Red Hat JBoss Enterprise Application Platform 7.1

Performance Tuning Guide

For Use with Red Hat JBoss Enterprise Application Platform 7.1

Last Updated: 2018-10-11

Red Hat JBoss Enterprise Application Platform 7.1 Performance Tuning Guide

For Use with Red Hat JBoss Enterprise Application Platform 7.1

Legal Notice

Abstract

This book is a guide of performance tuning for Red Hat JBoss Enterprise Application Platform 7.1.

Table of Contents

CHAPTER 1. INTRODUCTION

A JBoss EAP installation is optimized by default. However, configurations to your environment, applications, and use of JBoss EAP subsystems can impact performance, meaning additional configuration might be needed.

This guide provides optimization recommendations for common JBoss EAP use cases, as well as instructions for monitoring performance and diagnosing performance issues.

IMPORTANT

You should stress test and verify all performance configuration changes under anticipated conditions in a development or testing environment prior to deploying them to production.

1.1. ABOUT THE USE OF EAP_HOME IN THIS DOCUMENT

In this document, the variable ***EAP_HOME*** is used to denote the path to the JBoss EAP installation. Replace this variable with the actual path to your JBoss EAP installation.

- If you installed JBoss EAP using the ZIP install method, the install directory is the **`jboss-eap-7.1`** directory where you extracted the ZIP archive.
- If you installed JBoss EAP using the RPM install method, the install directory is **`/opt/rh/eap7/root/usr/share/wildfly/`**.
- If you used the installer to install JBoss EAP, the default path for ***EAP_HOME*** is **`${user.home}/EAP-7.1.0`**:
  - For Red Hat Enterprise Linux, Solaris, and HP-UX: **`/home/`*`USER_NAME`*`/EAP-7.1.0/`**
  - For Microsoft Windows: **`C:\Users\`*`USER_NAME`*`\EAP-7.1.0\`**
- If you used the JBoss Developer Studio installer to install and configure the JBoss EAP server, the default path for ***EAP_HOME*** is **`${user.home}/jbdevstudio/runtimes/jboss-eap`**:
  - For Red Hat Enterprise Linux: **`/home/`*`USER_NAME`*`/jbdevstudio/runtimes/jboss-eap/`**
  - For Microsoft Windows: **`C:\Users\`*`USER_NAME`*`\jbdevstudio\runtimes\jboss-eap`** or **`C:\Documents and Settings\`*`USER_NAME`*`\jbdevstudio\runtimes\jboss-eap\`**

NOTE

EAP_HOME is not an environment variable. ***JBOSS_HOME*** is the environment variable used in scripts.

CHAPTER 2. MONITORING PERFORMANCE

You can monitor JBoss EAP performance using any tool that can examine JVMs running on your machine. Red Hat recommends that you use either JConsole, for which JBoss EAP includes a preconfigured wrapper script, or Java VisualVM. Both these tools provide basic monitoring of JVM processes, including memory usage, thread utilization, loaded classes, and other JVM metrics.

If you will be running one of these tools locally on the same machine that JBoss EAP is running on, then no configuration is necessary. However, if you will be running one of these tools to monitor JBoss EAP running on a remote machine, then some configuration is required for JBoss EAP to accept remote JMX connections.

2.1. CONFIGURING JBOSS EAP FOR REMOTE MONITORING CONNECTIONS

For a Standalone Server

1. Ensure that you have created a management user. You might want to create a separate management user to monitor your JBoss EAP server. See the JBoss EAP *Configuration Guide* for details.

2. When starting JBoss EAP, bind the management interface to the IP address that you will use to remotely monitor the server:

```
$ EAP_HOME/bin/standalone.sh -bmanagement=IP_ADDRESS
```

WARNING

This exposes all the JBoss EAP management interfaces, including the management console and management CLI, to the specified network. Ensure that you only bind the management interface to a private network.

3. Use the following URI with your management user name and password in your JVM monitoring tool to connect to the JBoss EAP server. The URI below uses the default management port (**9990**).

```
service:jmx:remote+http://IP_ADDRESS:9990
```

For a Managed Domain Host

Using the above procedure of binding the management interface on a managed domain host will only expose the host controller JVM for remote monitoring, and not the individual JBoss EAP servers running on that host.

To configure JBoss EAP to remotely monitor individual servers on a managed domain host, follow the procedure below.

1. Create a new user in the **ApplicationRealm** that you will use to connect to the JBoss EAP servers for remote monitoring. See the JBoss EAP *Configuration Guide* for details.

2. In the management CLI, run the following commands to add a remoting port to the socket binding group, and add remoting to the **`ApplicationRealm`**. If necessary, replace the profile name and socket binding group in the following commands with the ones that you are using.

```
/profile=full/subsystem=jmx/remoting-connector=jmx:add(use-
management-endpoint=false)
/socket-binding-group=full-sockets/socket-
binding=remoting:add(port=4447)
/profile=full/subsystem=remoting/connector=remoting-
connector:add(socket-binding=remoting,security-
realm=ApplicationRealm)
```

3. When starting your JBoss EAP managed domain host, bind one or both of the following interfaces to an IP address that you will use for monitoring.

   - If you want to connect to individual JBoss EAP server JVMs running on your managed domain host, bind the public interface:

```
$ EAP_HOME/bin/domain.sh -b=IP_ADDRESS
```

   - If you want to connect to the JBoss EAP host controller JVM, also bind the management interface:

```
$ EAP_HOME/bin/domain.sh -bmanagement=IP_ADDRESS
```

WARNING

This exposes all the JBoss EAP management interfaces, including the management console and management CLI, to the specified network. Ensure that you only bind the management interface to a private network.

4. Use the following details in your JVM monitoring tool:

   - To connect to individual JBoss EAP server JVMs running on your managed domain host, use the following URI with your **`ApplicationRealm`** user name and password that was created earlier.

```
service:jmx:remote://IP_ADDRESS:4447
```

     To connect to different JBoss EAP servers on a single host, add the respective server's port offset value to the above port number.

   - To connect to the JBoss EAP host controller JVM, use the following URI with a management user name and password.

```
service:jmx:remote://IP_ADDRESS:9999
```

2.2. JCONSOLE

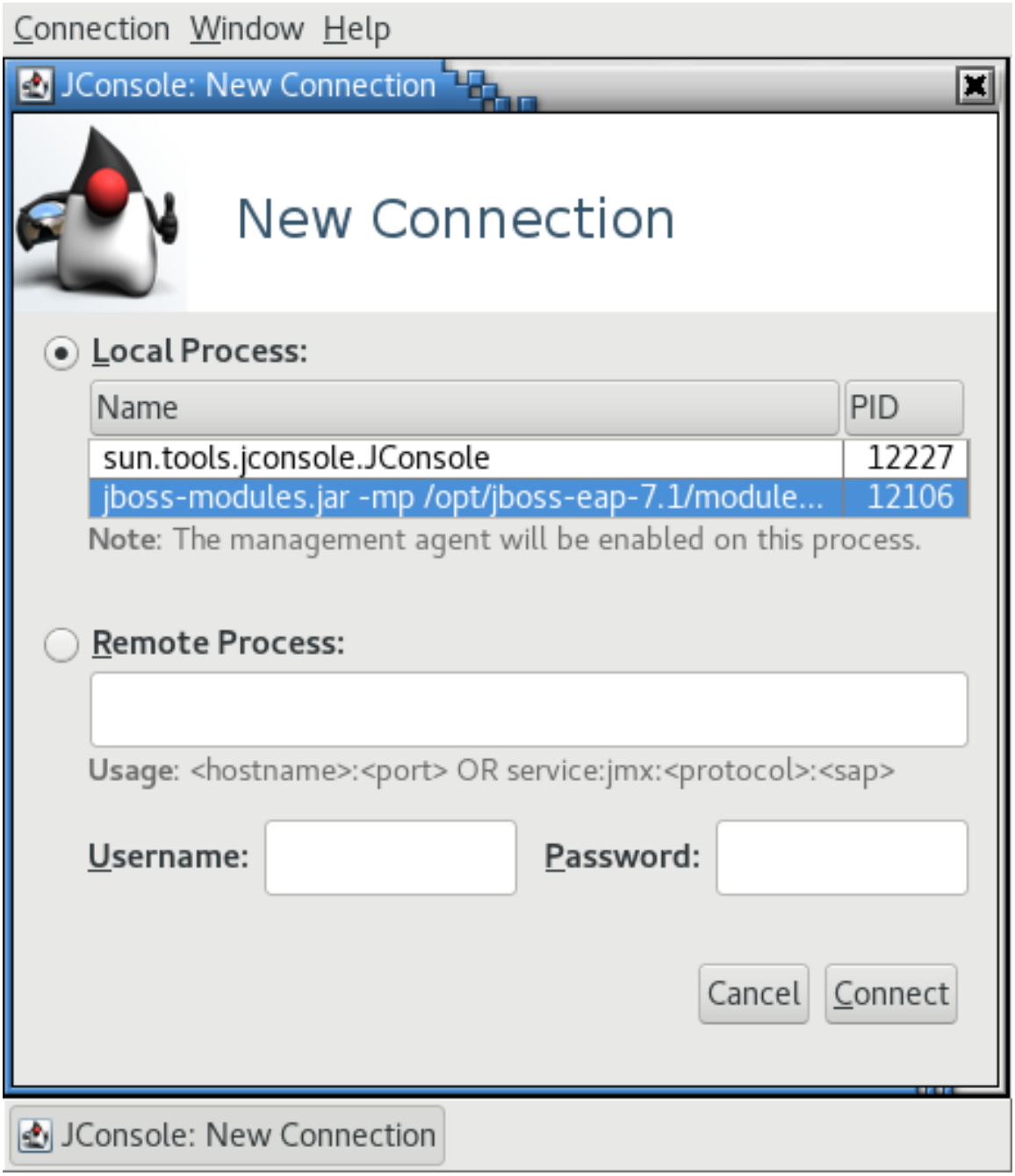
A preconfigured JConsole wrapper script is bundled with JBoss EAP. Using this wrapper script ensures that all the required libraries are added to the class path, and also provides access to the JBoss EAP management CLI from within JConsole.

2.2.1. Connecting to a Local JBoss EAP JVM Using JConsole

To connect to a JBoss EAP JVM running on the same machine as JConsole:

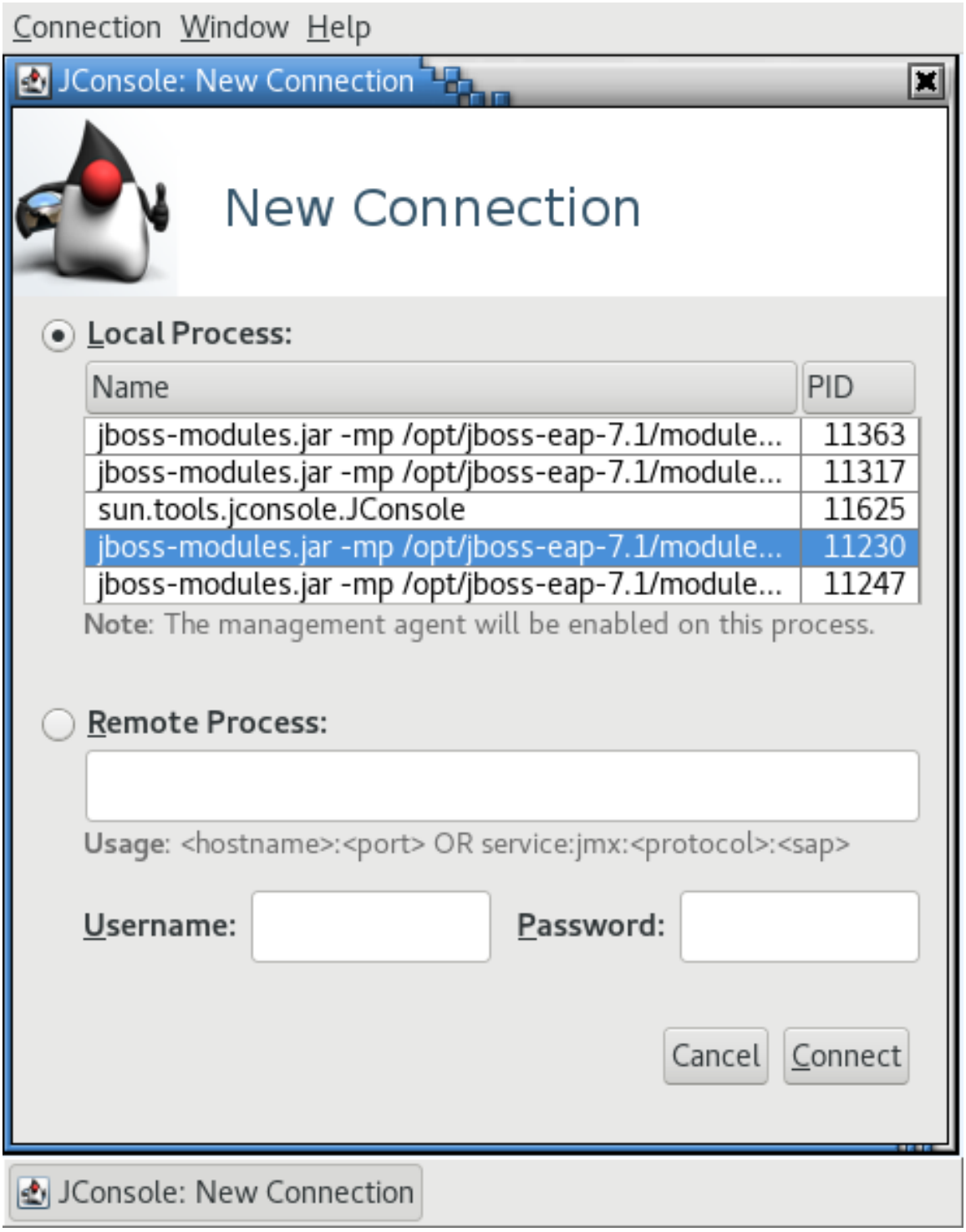
1. Run the `jconsole` script in ***EAP_HOME*/bin**.
2. Under **Local Process**, select the JBoss EAP JVM process that to want to monitor.
   - For a standalone JBoss EAP server, there is one JBoss EAP JVM process.

Figure 2.1. JConsole Local Standalone JBoss EAP Server JVM



- A JBoss EAP managed domain host has multiple JVM processes you can connect to: a host controller JVM process, a process controller JVM process, and a JVM process for each JBoss EAP server on the host. You can determine which JVM you have connected to by looking at the JVM arguments.

Figure 2.2. JConsole Local Managed Domain JBoss EAP JVMs



3. Click **Connect**.

2.2.2. Connecting to a Remote JBoss EAP JVM Using JConsole

Prerequisites

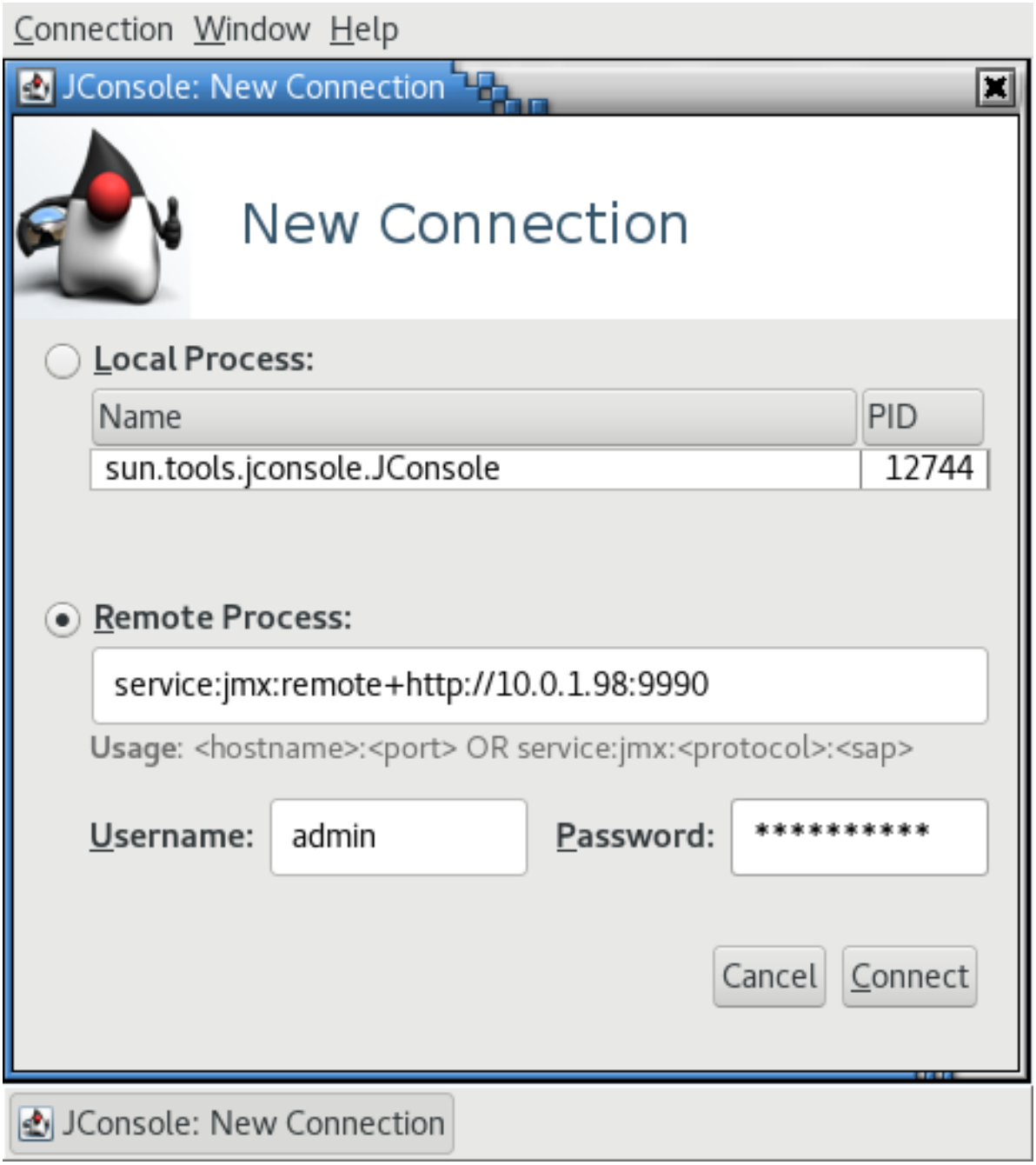
- Configure JBoss EAP for remote monitoring connections.

- Download and extract a ZIP installation of JBoss EAP to your local machine. See the JBoss EAP *Installation Guide* for details.

1. Run the **`jconsole`** script in ***`EAP_HOME`*`/bin`**.

2. Under **Remote Process**, insert the URI for the remote JBoss EAP JVM process that to want to monitor.
   See the instructions on configuring JBoss EAP for remote monitoring connections for the URI to use.

Figure 2.3. JConsole Remote JBoss EAP JVM



3. Ensure that you provide the user name and password for the monitoring connection.

4. Click **Connect**.

2.3. JAVA VISUALVM

Java VisualVM is included with the Oracle JDK, and is located at ***`JAVA_HOME`*`/bin/jvisualvm`**. If you are not using the Oracle JDK, VisualVM is also available for download from the VisualVM website. Note that VisualVM does not work with the IBM JDK.

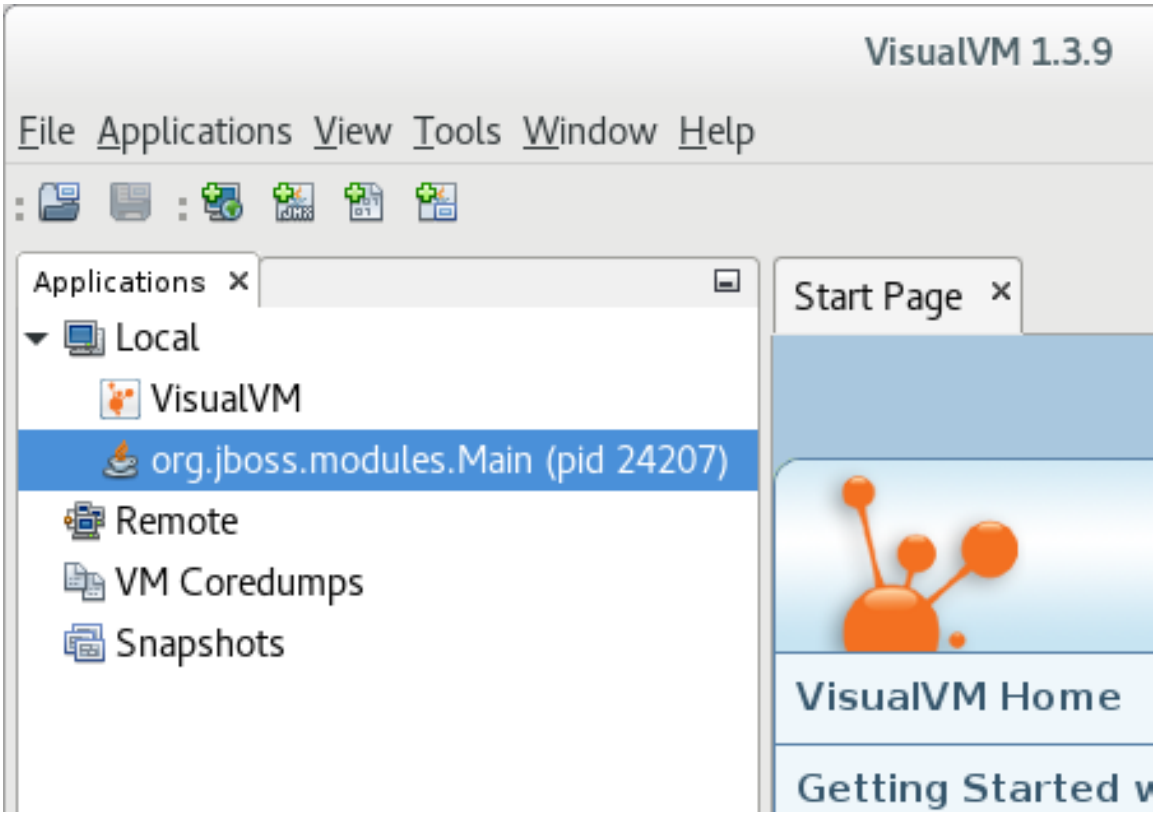
The following sections provide instructions for using VisualVM to connect to a local or remote JBoss EAP JVM. See the VisualVM documentation for other information on using VisualVM.

2.3.1. Connecting to a Local JBoss EAP JVM Using VisualVM

To connect to a JBoss EAP JVM running on the same machine as VisualVM:

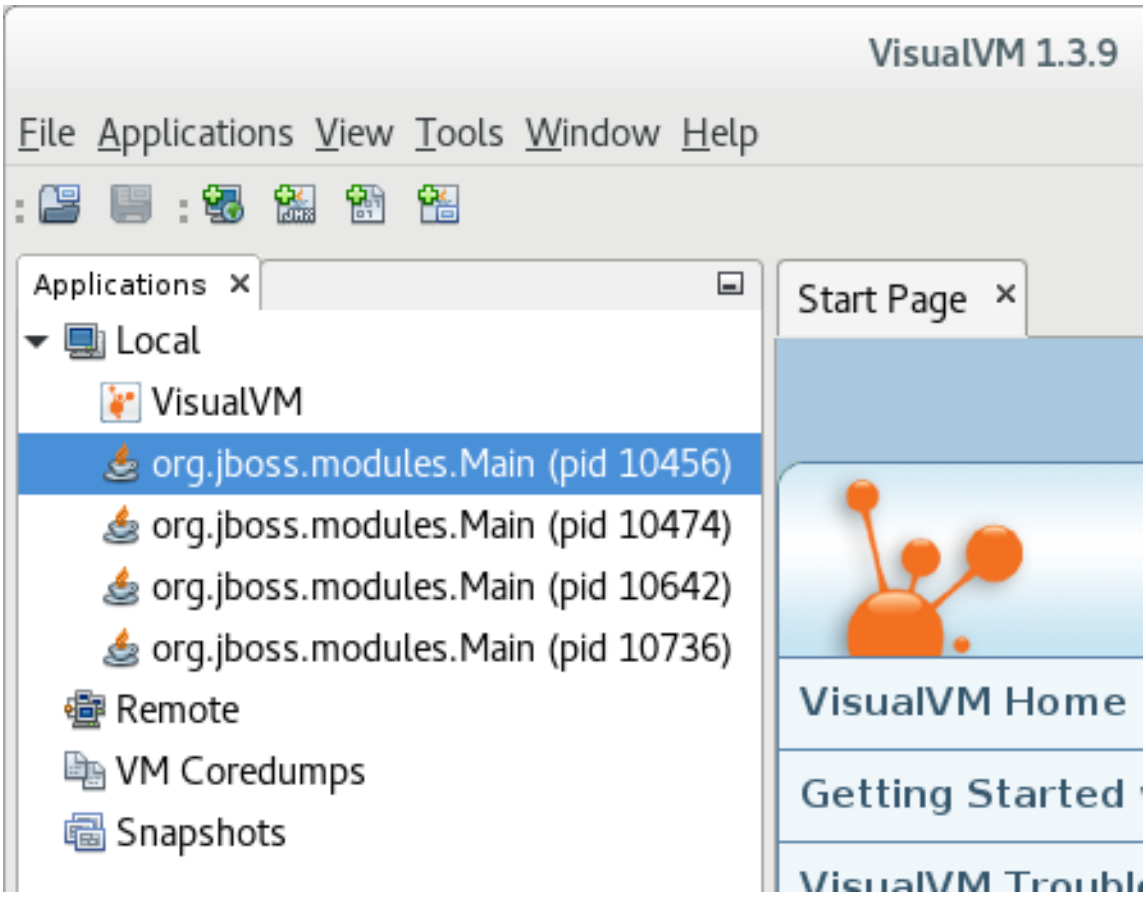
1. Open VisualVM, and find the **Applications** pane on the left side of the VisualVM window.
2. Under **Local**, double-click the JBoss EAP JVM process that you want to monitor.
   - For a standalone JBoss EAP server, there is one JBoss EAP JVM process.

     Figure 2.4. VisualVM Local Standalone JBoss EAP Server JVM

   - A JBoss EAP managed domain host has multiple JVM processes you can connect to: a host controller JVM process, a process controller JVM process, and a JVM process for each JBoss EAP server on the host. You can determine which JVM you have connected to by looking at the JVM arguments.

Figure 2.5. VisualVM Local Managed Domain JBoss EAP JVMs



2.3.2. Connecting to a Remote JBoss EAP JVM Using VisualVM

Prerequisites

- Configure JBoss EAP for remote monitoring connections.
- Download and extract a ZIP installation of JBoss EAP to your local machine. See the JBoss EAP *Installation Guide* for details.

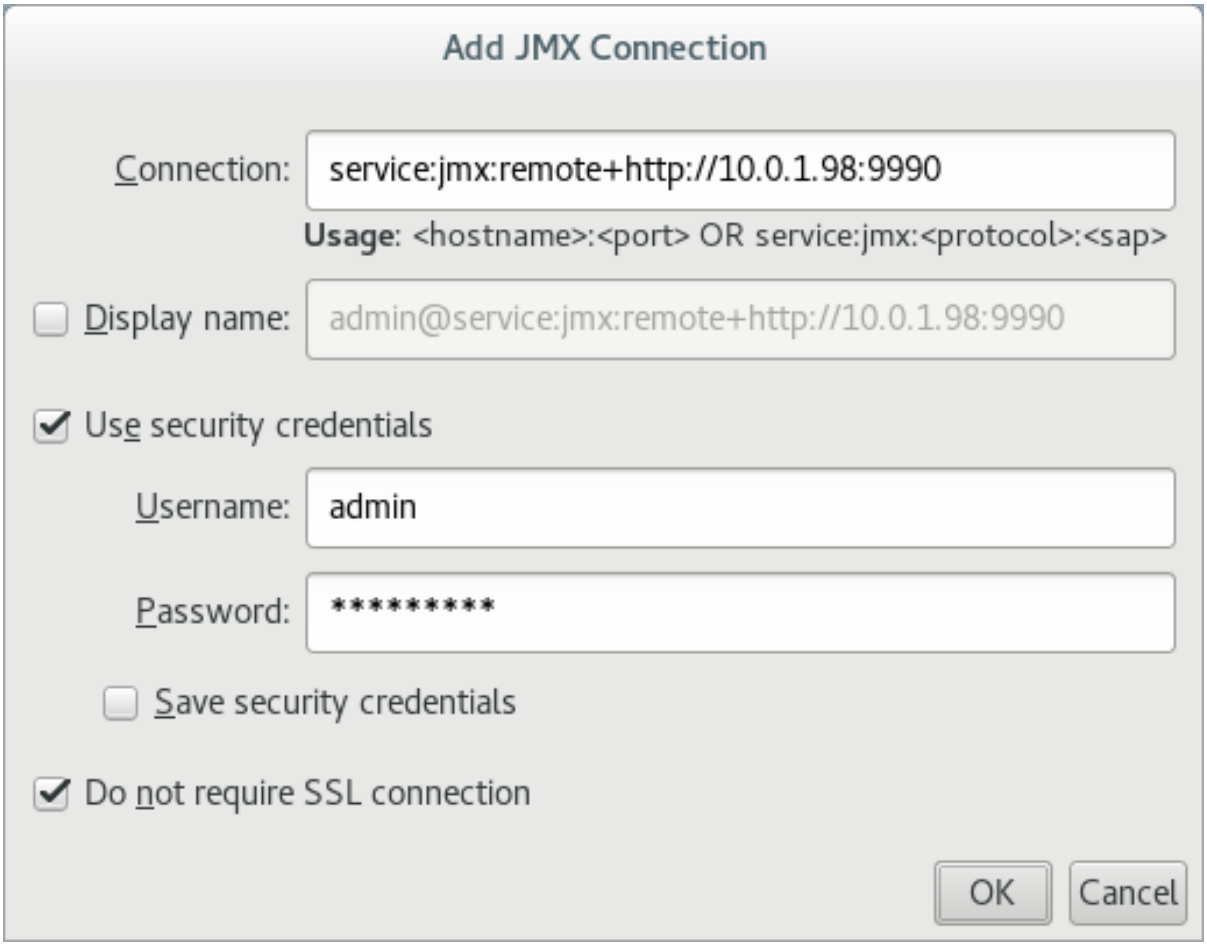
1. You must add the required JBoss EAP libraries to your class path to remotely monitor a JBoss EAP JVM. Start VisualVM with the arguments for required libraries on your local machine. For example:

```
$ visualvm -cp:a EAP_HOME/bin/client/jboss-cli-client.jar  -J-
Dmodule.path=EAP_HOME/modules
```

2. In the **File** menu, select **Add JMX Connection**.
3. Complete the details for your remote JBoss EAP JVM:
   - In the **Connection** field, insert the URI for the remote JBoss EAP JVM process that to want to monitor. See the instructions on configuring JBoss EAP for remote monitoring connections for the URI to use.
   - Select the **Use security credentials** check box, and enter the user name and password for the monitoring connection.

- If you are not using an SSL connection, select the **Do not require SSL connection** check box.

Figure 2.6. VisualVM Remote JBoss EAP JVM



4. Click **OK**.

5. In the **Applications** pane on the left side of the VisualVM window, double-click on the JMX item under the remote host to open the monitoring connection.

CHAPTER 3. DIAGNOSING PERFORMANCE ISSUES

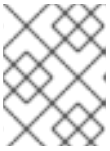
3.1. ENABLING GARBAGE COLLECTION LOGGING

Examining garbage collection logs can be useful when attempting to troubleshoot Java performance issues, especially those related to memory usage.

Other than some additional disk I/O activity for writing the log files, enabling garbage collection logging does not significantly affect server performance.

Garbage collection logging is already enabled by default for a standalone JBoss EAP server running on OpenJDK or Oracle JDK. For a JBoss EAP managed domain, garbage collection logging can be enabled for the host controller, process controller, or individual JBoss EAP servers.

1. Get the correct JVM options for enabling garbage collection logging for your JDK. Replace the path in the options below to where you want the log to be created.



NOTE

The Red Hat Customer Portal has a JVM Options Configuration Tool that can help you generate optimal JVM settings.

   - For OpenJDK or Oracle JDK:

```
-verbose:gc -Xloggc:/path/to/gc.log -XX:+PrintGCDetails -
XX:+PrintGCTimeStamps -XX:+PrintGCApplicationStoppedTime
```

   - For IBM JDK:

```
-verbose:gc -Xverbosegclog:/path/to/gc.log
```

2. Apply the garbage collection JVM options to your JBoss EAP server.
   See the JBoss EAP *Configuration Guide* for instructions on how to apply JVM options to a standalone server or servers in a managed domain.

3.2. JAVA HEAP DUMPS

A Java heap dump is a snapshot of a JVM heap created at a certain point in time. Creating and analyzing heap dumps can be useful for diagnosing and troubleshooting issues with Java applications.

Depending on which JDK you are using, there are different ways of creating and analyzing a Java heap dump for a JBoss EAP process. This section covers common methods for Oracle JDK, OpenJDK, and IBM JDK.

3.2.1. Creating a Heap Dump

3.2.1.1. OpenJDK and Oracle JDK

Create an On-Demand Heap Dump

You can use the **`jcmd`** command to create an on-demand heap dump for JBoss EAP running on OpenJDK or Oracle JDK.

1. Determine the process ID of the JVM that you want to create a heap dump from.
2. Create the heap dump with the following command:

```
$ jcmd JAVA_PID GC.heap_dump -all=true FILENAME.hprof
```

This creates a heap dump file in the HPROF format, usually located in ***EAP_HOME*** or ***EAP_HOME*****/bin**. Alternatively, you can specify a file path to another directory.

Create a Heap Dump Automatically on OutOfMemoryError

You can use the **-XX:+HeapDumpOnOutOfMemoryError** JVM option to automatically create a heap dump when an **OutOfMemoryError** exception is thrown.

This creates a heap dump file in the HPROF format, usually located in ***EAP_HOME*** or ***EAP_HOME*****/bin**. Alternatively, you can set a custom path for the heap dump using **-XX:HeapDumpPath=*/path/***. If you specify a file name using **-XX:HeapDumpPath**, for example, **-XX:HeapDumpPath=*/path/filename.hprof***, the heap dumps will overwrite each other.

See the JBoss EAP *Configuration Guide* for instructions on how to apply JVM options to a standalone server or servers in a managed domain.

3.2.1.2. IBM JDK

When using the IBM JDK, heap dumps are automatically generated when an **OutOfMemoryError** is thrown.

Heap dumps from the IBM JDK are saved in the **/tmp/** directory as a portable heap dump (PHD) formatted file.

3.2.2. Analyzing a Heap Dump

Heap Dump Analysis Tools

There are many tools that can analyze heap dump files and help identify issues. Red Hat Support recommends using the Eclipse Memory Analyzer tool (MAT), which can analyze heap dumps formatted in either HPROF or PHD formats.

For information on using Eclipse MAT, see the Eclipse MAT documentation.

Heap Dump Analysis Tips

Sometimes the cause of the heap performance issues are obvious, but other times you may need an understanding of your application's code and the specific circumstances that cause issues like an **OutOfMemoryError**. This can help to identify whether an issue is a memory leak, or if the heap is just not large enough.

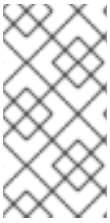
Some suggestions for identifying memory usage issues include:

- If a single object is not found to be consuming too much memory, try grouping by class to see if many small objects are consuming a lot of memory.
- Check if the biggest usage of memory is a thread. A good indicator of this is if the **OutOfMemoryError**-triggered heap dump is much smaller than the specified**Xmx** maximum heap size.

- A technique to make memory leaks more detectable is to temporarily double the normal maximum heap size. When an **`OutOfMemoryError`** occurs, the size of the objects related to the memory leak will be about half the size of the heap.

When the source of a memory issue is identified, you can view the paths from garbage collection roots to see what is keeping the objects alive.

3.3. IDENTIFYING HIGH CPU UTILIZATION BY JAVA THREADS



NOTE

For customers using JBoss EAP on Red Hat Enterprise Linux or Solaris, the JVMPeg lab tool on the Red Hat Customer Portal helps collect and analyze Java thread information to identify high CPU utilization. Follow the instructions for using the JVMPeg lab tool instead of using the following procedure.

For OpenJDK and Oracle JDK environments, Java thread diagnostic information is available using the **`jstack`** utility.

1. Identify the process ID of the Java process that is utilizing a high percentage of the CPU. It can also be useful to obtain per-thread CPU data on high-usage processes. This can be done using the **`top -H`** command on Red Hat Enterprise Linux systems.

2. Using the **`jstack`** utility, create a stack dump of the Java process. For example, on Linux and Solaris:

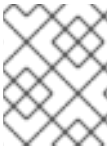
   ```
   jstack -l JAVA_PROCESS_ID > high-cpu-tdump.out
   ```

   You might need to create multiple dumps at intervals to see any changes or trends over a period of time.

3. Analyze the stack dumps. You can use a tool such as the Thread Dump Analyzer (TDA).

CHAPTER 4. JVM TUNING

Configuring optimal JVM options for your applications and JBoss EAP environment is one of the most fundamental ways to tune performance. This chapter covers configuring some general JVM options.



NOTE

Many of the JVM options listed in this chapter can be easily generated using the JVM Options Configuration Tool on the Red Hat Customer Portal.

See the JBoss EAP *Configuration Guide* for instructions on how to apply JVM options to a standalone server or servers in a managed domain.

4.1. SETTING A FIXED HEAP SIZE

You must set an appropriate heap size to prevent out of memory errors.

The **`-Xms`** option sets the initial heap size, and **`-Xmx`** sets the maximum heap size. It is recommended for production environments that you set the initial and maximum heap size options to the same size, so that the heap size is fixed and pre-allocated.

For example, the following options set a 2048 MB heap size:

```
-Xms2048M -Xmx2048M
```

It is recommended that you test your applications under load in a development environment to determine the maximum memory usage. Your production heap size should be at least 25% higher than the tested maximum to allow room for overhead.

4.2. CONFIGURING THE GARBAGE COLLECTOR

Although the parallel garbage collector, also known as the throughput garbage collector, is the default garbage collector in Java 8 for server-class machines, Red Hat recommends using the G1 garbage collector, which is expected to be the default from Java 9 onward. The G1 garbage collector generally performs better than the CMS and parallel garbage collectors in most scenarios.

To enable the G1 collector, use the following JVM option:

```
-XX:+UseG1GC
```

Garbage Collection Logging Options

Garbage collection logging is enabled by default for standalone JBoss EAP servers. To enable garbage collection logging for a JBoss EAP managed domain, see Section 3.1, “Enabling Garbage Collection Logging”.

4.3. ENABLING LARGE PAGES

Enabling large pages for JBoss EAP JVMs results in pages that are locked in memory and cannot be swapped to disk like regular memory.

Especially for memory-intensive applications, the advantage of using large pages is that the heap cannot be paged or swapped to disk, and is thus always readily available.

One disadvantage of using large pages is that other processes running on the system might not have quick access to memory, which might result in excessive paging for these processes.

As with any other performance configuration change, it is recommended that you test the impact of the change in a testing environment.

1. You must ensure that your operating system configuration allows for processes to use large pages.
   - For Red Hat Enterprise Linux systems, you must explicitly configure **`HugeTLB`** pages to guarantee that JBoss EAP processes will have access to large pages.
     For information on configuring Red Hat Enterprise Linux memory options, see the Memory chapter in the Red Hat Enterprise Linux *Performance Tuning Guide*.
   - For Windows Server systems, the user that is running JBoss EAP must have the large pages privilege assigned:
     1. Select **Control Panel** → **Administrative Tools** → **Local Security Policy**.
     2. Select **Local Policies** → **User Rights Assignment**.
     3. Double-click **Lock pages in memory**.
     4. Add the Windows Server users and user groups that you want to use large pages.
     5. Restart the machine.
2. Enable or disable large page support:
   - To explicitly enable large page support for JBoss EAP JVMs, use the following JVM option:

     ```
     -XX:+UseLargePages
     ```

   - To explicitly disable large page support for JBoss EAP JVMs, use the following JVM option:

     ```
     -XX:-UseLargePages
     ```

3. When starting JBoss EAP, ensure that there are no warnings related to reserving memory.
   - On Red Hat Enterprise Linux, an error might look like:

     ```
     OpenJDK 64-Bit Server VM warning: Failed to reserve shared
     memory. (error = 1)
     ```

   - On Windows Server, an error might look like:

     ```
     Java HotSpot(TM) 64-Bit Server VM warning: JVM cannot use large
     page memory because it does not have enough privilege to lock
     pages in memory.
     ```

   If you do see warnings, verify that your operating system configuration and JVM options are configured correctly.

For more information, see the Oracle documentation on Java support for large pages.

4.4. ENABLING AGGRESSIVE OPTIMIZATIONS

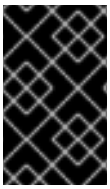
Using the aggressive optimizations (AggressiveOpts) JVM option can provide performance improvements for your environment. This option enables Java performance optimization features that are expected to become default in future Java releases.

To enable AggressiveOpts, use the following JVM option:

```
-XX:+AggressiveOpts
```

4.5. SETTING ULIMITS

For Red Hat Enterprise Linux and Solaris platforms, you must configure appropriate **`ulimit`** values for JBoss EAP JVM processes. The "soft" **`ulimit`** can be temporarily exceeded, while the "hard" **`ulimit`** is the strict ceiling for the usage of a resource. Appropriate **`ulimit`** values vary depending on your environment and applications.



IMPORTANT

If you are using IBM JDK, it is important to note that IBM JDK uses the soft limit for the maximum number of open files used by a JVM process. On Red Hat Enterprise Linux, the default soft limit (**`1024`**) is considered too low for JBoss EAP processes using IBM JDK.

If the limits applied to JBoss EAP processes are too low, you will see a warning like the following when starting JBoss EAP:

```
WARN  [org.jboss.as.warn.fd-limit] (main) WFLYSRV0071: The operating
system has limited the number of open files to 1024 for this process; a
value of at least 4096 is recommended.
```

To see your current **`ulimit`** values, use the following commands:

- For soft **`ulimit`** values:

  ```
  ulimit -Sa
  ```

- For hard **`ulimit`** values:

  ```
  ulimit -Ha
  ```

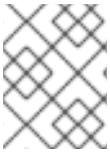
To set the **`ulimit`** for the maximum number of open files, use the following commands with the number you want to apply:

- To set the soft **`ulimit`** for the maximum number of open files:

  ```
  ulimit -Sn 4096
  ```

- To set the hard **`ulimit`** for the maximum number of open files:

  ```
  ulimit -Hn 4096
  ```



NOTE

To guarantee that a **`ulimit`** setting is effective, it is recommended on production systems to set the soft and hard limits to the same value.

For more information on setting **`ulimit`** values using a configuration file, see How to set ulimit values on the Customer Portal.

4.6. HOST CONTROLLER AND PROCESS CONTROLLER JVM TUNING

JBoss EAP managed domain hosts have separate JVMs for the host controller and process controller. See the JBoss EAP *Configuration Guide* for more information on the roles of host controllers and process controllers.

You can tune the host controller and process controller JVM settings, but even for large managed domain environments, the default JVM configuration for the host controller and process controller should suffice.

The default configurations for host controller and process controller JVMs have been tested with a managed domain size of up to 20 JBoss EAP hosts each running 10 JBoss EAP servers, for a total domain size of 200 JBoss EAP servers.

If you experience issues with larger managed domains, you might need to monitor the host controller or process controller JVMs in your environment to determine appropriate values for JVM options such as heap size.

CHAPTER 5. EJB SUBSYSTEM TUNING

JBoss EAP can cache Enterprise Java Beans (EJBs) to save initialization time. This is accomplished using bean pools.

There are two different bean pools that can be tuned in JBoss EAP: bean instance pools and bean thread pools.

Appropriate bean pool sizes depend on your environment and applications. It is recommended that you experiment with different bean pool sizes and perform stress testing in a development environment that emulates your expected real-world conditions.

5.1. BEAN INSTANCE POOLS

Bean instance pools are used for Stateless Session Beans (SLSBs) and Message Driven Beans (MDBs). By default, SLSBs use the instance pool **`default-slsb-instance-pool`**, and MDBs use the instance pool **`default-mdb-instance-pool`**.

The size of a bean instance pool limits the number of instances of a particular EJB that can be created at one time. If the pool for a particular EJB is full, the client will block and wait for an instance to become available. If a client does not get an instance within the time set in the pool's **`timeout`** attributes, an exception is thrown.

The size of a bean instance pool is configured using either **`derive-size`** or **`max-pool-size`**. The **`derive-size`** attribute allows you to configure the pool size using one of the following values:

- **`from-worker-pools`**, which indicates that the maximum pool size is derived from the size of the total threads for all worker pools configured on the system.
- **`from-cpu-count`**, which indicates that the maximum pool size is derived from the total number of processors available on the system. Note that this is not necessarily a 1:1 mapping, and might be augmented by other factors.

If **`derive-size`** is undefined, then the value of **`max-pool-size`** is used for the size of the bean instance pool.

NOTE

The **`derive-size`** attribute overrides any value specified in **`max-pool-size`**. **`derive-size`** must be undefined for the **`max-pool-size`** value to take effect.

You can configure an EJB to use a specific instance pool. This allows for finer control of the instances available to each EJB type.

5.1.1. Creating a Bean Instance Pool

This section shows you how to create a new bean instance pool using the management CLI. You can also configure bean instance pools using the management console by navigating to the **EJB 3** subsystem from the **Configuration** tab, and then selecting the **Bean Pools** tab.

To create a new instance pool, use one of the following commands:

- To create a bean instance pool with a derived maximum pool size:

```
/subsystem=ejb3/strict-max-bean-instance-pool=POOL_NAME:add(derive-size=DERIVE_OPTION,timeout-unit=TIMEOUT_UNIT,timeout=TIMEOUT_VALUE)
```

The following example creates a bean instance pool named **my_derived_pool** with a maximum size derived from the CPU count, and a timeout of 2 minutes:

```
/subsystem=ejb3/strict-max-bean-instance-pool=my_derived_pool:add(derive-size=from-cpu-count,timeout-unit=MINUTES,timeout=2)
```

- To create a bean instance pool with an explicit maximum pool size:

```
/subsystem=ejb3/strict-max-bean-instance-pool=POOL_NAME:add(max-pool-size=POOL_SIZE,timeout-unit=TIMEOUT_UNIT,timeout=TIMEOUT_VALUE)
```

The following example creates a bean instance pool named **my_pool** with a maximum of 30 instances and a timeout of 30 seconds:

```
/subsystem=ejb3/strict-max-bean-instance-pool=my_pool:add(max-pool-size=30,timeout-unit=SECONDS,timeout=30)
```

5.1.2. Specifying the Instance Pool a Bean Should Use

You can set a specific instance pool that a particular bean will use either by using the **@org.jboss.ejb3.annotation.Pool** annotation, or by modifying the **jboss-ejb3.xml** deployment descriptor of the bean. See the **jboss-ejb3.xml** Deployment Descriptor Reference in *Developing EJB Applications* for more information.

5.1.3. Disabling the Default Bean Instance Pool

The default bean instance pool can be disabled, which results in EJBs not using any instance pool by default. Instead, a new EJB instance is created when a thread needs to invoke a method on an EJB. This might be useful if you do not want any limit on the number of EJB instances that are created.

To disable the default bean instance pool, use the following management CLI command:

```
/subsystem=ejb3:undefine-attribute(name=default-slsb-instance-pool)
```

NOTE

If a bean is configured to use a particular bean instance pool, disabling the default instance pool does not affect the pool that the bean uses.

5.2. BEAN THREAD POOLS

By default, a bean thread pool named **default** is used for asynchronous EJB calls and EJB timers.

NOTE

From JBoss EAP 7 onward, remote EJB requests are handled in the worker defined in the **io** subsystem by default.

If required, you can configure each of these EJB services to use a different bean thread pool. This can be useful if you want finer control of each service's access to a bean thread pool.

When determining an appropriate thread pool size, consider how many concurrent requests you expect will be processed at once.

5.2.1. Creating a Bean Thread Pool

This section shows you how to create a new bean thread pool using the management CLI. You can also configure bean thread pools using the management console by navigating to the **EJB 3** subsystem from the **Configuration** tab and selecting **Thread Pools** in the left menu.

To create a new thread pool, use the following command:

```
/subsystem=ejb3/thread-pool=POOL_NAME:add(max-threads=MAX_THREADS)
```

The following example creates a bean thread pool named **my_thread_pool** with a maximum of 30 threads:

```
/subsystem=ejb3/thread-pool=my_thread_pool:add(max-threads=30)
```

5.2.2. Configuring EJB Services to Use a Specific Bean Thread Pool

The EJB3 asynchronous invocation service and timer service can each be configured to use a specific bean thread pool. By default, both these services use the **default** bean thread pool.

This section shows you how to configure the above EJB services to use a specific bean thread pool using the management CLI. You can also configure these services using the management console by navigating to the **EJB 3** subsystem from the **Configuration** tab, and then selecting the **Services** tab.

To configure an EJB service to use a specific bean thread pool, use the following command:

```
/subsystem=ejb3/service=SERVICE_NAME:write-attribute(name=thread-pool-
name,value=THREAD_POOL_NAME)
```

Replace **SERVICE_NAME** with the EJB service you want to configure:

- **async** for the EJB3 asynchronous invocation service
- **timer-service** for the EJB3 timer service

The following example sets the EJB3 async service to use the bean thread pool named **my_thread_pool**:

```
/subsystem=ejb3/service=async:write-attribute(name=thread-pool-
name,value=my_thread_pool)
```

5.3. EXCEPTIONS THAT INDICATE EJB SUBSYSTEM TUNING MIGHT BE REQUIRED

- **The Stateless EJB instance pool is not large enough or the timeout is too low**

```
javax.ejb.EJBException: JBAS014516: Failed to acquire a permit
within 20 SECONDS
     at
org.jboss.as.ejb3.pool.strictmax.StrictMaxPool.get(StrictMaxPool.jav
a:109)
```

See Bean Instance Pools.

- **The EJB thread pool is not large enough, or an EJB is taking longer to process than the invocation timeout**

```
java.util.concurrent.TimeoutException: No invocation response
received in 300000 milliseconds
```

See Bean Thread Pools.

CHAPTER 6. DATASOURCE AND RESOURCE ADAPTER TUNING

Connection pools are the principal tool that JBoss EAP uses to optimize performance for environments that use datasources, such as relational databases, or resource adapters.

Allocating and deallocating resources for datasource and resource adapter connections is very expensive in terms of time and system resources. Connection pooling reduces the cost of connections by creating a 'pool' of connections that are available to applications.

Before configuring your connection pool for optimal performance, you must monitor the datasource pool statistics or resource adapter statistics under load to determine the appropriate settings for your environment.

6.1. MONITORING POOL STATISTICS

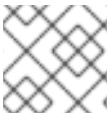
6.1.1. Datasource Statistics

When statistics collection is enabled for a datasource, you can view runtime statistics for the datasource.

6.1.1.1. Enabling Datasource Statistics

By default, datasource statistics are *not* enabled. You can enable datasource statistics collection using the management CLI or the management console.

Enable Datasource Statistics Using the Management CLI
The following management CLI command enables the collection of statistics for the **`ExampleDS`** datasource.



NOTE

In a managed domain, precede this command with **`/profile=PROFILE_NAME`**.

```
/subsystem=datasources/data-source=ExampleDS:write-
attribute(name=statistics-enabled,value=true)
```

Reload the server for the changes to take effect.

Enable Datasource Statistics Using the Management Console
Use the following steps to enable statistics collection for a datasource using the management console.

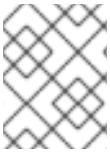
1. Navigate to **Configuration** → **Subsystems** → **Datasources**.
2. Select **Non-XA** or **XA**, depending on your datasource type.
3. Select the datasource and click **View**.
4. Click **Edit** under the **Attributes** tab.
5. Check the **Statistics enabled?** checkbox and click **Save**. A dialog appears indicating that the changes require a reload in order to take effect.
6. Reload the server.

- For a standalone server, click the **Reload Server Now** button to reload the server.
- In a managed domain, click the **Go to Runtime** button. From the **Runtime** tab, select the appropriate server and click the **Reload** drop down option to reload the server.

6.1.1.2. Viewing Datasource Statistics

You can view runtime statistics for a datasource using the management CLI or management console.

View Datasource Statistics Using the Management CLI
The following management CLI command retrieves the core *pool* statistics for the **ExampleDS** datasource.



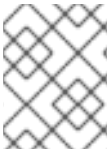
NOTE

In a managed domain, precede these commands with **/host=*HOST_NAME*/server=*SERVER_NAME***.

```
/subsystem=datasources/data-source=ExampleDS/statistics=pool:read-
resource(include-runtime=true)
{
    "outcome" => "success",
    "result" => {
        "ActiveCount" => 1,
        "AvailableCount" => 20,
        "AverageBlockingTime" => 0L,
        "AverageCreationTime" => 122L,
        "AverageGetTime" => 128L,
        "AveragePoolTime" => 0L,
        "AverageUsageTime" => 0L,
        "BlockingFailureCount" => 0,
        "CreatedCount" => 1,
        "DestroyedCount" => 0,
        "IdleCount" => 1,
        ...
}
```

The following management CLI command retrieves the *JDBC* statistics for the **ExampleDS** datasource.

```
/subsystem=datasources/data-source=ExampleDS/statistics=jdbc:read-
resource(include-runtime=true)
{
    "outcome" => "success",
    "result" => {
        "PreparedStatementCacheAccessCount" => 0L,
        "PreparedStatementCacheAddCount" => 0L,
        "PreparedStatementCacheCurrentSize" => 0,
        "PreparedStatementCacheDeleteCount" => 0L,
        "PreparedStatementCacheHitCount" => 0L,
        "PreparedStatementCacheMissCount" => 0L,
        "statistics-enabled" => true
    }
}
```



NOTE

Since statistics are runtime information, be sure to specify the **`include-runtime=true`** argument.

See Datasource Statistics for a detailed list of all available statistics.

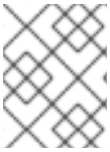
View Datasource Statistics Using the Management Console
To view datasource statistics from the management console, navigate to the **Datasources** subsystem from the **Runtime** tab. Use the **Datasources** tab to view statistics for non-XA datasources, and use the **XA Datasources** tab to view statistics for XA datasources.

See Datasource Statistics for a detailed list of all available statistics.

6.1.2. Resource Adapter Statistics

You can view core runtime statistics for deployed resource adapters. See the Resource Adapter Statistics appendix for a detailed list of all available statistics.

Enable Resource Adapter Statistics
By default, resource adapter statistics are *not* enabled. The following management CLI command enables the collection of statistics for a simple resource adapter **`myRA.rar`** with a connection factory bound in JNDI as **`java:/eis/AcmeConnectionFactory`**:

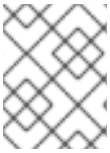


NOTE

In a managed domain, precede the command with **`/host=HOST_NAME/server=SERVER_NAME/`**.

```
/deployment=myRA.rar/subsystem=resource-
adapters/statistics=statistics/connection-
definitions=java\:\/eis\/AcmeConnectionFactory:write-
attribute(name=statistics-enabled,value=true)
```

View Resource Adapter Statistics
Resource adapter statistics can be retrieved from the management CLI. The following management CLI command returns statistics for the resource adapter **`myRA.rar`** with a connection factory bound in JNDI as **`java:/eis/AcmeConnectionFactory`**.

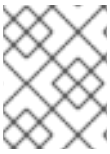


NOTE

In a managed domain, precede the command with **`/host=HOST_NAME/server=SERVER_NAME/`**.

```
deployment=myRA.rar/subsystem=resource-
adapters/statistics=statistics/connection-
definitions=java\:\/eis\/AcmeConnectionFactory:read-resource(include-
runtime=true)
{
    "outcome" => "success",
    "result" => {
        "ActiveCount" => "1",
        "AvailableCount" => "20",
```

```
        "AverageBlockingTime" => "0",
        "AverageCreationTime" => "0",
        "CreatedCount" => "1",
        "DestroyedCount" => "0",
        "InUseCount" => "0",
        "MaxCreationTime" => "0",
        "MaxUsedCount" => "1",
        "MaxWaitCount" => "0",
        "MaxWaitTime" => "0",
        "TimedOut" => "0",
        "TotalBlockingTime" => "0",
        "TotalCreationTime" => "0"
    }
}
```



NOTE

Since statistics are runtime information, be sure to specify the **`include-runtime=true`** argument.

6.2. POOL ATTRIBUTES

This section details advice for selected pool attributes that can be configured for optimal datasource or resource adapter performance. For instructions on how to configure each of these attributes, see:

- Configuring Datasource Pool Attributes
- Configuring Resource Adapter Pool Attributes

Minimum Pool Size

The **`min-pool-size`** attribute defines the minimum size of the connection pool. The default minimum is zero connections. With a zero **`min-pool-size`**, connections are created and placed in the pool when the first transactions occur.
If **`min-pool-size`** is too small, it results in increased latency while executing initial database commands because new connections might need to be established. If **`min-pool-size`** is too large, it results in wasted connections to the datasource or resource adapter.

During periods of inactivity the connection pool will shrink, possibly to the **`min-pool-size`** value.

Red Hat recommends that you set **`min-pool-size`** to the number of connections that allow for ideal on-demand throughput for your applications.

Maximum Pool Size

The **`max-pool-size`** attribute defines the maximum size of the connection pool. It is an important performance parameter because it limits the number of active connections, and thus also limits the amount of concurrent activity.
If **`max-pool-size`** is too small, it can result in requests being unnecessarily blocked. If **`max-pool-size`** is too large, it can result in your JBoss EAP environment, datasource, or resource adapter using more resources than it can handle.

Red Hat recommends that you set the **max-pool-size** to at least 15% higher than an acceptable **MaxUsedCount** observed after monitoring performance under load. This allows some buffer for higher than expected conditions.

Prefill

The **pool-prefill** attribute specifies whether JBoss EAP will prefill the connection pool with the minimum number of connections when JBoss EAP starts. The default value is **false**.
When **pool-prefill** is set to **true**, JBoss EAP uses more resources at startup, but there will be less latency for initial transactions.

Red Hat recommends to set **pool-prefill** to **true** if you have optimized the **min-pool-size**.

Strict Minimum

The **pool-use-strict-min** attribute specifies whether JBoss EAP allows the number of connections in the pool to fall below the specified minimum.
If **pool-use-strict-min** is set to **true**, JBoss EAP will not allow the number of connections to temporarily fall below the specified minimum. The default value is **false**.

Although a minimum number of pool connections is specified, when JBoss EAP closes connections, for instance, if the connection is idle and has reached the timeout, the closure may cause the total number of connections to temporarily fall below the minimum before a new connection is created and added to the pool.

Timeouts

There are a number of timeout options that are configurable for a connection pool, but a significant one for performance tuning is **idle-timeout-minutes**.
The **idle-timeout-minutes** attribute specifies the maximum time, in minutes, a connection may be idle before being closed. As idle connections are closed, the number of connections in the pool will shrink down to the specified minimum.

The longer the timeout, the more resources are used but requests might be served faster. The lower the timeout, the less resources are used but requests might need to wait for a new connection to be created.

6.3. CONFIGURING POOL ATTRIBUTES

6.3.1. Configuring Datasource Pool Attributes

Prerequisites

- Install a JDBC driver. See JDBC Drivers in the JBoss EAP *Configuration Guide*.
- Create a datasource. See Creating Datasources in the JBoss EAP *Configuration Guide*.

You can configure datasource pool attributes using either the management CLI or the management console:

- To use the management console, navigate to **Configuration** → **Subsystems** → **Datasources** → **Non-XA** | **XA**, select your datasource, and click **View**. The pool options are configurable

under the datasource **Pool** tab. Timeout options are configurable under the datasource **Timeouts** tab.

- To use the management CLI, execute the following command:

```
/subsystem=datasources/data-source=DATASOURCE_NAME/:write-
attribute(name=ATTRIBUTE_NAME,value=ATTRIBUTE_VALUE)
```

For example, to set the **ExampleDS** datasource **min-pool-size** attribute to a value of 5 connections, use the following command:

```
/subsystem=datasources/data-source=ExampleDS/:write-
attribute(name=min-pool-size,value=5)
```

6.3.2. Configuring Resource Adapter Pool Attributes

Prerequisites

- Deploy your resource adapter and add a connection definition. See Configuring Resource Adapters in the JBoss EAP *Configuration Guide*.

You can configure resource adapter pool attributes using either the management CLI or the management console:

- To use the management console, navigate to **Configuration** → **Subsystems** → **Resource Adapters**, select your resource adapter, click **View**, and select **Connection Definitions** in the left menu. The pool options are configurable under the **Pool** tab. Timeout options are configurable under the **Attributes** tab.
- To use the management CLI, execute the following command:

```
/subsystem=resource-adapters/resource-
adapter=RESOURCE_ADAPTER_NAME/connection-
definitions=CONNECTION_DEFINITION_NAME:write-
attribute(name=ATTRIBUTE_NAME,value=ATTRIBUTE_VALUE)
```

For example, to set the **my_RA** resource adapter **my_CD** connection definition **min-pool-size** attribute to a value of 5 connections, use the following command:

```
/subsystem=resource-adapters/resource-adapter=my_RA/connection-
definitions=my_CD:write-attribute(name=min-pool-size,value=5)
```

CHAPTER 7. MESSAGING SUBSYSTEM TUNING

Performance tuning advice for the **`messaging-activemq`** subsystem is covered in the Performance Tuning part of the JBoss EAP *Configuring Messaging* guide.

CHAPTER 8. LOGGING SUBSYSTEM TUNING

You can further improve upon JBoss EAP logging subsystem performance in production environments by disabling logging to console, configuring appropriate logging levels, and specifying the best location to store log files.

For more information on configuring the **logging** subsystem, see the logging chapter in the JBoss EAP *Configuration Guide*.

8.1. DISABLING LOGGING TO THE CONSOLE

Disabling console logging can improve JBoss EAP performance. Although outputting logs to the console can be useful in development and testing environments, for production environments it is not necessary in most cases. The JBoss EAP root logger includes a console log handler for all default standalone server profiles except **standalone-full-ha**. The default managed domain profiles do not include a console handler.

To remove the default console handler from the root logger, use the following management CLI command.

```
/subsystem=logging/root-logger=ROOT:remove-handler(name=CONSOLE)
```

8.2. CONFIGURING LOGGING LEVELS

For ideal performance, you must configure the logging levels for your production environment appropriately. For example, although **INFO** or **DEBUG** levels might be appropriate for development or testing environments, in most cases you should set your production environment logging level to something higher, such as **WARN** or **ERROR**.

For details on setting log levels for different logging handlers, see Configuring Log Handlers in the JBoss EAP *Configuration Guide*.

8.3. CONFIGURING THE LOCATION OF LOG FILES

You should consider the storage location of log files as a potential performance issue. If you save logs to a file system or disk configuration that has poor I/O throughput, it has the potential to affect the whole platform's performance.

To prevent logging from impacting JBoss EAP performance, it is recommended that you set log locations to high-performance dedicated disks that have a lot of space.

For details on configuring log file locations for different logging handlers, see Configuring Log Handlers in the JBoss EAP *Configuration Guide*.

CHAPTER 9. UNDERTOW SUBSYSTEM TUNING

The non-blocking I/O **undertow** subsystem introduced in JBoss EAP 7 has greatly improved performance compared to the previous **web** subsystem in JBoss EAP 6. Opportunities for tuning the **undertow** subsystem for your environment include configuring buffer caches, JSP settings, and listeners.

9.1. BUFFER CACHES

A buffer cache is used to cache static files handled by the **undertow** subsystem. This includes images, static HTML, CSS, and JavaScript files. You can specify a default buffer cache for each Undertow servlet container. Having an optimized buffer cache for your servlet container can improve Undertow performance for serving static files.

Buffers in a buffer cache are allocated in regions, and are of a fixed size. There are three configurable attributes for each buffer cache:

buffer-size

The size of an individual buffer, in bytes. The default is 1024 bytes. Red Hat recommends that you set the buffer size to entirely store your largest static file.

buffers-per-region

The number of buffers per region. The default is 1024.

max-regions

The maximum number of regions, which sets a maximum amount of memory allocated to the buffer cache. The default is 10 regions.

You can calculate the maximum amount memory used by a buffer cache by multiplying the buffer size, the number of buffers per region, and the maximum number of regions. For example, the default buffer cache is 1024 bytes * 1024 buffers per region * 10 regions = 10MB.

Configure your buffer caches based on the size of your static files, and the results from testing expected loads in a development environment. When determining the effect on performance, consider the balance of the buffer cache performance benefit versus the memory used.

For instructions on using the management CLI to configure buffer caches, see Configuring Buffer Caches in the JBoss EAP *Configuration Guide*.

9.2. JSP CONFIGURATION

There are JSP configuration options for Undertow servlet containers that provide optimizations for how JSP pages are compiled into Java bytecode.

generate-strings-as-char-arrays

If your JSPs contain a lot of **String** constants, enabling this option optimizes scriplets by converting the **String** constants to **char** arrays.

optimize-scriptlets

If your JSPs contain many **String** concatenations, enabling this option optimizes scriplets by removing **String** concatenation for every JSP request.

trim-spaces

If your JSPs contain a lot of white space, enabling this option trims the white space from HTTP requests and reduces HTTP request payload.

Configuring JSP Options

You can enable these Undertow JSP configuration options using the management console or management CLI.

- To enable them using the management console:
  1. Navigate to **Configuration → Subsystems → Web/HTTP - Undertow → Servlet/JSP →** and click **View**.
  2. Select the servlet container you want to configure and click **View**.
  3. Select **JSP**, and under **Attributes**, click **Edit**.
  4. Select the check boxes for the options you want to enable, and click **Save**.
- To enable them using the management CLI, use the following command:

```
/subsystem=undertow/servlet-
container=SERVLET_CONTAINER/setting=jsp/:write-
attribute(name=OPTION_NAME,value=true)
```

  For example, to enable **`generate-strings-as-char-arrays`** for the **`default`** servlet container, use the following command:

```
/subsystem=undertow/servlet-container=default/setting=jsp/:write-
attribute(name=generate-strings-as-char-arrays,value=true)
```

9.3. LISTENERS

Depending on your applications and environment, you can configure multiple listeners specific to certain types of traffic, for example, traffic on specific ports, and then configure options for each listener.

The following are selected performance-related options that can be configured on HTTP, HTTPS, and AJP listeners.

`max-connections`

The maximum number of concurrent connections that the listener can handle. By default this attribute is undefined, which results in unlimited connections.
You can use this option to set a ceiling on the number of connections a listener can handle, which might be useful to cap resource usage. In configuring this value you should consider your workload and traffic type. Also see **`no-request-timeout`** below.

`no-request-timeout`

The length of time in milliseconds that a connection is idle before it is closed. The default value is 60000 milliseconds (1 minute).
Tuning this option in your environment for optimal connection efficiency can help improve network performance. If idle connections are prematurely closed, there are overheads in re-establishing connections. If idle connections are open for too long, they unnecessarily use resources.

`max-header-size`

The maximum size of a HTTP request header, in bytes. The default is 1048576 (1024KB).
Limiting the header size can be useful to prevent certain types of denial of service attacks.

`buffer-pool`

Specifies the buffer pool in the **`io`** subsystem to use for the listener. By default, all listeners use the **`default`** buffer pool.

You can use this option to configure each listener to use a unique buffer pool, or have multiple listeners use the same buffer pool.

`worker`

The **`undertow`** subsystem relies on the **`io`** subsystem to provide XNIO workers. This option specifies the XNIO worker that the listener uses. By default, a listener uses the **`default`** worker in the **`io`** subsystem.

It might be useful to configure each listener to use a specific worker so you can assign different worker resources to certain types of network traffic.

Configuring Listener Options

You can configure listener options using the management console or management CLI.

- To configure them using the management console:
  1. Navigate to **Configuration** → **Subsystems** → **Web/HTTP - Undertow** → **HTTP** → and click **View**.
  2. Select the **HTTP Server** tab, then select the server you want to configure and click **View**.
  3. In the left menu, select type of listener to configure, for example **HTTP Listener**, and select the listener in the table.
  4. Under **Attributes**, click **Edit**.
  5. Modify the options you want to configure, and click **Save**.
- To configure them using the management CLI, use the following command:

```
/subsystem=undertow/server=SERVER_NAME/LISTENER_TYPE=LISTENER_NAME:w
rite-attribute(name=OPTION_NAME,value=OPTION_VALUE)
```

For example, to set **`max-connections`** to **`100000`** for the **`default`** HTTP listener in the **`default-server`** Undertow server, use the following command:

```
/subsystem=undertow/server=default-server/http-
listener=default:write-attribute(name=max-connections,value=100000)
```

CHAPTER 10. IO SUBSYSTEM TUNING

The **io** subsystem defines XNIO workers and buffer pools that are used by other JBoss EAP subsystems, such as Undertow and Remoting.

10.1. CONFIGURING WORKERS

You can create multiple separate workers that each have their own performance configuration and which handle different I/O tasks. For example, you could create one worker to handle HTTP I/O, and another worker to handle EJB I/O, and then separately configure the attributes of each worker for specific load requirements.

See the IO Subsystem Attributes appendix for the list of configurable worker attributes.

Worker attributes that significantly affect performance include **io-threads** which sets the total number of I/O threads that a worker can use, and **task-max-threads** which sets the maximum number of threads that can be used for a particular task. The defaults for these two attributes are calculated based on the server's CPU count.

See the JBoss EAP *Configuration Guide* for instructions on how to create and configure workers.

10.1.1. Monitoring Worker Statistics

You can view a worker's runtime statistics using the management CLI. This exposes worker statistics such as connection count, thread count, and queue size.

The following command displays runtime statistics for the **default** worker:

```
/subsystem=io/worker=default:read-resource(include-
runtime=true,recursive=true)
```

NOTE

The number of core threads, which is tracked by the **core-pool-size** statistic, is currently always set to the same value as the maximum number of threads, which is tracked by the **max-pool-size** statistic.

10.2. CONFIGURING BUFFER POOLS

A buffer pool in the **io** subsystem is a pooled NIO buffer instance that is used specifically for I/O operations. Like workers, you can create separate buffer pools which can be dedicated to handle specific I/O tasks.

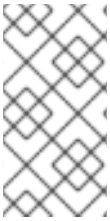
See the IO Subsystem Attributes appendix for the list of configurable buffer pool attributes.

The main buffer pool attribute that significantly affects performance is **buffer-size**. The default is calculated based on the RAM resources of your system, and is sufficient in most cases. If you are configuring this attribute manually, an ideal size for most servers is 16KB.

See the JBoss EAP *Configuration Guide* for instructions on how to create and configure buffer pools.

CHAPTER 11. JGROUPS SUBSYSTEM TUNING

For optimal network performance it is recommended that you use UDP multicast for JGroups in environments that support it.



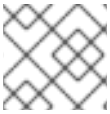
NOTE

TCP has more overhead and is often considered slower than UDP since it handles error checking, packet ordering, and congestion control itself. JGroups handles these features for UDP, whereas TCP guarantees them itself. TCP is a good choice when using JGroups on unreliable or high congestion networks, or when multicast is not available.

This chapter assumes that you have chosen your JGroups stack transport protocol (UDP or TCP) and communications protocols that JGroups cluster communications will use. See the JBoss EAP *Configuration Guide* for more information about cluster communication with JGroups.

11.1. MONITORING JGROUPS STATISTICS

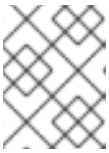
You can enable statistics for the **jgroups** subsystem to monitor JBoss EAP clustering using the management CLI or through JMX.



NOTE

Enabling statistics adversely affects performance. Only enable statistics when necessary.

1. Use the following command to enable statistics for a JGroups channel.

   NOTE

   In a managed domain, precede these commands with **/profile=*PROFILE_NAME***.

   ```
   /subsystem=jgroups/channel=CHANNEL_NAME:write-
   attribute(name=statistics-enabled,value=true)
   ```

   For example, use the following command to enable statistics for the default **ee** channel.

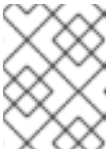
   ```
   /subsystem=jgroups/channel=ee:write-attribute(name=statistics-
   enabled,value=true)
   ```

2. Reload the JBoss EAP server.

   ```
   :reload
   ```

3. You can now see JGroups statistics using either the management CLI, or through JMX with a JVM monitoring tool:

   - To use the management CLI, use the **:read-resource(include-runtime=true)** command on the JGroups channel or protocol that you want to see the statistics for.



NOTE

In a managed domain, precede these commands with `/host=HOST_NAME/server=SERVER_NAME`.

For example:

- To see the statistics for the **ee** channel, use the following command:

```
/subsystem=jgroups/channel=ee:read-resource(include-
runtime=true)
```

- To see the statistics for the **FD_ALL** protocol in the **ee** channel, use the following command:

```
/subsystem=jgroups/channel=ee/protocol=FD_ALL:read-
resource(include-runtime=true)
```

- To connect to JBoss EAP using a JVM monitoring tool, see the Monitoring Performance chapter. You can see the statistics on JGroups MBeans through the JMX connection.

11.2. NETWORKING AND JUMBO FRAMES

Where possible, it is recommended that the network interface for JGroups traffic should be part of a dedicated Virtual Local Area Network (VLAN). This allows you to separate cluster communications from other JBoss EAP network traffic to more easily control cluster network performance, throughput, and security.

Another network configuration to consider to improve cluster performance is to enable jumbo frames. If your network environment supports it, enabling jumbo frames by increasing the Maximum Transmission Unit (MTU) can help boost network performance, especially in high throughput environments.

To use jumbo frames, all NICs and switches in your network must support it. See the Red Hat Customer Portal for instructions on enabling jumbo frames for Red Hat Enterprise Linux.

11.3. MESSAGE BUNDLING

Message bundling in JGroups improves network performance by assembling multiple small messages into larger bundles. Rather than sending out many small messages over the network to cluster nodes, instead messages are queued until the maximum bundle size is reached or there are no more messages to send. The queued messages are assembled into a larger message bundle and then sent.

This bundling reduces communications overhead, especially in TCP environments where there is a higher overhead for network communications.

Configuring Message Bundling

JGroups message bundling is configured using the **max_bundle_size** property. The default **max_bundle_size** is 64KB.

The performance improvements of tuning the bundle size depend on your environment, and whether more efficient network traffic is balanced against a possible delay of communications while the bundle is assembled.

Use the following management CLI command to configure **max_bundle_size**.

```
/subsystem=jgroups/stack=STACK_NAME/transport=TRANSPORT_TYPE/property=max_
bundle_size:add(value=BUNDLE_SIZE)
```

For example, to set **max_bundle_size** to **60K** for the default **udp** stack:

```
/subsystem=jgroups/stack=udp/transport=UDP/property=max_bundle_size:add(va
lue=60K)
```

11.4. JGROUPS THREAD POOLS

The **jgroups** subsystem uses its own thread pools for processing cluster communication. JGroups contains thread pools for **default**, **internal**, **oob**, and **timer** functions which you can configure individually. Each JGroups thread pool includes configurable attributes for **keepalive-time**, **max-threads**, **min-threads**, and **queue-length**.

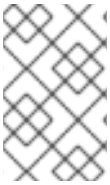
Appropriate values for each thread pool attribute depend on your environment, but for most situations the default values should suffice.

See the JBoss EAP *Configuration Guide* for instructions on how to configure JGroups thread pools.

11.5. JGROUPS SEND AND RECEIVE BUFFERS

The **jgroups** subsystem has configurable send and receive buffers for both UDP and TCP stacks.

Appropriate values for JGroups buffers depend on your environment, but for most situations the default values should suffice. It is recommended that you test your cluster under load in a development environment to tune appropriate values for the buffer sizes.



NOTE

Your operating system may limit the available buffer sizes and JBoss EAP may not be able to use its configured buffer values. See Resolving Buffer Size Warnings in the JBoss EAP *Configuration Guide*.

See the JBoss EAP *Configuration Guide* for instructions on how to configure JGroups send and receive buffers.

CHAPTER 12. TRANSACTIONS SUBSYSTEM TUNING

If your environment uses XA distributed transactions, you can tune the transaction manager's log store for better performance.

The default transaction log store uses a simple file store. For XA transactions this type of log store can be inefficient, as it creates one system file for each transaction log. Especially for XA transactions, a journal store is much more efficient as it uses a journal that consists of one file for all transactions.

For better XA transaction performance, it is recommended that you use a journal log store. For Red Hat Enterprise Linux systems, you can additionally enable asynchronous I/O (AIO) on the journal store to further improve performance.

NOTE

For Red Hat Enterprise Linux systems, if you are enabling asynchronous I/O (AIO) on the journal store, ensure that the **`libaio`** package is installed.

Enable the Journal Log Store Using the Management Console

1. Navigate to **Configuration** → **Subsystems** → **Transactions** → and click **View**.
2. Select the **Store** tab, and click **Edit**.
3. Select the **Use journal store** check box.
4. **Optional**: For Red Hat Enterprise Linux systems, select the **Journal store enable async io** check box.
5. Click **Save**.

Enable the Journal Log Store Using the Management CLI

1. To enable the journal log store using the management CLI, use the following command:

```
/subsystem=transactions:write-attribute(name=use-journal-
store,value=true)
```

2. **Optional**: For Red Hat Enterprise Linux systems, use the following command to enable journal log store asynchronous I/O:

```
/subsystem=transactions:write-attribute(name=journal-store-enable-
async-io, value=true)
```

APPENDIX A. REFERENCE MATERIAL

A.1. DATASOURCE STATISTICS

Table A.1. Core Pool Statistics

Name	Description
ActiveCount	The number of active connections. Each of the connections is either in use by an application or available in the pool.
AvailableCount	The number of available connections in the pool.
AverageBlockingTime	The average time spent blocking on obtaining an exclusive lock on the pool. This value is in milliseconds.
AverageCreationTime	The average time spent creating a connection. This value is in milliseconds.
AverageGetTime	The average time spent obtaining a connection.
AveragePoolTime	The average time that a connection spent in the pool.
AverageUsageTime	The average time spent using a connection.
BlockingFailureCount	The number of failures trying to obtain a connection.
CreatedCount	The number of connections created.
DestroyedCount	The number of connections destroyed.
IdleCount	The number of connections that are currently idle.
InUseCount	The number of connections currently in use.
MaxCreationTime	The maximum time it took to create a connection. This value is in milliseconds.
MaxGetTime	The maximum time for obtaining a connection.
MaxPoolTime	The maximum time for a connection in the pool.
MaxUsageTime	The maximum time using a connection.
MaxUsedCount	The maximum number of connections used.
MaxWaitCount	The maximum number of requests waiting for a connection at the same time.

Name	Description
MaxWaitTime	The maximum time spent waiting for an exclusive lock on the pool.
TimedOut	The number of timed out connections.
TotalBlockingTime	The total time spent waiting for an exclusive lock on the pool. This value is in milliseconds.
TotalCreationTime	The total time spent creating connections. This value is in milliseconds.
TotalGetTime	The total time spent obtaining connections.
TotalPoolTime	The total time spent by connections in the pool.
TotalUsageTime	The total time spent using connections.
WaitCount	The number of requests that had to wait to obtain a connection.
XACommitAverageTime	The average time for an XAResource commit invocation.
XACommitCount	The number of XAResource commit invocations.
XACommitMaxTime	The maximum time for an XAResource commit invocation.
XACommitTotalTime	The total time for all XAResource commit invocations.
XAEndAverageTime	The average time for an XAResource end invocation.
XAEndCount	The number of XAResource end invocations.
XAEndMaxTime	The maximum time for an XAResource end invocation.
XAEndTotalTime	The total time for all XAResource end invocations.
XAForgetAverageTime	The average time for an XAResource forget invocation.
XAForgetCount	The number of XAResource forget invocations.
XAForgetMaxTime	The maximum time for an XAResource forget invocation.
XAForgetTotalTime	The total time for all XAResource forget invocations.
XAPrepareAverageTime	The average time for an XAResource prepare invocation.

Name	Description
XAPrepareCount	The number of XAResource prepare invocations.
XAPrepareMaxTime	The maximum time for an XAResource prepare invocation.
XAPrepareTotalTime	The total time for all XAResource prepare invocations.
XARecoverAverageTime	The average time for an XAResource recover invocation.
XARecoverCount	The number of XAResource recover invocations.
XARecoverMaxTime	The maximum time for an XAResource recover invocation.
XARecoverTotalTime	The total time for all XAResource recover invocations.
XARollbackAverageTime	The average time for an XAResource rollback invocation.
XARollbackCount	The number of XAResource rollback invocations.
XARollbackMaxTime	The maximum time for an XAResource rollback invocation.
XARollbackTotalTime	The total time for all XAResource rollback invocations.
XAStartAverageTime	The average time for an XAResource start invocation.
XAStartCount	The number of XAResource start invocations.
XAStartMaxTime	The maximum time for an XAResource start invocation.
XAStartTotalTime	The total time for all XAResource start invocations.

Table A.2. JDBC Statistics

Name	Description
PreparedStatementCacheAccessCount	The number of times that the statement cache was accessed.
PreparedStatementCacheAddCount	The number of statements added to the statement cache.
PreparedStatementCacheCurrentSize	The number of prepared and callable statements currently cached in the statement cache.
PreparedStatementCacheDeleteCount	The number of statements discarded from the cache.
PreparedStatementCacheHitCount	The number of times that statements from the cache were used.

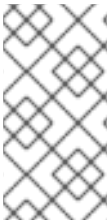
Name	Description
PreparedStatementCacheMissCount	The number of times that a statement request could not be satisfied with a statement from the cache.

A.2. RESOURCE ADAPTER STATISTICS

Table A.3. Resource Adapter Statistics

Name	Description
ActiveCount	The number of active connections. Each of the connections is either in use by an application or available in the pool
AvailableCount	The number of available connections in the pool.
AverageBlockingTime	The average time spent blocking on obtaining an exclusive lock on the pool. The value is in milliseconds.
AverageCreationTime	The average time spent creating a connection. The value is in milliseconds.
CreatedCount	The number of connections created.
DestroyedCount	The number of connections destroyed.
InUseCount	The number of connections currently in use.
MaxCreationTime	The maximum time it took to create a connection. The value is in milliseconds.
MaxUsedCount	The maximum number of connections used.
MaxWaitCount	The maximum number of requests waiting for a connection at the same time.
MaxWaitTime	The maximum time spent waiting for an exclusive lock on the pool.
TimedOut	The number of timed out connections.
TotalBlockingTime	The total time spent waiting for an exclusive lock on the pool. The value is in milliseconds.
TotalCreationTime	The total time spent creating connections. The value is in milliseconds.
WaitCount	The number of requests that had to wait for a connection.

A.3. IO SUBSYSTEM ATTRIBUTES



NOTE

Attribute names in these tables are listed as they appear in the management model, for example, when using the management CLI. See the schema definition file located at ***`EAP_HOME`*`/docs/schema/wildfly-io_2_0.xsd`** to view the elements as they appear in the XML, as there may be differences from the management model.

Table A.4. worker Attributes

Attribute	Default	Description
io-threads		The number of I/O threads to create for the worker. If not specified, the number of threads is set to the number of CPUs × 2.
stack-size	0	The stack size, in bytes, to attempt to use for worker threads.
task-keepalive	60000	The number of milliseconds to keep non-core task threads alive. *This attribute should not be used as it is currently ignored.*
task-max-threads		The maximum number of threads for the worker task thread pool. If not specified, the maximum number of threads is set to the number of CPUs × 16, taking the **`MaxFileDescriptorCount`** JMX property, if set, into account.

Table A.5. buffer-pool Attributes

Attribute	Default	Description
buffer-size		The size, in bytes, of each buffer slice. If not specified, the size is set based on the available RAM of your system: • 512 bytes for less than 64 MB RAM • 1024 bytes (1 KB) for 64 MB - 128 MB RAM • 16384 bytes (16 KB) for more than 128 MB RAM For performance tuning advice on this attribute, see Configuring Buffer Pools in the JBoss EAP *Performance Tuning Guide*.

Attribute	Default	Description
buffers-per-slice		How many slices, or sections, to divide the larger buffer into. This can be more memory efficient than allocating many separate buffers. If not specified, the number of slices is set based on the available RAM of your system: • 10 for less than 128 MB RAM • 20 for more than 128 MB RAM
direct-buffers		Whether the buffer pool uses direct buffers, which are faster in many cases with NIO. Note that some platforms do not support direct buffers.

Revised on 2018-10-11 12:33:03 UTC