ScienceLogic

Tomcat Monitoring PowerPack Guide

December 11, 2019

Contents

Introduction

What does the PowerPack Monitor?

This PowerPack provides monitoring for Tomcat Server configuration and performance metrics. The PowerPack uses two different methods to pull data from a Tomcat Server and you can choose any one method (dynamic app) as you see fit.

Method #1: Connect to the Tomcat Webserver Status page, OR

Method#2: Connect to the SNMP agent on the Tomcat server's JVM.
https://docs.oracle.com/javase/7/docs/technotes/guides/management/snmp.html

Supported Tomcat Versions & Operating Systems

The powerpack is not dependent on any operating system since it connects to the server-status page or connects via SNMP.

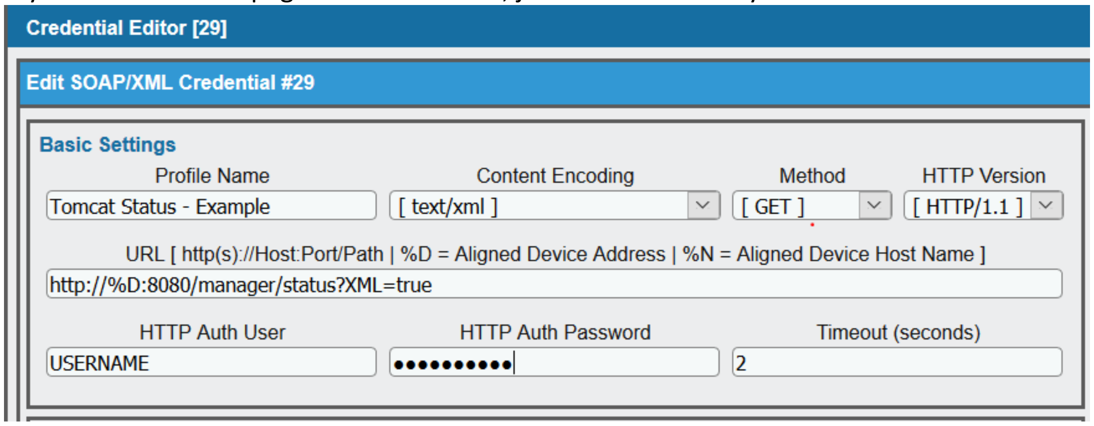
This version of the PowerPack has been certified with Tomcat Version 8.5 and 9.0 but it is expected to run on other versions as well.

Prerequisites

- For **Method 1** which connects via server-status page:
    - IP address of the server where Tomcat is running.
    - Port on which Tomcat is running.
    - Note: If your Tomcat Server server-status page is protected by a username/password, then create credentials as follows
- For **Method 2** which connects via snmp
    - IP address of the server where Tomcat is running.
    - SNMP Port (default is UDP 161); ensure this port is open from the Collector to the server.
    - SNMP read community string
    - For detailed instruction on how to set up your JVM to report metrics via SNMP, please refer to this webpage: https://docs.oracle.com/javase/7/docs/technotes/guides/management/snmp.html

Creating Credentials

- To connect to your server status page, create a soap/xml credentials in SL
    - Use the Example credential provided with the pack as a reference and use the "save-as" option to save a copy of the credential for you to edit. Please avoid modifying the example credential as a matter of best practice.
- If your server-status page is not restricted, just create a dummy credential



Credential Editor [29]

Edit SOAP/XML Credential #29

Basic Settings

Profile Name	Content Encoding	Method	HTTP Version
Tomcat Status - Example	[text/xml]	[GET]	[HTTP/1.1]

URL [http(s)://Host:Port/Path \| %D = Aligned Device Address \| %N = Aligned Device Host Name]

http://%D:8080/manager/status?XML=true

HTTP Auth User	HTTP Auth Password	Timeout (seconds)
USERNAME	••••••••••	2

*If using the SNMP option, please create as standard SNMP credential.

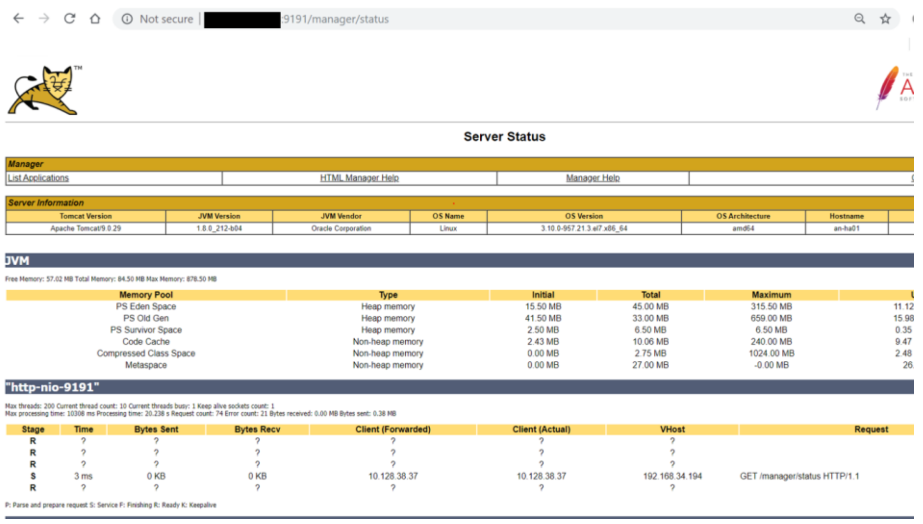
Collected Metrics

Overview

Depending on the method of collection, we collect different sets of metrics.

1) Server status page - http://<ip_address>:<port_number>/server-status

- Below is the sample server-status page from a Tomcat server.



Not secure | [redacted]:9191/manager/status

Server Status

Manager		
List Applications	HTML Manager Help	Manager Help

Server Information

Tomcat Version	JVM Version	JVM Vendor	OS Name	OS Version	OS Architecture	Hostname
Apache Tomcat/9.0.29	1.8.0_212-b04	Oracle Corporation	Linux	3.10.0-957.21.3.el7.x86_64	amd64	an-ha01

JVM

Free Memory: 57.02 MB Total Memory: 84.50 MB Max Memory: 878.50 MB

Memory Pool	Type	Initial	Total	Maximum	
PS Eden Space	Heap memory	15.50 MB	45.00 MB	315.50 MB	11.12
PS Old Gen	Heap memory	41.50 MB	33.00 MB	659.00 MB	15.98
PS Survivor Space	Heap memory	2.50 MB	6.50 MB	6.50 MB	0.35
Code Cache	Non-heap memory	2.43 MB	10.06 MB	240.00 MB	9.47
Compressed Class Space	Non-heap memory	0.00 MB	2.75 MB	1024.00 MB	2.48
Metaspace	Non-heap memory	0.00 MB	27.00 MB	-0.00 MB	26.

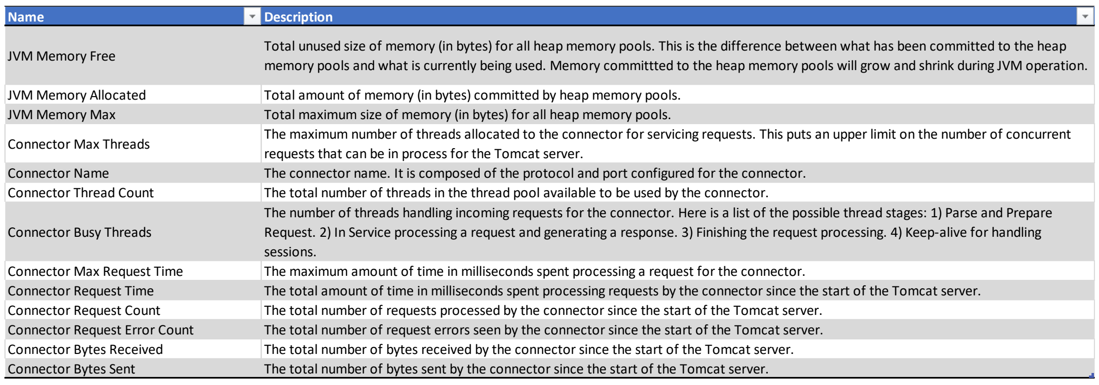
"http-nio-9191"

Max threads: 200 Current thread count: 10 Current threads busy: 1 Keep alive sockets count: 1
Max processing time: 10308 ms Processing time: 20.238 s Request count: 74 Error count: 21 Bytes received: 0.00 MB Bytes sent: 0.38 MB

Stage	Time	Bytes Sent	Bytes Recv	Client (Forwarded)	Client (Actual)	VHost	Request
R	?	?	?	?	?	?	
R	?	?	?	?	?	?	
R	?	?	?	?	?	?	
S	3 ms	0 KB	0 KB	10.128.38.37	10.128.38.37	192.168.34.194	GET /manager/status HTTP/1.1
R	?	?	?	?	?	?	

P: Parse and prepare request S: Service F: Finishing R: Ready K: Keepalive

List of Metrics collected:



Name	Description
JVM Memory Free	Total unused size of memory (in bytes) for all heap memory pools. This is the difference between what has been committed to the heap memory pools and what is currently being used. Memory committted to the heap memory pools will grow and shrink during JVM operation.
JVM Memory Allocated	Total amount of memory (in bytes) committed by heap memory pools.
JVM Memory Max	Total maximum size of memory (in bytes) for all heap memory pools.
Connector Max Threads	The maximum number of threads allocated to the connector for servicing requests. This puts an upper limit on the number of concurrent requests that can be in process for the Tomcat server.
Connector Name	The connector name. It is composed of the protocol and port configured for the connector.
Connector Thread Count	The total number of threads in the thread pool available to be used by the connector.
Connector Busy Threads	The number of threads handling incoming requests for the connector. Here is a list of the possible thread stages: 1) Parse and Prepare Request. 2) In Service processing a request and generating a response. 3) Finishing the request processing. 4) Keep-alive for handling sessions.
Connector Max Request Time	The maximum amount of time in milliseconds spent processing a request for the connector.
Connector Request Time	The total amount of time in milliseconds spent processing requests by the connector since the start of the Tomcat server.
Connector Request Count	The total number of requests processed by the connector since the start of the Tomcat server.
Connector Request Error Count	The total number of request errors seen by the connector since the start of the Tomcat server.
Connector Bytes Received	The total number of bytes received by the connector since the start of the Tomcat server.
Connector Bytes Sent	The total number of bytes sent by the connector since the start of the Tomcat server.

List of Metrics collected via snmp and Java Management MIB:

Name	SNMP OID	Description
jvmClassesLoadedCount	.1.3.6.1.4.1.42.2.145.3.163.1.1.1.1	The number of classes currently loaded in the JVM. See java.lang.management.ClassLoadingMXBean.getLoadedClassCount()
jvmMemoryPendingFinalCount	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.1	The approximate number objects that are pending for finalization. See java.lang.management.MemoryMXBean. getObjectPendingFinalizationCount()
jvmMemPoolName	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.110.1.2	The name of this memory pool, as returned by MemoryPoolMXBean.getName(). See java.lang.management.MemoryPoolMXBean.getName()
jvmThreadCount	.1.3.6.1.4.1.42.2.145.3.163.1.1.3.1	The current number of live threads. See java.lang.management.ThreadMXBean.getThreadCount()
jvmThreadDaemonCount	.1.3.6.1.4.1.42.2.145.3.163.1.1.3.2	The current number of daemon threads. See java.lang.management.ThreadMXBean.getDaemonThreadCount()
jvmThreadPeakCount	.1.3.6.1.4.1.42.2.145.3.163.1.1.3.3	The peak thread count since the execution of the application. See java.lang.management.ThreadMXBean.getPeakThreadCount()
jvmClassesTotalLoadedCount	.1.3.6.1.4.1.42.2.145.3.163.1.1.1.2	The total number of classes that have been loaded since the JVM has started execution. See java.lang.management.ClassLoadingMXBean. getTotalLoadedClassCount()
jvmClassesUnloadedCount	.1.3.6.1.4.1.42.2.145.3.163.1.1.1.3	The total number of classes that have been unloaded since the JVM has started execution. See java.lang.management.ClassLoadingMXBean.getUnloadedClassCount()
jvmMemoryHeapInitSize	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.10	Total amount of memory (in bytes) that the Java virtual machine initially requests from the operating system for memory management for heap memory pools. See java.lang.management.MemoryMXBean.getHeapMemoryUsage().getInit()
jvmMemoryHeapUsed	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.11	Total amount of used memory (in bytes) from heap memory pools. See java.lang.management.MemoryMXBean.getHeapMemoryUsage().getUsed()
jvmMemoryHeapCommitted	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.12	Total amount of memory (in bytes) committed by heap memory pools. See java.lang.management.MemoryMXBean.getHeapMemoryUsage(). getCommitted()
jvmMemoryHeapMaxSize	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.13	Total maximum size of memory (in bytes) for all heap memory pools. See java.lang.management.MemoryMXBean.getHeapMemoryUsage().getMax()
jvmMemoryNonHeapInitSize	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.20	Total amount of memory (in bytes) that the Java virtual machine initially requests from the operating system for memory management for non heap memory pools. See java.lang.management.MemoryMXBean.getNonHeapMemoryUsage().getInit()
jvmMemoryNonHeapUsed	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.21	Total amount of used memory (in bytes) from non heap memory pools. See java.lang.management.MemoryMXBean.getNonHeapMemoryUsage().getUsed()
jvmMemoryNonHeapCommitted	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.22	Total amount of memory (in bytes) committed by non heap memory pools. See java.lang.management.MemoryMXBean. getNonHeapMemoryUsage().getCommitted()
jvmMemoryNonHeapMaxSize	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.23	Total maximum size of memory (in bytes) for all non heap memory pools. See java.lang.management.MemoryMXBean.getNonHeapMemoryUsage().getMax()
jvmMemGCCount	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.101.1.2	The total number of collections that have occurred, as returned by GarbageCollectorMXBean.getCollectionCount(). If garbage collection statistics are not available, this object is set to 0. See java.lang.management.GarbageCollectorMXBean.getCollectionCount()
jvmMemGCTimeMs	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.101.1.3	The approximate accumulated collection elapsed time in milliseconds, since the Java virtual machine has started. This object is set to 0 if the collection elapsed time is undefined for this collector. See java.lang.management.GarbageCollectorMXBean.getCollectionTime()
jvmMemPoolPeakReset	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.110.1.5	This object indicates the last time - in milliseconds - at which the peak memory usage statistic of this memory pool was reset to the current memory usage. This corresponds to a time stamp as returned by java.lang.System.currentTimeMillis(); Setting this object to a time earlier than its current time value has no effect. Setting this object to a time later than its current time value causes the peak memory usage statistic of this memory pool to be reset to the current memory usage. The new value of this object will be the time at which the reset operation is triggered. There could be a delay between the time at which the reset operation is triggered and the time at which the actual resetting happens, so this value is only indicative. See java.lang.management.MemoryPoolMXBean.resetPeakUsage()
jvmMemPoolInitSize	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.110.1.10	Initial size of this memory pool. See java.lang.management.MemoryPoolMXBean.getUsage().getInit()
jvmMemPoolUsed	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.110.1.11	Amount of used memory in this memory pool. See java.lang.management.MemoryPoolMXBean.getUsage().getUsed()
jvmMemPoolCommitted	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.110.1.12	Amount of committed memory in this memory pool. See java.lang.management.MemoryPoolMXBean.getUsage().getCommitted()
jvmMemPoolMaxSize	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.110.1.13	Maximal size of this memory pool. See java.lang.management.MemoryPoolMXBean.getUsage().getMax()
jvmMemPoolPeakUsed	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.110.1.21	Amount of used memory in this memory pool when the peak usage was reached. See java.lang.management.MemoryPoolMXBean.getPeakUsage().getUsed()
jvmMemPoolPeakCommitted	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.110.1.22	Amount of committed memory in this memory pool when the peak usage was reached. See java.lang.management.MemoryPoolMXBean.getPeakUsage().getCommitted()
jvmMemPoolPeakMaxSize	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.110.1.23	Maximal size of this memory pool when the peak usage was reached. See java.lang.management.MemoryPoolMXBean.getPeakUsage().getMax()
jvmMemPoolCollectUsed	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.110.1.31	The amount of used memory at the most recent time that the Java virtual machine has expended effort in recycling unused objects in this memory pool. See java.lang.management.MemoryPoolMXBean.getCollectionUsage().getUsed()
jvmMemPoolCollectCommitted	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.110.1.32	The amount of committed memory at the most recent time that the Java virtual machine has expended effort in recycling unused objects in this memory pool. See java.lang.management.MemoryPoolMXBean.getCollectionUsage(). getCommitted()
jvmMemPoolCollectMaxSize	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.110.1.33	The value of the maximum amount of memory at the most recent time that the Java virtual machine has expended effort in recycling unused objects in this memory pool. See java.lang.management.MemoryPoolMXBean.getCollectionUsage().getMax()
jvmMemPoolThreshdCount	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.110.1.111	The number of times that the memory usage has crossed the usage threshold, as detected by the Java virtual machine. If memory usage threshold is not supported, then this object, if implemented, will always be equals to 0. See also jvmMemPoolThresholdSupport. See java.lang.management.MemoryPoolMXBean.getUsageThresholdCount(), java.lang.management.MemoryPoolMXBean.isUsageThresholdSupported()
jvmMemPoolCollectThreshdCount	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.110.1.132	The number of times that the memory usage has crossed the collection usage threshold, as detected by the Java virtual machine. If memory usage threshold is not supported, then this object, if implemented, will always be equals to 0. See also jvmMemPoolCollectThreshdSupport. See java.lang.management.MemoryPoolMXBean. getCollectionUsageThresholdCount(), java.lang.management.MemoryPoolMXBean. isCollectionUsageThresholdSupported()
jvmThreadTotalStartedCount	.1.3.6.1.4.1.42.2.145.3.163.1.1.3.4	The total number of threads created and started since the Java Virtual Machine started. See java.lang.management.ThreadMXBean.getTotalStartedThreadCount()
jvmRTUptimeMs	.1.3.6.1.4.1.42.2.145.3.163.1.1.4.11	Uptime of the Java virtual machine, in milliseconds. This is equivalent to (System.currentTimeMillis() - jvmStartTimeMs). See also jvmRTStartTimeMs. See java.lang.management.RuntimeMXBean.getUptime()
jvmJITCompilerTimeMs	.1.3.6.1.4.1.42.2.145.3.163.1.1.5.2	Gets the approximate accumulated elapsed time (in milliseconds) spent in compilation since the Java virtual machine has started. If multiple threads are used for compilation, this value is the summation of the approximate time that each thread spent in compilation. If compiler time monitoring is not supported, then this object remains set to 0. See java.lang.management.CompilationMXBean.getTotalCompilationTime()
jvmMemManagerName	.1.3.6.1.4.1.42.2.145.3.163.1.1.2.100.1.2	The name of this memory manager, as returned by MemoryManagerMXBean.getName(). See java.mangement.MemoryManagerMXBean.getName().
jvmThreadInstBlockCount	.1.3.6.1.4.1.42.2.145.3.163.1.1.3.10.1.4	The total number of times that this thread has blocked to enter or re-enter a monitor.. See java.lang.management.ThreadMXBean.getThreadInfo(long,boolean). getBlockedCount()
jvmThreadInstBlockTimeMs	.1.3.6.1.4.1.42.2.145.3.163.1.1.3.10.1.5	The approximate accumulated elapsed time (in millisecond) that a thread has blocked to enter or re-enter a monitor since it has started - or since thread contention monitoring was enabled. This object is always set to 0 if thread contention monitoring is disabled or not supported. See java.lang.management.ThreadMXBean.getThreadInfo(long,boolean). getBlockedTime()
jvmThreadInstWaitCount	.1.3.6.1.4.1.42.2.145.3.163.1.1.3.10.1.6	The total number of times that this thread has waited for notification. See java.lang.management.ThreadMXBean.getThreadInfo(long,boolean). getWaitedCount()
jvmThreadInstWaitTimeMs	.1.3.6.1.4.1.42.2.145.3.163.1.1.3.10.1.7	The approximate accumulated elapsed time (in millisecond) that a thread has waited on a monitor through a java.lang.Object.wait method since it has started - or since thread contention monitoring wasenabled. This object is always set to 0 if thread contention monitoring is disabled or not supported. See java.lang.management.ThreadMXBean.getThreadInfo(long,boolean). getWaitedTime()
jvmThreadInstCpuTimeNs	.1.3.6.1.4.1.42.2.145.3.163.1.1.3.10.1.8	The approximate accumulated CPU time (in nanosecond) for a thread since it has started - or since thread CPU time monitoring was enabled. If the thread of the specified ID is not alive or does not exist, or the CPU time measurement is disabled or not supported, this object is set to 0. See java.lang.management.ThreadMXBean.getThreadCpuTime(long), java.lang.management.ThreadMXBean.isThreadCpuTimeSupported(), java.lang.management.ThreadMXBean.isThreadCpuTimeEnabled()
jvmThreadInstName	.1.3.6.1.4.1.42.2.145.3.163.1.1.3.10.1.9	This thread name - as returned by Thread.getThreadName(). See java.lang.management.ThreadInfo.getThreadName()

Onboarding Suggestions

How to verify if Tomcat is configured for monitoring?

- If you are getting server-status http://<ip_address>:<port_number>/server-status, then your tomcat server is running and you can use Method#1.
- Execute the nmap command from collector to the tomcat machine to ensure Tomcat port is open *Nmap -sS <ip address of tomcat server> -p <port number>*