PingFederate®

Version 8.x

Deployment Note

Performance Tuning Guide

Ping Identity

PingFederate *Performance Tuning Guide*
Version 4.0 (for PingFederate 8.x)

July, 2015

About Development and Deployment Notes

The PingFederate software development and specialty teams provide supplemental documentation periodically to give our customers insight into configuration, deployment, prototypes, or use cases that are not part of the main product documentation or basic product deployment.

Disclaimer

This document is proprietary and not for general publication. It may be provided to customers for informational purposes only, and the information herein is subject to change without notice. Ping Identity does not provide any warranties and specifically disclaims any liability in connection with this document.

Note that Ping Identity may not provide support for any sample configurations provided in this document. The variability inherent among security environments prevents full testing and support for all possible platform configurations. If you need special assistance or would like to inquire about implementation or support programs, please contact Ping Identity Support (ping.force.com/Support).

Contact Information

Ping Identity Corporation
1001 17th Street, Suite 100
Denver, CO 80202
U.S.A.

Phone: 877.898.2905 (+1 303.468.2882 outside North America)
Fax: 303.468.2909
E-mail: info@pingidentity.com
Web Site: www.pingidentity.com

Contents

Introduction

The "out-of-the-box" configuration for PingFederate is acceptable for most small size deployments; however, for mission-critical, high-transaction volume deployments additional tuning is recommended. Fine-tuning a few simple application and system level settings will enable you to achieve maximum performance out of the hardware chosen for your deployment.

This guide addresses several areas of tuning such as concurrency, memory, logging and Java™ specific tuning options. The guide is not designed as a "one-size-fits-all" set of instructions to optimize PingFederate, but more as a checklist of suggestions for areas of the product that can be tuned to improve performance, and any tradeoffs associated with those changes.

For ultimate reassurance that any fine-tuned settings will meet your expectations, performance testing in a lab environment is recommended. A companion document to this guide, the PingFederate Capacity Planning Guide, is available as a reference for sizing. It is based on performance tests performed by Ping Identity in lab environments to simulate customer deployments.

Logging

Logging is a useful feature for tracking various aspects of the system's operation. Logging requires a certain amount of system resources, which affects the system's overall performance. Of particular expense is the actual writing to the log files. PingFederate 8.x uses the high-performance asynchronous logger from Log4J2 to minimize the performance impact of logging runtime and administrative events, including status and error messages that can be used for troubleshooting. Audit information is logged synchronously to preserve transactional integrity.

Although the bulk of logging is executed asynchronously, decreasing the amount of information written to log files always provides the best possible performance. To learn more about configuring logging see the "Managing Log Files" section of the Administrator's Manual. For an example of a minimal logging configuration file, see the `terse.example.log4j2.xml` file in the `pingfederate/server/default/conf` directory.

Concurrency

In most cases, the more requests that are processed concurrently increases the number of requests that are processed over all. Given the appropriate amount of hardware, processing *N* requests concurrently is typically faster than processing *N* requests serially.

In PingFederate there are two main pools of threads that control the level of concurrent user requests: Acceptor Threads and Server Threads. Acceptor threads receive the HTTPS/SSL requests, and pass those requests on to available Server threads to be processed.

Acceptor Thread Pool

By default, the runtime SSL port (9031) is configured to run 2 acceptor threads processing a request queue that can hold a maximum of 512 connection requests. The acceptor threads pass the request to a number of non-blocking I/O selectors (by default 1 per CPU core) that reads the request.

For optimal performance, particularly in larger deployments, we recommend switching to a purely non-blocking I/O model.

Tuning the Acceptor Thread Pool

1. Shut down PingFederate if it is running.
2. Edit the file *<Install Dir>*\pingfederate\etc\jetty-runtime.xml, where *<Install Dir>* is the directory where you installed PingFederate.
3. Find the following section:

```
<Call id="httpsPrimaryConnector" name="addConnector">
<Arg>
<New
class="com.pingidentity.appserver.jetty.server.connector.ServerConnector
">
<Arg name="server"><Ref refid="RuntimeServer" /></Arg>
<Arg name="acceptors" type="int">2</Arg>
<Arg name="selectors" type="int"><Property name="ssl.selectors"
default="-1"/></Arg>
<Arg name="factories">
<Array type="org.eclipse.jetty.server.ConnectionFactory">
<Item>
<New class="org.eclipse.jetty.server.SslConnectionFactory">
<Arg name="next">http/1.1</Arg>
<Arg name="sslContextFactory"><Ref
refid="primarySSLContextFactory"/></Arg>
</New>
</Item>
<Item>
<New class="org.eclipse.jetty.server.HttpConnectionFactory">
<Arg name="config"><Ref refid="sslHttpConfig"/></Arg>
</New>
</Item>
</Array>
</Arg>
<Set name="host"><SystemProperty name="pf.engine.bind.address"
default="0.0.0.0"/></Set>
<Set name="port"><SystemProperty name="pf.https.port" default="9031"
/></Set>
<Set name="idleTimeout">30000</Set>
<Set name="soLingerTime"><Property name="https.soLingerTime" default="-
1"/></Set>
<Set name="acceptorPriorityDelta"><Property
name="ssl.acceptorPriorityDelta" default="0"/></Set>
<Set name="selectorPriorityDelta"><Property
name="ssl.selectorPriorityDelta" default="0"/></Set>
<Set name="acceptQueueSize">512</Set>
</New>
</Arg>
</Call>
```

4. To switch to purely non-blocking I/O, disable Acceptor threads by changing the value of the `acceptors` entry to `0`.
5. Set the value of `selectors` to `1`.

   If the queue reaches its maximum size, additional requests will receive a connection (refused) error. If this occurs in your environment you can increase the value of the `acceptQueueSize` attribute.

Server Thread Pool

The number of concurrent requests that PingFederate is capable of processing is controlled by the server container's Server Threads pool configuration. When an HTTPS request is received, an Acceptor thread picks an idle Server thread from the pool and dispatches it to process the request.

The default configuration defines a minimum of 15 concurrent threads, with 50 for the "low" and "maximum" number of threads. This means that initially the server creates 15 threads for the pool. If that limit is exceeded, it generates up to 50 threads. This configuration effectively limits the total concurrency of the server to a maximum of 50 requests. If all threads in the pool are processing, additional requests accumulate in the Acceptor Queue and wait until a free Server thread is available from the pool.

Tuning the Server Thread Pool

When tuning the server thread pool, it is best to size the system based on expected user load, setting the minimum number of threads to between 75% and 100% of the number of requests you expect the system to handle most often. Set the maximum to between 25% and 50% higher than the minimum to handle load spikes. Testing has shown that PingFederate performs quite well when the server thread pool is sized between 25 and 50 server threads per available CPU core, assuming sufficient memory (see Memory section for memory-tuning details). Note that this is guidance and should not necessarily be scaled directly on larger systems. For example, if you are running PingFederate on a system with 24 CPU cores, it does not make sense to size the thread pool at a minimum of 600 threads and a maximum of 1200, unless you expect to normally handle at least 800 concurrent requests.

1. Shut down PingFederate if it is running.

2. Edit the file `<Install Dir>\pingfederate\etc\jetty-runtime.xml`, where *<Install Dir>* is the directory where you installed PingFederate.

3. Find the following section:

```
<Get name="ThreadPool">
    <Set name="minThreads" type="int">10</Set>
    <Set name="maxThreads" type="int">200</Set>
    <Set name="detailedDump">false</Set>
</Get>
```

4. Modify it using the following guidelines:
   - Set `minThreads` to (Available CPU cores * 25)
   - Set `maxThreads` to (Available CPU cores * 50)

For example, if your PingFederate system has 1 CPU with 4 cores, the total available CPU/Cores is 4. The configuration would be:

```
<Get name="ThreadPool">
     <Set name="minThreads" type="int">100</Set>
```

```
        <Set name="maxThreads" type="int">200</Set>
        <Set name="detailedDump">false</Set>
    </Get>
```

Data Stores

LDAP and JDBC data stores use connection pooling to improve the performance and efficiency of communicating with external systems. Connection pools improve efficiency by maintaining persistent connections to the Database or LDAP server thus preventing the expense of creating the connection on demand. Connection pools also allow more control over the load placed on the backend server.

Much like the server thread pool, for optimal performance we want to have enough connections to handle most to all of the requests concurrently. It may not be necessary to have a connection available for every concurrent request received by the server but having too few available will cause requests to wait when accessing LDAP or Database resources. For optimal performance connection pools should be sized large enough to handle between 50% and 100% of the number of concurrent requests the server is expected to encounter most often.

Of course, it is also very important to understand the capacity and limits of the LDAP or Database server in use and size the connection pool accordingly. Sizing the connection pool to be larger than the number of concurrent requests the backend server can handle will allow PingFederate to flood the data store and not improve performance.

Tuning the Connection Pools

Database

In the “Database Config” tab (of the Data Store screen) connection pooling configuration is accessed via the “Advanced…” button in the lower right corner of the screen. For optimal performance you will want to set the “Minimum Pool Size” to 50% of the maxThreads value (in the Jetty server configuration) and “Maximum Pool Size” to between 75% and 100% of the maxThreads value. Assuming, of course, the backend database server is capable of handling these loads, if not appropriate limits should be applied.

LDAP

In the `LDAP Configuration` tab of the `Data Store` page, connection pooling configuration is accessed via the `Advanced` button in the lower right corner of the screen. For optimal performance you will want to set the `Minimum Connections` to 50% of the `maxThreads` value (in the Jetty server configuration) and `Maximum Connections` to between 50% and 75% of the `maxThreads` value. Assuming, of course, the backend LDAP server is capable of handling these loads; if it cannot, appropriate limits should be applied.

Note: As of PingFederate version 7.2 LDAP connection pooling is used for bind requests. Separate connection pools are created for bind requests and for search requests. This dramatically improves performance of LDAP authentications, especially when using LDAPS, as bind requests no longer require unique connections. Each pool (bind and search) is sized individually per the values set for minimum and maximum connections.

Caveats

It is important to note that this is not a magic formula, but rather guidance for optimizing the concurrency of your PingFederate deployment. On a large multi-core system, a thread pool that is too

small will under-utilize the available processor resources. A thread pool that is too large can cause the system to become flooded and unusable.

A good target for the CPU is between 60%-80% utilization when under nominal (reasonably expected) user load. This way, CPU resources are not under-utilized while still allowing room for occasional load spikes. The level of concurrency in PingFederate may need to be decreased (or even increased) depending on the system's configuration (Integration Kits and/or Adapters in use) and available memory (see the Memory section of this document), as well as other processes competing for resources. All deployments are different. This should serve as a guideline of where to start when tuning the server.

Memory

After the CPU, memory is the most important resource in your deployment. In the previous section we looked at how to configure PingFederate to support more concurrent requests. With increased concurrency comes an increase in the memory required. Moreover, PingFederate is a Java™ application, and although this document is not to be considered a guide to garbage collection theory or ergonomics, consideration must be given to tuning that affects, or is affected by, garbage collection.

PingFederate will automatically set many of its own memory settings based on the memory that is currently available on the system, not the total memory of the system.

JVM Heap

Probably the most important tuning we can apply to the Java™ Virtual Machine (JVM) is to configure how much heap memory (used for allocations at runtime) to use. If the demands on PingFederate require more memory than is *currently* available, the JVM must grow the heap (if it can) or perform garbage collection to provide memory to allocate. Resizing the heap and garbage collecting can be expensive processes, and thus detrimental to performance. Sizing the heap to ensure an adequate amount of memory is available but still manageable to garbage collection is important in optimizing overall performance.

By default, the PingFederate launch script sets the size of the Java heap based on the amount of free memory detected at start-up time. In the case of the Windows™ service, heap size configuration is set based on the amount of free memory available when the service is installed. If less than 2 GB is free, the heap is set to a minimum of 256 MB, growing to a maximum of 1GB. This means that when PingFederate is launched, 256 MB are available for newly created objects. If more than 256 MB are required, the JVM garbage collects and, if necessary, expands the heap until it reaches 1 GB. At this point, the JVM must invoke garbage collection in order to free up released memory to be re-used. If at least 2 GB of free memory is detected, the Java heap is set to 1.5 GB. If 2.5 GB or more of free memory is detected, the Java heap is set to 2 GB. Again, once the heap is full, the JVM will initiate garbage collection in order to free up memory to be re-used.

Tuning the JVM Heap

To specify the initial (minimum) size of the JVM heap, we use the `minimumHeap` variable. To specify the maximum size of the heap, we use the `maximumHeap` variable. These variables are found in the `run.bat` file for Windows and `run.sh` file for Unix and Linux (in `pingfederate/server/bin`). In the case of the Windows service, the minimum heap size is defined by the wrapper.java.initmemory variable, and the maximum heap size is defined by wrapper.java.maxmemory variable found in pingfederate/sbin/wrapper.

Adjusting the Heap Size

As stated previously, the JVM can grow the heap from the value of the minimum heap variable to the value of the maximum heap variable. Growing the heap however, has expense to it. Requesting memory from the operating system is not an entirely inexpensive exercise, plus the JVM must now reorganize the heap to account for the memory being added. If conserving memory in your deployment is important, then setting a lower value for the minimum heap than that of the maximum heap ensures that you are not reserving memory that you are not specifically using. If, however, you have enough memory such that a certain amount is easily earmarked for the PingFederate server, we recommend that you adjust the size of the heap by setting minimum heap and maximum heap to the same value. This allows the JVM to reserve its entire heap and decrease the amount of resizing that the JVM needs to perform if the amount of memory being used exceeds the value of minimum heap.

Young Generation Bias

Without getting into too much detail on the generational memory management model in Java™, the basic principal is that new objects are created in the "young" generation and are garbage collected when they are no longer used. If an object is created, and a reference is maintained, that object is eventually moved to the "old" generation.

The processing model of PingFederate is mostly geared towards short-lived transactions (Single Sign On and Token Processing) and not long held, interactive, user sessions. As such, most of the objects that are created are relatively short-lived. It doesn't make sense to "promote" short-lived objects to the old generation because they are probably not going to be needed for long anyway. The problem is that when the young generation fills up, space must be made for new objects. So objects that are in the young generation and still in use must be moved to the old generation, which has a cost. Moreover, those objects will eventually be “garbage” and need to be collected from the old generation.

The old generation is typically more expensive to clean up than the young generation because the old generation is cleaned only during a full garbage collection (meaning that the JVM has almost reached the value of `-Xmx` and the entire heap must be cleaned). However, the young generation is garbage collected more frequently and with multiple threads by default (on multi-core systems), so the pauses for collections are shorter.

By default, the JVM tends to size the generations biased to the old generation, giving it most of the total space of the heap. This means more frequently moving objects from the young generation into the old generation to make space for new objects, and more frequent "full" collections as the old generation fills up. By telling the JVM to provide more memory to the "young" generation, we reduce the frequency of the more costly full collections.

To do this we can either specify fixed values for the size of the young generation or modify the ratio of young generation to old generation. To specify a fixed value for the young generation, use the `-XX:NewSize=` and `-XX:MaxNewSize=` arguments. These arguments are to the young generation what `-Xms` and `-Xmx` are to the entire heap. Like `-Xms`, `-XX:NewSize=` defines the initial (or minimum) size. And, like `-Xmx`, `-XX:MaxNewSize=` defines the maximum size. The same reasoning applies for adjusting these values.

To specify a ratio between the old and new generation size, use the `-XX:NewRatio=` argument. For example, setting `-XX:NewRatio=3` means that the ratio between the young and old generation is 1:3. Put another way, the size of the young generation is one fourth of the total heap size.

In a mostly short-lived object environment, it is recommended that 50-60% of the heap be given to the young generation.

Examples

- Fixed Heap size:
  `-Xms2048m -Xmx-2048m`
- Fixed Heap Size with 50% Young Generation Bias using NewSize:
  `-Xms2048m -Xmx-2048m -XX:NewSize=1024m -XX:MaxNewSize=1024m`
- Fixed Heap Size with 50% Young Generation Bias using NewRatio:
  `-Xms2048m -Xmx-2048m -XX:NewRatio=1`

Additional JVM Options

The JVM comes with a wide variety of tuning options. Although the virtual machine should configure itself to run the best it can in most situations; it is sometimes advantageous to configure it for a specific application. In this section we look at a few additional options that can be applied to potentially improve the operation of PingFederate in your deployment. In all cases, it is recommended that you test these options to ensure that they do benefit your deployment before enabling them in a production environment. To add additional JVM arguments, see Adding JVM Command Line Options to PingFederate on page 11 for instructions.

Parallel Compacting Garbage Collector

By default, the "server" HotSpot™ Java Virtual Machine selects the Parallel Scavenge (Copying) Collector for the Young Generation, and the Serial Compacting Collector for the Old Generation. PingFederate will enable the Parallel Garbage Collector automatically if more than 1 CPU is detected on the system.

Both the serial and parallel compacting collectors are "stop-the-world" collectors. The key difference is that the parallel collector uses multiple threads to perform the collection and compaction whereas the serial collector is limited to a single thread. By default, the number of threads used for collection and compaction is equal to the number of CPU cores available on the system. This can be adjusted by modifying the `garbageCollector` variable adding `-XX:ParallelGCThreads=<N>`. For example: `garbageCollector=''-XX:+UseParallelOldGC-XX:ParallelGCThreads=<N>''`

Concurrent Mark Sweep Garbage Collector

Also known as the "concurrent low pause" collector, the Concurrent Mark Sweep (CMS) collector is designed to collect the old (or "tenured") generation while the application threads are handling requests. This is advantageous when PingFederate is deployed on systems with abundant CPU resources as the JVM can use some of the CPU cores to collect garbage while other CPU cores can be used for handling user requests. With little to no "stop-the-world" effect on application threads, a user is unlikely to perceive poor response time when garbage collection occurs. To enable this collector, set the `garbageCollector` variable value to `-XX:+UseConcMarkSweepGC`.

One downside to the CMS collector is that it is not a compacting collector. A compacting collector removes objects from the generation and then reorganizes it such that all free memory is in a contiguous chunk. Since moving objects can be an expensive process, requiring that all application threads be paused, the CMS collector does not do it. This leads to memory fragmentation, where free and used space are scattered throughout the tenured generation's space. Fragmentation can cause the JVM to work harder to find the right size space to put newly promoted objects, which can decrease performance. To minimize the impact of fragmentation, you can increase the size of your heap by 25%-50% above that suitable for use with the non-concurrent collector.

Modifying and Adding JVM Command Line Options to PingFederate

Launching as a Console Application

Scripts found in the <*Install Dir*>\pingfederate\bin directory control how PingFederate launches. For Windows deployments, the script used is `run.bat`, for Unix or Linux deployments the script is `run.sh`. These scripts construct the JVM command line to launch PingFederate and are the location where any modifications or additions to the JVM command line are made.

JVM arguments are space-delimited. Arguments need not appear in any specific order, however it is recommended that you edit specific entries in the script where noted below. We also *highly* recommend that you make a backup copy of the launch script prior to making any changes so that you can revert to the default configuration as necessary.

If you want to override the resource-based heap size and/or garbage collector configuration determined at startup, specify values in the `override` section of the launch scripts.

Windows

Edit the `run.bat` file and locate the section that contains the following:

```
REM Sun JVM Optimizations based on system resources. Modify as
appropriate.

SET minimumHeap=-Xms256m
SET maximumHeap=-Xmx1024m
SET minimumPermSize=-XX:PermSize=96m
SET minimumNewSize=
SET maximumNewSize=
SET garbageCollector=
```

For the heap size, modify the values of `minimumHeap=-Xms` and `maximumHeap=-Xmx` to the values you intend to use. Valid unit qualifiers are `k` for kilobytes, `m` for megabytes and `g` for gigabytes. Note that `maximumHeap=-Xmx1536m` and `maximumHeap=-Xmx1.5g` (for example) are equivalent.

To modify other heap sizing, or the garbage collector configuration, simply uncomment (remove the `REM` qualifier) and adjust the value of the arguments you want to change.

To add additional arguments to the JVM command line, you must create a script variable to contain the text of the JVM argument, then append the new script variable to the script variable that makes up the final JVM command line.

The following two lines are an example of how you would add the Aggressive Options flag to the JVM command line:

```
set agressiveOpts=-XX:+AggressiveOpts
set PF_JAVA_OPTS=%PF_JAVA_OPTS% %agressiveOpts%
```

Linux/Solaris

Edit the `run.sh` file and locate the section that contains the following:

```
#JVM Optimizations (for non MacOSX) systems based on system resources.
Modify as appropriate.

totalMem=0
numberCPUCores=1
minimumHeap="-Xms256m"
maximumHeap="-Xmx1024m"
minimumPermSize="-XX:PermSize=96m"
minimumNewSize=""
maximumNewSize=""
garbageCollector=""
```

For heap size, modify the values of `minimumHeap=-Xms` and `maximumHeap=-Xmx` to the values you intend to use. Valid unit qualifiers are `k` for kilobytes, `m` for megabytes and `g` for gigabytes. Note that `maximumHeap=-Xmx1536m` and `maximumHeap=-Xmx1.5g` are equivalent.

To modify other heap sizing or the garbage collector configuration, simply uncomment (remove the `#` qualifier) and adjust the value of the argument(s) you want to change.

To add additional arguments to the JVM command line, you must create a script variable to contain the text of the JVM argument, then append the new script variable to the script variable that makes up the final JVM command line.

The following two lines are an example of how you would add the "Aggressive Options" flag to the JVM command line:

```
aggressiveOpts=''-XX:+AggressiveOpts''
JAVA_OPTS="$JAVA_OPTS $aggressiveOpts''
```

Running as a Windows Service

If you have configured PingFederate to run as a Windows service, the `PingFederateService.conf` file located in the `<Install Dir>\pingfederate\sbin\wrapper` directory controls the JVM command line. The values defined in this file were determined (based on available resources) at the time the service was installed.

To change heap size, edit the `PingFederateService.conf` file and locate the following lines:

```
# Initial Java Heap Size (in MB)
wrapper.java.initmemory=256
# Maximum Java Heap Size (in MB)
wrapper.java.maxmemory=1024
```

Modify the value of `initmemory` to be the value you would apply for `-Xms` and, similarly, set `maxmemory` to what you would apply for `-Xmx`. Note that units are strictly megabytes in this instance. Due to the way the service wrapper constructs the JVM command line, PingFederate is unable to use the "AggressiveHeap" option.

For any other JVM arguments, add a line like the following:

```
wrapper.java.additional.N=-argument
```

`N` represents the number of additional arguments added to the JVM command line. By default, there are eight (a ninth is provided, but commented out, to demonstrate debugging functionality). The string "`-argument''` represents the new argument you are adding.

For example:

```
wrapper.java.additional.9=-XX:+AggressiveOpts
```

If you encounter an issue launching PingFederate after adding additional arguments to the script, either delete your entry or comment it out by preceding the line with the comment character #. As in:

```
#wrapper.java.additional.9=-XX:+AggressiveOpts
```

References

Memory Management

http://www.oracle.com/technetwork/java/javase/memorymanagement-whitepaper-150215.pdf

Hotspot™ JVM Command line arguments

http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html