ACTION;
ALTER TABLE JBATCH.STEPTHREADEXECUTION ADD CONSTRAINT
  STPTHRDXCTNFKJBXCD FOREIGN KEY (FK_JOBEXECID)
  REFERENCES JBATCH.JOBEXECUTION (JOBEXECID);
ALTER TABLE JBATCH.JOBPARAMETER ADD CONSTRAINT
  JBPRMETERFKJBXECID FOREIGN KEY (FK_JOBEXECID)
  REFERENCES JBATCH.JOBEXECUTION (JOBEXECID);
```

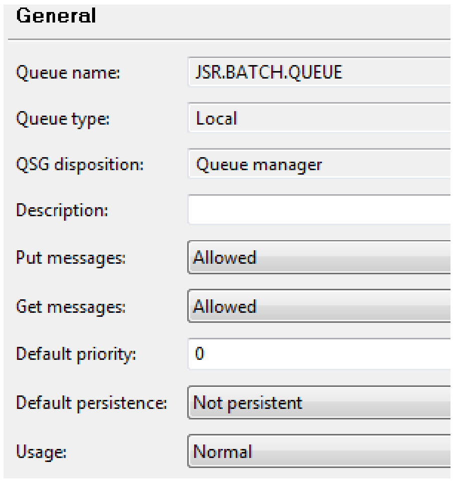
Notes:

- Do **not** simply copy and paste from this listing. You should be in the practice of generating the DDL using the `ddlGen` utility and using the generated DDL, *not* DDL from a document that may be backlevel from the level of Liberty you're using.
- The DDL listing above does not include any *other* DB2 definitions you may require, such as STOGROUP, or GRANT. Review this with your DB administrator and implement according to your local policies.
- The `ON DELETE NO ACTION` statement (in **red** above) was added manually. Our local CURRENT RULES was set to DB2, which mandated an 'ON DELETE' action when a constraint references the same table. You may not require this, depending on your CURRENT RULES.

Job dispatching queue

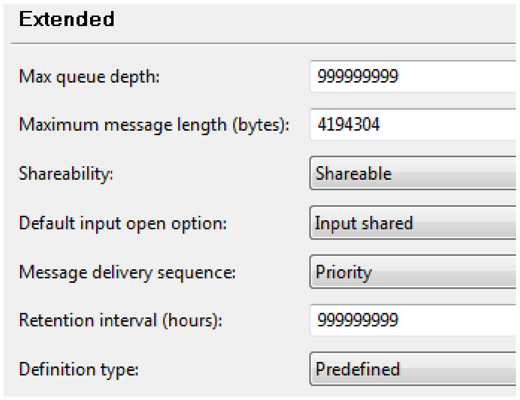
This is a defined queue that sits between the Dispatcher server (`JSRDISP`) and the Executor servers (`JSREXEC1` and `JSREXEC2`). The Dispatcher server is defined to put its job submission messages on this queue; the Executor servers have JMS activation specifications defined to listen on ths queue.

The queue we created was called `JSR.BATCH.QUEUE`, and it was defined as a local queue in our environment. Using MQ Explorer, the queue definition looked like this:



General

Queue name:	JSR.BATCH.QUEUE
Queue type:	Local
QSG disposition:	Queue manager
Description:	
Put messages:	Allowed
Get messages:	Allowed
Default priority:	0
Default persistence:	Not persistent
Usage:	Normal



Extended

Max queue depth:	999999999
Maximum message length (bytes):	4194304
Shareability:	Shareable
Default input open option:	Input shared
Message delivery sequence:	Priority
Retention interval (hours):	999999999
Definition type:	Predefined

Dispatcher server (JSRDISP)

/dropins directory

No applications were deployed to this server.

server.xml

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

    <!-- Enable features -->
    <featureManager>
        <feature>servlet-3.1</feature>
        <feature>batch-1.0</feature>
        <feature>batchManagement-1.0</feature>
        <feature>zosLocalAdapters-1.0</feature>
        <feature>appSecurity-2.0</feature>
        <feature>wmqJmsClient-2.0</feature>
        <feature>adminCenter-1.0</feature>
    </featureManager>

    <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">
        <special-subject type="EVERYONE"/>
        <user name="Fred" />
      </security-role>
    </authorization-roles>

    <administrator-role>
      <user>Fred</user>
    </administrator-role>

    <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="xxxxxx" password="xxxxxx" />

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

    <variable name="wmqJmsClient.rar.location"
      value="${server.config.dir}/wmq.jmsra.rar" />

    <wmqJmsClient startupRetryCount="999"
      startupRetryInterval="1000ms"
      reconnectionRetryCount="10"
      reconnectionRetryInterval="5m">
    </wmqJmsClient>

    <batchJmsEvents connectionFactoryRef="batchConnectionFactory" />

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

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

    <zosLocalAdapters wolaGroup="LIBERTY"
      wolaName2="BATCH"
      wolaName3="MANAGER"/>

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

</server>
```

Notes:

- This sample uses "basic" security, which means RACF is not used for SSL certificates, authentication, or application authorization. Liberty itself is doing that.

  The `<keyStore>` element is auto-creating a self-signed SSL certificate.

  The `<basicRegistry>` element is defining a user registry, which has one user: "Fred".

  The `<authorization-roles>` element is creating one Java batch role of "batchAdmin" and providing two users access:

Fred	This is used when `batchManager` is the command line client, and Fred's ID and password are supplied on the command. This grants Fred the authority to submit jobs.

EVERYONE	This is used when `batchManagerZos` is the command line client and basic security is in place. Long story short: basic security can't determine who the ID is on the other side of the WOLA connection, so it can't determine whether to let them in or not. The special-subect EVERYONE permits access. If we were using SAF registry and SAF authorization, then we could properly authorize the user (USER1) to the EJBROLE for access. But we're using basic registry to keep that part simple, and thus this EVERYONE is needed.

The `<administrator-role>` is for the AdminCenter. This is granting Fred access to the AdminCenter as an administrator.

- JDBC Type 4 is used for access to the Job Repository tables in DB2. The alias ID defined must have `SELECT`, `INSERT`, `UPDATE` and `DELETE` authority.
- Two JMS functions are defined: `<batchJmsDispatcher>` to queue job submission requests for Executor servers to pick up and run; and `<batchJmsEvents>` to publish events to the MQ topic for subcribers to monitor[4].
- We're showing CLIENT mode connection to MQ. If you use BINDINGS mode, you must add the `zosTransaction-1.0` feature to the feature list, as well as grant the server `READ` to the `BBG.AUTHMOD.BBGZSAFM.TXRRS SERVER` profile. The Angel must be started prior to the server starting.
- The `<zosLocalAdapters>` element defines the WOLA "three part name" that will be used on the `batchManagerZos` command line client. This three part name also relates back to the CBIND profile we saw illustrated in the RACF definitions.x

server.env

```
JAVA_HOME=/shared/java/J8.0_64
```

This is simply a pointer to the 64-bit Java the server was to use.

wmq.jmsra.rar

This is the JCA resource adapter supplied with IBM MQ. It was copied to this server directory from the `/`*`<MQ_install_path>`*`/java/lib/jca` location. We point to this from `server.xml` with:

```
<variable name="wmqJmsClient.rar.location"
      value="${server.config.dir}/wmq.jmsra.rar" />
```

We could have located this in a common location and shared one file between the servers. The key point is that the server needed this RAR to do JMS work.

JSRDISP server JCL start procedure

This was copied from the install location:

`/`*`<install_path>`*`/templates/zos/procs/bbgzsrv.jcl`

and renamed to `JSRDISP`. Then it was customized:

```
//JSRDISP PROC PARMS='JSRDISP --clean'
//*------------------------------------------------------------------
//* This proc may be overwritten by fixpacks or iFixes.
//* You must copy to another location before customizing.
//*------------------------------------------------------------------
//* INSTDIR - the path to the WebSphere Liberty Profile install.
//*           This path is used to find the product code and is
//*           equivalent to the WLP_INSTALL_DIR environment variable
//*           in the Unix shell.
//* USERDIR - the path to the WebSphere Liberty Profile user area.
```

4 The JSRMON server application is monitoring for events that only the Executor servers publish, so in theory we don't really need the `<batchJmsEvents>` element here. But you may have a monitor that is interested in the events published by the Dispatcher server, so we illustrate that here.

```
//*           This path is used to store shared and server specific
//*           configuration information and is equivalent to the
//*           WLP_USER_DIR environment variable in the Unix shell.
//*-------------------------------------------------------------------
//  SET INSTDIR='/shared/zWebSphere/Liberty/V16004'
//  SET USERDIR='/shared/jsrhome'
//*-------------------------------------------------------------------
//* Start the Liberty server
//*
//* WLPUDIR - PATH DD that points to the Liberty Profile's "user"
//*           directory. If the DD is not allocated, the user
//*           directory location defaults to the wlp/usr directory
//*           in the install tree.
//* STDOUT  - Destination for stdout (System.out)
//* STDERR  - Destination for stderr (System.err)
//* MSGLOG  - Destination for messages.log (optional)
//* STDENV  - Initial Unix environment - read by the system.  The
//*           installation default and server specific server
//*           environment files will be merged into this environment
//*           before the JVM is launched.
//*-------------------------------------------------------------------
//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=*
//*MSGLOG   DD SYSOUT=*
//*STDENV   DD PATH='/etc/system.env',PATHOPTS=(ORDONLY)
//*STDOUT   DD PATH='&ROOT/std.out',
//*            PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//*            PATHMODE=SIRWXU
//*STDERR   DD PATH='&ROOT/std.err',
//*            PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//*            PATHMODE=SIRWXU
//* ================================================================ */
//* PROPRIETARY-STATEMENT:                                           */
//* Licensed Material - Property of IBM                              */
//*                                                                  */
//* (C) Copyright IBM Corp. 2011, 2012                               */
//* All Rights Reserved                                              */
//* US Government Users Restricted Rights - Use, duplication or      */
//* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*/
//* ================================================================ */
```

Notes:

- Since we are dedicating a proc to each server, we can code the server name on the PROC statement `PARMS=' '`. The `--clean` is not strictly necessary; it simply tells the server to clear cache prior to starting.
- `INSTDIR=` is set to the installation directory path.
- `USERDIR=` is set to the `WLP_USER_DIR` value for this server.

JSRZANGL JCL start procedure

This was copied from the install location:

`/<install_path>/templates/zos/procs/bbgzangl.jcl`

and renamed to `JSRZANGL`. Then it was customized.

Note: We did a slightly "bad practice" thing here – by naming this "JSRZANGL," it implies this Angel is exclusive to the "JSR" servers. In truth, an Angel can be shared[5] between Liberty z/OS servers being used for different things. We named this "JSRZANGL" so a listing of active tasks with `PREFIX JSR*` would yield all five started tasks. In reality you would likely leave the Angel name the default `BBGZANGL`, or some other generic name.

```
//JSRZANGL PROC PARMS='',COLD=N,NAME=''
//*------------------------------------------------------------------
//  SET ROOT='/shared/zWebSphere/Liberty/V16004'
//*------------------------------------------------------------------
//* Start the Liberty angel process
//*------------------------------------------------------------------
//* This proc may be overwritten by fixpacks or iFixes.
//* You must copy to another location before customizing.
//*------------------------------------------------------------------
//STEP1   EXEC PGM=BPXBATA2,REGION=0M,TIME=NOLIMIT,
//      PARM='PGM &ROOT./lib/native/zos/s390x/bbgzangl COLD=&COLD NAME=X
//             &NAME &PARMS'
//STDOUT    DD SYSOUT=*
//STDERR    DD SYSOUT=*
//* ================================================================ */
//* PROPRIETARY-STATEMENT:                                           */
//* Licensed Material - Property of IBM                              */
//*                                                                  */
//* (C) Copyright IBM Corp. 2011, 2012                               */
//* All Rights Reserved                                              */
//* US Government Users Restricted Rights - Use, duplication or      */
//* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*/
//* ================================================================ */
```

First executor (JSREXEC1)

/dropins directory

The `SleepyBatchletSample-1.0.war` application was deployed to this server.

server.xml

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

    <!-- Enable features -->
    <featureManager>
        <feature>servlet-3.1</feature>
        <feature>batch-1.0</feature>
        <feature>batchManagement-1.0</feature>
        <feature>appSecurity-2.0</feature>
        <feature>wmqJmsClient-2.0</feature>
    </featureManager>

    <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">
```

5 In 16.0.0.4 the ability to create multiple "named Angels" was introduced. This allows you to specify which Angel, among several, your server will make use of. Look at the Angel JCL proc and the `NAME=''` parameter. That hints at this new "named Angel" function. A blank value means it's an unnamed Angel. We chose to **not** illustrate "named Angels to maintain focus on the key Java Batch functionality.

```
      <special-subject type="EVERYONE"/>
      <user name="Fred" />
    </security-role>
  </authorization-roles>

  <batchJmsEvents connectionFactoryRef="batchConnectionFactory" />

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

  <batchPersistence jobStoreRef="BatchDatabaseStore" />

  <databaseStore id="BatchDatabaseStore"
   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="xxxxxx" password="xxxxxx" />

  <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>

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

 <variable name="wmqJmsClient.rar.location"
    value="${server.config.dir}/wmq.jmsra.rar"/>

  <wmqJmsClient startupRetryCount="999"
    startupRetryInterval="1000ms"
    reconnectionRetryCount="10"
    reconnectionRetryInterval="5m">
  </wmqJmsClient>

  <jmsActivationSpec id="batchActivationSpec1" >
    <properties.wmqJms
      destinationRef="batchJobSubmissionQueue"
 messageSelector="com_ibm_ws_batch_applicationName = 'SleepyBatchletSample-1.0'"
      maxPoolDepth="1"
```

```
        transportType="CLIENT"
        channel="SYSTEM.DEF.SVRCONN"
        destinationType="javax.jms.Queue"
        queueManager="MQS1"
        hostName="wg31.washington.ibm.com"
        port="1414">
      </properties.wmqJms>
    </jmsActivationSpec>

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

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="26080"
                  httpsPort="26443" />

</server>
```

Notes:

- This server has no `zosLocalAdapters-1.0` feature, and no `adminCenter-1.0` feature.
- JDBC Type 4 is used for access to the Job Repository tables in DB2. The alias ID defined must have `SELECT`, `INSERT`, `UPDATE` and `DELETE` authority.
- This has a `<batchJmsExecutor>` element rather than `<batchJmsDispatcher>`.
- This has a `<jmsActivationSpec>` element to define the queue on which to listen for job submission messages.
- We're showing CLIENT mode connection to MQ. If you use BINDINGS mode, you must add the `zosTransaction-1.0` feature to the feature list, as well as grant the server `READ` to the `BBG.AUTHMOD.BBGZSAFM.TXRRS SERVER` profile. The Angel must be started prior to the server starting.
- Notice the `messageSelector=` property. This defines what messages to pick up. In this example it will only pick up job submissions messages for the SleepyBatchlet application.
- The HTTP ports are unique from other servers.

server.env

```
JAVA_HOME=/shared/java/J8.0_64
```

This is simply a pointer to the 64-bit Java the server was to use. It's the exact same content as seen in the `server.env` for the other servers.

wmq.jmsra.rar

This is the JCA resource adapter supplied with IBM MQ. It was copied to this server directory from the `/`*`<MQ_install_path>`*`/java/lib/jca` location. We point to this from `server.xml` with:

```
<variable name="wmqJmsClient.rar.location"
      value="${server.config.dir}/wmq.jmsra.rar" />
```

We could have located this in a common location and shared one file between the servers. The key point is that the server needed this RAR to do JMS work.

JSREXEC1 server JCL start procedure

This was copied from the install location:

/<install_path>/templates/zos/procs/bbgzsrv.jcl

and renamed to JSREXEC1. Then it was customized:

```
//JSREXEC1 PROC PARMS='JSREXEC1 --clean'
//*------------------------------------------------------------------
//* This proc may be overwritten by fixpacks or iFixes.
//* You must copy to another location before customizing.
//*------------------------------------------------------------------
//* INSTDIR - the path to the WebSphere Liberty Profile install.
//*           This path is used to find the product code and is
//*           equivalent to the WLP_INSTALL_DIR environment variable
//*           in the Unix shell.
//* USERDIR - the path to the WebSphere Liberty Profile user area.
//*           This path is used to store shared and server specific
//*           configuration information and is equivalent to the
//*           WLP_USER_DIR environment variable in the Unix shell.
//*------------------------------------------------------------------
//  SET INSTDIR='/shared/zWebSphere/Liberty/V16004'
//  SET USERDIR='/shared/jsrhome'
//*------------------------------------------------------------------
//* Start the Liberty server
//*
//* WLPUDIR - PATH DD that points to the Liberty Profile's "user"
//*           directory. If the DD is not allocated, the user
//*           directory location defaults to the wlp/usr directory
//*           in the install tree.
//* STDOUT  - Destination for stdout (System.out)
//* STDERR  - Destination for stderr (System.err)
//* MSGLOG  - Destination for messages.log (optional)
//* STDENV  - Initial Unix environment - read by the system.  The
//*           installation default and server specific server
//*           environment files will be merged into this environment
//*           before the JVM is launched.
//*------------------------------------------------------------------
//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=*
//*MSGLOG   DD SYSOUT=*
//*STDENV   DD PATH='/etc/system.env',PATHOPTS=(ORDONLY)
//*STDOUT   DD PATH='&ROOT/std.out',
//*            PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//*            PATHMODE=SIRWXU
//*STDERR   DD PATH='&ROOT/std.err',
//*            PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//*            PATHMODE=SIRWXU
//* =============================================================== */
//* PROPRIETARY-STATEMENT:                                          */
//* Licensed Material - Property of IBM                             */
//*                                                                 */
//* (C) Copyright IBM Corp. 2011, 2012                              */
//* All Rights Reserved                                             */
//* US Government Users Restricted Rights - Use, duplication or     */
//* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*/
//* =============================================================== */
```

Notes:

- Since we are dedicating a proc to each server, we can code the server name on the PROC statement `PARMS=' '`. The `--clean` is not strictly necessary; it simply tells the server to clear cache prior to starting.
- `INSTDIR=` is set to the installation directory path.
- `USERDIR=` is set to the `WLP_USER_DIR` value for this server.

Second executor (JSREXEC2)

/dropins directory

Two applications were deployed into this server:

- `SleepyBatchletSample-1.0.war`
- `BonusPayout-1.0.war`

server.xml

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

    <!-- Enable features -->
    <featureManager>
        <feature>servlet-3.1</feature>
        <feature>batch-1.0</feature>
        <feature>batchManagement-1.0</feature>
        <feature>wmqJmsClient-2.0</feature>
        <feature>appSecurity-1.0</feature>
    </featureManager>

    <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">
        <special-subject type="EVERYONE"/>
        <user name="Fred" />
      </security-role>
    </authorization-roles>

    <batchJmsEvents connectionFactoryRef="batchConnectionFactory" />

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

    <batchPersistence jobStoreRef="BatchDatabaseStore" />

    <databaseStore id="BatchDatabaseStore"
     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="xxxxxx" password="xxxxxx" />

   <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>

   <authData id="bonusAlias" user="xxxxxx" password="xxxxxx" />

   <dataSource id="bonusDB" jndiName="jdbc/bonus"
     containerAuthDataRef="bonusAlias"
    type="javax.sql.XADataSource"
    jdbcDriverRef="DB2T4">
    <properties.db2.jcc
     serverName="wg31.washington.ibm.com"
     portNumber="9446"
     databaseName="WG31DB2"
     driverType="4" />
   </dataSource>

   <batchJmsExecutor activationSpecRef="batchActivationSpec"
     queueRef="batchJobSubmissionQueue"/>

   <variable name="wmqJmsClient.rar.location"
     value="${server.config.dir}/wmq.jmsra.rar"/>

   <wmqJmsClient startupRetryCount="999"
     startupRetryInterval="1000ms"
     reconnectionRetryCount="10"
     reconnectionRetryInterval="5m">
   </wmqJmsClient>

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

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

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="27080"
                  httpsPort="27443" />

</server>
```

Notes:

- We're showing CLIENT mode connection to MQ. If you use BINDINGS mode, you must add the `zosTransaction-1.0` feature to the feature list, as well as grant the server `READ` to the `BBG.AUTHMOD.BBGZSAFM.TXRRS SERVER` profile. The Angel must be started prior to the server starting.
- This has an additional JDBC Type 4 data source which is used by the BonusPayout application.
- The JMS activation specification element has a conditional "or" operator that allows it to pick up a SleepyBatchlet or BonusPayout application job submission.

server.env

```
JAVA_HOME=/shared/java/J8.0_64
```

This is simply a pointer to the 64-bit Java the server was to use. It's the exact same content as seen in the `server.env` for the other servers.

wmq.jmsra.rar

This is the JCA resource adapter supplied with IBM MQ. It was copied to this server directory from the /*<MQ_install_path>*/java/lib/jca location. We point to this from `server.xml` with:

```
<variable name="wmqJmsClient.rar.location"
      value="${server.config.dir}/wmq.jmsra.rar" />
```

We could have located this in a common location and shared one file between the servers. The key point is that the server needed this RAR to do JMS work.

JSREXEC2 server JCL start procedure

This was copied from the install location:

/*<install_path>*/templates/zos/procs/bbgzsrv.jcl

and renamed to `JSREXEC2`. Then it was customized:

```
//JSREXEC2 PROC PARMS='JSREXEC2 --clean'
//*------------------------------------------------------------------
//* This proc may be overwritten by fixpacks or iFixes.
//* You must copy to another location before customizing.
//*------------------------------------------------------------------
//* INSTDIR - the path to the WebSphere Liberty Profile install.
//*           This path is used to find the product code and is
//*           equivalent to the WLP_INSTALL_DIR environment variable
//*           in the Unix shell.
//* USERDIR - the path to the WebSphere Liberty Profile user area.
//*           This path is used to store shared and server specific
//*           configuration information and is equivalent to the
//*           WLP_USER_DIR environment variable in the Unix shell.
//*-------------------------------------------------------------------
//  SET INSTDIR='/shared/zWebSphere/Liberty/V16004'
//  SET USERDIR='/shared/jsrhome'
//*-------------------------------------------------------------------
//* Start the Liberty server
//*
//* WLPUDIR - PATH DD that points to the Liberty Profile's "user"
//*           directory. If the DD is not allocated, the user
//*           directory location defaults to the wlp/usr directory
//*           in the install tree.
//* STDOUT  - Destination for stdout (System.out)
//* STDERR  - Destination for stderr (System.err)
//* MSGLOG  - Destination for messages.log (optional)
//* STDENV  - Initial Unix environment - read by the system.  The
//*           installation default and server specific server
//*           environment files will be merged into this environment
//*           before the JVM is launched.
//*-------------------------------------------------------------------
//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=*
//*MSGLOG   DD SYSOUT=*
//*STDENV   DD PATH='/etc/system.env',PATHOPTS=(ORDONLY)
//*STDOUT   DD PATH='&ROOT/std.out',
//*            PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//*            PATHMODE=SIRWXU
//*STDERR   DD PATH='&ROOT/std.err',
//*            PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//*            PATHMODE=SIRWXU
//* ================================================================ */
//* PROPRIETARY-STATEMENT:                                           */
//* Licensed Material - Property of IBM                              */
//*                                                                  */
//* (C) Copyright IBM Corp. 2011, 2012                               */
//* All Rights Reserved                                              */
//* US Government Users Restricted Rights - Use, duplication or      */
//* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*/
//* ================================================================ */
```

Notes:

- Since we are dedicating a proc to each server, we can code the server name on the PROC statement `PARMS=' '`. The `--clean` is not strictly necessary; it simply tells the server to clear cache prior to starting.
- `INSTDIR=` is set to the installation directory path.
- `USERDIR=` is set to the `WLP_USER_DIR` value for this server.

Bonus application account table DDL

The `BonusPayout-1.0.war` application requires a single table for its processing. The DDL for this table was the following:

```
CREATE TABLE BONUSDB.ACCOUNT(
 ACCTNUM INTEGER NOT NULL,
 BALANCE INTEGER NOT NULL,
 INSTANCEID BIGINT NOT NULL,
 ACCTCODE VARCHAR(30),
 CONSTRAINT ACCOUNT_PK PRIMARY KEY (ACCTNUM,INSTANCEID));

CREATE UNIQUE INDEX ACCT_IDX ON BONUSDB.ACCOUNT(ACCTNUM,INSTANCEID);

COMMIT;
```

Notes:

- This DDL listing does not include any other DB2 definitions you may require, such as STOGROUP, or GRANT. Review this with your DB administrator and implement according to your local policies.
- The ID that accesses this table must have SELECT, INSERT and UPDATE authority at a minimum.

Monitor server (JSRMON)

This server hosts an MDB application that listens on the batch event topic that the servers are publishing to. It is watching for the job log event, and will produce a folder and file for each job log that it sees.

/dropins directory

The application `JobLogEventsDirCreator-1.0.war` was deployed in this server. This was pulled fromt his Git location:

https://github.com/WASdev/sample.batch.joblogevents

server.xml

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

    <!-- Enable features -->
    <featureManager>
        <feature>batch-1.0</feature>
        <feature>batchManagement-1.0</feature>
        <feature>jsonp-1.0</feature>
        <feature>servlet-3.1</feature>
        <feature>wmqJmsClient-2.0</feature>
    </featureManager>

    <variable name="wmqJmsClient.rar.location"
      value="${server.config.dir}/wmq.jmsra.rar" />

    <wmqJmsClient startupRetryCount="999"
      startupRetryInterval="1000ms"
      reconnectionRetryCount="10"
      reconnectionRetryInterval="5m">
    </wmqJmsClient>

    <jmsTopic id="JobLogEventTopic"
      jndiName="jms/batch/batchJobTopic">
      <properties.wmqJms
      baseTopicName="batch/jobs/execution/jobLogPart" />
    </jmsTopic>

    <jmsActivationSpec id="JobLogEventsDirCreator-1.0/JobLogEventsSubscriber">
      <properties.wmqJms
       destinationRef="JobLogEventTopic"
       destinationType="javax.jms.Topic"
       transportType="CLIENT"
       channel="SYSTEM.DEF.SVRCONN"
       queueManager="MQS1"
       hostName="wg31.washington.ibm.com"
       port="1414" />
    </jmsActivationSpec>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="28080"
                  httpsPort="28443" />

</server>
```

Notes:

- The application requires the `jsonp-1.0` feature to operate.
- The baseTopicName= attribute specifies which topic leaf to listen on. The application is written to act upon the "jobLogPart" events only, so that's what we specify here[6].
- We're showing CLIENT mode connection to MQ. If you use BINDINGS mode, you must add the `zosTransaction-1.0` feature to the feature list, as well as grant the server `READ` to the `BBG.AUTHMOD.BBGZSAFM.TXRRS SERVER` profile. The Angel must be started prior to the server starting.

server.env

```
JAVA_HOME=/shared/java/J8.0_64
```

This is simply a pointer to the 64-bit Java the server was to use. It's the exact same content as seen in the `server.env` for the other servers.

wmq.jmsra.rar

This is the JCA resource adapter supplied with IBM MQ. It was copied to this server directory from the `/`*`<MQ_install_path>`*`/java/lib/jca` location. We point to this from `server.xml` with:

```
<variable name="wmqJmsClient.rar.location"
      value="${server.config.dir}/wmq.jmsra.rar" />
```

We could have located this in a common location and shared one file between the servers. The key point is that the server needed this RAR to do JMS work.

JSRMON server JCL start procedure

This was copied from the install location:

`/`*`<install_path>`*`/templates/zos/procs/bbgzsrv.jcl`

and renamed to `JSRMON`. Then it was customized:

```
//JSRMON  PROC PARMS='JSRMON --clean'
//*------------------------------------------------------------------
//* This proc may be overwritten by fixpacks or iFixes.
//* You must copy to another location before customizing.
//*------------------------------------------------------------------
//* INSTDIR - the path to the WebSphere Liberty Profile install.
//*           This path is used to find the product code and is
//*           equivalent to the WLP_INSTALL_DIR environment variable
//*           in the Unix shell.
//* USERDIR - the path to the WebSphere Liberty Profile user area.
//*           This path is used to store shared and server specific
//*           configuration information and is equivalent to the
//*           WLP_USER_DIR environment variable in the Unix shell.
//*------------------------------------------------------------------
```

6 We could specify the topic root of `batch/#` and it would listen on everything. But the application would only act upon events it sees under the `batch/jobs/execution/jobLogPart` leaf.

```
//   SET INSTDIR='/shared/zWebSphere/Liberty/V16004'
//   SET USERDIR='/shared/jsrhome'
//*-------------------------------------------------------------------
//* Start the Liberty server
//*
//* WLPUDIR - PATH DD that points to the Liberty Profile's "user"
//*           directory. If the DD is not allocated, the user
//*           directory location defaults to the wlp/usr directory
//*           in the install tree.
//* STDOUT  - Destination for stdout (System.out)
//* STDERR  - Destination for stderr (System.err)
//* MSGLOG  - Destination for messages.log (optional)
//* STDENV  - Initial Unix environment - read by the system.  The
//*           installation default and server specific server
//*           environment files will be merged into this environment
//*           before the JVM is launched.
//*-------------------------------------------------------------------
//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=*
//*MSGLOG   DD SYSOUT=*
//*STDENV   DD PATH='/etc/system.env',PATHOPTS=(ORDONLY)
//*STDOUT   DD PATH='&ROOT/std.out',
//*            PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//*            PATHMODE=SIRWXU
//*STDERR   DD PATH='&ROOT/std.err',
//*            PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//*            PATHMODE=SIRWXU
//* ================================================================ */
//* PROPRIETARY-STATEMENT:                                           */
//* Licensed Material - Property of IBM                              */
//*                                                                  */
//* (C) Copyright IBM Corp. 2011, 2012                               */
//* All Rights Reserved                                              */
//* US Government Users Restricted Rights - Use, duplication or      */
//* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*/
//* ================================================================ */
```

Notes:

- Since we are dedicating a proc to each server, we can code the server name on the PROC statement `PARMS=' '`. The `--clean` is not strictly necessary; it simply tells the server to clear cache prior to starting.
- `INSTDIR=` is set to the installation directory path.
- `USERDIR=` is set to the `WLP_USER_DIR` value for this server.

Operation Illustration

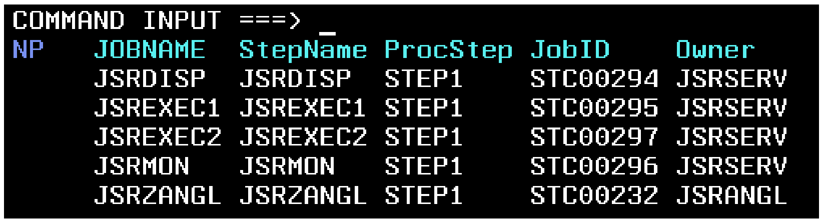
Given the configuration illustrated above, in this section we will illustrate basic operations of this environment, including job submission and event monitoring.

Start servers

The servers were started in this order:

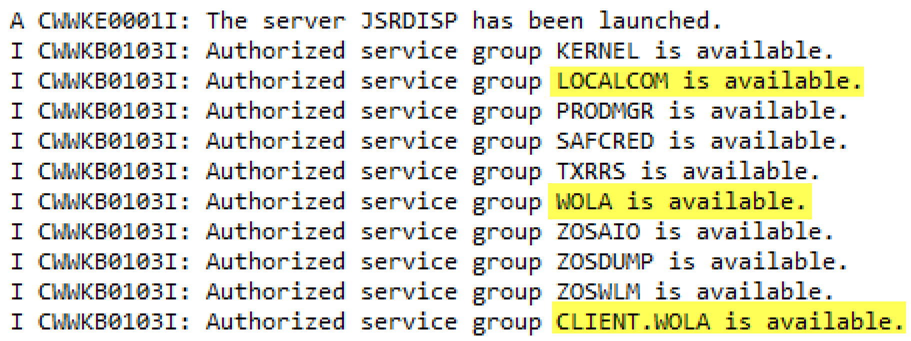
Server START[7]	*Notes*
/S JSRZANGL	In terms of sequence, this is the important point: the Angel process must be up before any Liberty z/OS server requiring the Angel is started.
/S JSRDISP	These can be started in any order.
/S JSREXEC1	
/S JSREXEC2	
/S JSRMON	

A listing of the jobs showed:



```
COMMAND INPUT ===> _
NP   JOBNAME  StepName ProcStep JobID    Owner
     JSRDISP  JSRDISP  STEP1    STC00294 JSRSERV
     JSREXEC1 JSREXEC1 STEP1    STC00295 JSRSERV
     JSREXEC2 JSREXEC2 STEP1    STC00297 JSRSERV
     JSRMON   JSRMON   STEP1    STC00296 JSRSERV
     JSRZANGL JSRZANGL STEP1    STC00232 JSRANGL
```

Only one server required the Angel process, and that was `JSRDISP`. The requirement for the Angel was driven by the use of `batchManagerZos` and WOLA[8]. A look in the `messages.log` for that server showed:



```
A CWWKE0001I: The server JSRDISP has been launched.
I CWWKB0103I: Authorized service group KERNEL is available.
I CWWKB0103I: Authorized service group LOCALCOM is available.
I CWWKB0103I: Authorized service group PRODMGR is available.
I CWWKB0103I: Authorized service group SAFCRED is available.
I CWWKB0103I: Authorized service group TXRRS is available.
I CWWKB0103I: Authorized service group WOLA is available.
I CWWKB0103I: Authorized service group ZOSAIO is available.
I CWWKB0103I: Authorized service group ZOSDUMP is available.
I CWWKB0103I: Authorized service group ZOSWLM is available.
I CWWKB0103I: Authorized service group CLIENT.WOLA is available.
```

The "is available" messages validated access to the WOLA authorized service was present.

A bit further down in the log file we saw:

```
CWWKB0501I: The WebSphere Optimized Local Adapter channel registered with the
Liberty profile server using the following name: LIBERTY BATCH MANAGER
```

That verified the "three part name" we defined in the `server.xml` was in effect. When we used `batchManagerZos` to submit jobs that three-part name was important to know.

Submit using batchManager

The `batchManager` command line client uses REST to submit jobs to the dispatcher server. It can be run from any shell environment. We will illustrate it being run from SSH using the PuTTY client tool.

7 Because we had a separate JCL proc for each server with the server name coded on the `PARMS=` on the PROC statement, we could start each server with `/S` *<proc>*. If we were using a common proc, then `/S <proc>,PARMS='`*<server_name>*`'`

8 The `batchManager` client is network based and does not use WOLA. That does not require the Angel Process.

We logged into PuTTY using the `USER1` ID. Recall that `USER1` was the ID we granted `READ` to for WOLA-related `CBIND` profile. That is what gives `USER1` the authority to issue the `batchManagerZos` command to connect with WOLA into the dispatcher server[9].

We set the following environment variables:

```
export JAVA_HOME=/shared/java/J8.0_64
```

The `batchManager` command line client is Java-based, and this provides the shell knowledge of the location for the 64-bit Java installation.

The `batchManagerZos` command line client is *not* Java-based, so `JAVA_HOME` is not necessary for that.

```
export JVM_ARGS="-Djavax.net.ssl.trustStore=/u/user1/key.jks"
```

We explained earlier that the design of this document was to use "basic" security so focus could be more clearly maintained on the key Java batch elements. If you look back in the `server.xml` examples, you'll see the "basic" security defined with the `<keyStore>`, `<basicRegistry>`, and `<authorization-roles>` elements. This export of `JVM_ARGS` is related to the `<keyStore>` basic security element.

The `batchManager` REST-based command line client is going to use the HTTPS ("SSL") port of the Liberty z/OS server. That means the client must have access to the file that contains the certificate used to validate the server certificate that is presented.

The "basic" SSL setup with `<keyStore>` creates a simple, self-signed certificate. It stores that certificate in a file called `key.jks`, which is located under the `/resources/security` directory for the server. To make SSL work we need to give the `batchManager` client access to that file.

The `JVM_ARGS` you see above is providing the `batchManager` client access to the file, but that file is *not* located under the server's `/resources/security` directory. We *copied* that file to the `/u/user1` directory and gave it permissions `777`.

Why? Because it was easier than granting the `USER1` ID read access all the way down the server's path to the `key.jks` file. The server's directory structure was owned by the `JSRSERV` ID, and the `USER1` ID was considered "other" with no permissions. To accomplish the setup of SSL it was easier to simply copy the `key.jks` file to some location easily accessible by the `USER1` ID.

In "the real world" you would not do this. In a real-world setting you would not use "basic" security, you would use SAF-based keyrings for the holding of certificates that have been properly signed by a valid Certificate Authority. Here we are demonstrating *functionality*, and copying out the `key.jks` file was the easy way to accomplish that.

We were ready to submit our first job. We changed directories to:

```
cd /shared/zWebSphere/Liberty/V16004/bin
```

That's where the `batchManager` command line client resided.

We submitted the following command[10]:

```
./batchManager submit --batchManager=localhost:25443
    --user=Fred --password=fredpwd --applicationName=SleepyBatchletSample-1.0
                                         --jobXMLName=sleepy-batchlet.xml --wait
```

We saw[11]:

9 If we were illustrating *just* batchManager (the REST-based command line client), then the ID we logged in with would not matter as much.
10 That is a one-line command. It is broken across lines here.
11 The instance and execution number here was 56. We ran a lot of different tests prior to capturing this output for the document.

```
[2017/01/24 12:07:41.110 -0500] CWWKY0101I: Job  with instance ID 56 has been
submitted.
[2017/01/24 12:07:41.112 -0500] CWWKY0106I: JobInstance:
{"jobName":"","instanceId":56,"appName":"SleepyBatchletSample-
1.0#SleepyBatchletSample-
1.0.war","submitter":"Fred","batchStatus":"STARTING","jobXMLName":"sleepy-
batchlet.xml","instanceState":"JMS_QUEUED","lastUpdatedTime":"2017/01/24
17:07:37.744 +0000"}
[2017/01/24 12:08:11.261 -0500] CWWKY0105I: Job  with instance ID 56 has
finished. Batch status: COMPLETED. Exit status: COMPLETED
[2017/01/24 12:08:11.261 -0500] CWWKY0107I: JobExecution:{"jobName":"sleepy-
batchlet","executionId":56,"instanceId":56,"batchStatus":"COMPLETED","exitSta
tus":"COMPLETED","createTime":"2017/01/24 17:07:37.686
+0000","endTime":"2017/01/24 17:07:58.312
+0000","lastUpdatedTime":"2017/01/24 17:07:58.312
+0000","startTime":"2017/01/24 17:07:42.966 +0000","jobParameters":
{},"restUrl":"https://192.168.17.219:27443/ibm/api/batch","serverId":"localho
st//shared/jsrhome/JSREXEC2","logpath":"/shared/jsrhome/servers/JSREXEC2/logs
/joblogs/sleepy-batchlet/2017-01-
24/instance.56/execution.56/","stepExecutions":
[{"stepExecutionId":78,"stepName":"step1","batchStatus":"COMPLETED","exitStat
us":"SleepyBatchlet:i=15;stopRequested=false","stepExecution":"https://localh
ost:25443/ibm/api/batch/jobexecutions/56/stepexecutions/step1"}]}
USER1:/shared/zWebSphere/Liberty/V16004/bin:>
```

It ended up being executed in the `JSREXEC2` server even though it was deployed in both `JSREXEC1` and `JSREXEC2`. It just so happened `JSREXEC2` picked it up first.

Next, we looked under the `JSRMON` server directory where the monitoring MDB application writes its output. We saw this:

```
/shared/jsrhome/servers/JSRMON/JobLogEvents
                 /sleepy-batchlet/2017-01-24/instance.56/execution.56/part1.log
```

The contents of the `part1.log` file were:

```
[1/24/17 17:07:42:965 GMT] com.ibm.ws.batch.JobLogger
=============================================================
Started invoking execution for a job
 JobInstance id = 56
 JobExecution id = 56
 Job Name = sleepy-batchlet
 Job Parameters = {}
=============================================================

[1/24/17 17:07:42:997 GMT] com.ibm.ws.batch.JobLogger
CWWKY0009I: Job sleepy-batchlet started for job instance 56 and job execution
56.
[1/24/17 17:07:43:204 GMT] com.ibm.ws.batch.JobLogger
=============================================================
For step name = step1
 New top-level step execution id = 78
=============================================================

[1/24/17 17:07:43:247 GMT] com.ibm.ws.batch.JobLogger
CWWKY0018I: Step step1 started for job instance 56 and job execution 56.
[1/24/17 17:07:58:311 GMT] com.ibm.ws.batch.JobLogger
CWWKY0020I: Step step1 ended with batch status COMPLETED and exit status
SleepyBatchlet:i=15;stopRequested=false for job instance 56 and job execution
56.
```

```
[1/24/17 17:07:58:335 GMT] com.ibm.ws.batch.JobLogger
CWWKY0010I: Job sleepy-batchlet ended with batch status COMPLETED and exit
status COMPLETED for job instance 56 and job execution 56.
[1/24/17 17:07:58:336 GMT] com.ibm.ws.batch.JobLogger
==========================================================
Completed invoking execution for a job
 JobInstance id = 56
 JobExecution id = 56
 Job Name = sleepy-batchlet
 Job Parameters = {}
 Job Batch Status = COMPLETED, Job Exit Status = COMPLETED
==========================================================
```

The presence of that file indicated that the `JSREXEC2` server published its events to the topic, and the MDB application running in the `JSRMON` server successfully subscribed and pulled the message from the topic and wrote it to the file system[12].

Next, we submitted the BonusPayout job submission. This was deployed in JSREXEC2 only, so we expect it to run there. The command was:

```
./batchManager submit --batchManager=localhost:25443
             --user=Fred --password=fredpwd --applicationName=BonusPayout-1.0
       --jobXMLName=SimpleBonusPayoutJob.xml --jobParameter=dsJNDI=jdbc/bonus
                                --jobParameter=tableName=BONUSDB.ACCOUNT --wait
```

We saw:

```
[2017/01/24 12:21:31.358 -0500] CWWKY0101I: Job  with instance ID 57 has been
submitted.
[2017/01/24 12:21:31.361 -0500] CWWKY0106I: JobInstance:
{"jobName":"","instanceId":57,"appName":"BonusPayout-1.0#BonusPayout-
1.0.war","submitter":"Fred","batchStatus":"STARTING","jobXMLName":"SimpleBonu
sPayoutJob.xml","instanceState":"JMS_CONSUMED","lastUpdatedTime":"2017/01/24
17:21:31.127 +0000"}
[2017/01/24 12:22:01.565 -0500] CWWKY0105I: Job  with instance ID 57 has
finished. Batch status: COMPLETED. Exit status: COMPLETED
[2017/01/24 12:22:01.566 -0500] CWWKY0107I: JobExecution:
{"jobName":"SimpleBonusPayoutJob","executionId":57,"instanceId":57,"batchStat
us":"COMPLETED","exitStatus":"COMPLETED","createTime":"2017/01/24
17:21:31.062 +0000","endTime":"2017/01/24 17:21:36.625
+0000","lastUpdatedTime":"2017/01/24 17:21:36.625
+0000","startTime":"2017/01/24 17:21:31.466 +0000","jobParameters":
{"tableName":"BONUSDB.ACCOUNT","dsJNDI":"jdbc/bonus"},"restUrl":"https://192.
168.17.219:27443/ibm/api/batch","serverId":"localhost//shared/jsrhome/JSREXEC
2","logpath":"/shared/jsrhome/servers/JSREXEC2/logs/joblogs/SimpleBonusPayout
Job/2017-01-24/instance.57/execution.57/","stepExecutions":
[{"stepExecutionId":79,"stepName":"generate","batchStatus":"COMPLETED","exitS
tatus":"COMPLETED","stepExecution":"https://localhost:25443/ibm/api/batch/job
executions/57/stepexecutions/generate"},
{"stepExecutionId":80,"stepName":"addBonus","batchStatus":"COMPLETED","exitSt
atus":"COMPLETED","stepExecution":"https://localhost:25443/ibm/api/batch/jobe
xecutions/57/stepexecutions/addBonus"}]}
USER1:/shared/zWebSphere/Liberty/V16004/bin:>
```

12 The BonusPayout application, which we submitted next, was deployed in `JSREXEC2` only, so we knew that job would run there. So in a sense it's unfortunate the same server (`JSREXEC2`) picked up this SleepyBatchlet job for it means in this illustration we have not validated `JSREXEC1` is publishing its job events. But from previous tests we saw that it was, so we knew it was working. If you're running this on your system, make sure to validate all your servers successfully publish events. In this example, shutting down `JSREXEC2` and submitting SleepyBatchlet would have insured it ran in `JSREXEC1`.

Back to the `JSRMON` server directory to look at the job log output from the monitoring application. We saw:

```
/shared/jsrhome/servers/JSRMON/JobLogEvents
           /SimpleBonusPayoutJob/2017-01-24/instance.57/execution.57/part1.log
```

The contents of the `part1.log` file were:

```
[1/24/17 17:21:31:465 GMT] com.ibm.ws.batch.JobLogger
==========================================================
Started invoking execution for a job
 JobInstance id = 57
 JobExecution id = 57
 Job Name = SimpleBonusPayoutJob
 Job Parameters = {tableName=BONUSDB.ACCOUNT, dsJNDI=jdbc/bonus}
==========================================================

[1/24/17 17:21:31:483 GMT] com.ibm.ws.batch.JobLogger
CWWKY0009I: Job SimpleBonusPayoutJob started for job instance 57 and job
execution 57.
[1/24/17 17:21:31:504 GMT] com.ibm.ws.batch.JobLogger
==========================================================
For step name = generate
 New top-level step execution id = 79
==========================================================

[1/24/17 17:21:31:566 GMT] com.ibm.ws.batch.JobLogger
CWWKY0018I: Step generate started for job instance 57 and job execution 57.
[1/24/17 17:21:31:751 GMT] BonusPayout
In GenerateDataBatchlet, using account code = CHK
[1/24/17 17:21:31:782 GMT] com.ibm.ws.batch.JobLogger
CWWKY0020I: Step generate ended with batch status COMPLETED and exit status
COMPLETED for job instance 57 and job execution 57.
[1/24/17 17:21:31:858 GMT] com.ibm.ws.batch.JobLogger
==========================================================
For step name = addBonus
 New top-level step execution id = 80
==========================================================

[1/24/17 17:21:31:876 GMT] com.ibm.ws.batch.JobLogger
CWWKY0018I: Step addBonus started for job instance 57 and job execution 57.
[1/24/17 17:21:36:625 GMT] com.ibm.ws.batch.JobLogger
CWWKY0020I: Step addBonus ended with batch status COMPLETED and exit status
COMPLETED for job instance 57 and job execution 57.
[1/24/17 17:21:36:639 GMT] com.ibm.ws.batch.JobLogger
CWWKY0010I: Job SimpleBonusPayoutJob ended with batch status COMPLETED and
exit status COMPLETED for job instance 57 and job execution 57.
[1/24/17 17:21:36:640 GMT] com.ibm.ws.batch.JobLogger
==========================================================
Completed invoking execution for a job
 JobInstance id = 57
 JobExecution id = 57
 Job Name = SimpleBonusPayoutJob
 Job Parameters = {tableName=BONUSDB.ACCOUNT, dsJNDI=jdbc/bonus}
 Job Batch Status = COMPLETED, Job Exit Status = COMPLETED
==========================================================
```

Submit using batchManagerZos

This is a different command line utility, so the job submission syntax is different. But every other part of the infrastructure we illustrated remains the same.

We changed directories to:

```
cd /shared/zWebSphere/Liberty/V16004/lib/native/zos/s390x
```

That's where the `batchManagerZos` client program resided.

We submitted the SleepyBatchlet job:

```
./batchManagerZos submit --batchManager=LIBERTY+BATCH+MANAGER
    --applicationName=SleepyBatchletSample-1.0 --jobXMLName=sleepy-batchlet.xml --wait
```

Note: There is no ID or password specified on that command. The ID was the ID we logged into the shell with, which was `USER1`. The `CBIND` profile based on the three part name we used was checked to see if `USER1` had `READ`, which it did. Therefore, this command was allowed.

We saw[13]:

```
INFO: CWWKY0101I: Job  with instance ID 58 has been submitted.

INFO: CWWKY0106I: JobInstance:
{"jobName":"","instanceId":58,"appName":"SleepyBatchletSample-
1.0#SleepyBatchletSample-
1.0.war","submitter":"","batchStatus":"STARTING","jobXMLName":"sleepy-
batchlet.xml","instanceState":"JMS_CONSUMED","lastUpdatedTime":"2017/01/24
17:37:00.177 +0000"}

INFO: CWWKY0105I: Job  with instance ID 58 has finished. Batch status:
COMPLETED. Exit status: COMPLETED

INFO: CWWKY0107I: JobExecution:{"jobName":"sleepy-
batchlet","executionId":58,"instanceId":58,"batchStatus":"COMPLETED","exitSta
tus":"COMPLETED","createTime":"2017/01/24 17:37:00.117
+0000","endTime":"2017/01/24 17:37:15.416
+0000","lastUpdatedTime":"2017/01/24 17:37:15.416
+0000","startTime":"2017/01/24 17:37:00.325 +0000","jobParameters":
{"com.ibm.ws.batch.submitter.jobId":"STC00304","com.ibm.ws.batch.submitter.jo
bName":"USER12
"},"restUrl":"https://192.168.17.219:27443/ibm/api/batch","serverId":"localho
st//shared/jsrhome/JSREXEC2","logpath":"/shared/jsrhome/servers/JSREXEC2/logs
/joblogs/sleepy-batchlet/2017-01-
24/instance.58/execution.58/","stepExecutions":
[{"stepExecutionId":81,"stepName":"step1","batchStatus":"COMPLETED","exitStat
us":"SleepyBatchlet:i=15;stopRequested=false"}]}

USER1:/shared/zWebSphere/Liberty/V16004/lib/native/zos/s390x:>
```

We looked under the `JSRMON` server directory and saw that there was a directory structure for this instance of the job. The contents of the `part1.log` file was essentially the same as we showed for the `batchManager` example. So we we won't show it again here.

Next, we submitted the BonusPayout job with this command:

```
./batchManagerZos submit --batchManager=LIBERTY+BATCH+MANAGER
      --applicationName=BonusPayout-1.0 --jobXMLName=SimpleBonusPayoutJob.xml
                                           --jobParameter=dsJNDI=jdbc/bonus
                              --jobParameter=tableName=BONUSDB.ACCOUNT --wait
```

We saw:

```
INFO: CWWKY0101I: Job  with instance ID 59 has been submitted.

INFO: CWWKY0106I: JobInstance:
{"jobName":"","instanceId":59,"appName":"BonusPayout-1.0#BonusPayout-
```

13 It ran in `JSREXEC2` again, which means in this case we still have not seen `JSREXEC1` run a job and publish events. As we mentioned earlier, in other tests we did see it work, so we know our infrastructure as okay. If you wanted to prove the `JSREXEC1` server was working properly you would stop the `JSREXEC2` server and re-submit SleepyBatchlet. With only `JSREXEC1` listening on the submission queue it would pick up the job and run it.

```
1.0.war","submitter":"","batchStatus":"STARTING","jobXMLName":"SimpleBonusPay
outJob.xml","instanceState":"JMS_CONSUMED","lastUpdatedTime":"2017/01/24
17:46:48.125 +0000"}
INFO: CWWKY0105I: Job  with instance ID 59 has finished. Batch status:
COMPLETED. Exit status: COMPLETED
INFO: CWWKY0107I: JobExecution:
{"jobName":"SimpleBonusPayoutJob","executionId":59,"instanceId":59,"batchStat
us":"COMPLETED","exitStatus":"COMPLETED","createTime":"2017/01/24
17:46:48.067 +0000","endTime":"2017/01/24 17:46:50.059
+0000","lastUpdatedTime":"2017/01/24 17:46:50.059
+0000","startTime":"2017/01/24 17:46:48.279 +0000","jobParameters":
{"tableName":"BONUSDB.ACCOUNT","com.ibm.ws.batch.submitter.jobId":"STC00304",
"com.ibm.ws.batch.submitter.jobName":"USER13
","dsJNDI":"jdbc/bonus"},"restUrl":"https://192.168.17.219:27443/ibm/api/batc
h","serverId":"localhost//shared/jsrhome/JSREXEC2","logpath":"/shared/jsrhome
/servers/JSREXEC2/logs/joblogs/SimpleBonusPayoutJob/2017-01-
24/instance.59/execution.59/","stepExecutions":
[{"stepExecutionId":82,"stepName":"generate","batchStatus":"COMPLETED","exitS
tatus":"COMPLETED"},
{"stepExecutionId":83,"stepName":"addBonus","batchStatus":"COMPLETED","exitSt
atus":"COMPLETED"}]}
USER1:/shared/zWebSphere/Liberty/V16004/lib/native/zos/s390x:>
```

We expected this to run in `JSREXEC2` since that's the only place the BonusPayout sample application is deployed.

We looked under the `JSRMON` server directory and saw that there was a directory structure for this instance of the job. The contents of the `part1.log` file was essentially the same as we showed for the `batchManager` example. So we we won't show it again here.

Submit using batchManagerZos and --queueManagerName

In the previous examples we showed use of the `--wait` parameter. This told the command line client, either `batchManager` or `batchManagerZos`, to hold off returning to the command prompt until the job status had been resolved (failed or completed). It did this by polling the dispatcher server to check for the status of the job. The dispatcher, in turn, checked the job respository database table each time it was polled. This involves some degree of overhead.

When using `batchManagerZos` and batch job events we have another option for determining when a job status is resolved. This involves having `batchManagerZos` watch for the batch job event indicating the job status resolution. This involves no polling, and thus no repeated calls to the database to check for status.

To use this we exported a `STEPLIB` environment variable[14]:

```
export STEPLIB='MQ800.SCSQAUTH'
```

Without this we saw "module not found" messages.

Then we issued the following command to submit the SleepyBatchlet job:

```
./batchManagerZos submit --batchManager=LIBERTY+BATCH+MANAGER
  --applicationName=SleepyBatchletSample-1.0 --jobXMLName=sleepy-batchlet.xml
                                      --wait --queueManagerName=MQS1
```

We saw:

```
INFO: CWWKY0101I: Job  with instance ID 60 has been submitted.
INFO: CWWKY0106I: JobInstance:
{"jobName":"","instanceId":60,"appName":"SleepyBatchletSample-
```

14 Or `export STEPLIB=$STEPLIB:'MQ800.SCSQAUTH'` if you wanted to preserve any existing `STEPLIB` values.

```
1.0#SleepyBatchletSample-
1.0.war","submitter":"","batchStatus":"STARTING","jobXMLName":"sleepy-
batchlet.xml","instanceState":"JMS_CONSUMED","lastUpdatedTime":"2017/01/24
18:02:40.968 +0000"}

INFO: CWWKY0105I: Job  with instance ID 60 has finished. Batch status:
COMPLETED. Exit status: COMPLETED

INFO: CWWKY0107I: JobExecution:{"jobName":"sleepy-
batchlet","executionId":60,"instanceId":60,"batchStatus":"COMPLETED","exitSta
tus":"COMPLETED","createTime":"2017/01/24 18:02:40.924
+0000","endTime":"2017/01/24 18:02:56.171
+0000","lastUpdatedTime":"2017/01/24 18:02:56.171
+0000","startTime":"2017/01/24 18:02:41.103 +0000","jobParameters":
{"com.ibm.ws.batch.submitter.jobId":"STC00304","com_ibm_ws_batch_events_corre
lationId":"d1feb34cd97cd24b0400016f0d5d9f87c0a811db2abdf676","com.ibm.ws.batc
h.submitter.jobName":"USER14
"},"restUrl":"https://192.168.17.219:27443/ibm/api/batch","serverId":"localho
st//shared/jsrhome/JSREXEC2","logpath":"/shared/jsrhome/servers/JSREXEC2/logs
/joblogs/sleepy-batchlet/2017-01-
24/instance.60/execution.60/","stepExecutions":
[{"stepExecutionId":84,"stepName":"step1","batchStatus":"COMPLETED","exitStat
us":"SleepyBatchlet:i=15;stopRequested=false"}]}

USER1:/shared/zWebSphere/Liberty/V16004/lib/native/zos/s390x:>
```

Note the correlation ID we highlighted in yellow. This is what `batchManagerZos` uses to know when the job it submitted has completed. It looks for a batch event with that correlator ID, and when it sees the message with that correlator it pulls the message and determines the status.

We won't show submitting the BonusPayout job using `--queueManagerName` because it is just like what we showed earlier, except with `--queueManagerName=MQS1` on the end of the job submission command. The key point being: the rest of the batchManagerZos command line job submission syntax is the same; the only difference is the addition of `--queueManagerName=` to indicate the MQ Queue Manager to connect to.

Document Change History

Check the date in the footer of the document for the version of the document.

January 25, 2017	Original document at time of Techdoc creation
January 27, 2017	Updated to better show the use of JDBC Type 4 authentication aliases.
June 6, 2017	Added notes about using MQ BINDINGS mode -- "If you use BINDINGS mode, you must add the `zosTransaction-1.0` feature to the feature list, as well as grant the server `READ` to the `BBG.AUTHMOD.BBGZSAFM.TXRRS SERVER` profile. The Angel must be started prior to the server starting."

End of WP102544