GlassFish v3 Monitoring Test Plan

Author: Sankar Neelakandan sankar.neelakandan@sun.com
Version: 0.2
Date: 09/28/2009

Overview

1. Introduction

This monitoring QA one pager describes the testing activity for Glassfish monitoring feature for final V3 release. Details like monitoring features to be tested, test scenarios, test deliverables, and schedules will be discussed in this document. Before the features testing begins the plan has to be approved by the QA lead and respective engineers working in the monitoring component.

2. Features to be tested

See section 4

3. Features not tested

1. Monitoring data thru JMX and MBean

4. Monitoring test scenarios

Please refer to this document for test scenarios that will be executed in V3 release

Enable/Disable Monitornig

The test scenarios are covered in admincli testspecs as this applies to command verification rather than monitoring.

JSP Monitoring

1. enable web-container montioring, deploy a web module and verify the following stats shows up and its value is 0

server.applications.webmodmonitoring.server.activejspsloadedcount-count = 0
server.applications.webmodmonitoring.server.maxjspsloadedcount-count = 0
server.applications.webmodmonitoring.server.totaljspsloadedcount-count = 0

server.web.jsp.activejspsloadedcount-count = 0
server.web.jsp.maxjspsloadedcount-count = 0
server.web.jsp.totaljspsloadedcount-count = 0

2. access the deployed jsp and verify the following stats values shows as 1

server.applications.webmodmonitoring.server.activejspsloadedcount-count = 1
server.applications.webmodmonitoring.server.totaljspsloadedcount-count = 1

server.web.jsp.totaljspsloadedcount-count = 1
server.web.jsp.activejspsloadedcount-count = 1

3. access 1 more jsp and verify the load count increases

server.applications.webmodmonitoring.server.activejspsloadedcount-count = 2
server.applications.webmodmonitoring.server.totaljspsloadedcount-count = 2

server.web.jsp.totaljspsloadedcount-count = 2
server.web.jsp.activejspsloadedcount-count = 2

4. access the same jsp again and verify the stats ramains the same

server.applications.webmodmonitoring.server.activejspsloadedcount-count = 2
server.applications.webmodmonitoring.server.totaljspsloadedcount-count = 2

server.web.jsp.totaljspsloadedcount-count = 2
server.web.jsp.activejspsloadedcount-count = 2

5. verify totaljspsloadedcount is equal to the number of jsps in all applications loaded

6. redeploy a application and verify the application specific stats starts at 0 but the container specific stats are cumulative

7. try to individually access the jsp stats and verify the get command returns the stats values

8. undeploy the application and verify the application specific stats disappear and container wide activejspsloadedcount is decremented

Servlet Monitoring

1. enable web-container montioring, deploy a web module and verify the following stats shows up and its value is 0

server.applications.webmodmonitoring.server.activeservletsloadedcount-count = 0
server.applications.webmodmonitoring.server.maxservletsloadedcount-count = 0
server.applications.webmodmonitoring.server.totalservletsloadedcount-count = 0

server.web.jsp.activeservletsloadedcount-count = 0
server.web.jsp.maxservletsloadedcount-count = 0
server.web.jsp.totalservletsloadedcount-count = 0

2. access the deployed servlet and verify the following stats values shows as 1

server.applications.webmodmonitoring.server.activeservletsloadedcount-count = 1
server.applications.webmodmonitoring.server.totalservletsloadedcount-count = 1

server.web.jsp.totalservletsloadedcount-count = 1
server.web.jsp.activeservletsloadedcount-count = 1

3. access 1 more servlet and verify the load count increases

server.applications.webmodmonitoring.server.activeservletsloadedcount-count = 2
server.applications.webmodmonitoring.server.totalservletsloadedcount-count = 2

server.web.jsp.totalservletsloadedcount-count = 2
server.web.jsp.activeservletsloadedcount-count = 2

4. access the same servlet again and verify the stats ramains the same

server.applications.webmodmonitoring.server.activeservletsloadedcount-count = 2
server.applications.webmodmonitoring.server.totalservletsloadedcount-count = 2

server.web.jsp.totalservletsloadedcount-count = 2
server.web.jsp.activeservletsloadedcount-count = 2

5. verify totaljspsloadedcount is equal to the number of servlets in all applications loaded

6. redeploy a application and verify the application specific stats starts at 0 but the container specific stats are cumulative

7. try to individually access the servlet stats and verify the get command returns the stats values

8. undeploy the application and verify the application specific stats disappear and container wide activeservletsloadedcount is decremented

Request Monitoring

1. deploy 2 different applications web applications. Access the deployed servlets 10 times each, access /index.html 10 times and verify the below stats
server.web.request.requestcount-count = 20 --- do we include the /index.html request counts in this stat ?.
server.applications.webmodmonitoring.server.requestcount-count = 10
server.applications.standalonewebmod1.server.requestcount-count = 10

2. access one of the app again and verify the container and app specific stats are updated
3. Throw error the deployed servlet with different response codes and verify the error count is updated.
server.web.request.errorcount-count = 50
Do we include /foo 404 requests in this stat ?.
4. verify reponse codes below 400 are counted as non error requests and codes above 400 are counted as error requests
5. Let the Servlet process the request for about 25 seconds and verify the max processing time is 25 seconds. server.web.request.maxtime-count = 25
6. Let the Servlet process the request for about 75 seconds and verify the max processing time is 75 seconds. server.web.request.maxtime-count = 75

7. Let the Servlet process the request for about 25 seconds and verify the max processing time is 75 seconds. server.web.request.maxtime-count = 75

Session Monitoring

1. Deploy a session application run 10 concurrent clients and verify the sessions created are 10.

server.applications.webmodmonitoring.server.activesessionscurrent-count = 10
server.web.session.activesessionscurrent-count = 10

2. Run another 10 clients from another application and verify

server.applications.webmodmonitoring.server.activesessionscurrent-count = 10
server.applications.standalonewebmod2.server.activesessionscurrent-count = 10
server.web.session.activesessionscurrent-count = 20

3. verify the active sessions high count is

server.applications.webmodmonitoring.server.activesessionshigh-count = 10
server.applications.standalonewebmod2.server.activesessionshigh-count = 10
server.web.session.activesessionshigh-count = 20

4. verify totalsessions-count is

server.applications.webmodmonitoring.server.sessionstotal-count = 10
server.applications.standalonewebmod2.server.sessionstotal-count = 10
server.web.session.sessionstotal-count = 20

5. set max-sessions as 5 for a application and send 12 clients, verify the rejected sessions is 7

6. create 10 sessions and restart the domain. After startup verify the sessions activated is 10

7. create 10 sessions and let it expire verify the sessions expired count for both application and container and it is equal to 10

8. create 10 more sessions and let it expire verify the sessions expired count for both application and container and it is equal to 20

9. not possible to test passivation,persistence since we don't have any support for that.

10. create another virtual server and verify all the above scenarios for the non default virtual server.

JDBC connection pool monitoring

All of the following scenarios are tested with live jdbc pools. Pool is created , jdbc app is deployed and run with specific number of clients and the stats are verified.

1. Turn on jdbc pool monitoring and verify the stats are displayed
2. get the number of connections used in jdbc connection pool when there are no connections are used
3. get the NumConnUsed attribute from the jdbc connection pool when some predetermined connections are used and verify it matches
4. get the NumConnFree for the jdbc connection pool when all the connections are free
5. get the NumConnFree attribute when 3 connections are free and rest of them are used
6. get the NumConnValidationFailed Attribute for the jdbc connection pool
7. timeout some connections and get the NumConnTimedOut attribute for the jdbc Connection pool and verify it matches
8. get connectionrequestwaittime attribute of a jdbc connection pool
9. get connectionrequestwaittime-highwatermark attribute of a jdbc connection pool
10. get connectionrequestwaittime-lowwatermark attribute of a jdbc connection pool
11. get numconnacquired and numconnreleased attributes of a jdbc connection pool
12. get connectioncreated and connectiondestroyed attributes of a jdbc connection pool
13. reset the monitoring level and verify all the stats are initialized to zero

Connector connection pool monitoring

All of the following scenarios are tested with live connector pools. Pool is created , connector app is deployed and run with specific number of clients and the stats are verified.

1. Turn on connector pool monitoring and verify the stats are displayed
2. get the number of connections used in connector connection pool when there are no connections are used
3. get the NumConnUsed attribute from the connector connection pool when some predetermined connections are used and verify it matches
4. get the NumConnFree for the connector connection pool when all the connections are free
5. get the NumConnFree attribute when 3 connections are free and rest of them are used
6. get the NumConnValidationFailed Attribute for the jdbc connection pool
7. timeout some connections and get the NumConnTimedOut attribute for the connector Connection pool and verify it matches
8. get connectionrequestwaittime attribute of a connector connection pool
9. get connectionrequestwaittime-highwatermark attribute of a connector connection pool
10. get connectionrequestwaittime-lowwatermark attribute of a connector connection pool
11. get numconnacquired and numconnreleased attributes of a connector connection pool
12. get connectioncreated and connectiondestroyed attributes of a connector connection pool
13. reset the monitoring level and verify all the stats are initialized to zero

work-management monitoring

1. deploy a work RA in the background and verify the activeworkcount equals the number of work being executed
2. while the work is being executed check the highwater mark value to be equal to activeworkcount
3. let all the work submitted and verify the submitted work stats is equal to the work submitted
4. let all the work complete its execution and verify the completed work count is equl to the number of work submitted
5. make the work to be rejected by specifying the wokstarttimeout to be 0 and sleep a while in work accepted listener method so that work is rejected. verify the work rejected is the correct number
6. verify the wait queue length - currently it is not possible to test this as there is no max-thread config to configure
7. workrequestwaittime -currently it is not possible to test this as there is no max-thread config to configure so that the remaining threads will wait.
8. turn off monitoring and verify the workra stats are removed
9. reset monitoring and verify the stats are initialized to 0
10. redeploy the workra and verify the stats are not cumulative and starts from 0 and adds-on

Transaction service monitoring

1. Turn on tx service monitoring and verify the stats are displayed
2. deploy tx app, commit 5 tx, rollback 7 tx's and get all the transaction attributes and verify it macthes
3. run the tx app in the background and verify the activeids are shown and its number matches with the ongoing transaction
4. reset the monitoring level and verify all the stats are initialized to zero

http-service

Request monitoring

1. turn monitoring on and verify the stats attributes are displayed
2. turn monitoring off and verify the stats attributes are not displayed
3. send http requests for response codes 101,200,201,202,203,204,205,206,300,301,302,303,304,305,307,400,401,402,403,404,405,406,407,408,409,410,411,412,413,414,415,416,417,500,501,502,503,504 and verify the requests-counts matches returned from the server

connection monitoring

1. send 50 non keepalive connections in the background and verify the
server.network.http-listener-1.connections.openconnections-count=50
server.network.http-listener-1.connections.totalconnections-count = 50

2. send 50 non keepalive connections in the background and verify the
server.network.http-listener-1.connections.openconnections-count=50
server.network.http-listener-1.connections.totalconnections-count = 100

keep-alive connection monitoring

3. send 23 keep-alive connections and don't close it , let it timeout . Verify the stat is equal to
server.network.http-listener-1.keep-alive.timeouts-count = 23

4. send 10 keep-alive connections in the background. Verify the stat is equal to
server.network.http-listener-1.keep-alive.keepaliveconnections-count = 10

5. open 10 keep-alive connections and send 10 requests in each connection. Verify the stat is equal to
server.network.http-listener-1.keep-alive.hits-count = 90

6. configure the max-keep-alive-connections to 25 and open 52 keep-alive connections to the server. verify the refusals count is
server.network.http-listener-1.keep-alive.refusals-count = 27

7. verify connection flushed count is 25 from the above test
server.network.http-listener-1.keep-alive.flushes-count = 25

Thread-pool monitoring

1. send 40 http connections to http-listener and 10 requests in each connection to the server and verify the stats equal to
server.network.http-listener-1.thread-pool.numberofactivethreads-count = 40

2. from the above test verify the stats equal to
server.network.http-listener-1.thread-pool.totalexecutedtasks-count = 400

3. right after the above test verify the stat equals to
server.network.http-listener-1.thread-pool.currentthreadpoolsize-count = 40

JRuby monitoring

1. deploy a jruby app and enable montioring. verify the jruby stats attributes are returned from get *
2. disable jruby monitoring and verify the stats attributes are removed
3. send the following http requests for response codes 202,302,304,305,400,401,403,404,405,500,503,600 and verify the requests-count is equal in those response codes

EJB

Stateless

Bean Methods

1. Run a stateless ejb application and get the following stats and verify it matches with the expected values

bean-methods.create.executiontime-count
bean-methods.create.totalnumerrors-count
bean-methods.create.totalnumsuccess-count
bean-methods.create.methodstatistic

Bean Pool

2. Run a ejb stateless session bean application and verify the bean-pool values are correct as expected

bean-pool.jmsmaxmessagesload-count
bean-pool.numbeansinpool-current
bean-pool.numthreadswaiting-current
bean-pool.totalbeanscreated-count
bean-pool.totalbeansdestroyed-count

3. Run a ejb stateless session bean application and verify the bean lifecycle values are correct as expected

createcount-count
removecount-count
methodready-count

Stateful

Bean-Cache

1. Run a ejb stateful session bean application and verify the bean-cache values are correct as expected

Bean-Cache
bean-cache.cachemisses
bean-cache.numbeansincache
bean-cache.numpassivationerrors
bean-cache.numexpiredsessionsremoved

Message Driven Beans

1. Run a MDB bean application and verify the following stats are correct

bean-pool.totalbeanscreated
bean-pool.numbeansinpool
bean-pool.totalbeansdestroyed
bean-pool.jmsmaxmessagesload
bean-pool.numthreadswaiting

messagecount
removecount

5. Test methodology

New tests will be automated with Tonga test framework and run as per the test matrix.

6. Test deliverables

This plan and test results of regression test suite.

7. Onepager Review

QE : Judy J Tang

Engineers: Sreenivas Munnangi, Prashanth Abbagani

8. Reference Documents

One Pager: Monitoring in GlassFish v3

Statistics: Monitoring in V3