redhat®
TRAINING

CHAPTER 13

CONFIGURATION OF BATCH SUBSYSTEM

General description	
Goal	Configure the batch subsystem to run batch jobs written with JBeret. • Describe the
Goals	functionality of batch jobs (JSR 352) and the JBeret implementation, supported by the JBoss EAP. • Configure the batch subsystem to support batch job execution. • Exploration of batch jobs (and
sections	quiz) • Configuration of the batch subsystem (and exercise Guided)
Laboratory work	• Configuration of batch subsystem

Batch Job Scanning

Goals

After completing this section, students should be able to do the following:

- **Describe the functionality of batch jobs and the implementation of JBeret.**

batch jobs

EAP 7 supports the new Java specification, JSR 352: Batch Applications. Batch applications are applications that can be used to customize and create batch jobs. A batch job refers to a series of tasks that are executed periodically without requiring interactive input. Batch jobs can be particularly useful for running resource-intensive tasks during times of low server usage. For example, an organization's point-of-sale system might generate a daily inventory report using a batch job that runs only when all stores are closed for the day, saving server resources used to process orders from being used for generate reports.

Batch applications are made up of a series of steps that can be executed sequentially or concurrently using separate threads. Each step is fragment-oriented or task-oriented. A fragment-oriented step refers to a task used to process information from a source and generally runs longer. Due to its nature of generally running for a long time, it is possible to set checkpoints to restart interrupted runs without losing progress.

By comparison, a task-oriented step runs quickly, with tasks like deleting old files on a file system or deleting an email, rather than large volumes of data processing and transformation, which is typical in fragment-oriented steps.

A fragment-oriented step has three major parts: the reader, the processor, and the writer. The reader is responsible for reading the data. The processor performs a process or an action on the data, such as filtering or transforming the data. The writer is then responsible for updating or writing the data change.

Batch jobs are defined using the Job Specification Language (JSL), which is an XML-based language that is part of the JSR-352 standard. Each job file is created with a unique name and placed in the META-INF/batch-jobs directory. The following is an example of an XML job definition file:

```
<job id="sampleJob"
xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="1.0">
        <step id="sampleStep" >
                <chunk item-count="3">
                        <reader ref="reader"/>
                        <processor ref="processor"/> <writer
                        ref="writer"/> </chunk>


        </step>
</job>
```

Based on this example, a single job named sampleJob has a single step, sampleStep, that contains a single fragment. Inside the fragment is a reader,

a processor and a writer that must be implemented in the application code of the batch. Implementing and developing batch applications is outside the scope of this course.

Steps independent of each other can be executed concurrently to improve performance and reduce batch execution times. This can be accomplished in the job specification through the use of splits and flows.

A flow is a set of steps that is executed sequentially. Steps in a flow cannot refer to steps outside of the flow. The flow proceeds to the next element when its last step is executed successfully.

A split is a set of flows that run concurrently. Each stream runs on a separate thread. The split moves on to the next element when all of its flows run successfully.

The batch subsystem also offers the concept of partitioning that allows processing of large data sets in parallel. This is generally used in fragment-oriented steps. For example, if a batch job needs to process an input file with one thousand (1000) records, you can split processing into two partitions, where the first processing logs from 1 to 500 and the second processing logs from 501 to 1000. To each partition it can be assigned a number of threads from the batch subsystem thread pool, and the batch runtime will process the records in parallel.

The batch subsystem

Batch jobs can be configured using the batch subsystem. In each EAP 7, the subsystem is based on JBeret's open source implementation of JSR 352. Some important functions provided by JBeret are:

- Components to read and write common file formats (CSV, XML, JSON)
- Components for accessing external resources, such as databases and message queues.
- Annotations to inject custom scopes (@JobScoped, @StepScoped, @PartitionScoped)
- Support for storing statistics in multiple places, such as databases (relationship and document-oriented), in-memory, and Infinispan

note

For EAP, only in-memory and database statistics storage is available.

The following represents the default configuration of the batch subsystem:

Job repository JBeret
: uses the job repository to configure where all log information about jobs is stored, such as the batch execution status, when the batch was started, and when the batch was completed. Due to the nature of the batch, any server failures or crashes must be stored and can be referenced to identify problems. JBeret can restart any batch that was interrupted. EAP provides a configuration

in memory by default, but if a server goes down, the statistics will be lost. Alternatively, JBeret supports database persistence.

Thread Pool The

max-threads element defines how many threads the batch subsystem can use to process jobs. Notice that two thread pool threads are reserved for partitioning jobs.

Quiz: Batch Job Exploration

Choose the correct answer to the following questions:

1. What type of step is used for long-running processes? (Choose one option).

to. **Fragment-oriented step**
b. **task-oriented pace**

2. Which of the following comprises the basis of fragment-oriented steps? (Choose three options).

to. **Reader**
b. **Processor**
Analyzer
CD **Writer**
and. **Editor**
F. **Action**

3. Which of the following is a good batch candidate? (Choose one option).

to. **An organization must create a PDF file containing a report of all items sold today. b. A user wants to upload new pictures from his vacation to a blog.**
c. **An organization wants to send an email to all administrators every time an error is logged.**
d. **An organization must update inventory immediately after each purchase.**

4. What implementation is the EAP 7 batch subsystem based on? (Choose one option).

to. **JSR 352**
b. **Spring Batch**
c. **JBeret**
d. **Easy Batch**
and. **None of the above.**

Solution

Choose the correct answer to the following questions:

1. What type of step is used for long-running processes? (Choose one option).

- to. **Fragment-oriented step**
- b. task-oriented pace

2. Which of the following comprises the basis of fragment-oriented steps? (Choose three options).

- to. **Reader**
- b. **Processor**
- c. Analyzer
- d. **Writer**
- and. Editor
- F. Action

3. Which of the following is a good batch candidate? (Choose one option).

to. An organization must create a PDF file containing a report of all items sold today. b. A user wants to upload new pictures from his vacation to a blog.
- c. An organization wants to send an email to all administrators every time an error is logged.
- d. An organization must update inventory immediately after each purchase.

4. What implementation is the EAP 7 batch subsystem based on? (Choose one option).

JSR 352
- ab Spring Batch
- c. **JBeret**
- d. Easy Batch
- and. None of the above.

Batch Subsystem Configuration

Goals

After completing this section, students should be able to do the following:

- **Configure the batch subsystem to support batch job execution.**

Batch Subsystem Configuration

Batch jobs are defined using an XML file, which comes with the application JAR or WAR file inside the META-INF/batch-jobs directory. Each job has a unique name associated with it. Jobs defined in these XML files can be deployed and controlled by the application itself using an API exposed by the batch subsystem, as well as using the EAP CLI.

```
<subsystem xmlns="urn:jboss:domain:batch-jberet:1.0">
        <default-job-repository name="in-memory"/>          1
        <default-thread-pool name="batch"/>            2
        <job-repository name="in-memory">          3
                <in-memory/> </4
        job-repository> <thread-
        pool name="batch">
                <max-threads count="10"/> </        5
        thread-pool> </
subsystem>
```

1 **JBeret uses the default-job-repository element to refer to the default job-repository configuration, defined within the subsystem. This setup allows an administrator to set up multiple working repositories in one place and activate them as needed.**

2 **The default-thread-pool element refers to which thread pool configuration JBeret should use to execute each step. Similar to the default-job repository, an administrator can use this approach to configure multiple sets of threads that can be activated as needed.**

3 **JBeret uses the job-repository element to configure where information about job execution is stored, such as the execution status of the batch, when it was started, and when it was finished. Because a batch job can have many steps, all failures during step execution must be stored and can be referenced to identify problems. JBeret can restart any interrupted or failed batch job due to invalid data.**

4 **The in-memory element configures JBeret to store batch registration information in EAP memory. Alternatively, it also supports a jdbc-job-repository element to store batch records in a database, which is configured as a data source in EAP.**

5 **The max-threads element defines how many threads the batch subsystem can use to process threads. Notice that two thread pool threads are reserved for partitioning jobs. The value of max-threads must be greater than three.**

The runtime environment configuration in the batch subsystem consists of working repositories and thread pools. A job repository stores batch job execution details and can be one of two types:

- in-memory: an in-memory job repository stores job details files executed on the server in RAM. Data is lost if the server is shut down or restarted. This is the default working repository. EAP 7 comes with a preconfigured in-memory working repository called in-memory.

- JDBC: A JDBC job repository stores the details of batch jobs executed on the server in a database. Data is persistent across reboots and shutdowns of the server. A JDBC repository job is configured within the <jdbc-job-repository> section of the bundle subsystem. References a data source attribute indicating a valid EAP 7 data source. The data source must be configured in the EAP data source system.

The batch subsystem can be configured with thread-pools, which maintain a set of threads that batch jobs can reuse. The thread pool attribute max-threads defines the maximum number of threads that batch jobs can use while the jobs are running. The keepalive-time attribute configures the duration for which the thread should remain idle when there are no jobs to run. EAP defines a thread pool with 10 max-threads and a keepalive-time of 30 seconds by default.

You can use the EAP CLI or the EAP management console to configure job repositories, thread pools, and the batch job runtime environment. You can use the EAP CLI to start, stop, and restart batch jobs and view the results of job runs.

The batch subsystem can be configured using the administration console. For a standalone server, navigate to Configuration > Subsystem > Bundle and click View to view the bundle subsystem configuration page.

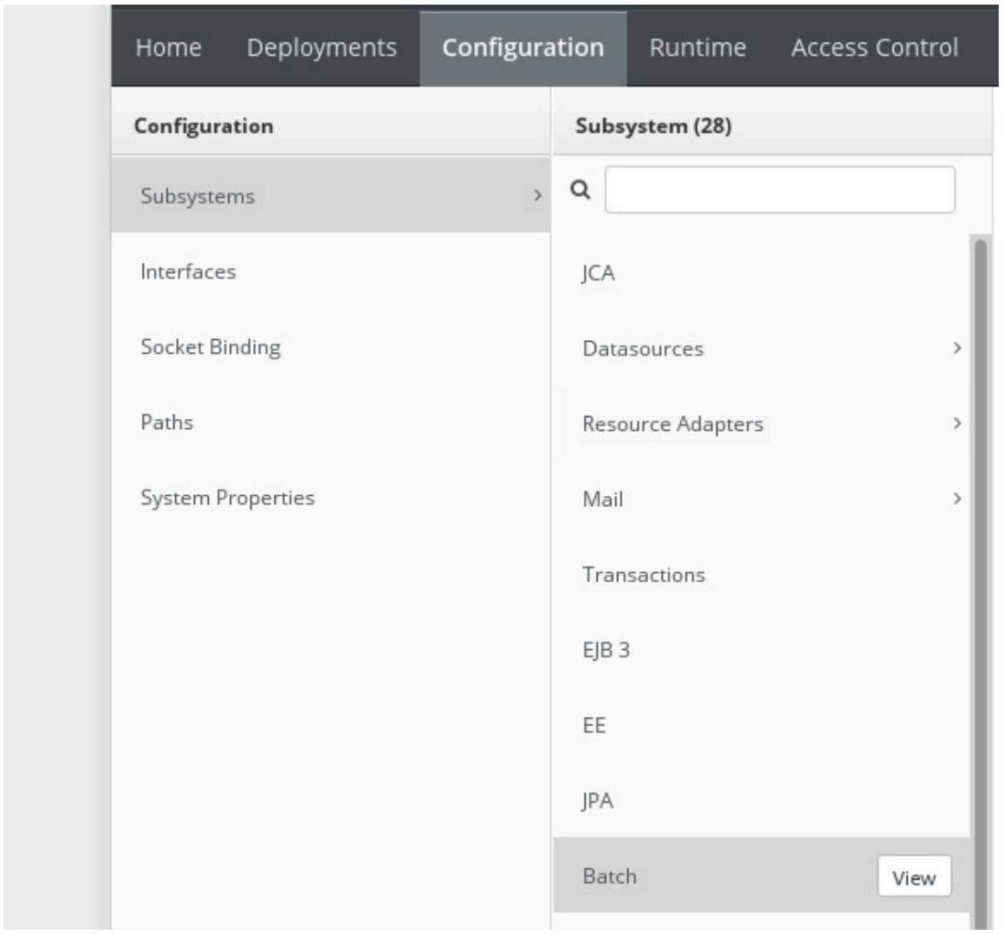


Figure 13.1: Batch subsystem for the standalone server

For an EAP managed domain, navigate to Configuration > Profiles > <profile name> > Subsystem > Bundle and click View to view the bundle subsystem configuration page.

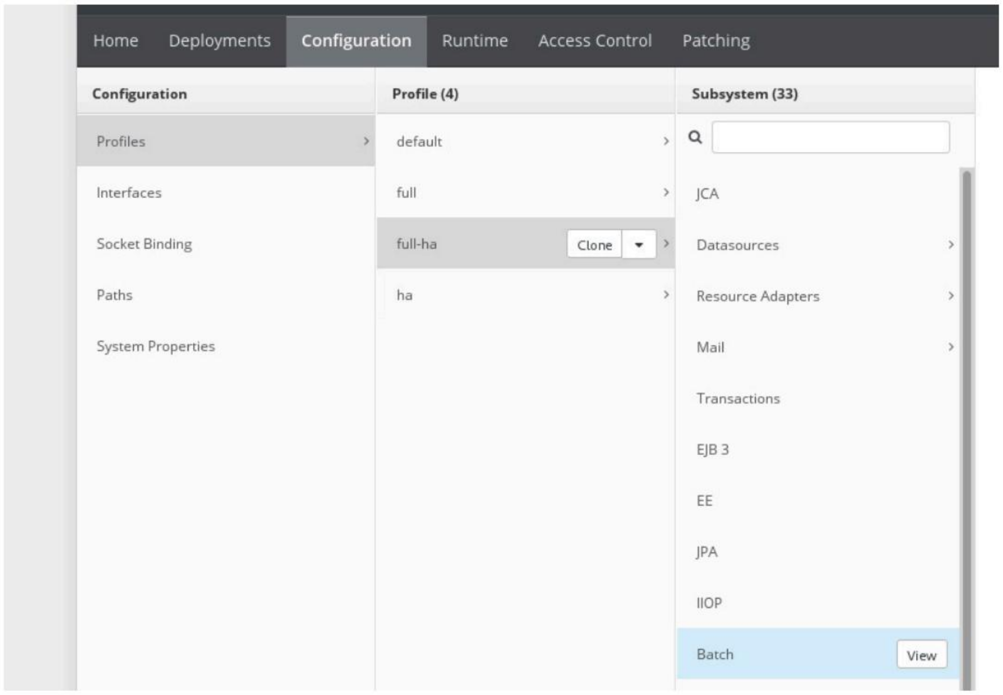
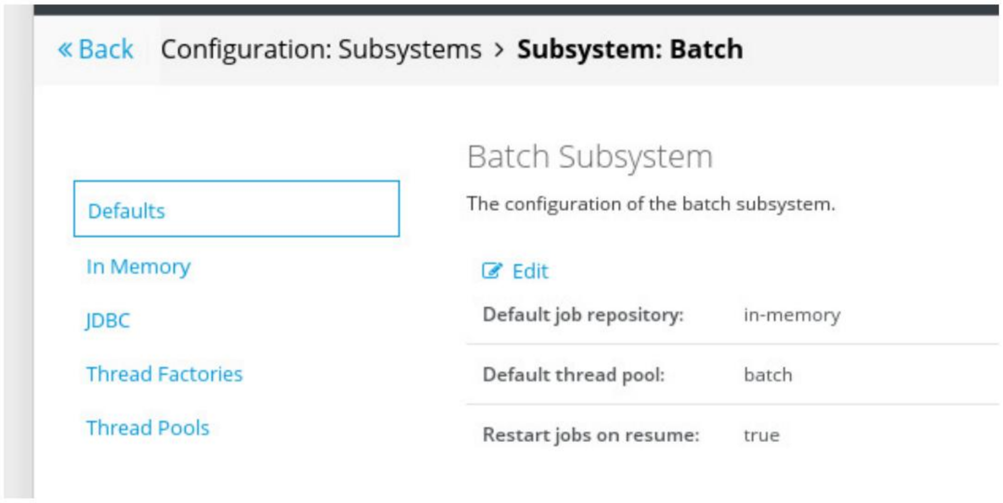


Figure 13.2: Packages subsystem for a managed domain

The default thread set and repository for batch jobs, job repositories, thread sets and thread factories can be configured using the links on the left sidebar of the batch subsystem configuration page (both for standalone server as for managed domain mode).



The configuration of the batch subsystem.

Default job repository:	in-memory
Default thread pool:	batch
Restart jobs on resume:	true

Figure 13.3: Configuration of the batch subsystem

The batch subsystem can also be configured using the EAP CLI. The EAP CLI has the added benefit of allowing an EAP administrator to start, stop, and restart jobs. You can also view job execution statistics using the EAP CLI.

To create an in-memory working repository named custom-mem-repo on a standalone server, run the following command:

```
[standalone@127.0.0.1:9990 /] /subsystem=batch-jberet/\ in-memory-job-
repository=custom-mem-repo:add()
```

To start a batch job named job1, which is defined in an application named test-batch-app.war on a standalone server, run the following command:

```
[standalone@127.0.0.1:9990 /] /deployment=test-batch-app.war/subsystem=\ batch-jberet:start-
job(job-xml-name=job1)
```

To stop a batch job on a standalone server, run the following command:

```
[standalone@127.0.0.1:9990 /] /deployment=test-batch-app.war/subsystem=\ batch-jberet:stop-
job(job-xml-name=job1)
```

To restart a batch job on a standalone server, run the following command:

```
[standalone@127.0.0.1:9990 /] /deployment=test-batch-app.war/subsystem=\ batch-
jberet:restart-job(job-xml-name=job1)
```

To view statistics about the execution of job1 on a standalone server, run the following command:

```
[standalone@127.0.0.1:9990 /] /deployment=test-batch-app.war/\ subsystem=batch-
jberet/job=job1:read-resource(recursive=true,include-runtime=true)
```

To create a thread-pool named custom-pool with a max-threads value of 15 and a keepalive-time value of 10 seconds that the batch subsystem will use to run jobs, run the following command:

```
[standalone@127.0.0.1:9990 /] /subsystem=batch-jberet/\ thread-
pool=custom-pool:add(max-threads=15,keepalive-time={unit=SECONDS,time=10})
```

To view the thread-pool statistics, run the following command:

```
[standalone@127.0.0.1:9990 /] /subsystem=batch-jberet/\ thread-
pool=custom-pool:read-resource(include-runtime=true,recursive=true)
```

To change the default thread-pool that the batch subsystem will use to run batch jobs, run the following command:

```
[standalone@127.0.0.1:9990 /] /subsystem=batch-jberet\ :write-
attribute(name=default-thread-pool,value=custom-pool)
```

To change the default job repository that the batch subsystem will use to store batch job execution details, run the following command:

```
[standalone@127.0.0.1:9990 /] /subsystem=batch-jberet\ :write-
attribute(name=default-job-repository,value=custom-mem-repo)
```

For an EAP managed domain consisting of multiple hosts and servers, the job repository and thread set configuration is similar to the standalone server. The CLI commands to configure the batch subsystem will be in the /profile=<profile_name>/subsystem=batch-jberet namespace.

The main difference between running jobs and viewing job statistics in a managed domain versus a standalone server is the fact that you must provide the name of the server where you want the jobs to run. This is because there can be multiple hosts and servers in a managed domain and the batch application WAR file is deployed at the server group level, so it can be running on multiple servers. To avoid duplication of job execution, provide the server name on which the job should be started, stopped, or restarted. For example, to start a job on server-one running on host1:

```
[domain@172.25.250.254:9990 /] /host=host1/server=server-one\ /deployment=test-
batch-app.war/subsystem=\ batch-jberet:start-job(job-
xml-name=job1 )
```

Similarly, to view job statistics in a managed domain:

```
[domain@172.25.250.254:9990 /] /host=host1/server=server-one\ /deployment=test-
batch-app.war/subsystem=\ batch-jberet/job=job1:\
read-resource(include-
runtime=true,recursive=true)
```

Configuring the batch subsystem for persistent records

By default, EAP is configured with an in-memory job repository, which stores job execution details in RAM. To ensure that details of job executions are not lost after a server restart, you can set up a persistent store of job statistics by configuring a jdbc-job repository, which stores the details in a database. Red Hat recommends that you create a separate database to store batch job statistics, and you need to create an EAP data source pointing to this database.

This data source is referenced in the jdbc-job-repository configuration.
When a new jdbc-job-repository is configured, the tables are automatically created after the server configuration has been reloaded.

To create a JDBC working repository named custom-jdbc-repo, which references a data source named custom-ds on a standalone server, run the following command:

```
[standalone@127.0.0.1:9990 /] /subsystem=batch-jberet/\ jdbc-job-
repository=custom-jdbc-repo:add(data-source=custom-ds)
```

After setting the default working repository to the newly created custom-jdbc-repo, and having the server configuration reloaded, you should see the following four tables created in the database:

```
+--------------------+
 | tables_in_bkjobs                  |
+---------------------+
 | JOB_EXECUTION |
 | JOB_INSTANCE |
 | PARTITION_EXECUTION |
 | STEP_EXECUTION |
+---------------------+
```

After running some batch jobs, the statistics of the job executions should still be in the database, and you should be able to check the data in the tables using SELECT queries. For example:

```
mysql> select JOBINSTANCEID,STARTTIME,ENDTIME,BATCHSTATUS from JOB_EXECUTION;
+---------------+--------------------+----------- ----------+-------------+
 | JOBINSTANCEID | STARTTIME                          | ENDTIME                          | BATCHSTATUS |
+---------------+--------------------+----------- ----------+-------------+
                          1 | 2016-06-09 07:17:02 | 2016-06-09 07:17:04 | COMPLETED |
                          2 | 2016-06-09 07:24:28 | 2016-06-09 07:24:30 | COMPLETED |
                          3 | 2016-06-09 07:33:05 | 2016-06-09 07:33:07 | COMPLETED |
                          4 | 2016-06-09 07:33:15 | 2016-06-09 07:33:18 | COMPLETED |
+---------------+--------------------+----------- ----------+-------------+
```

The JOBINSTANCEID is a unique identifier for the execution of a job. STARTTIME and ENDTIME denote when the job started and completed respectively. BATCHSTATUS indicates whether the job execution completed successfully.