LoadGen

Goals for the creation of this utility was to generate loads against REST endpoints and then to be able to describe a platform profile. There generally seems to be two types of load testing.

1. Vendor testing:
    - vehicle's top speed
    - top ops per second under specific conditions (i.e. 100 million entries with xxxxx authentications per second across two OpenAM instance with 4 cores and 16GB memory each)
2. Customer testing/platform profile
    - driving this vehicle with this load over this type of terrain under these conditions **should** yield a fuel consumption rate of X.
    - A 4 core system with 16GB of memory with 100 million entries and a 10% concurrency rate performing these types of task(s), for example:
        1. Authenticate to OpenAM
        2. sleep 10 seconds
        3. Validate session
        4. sleep 10 seconds
        5. Update user data
        6. Logout

        should provide a transaction rate of X with a 200ms or less latency.

Why I put the effort into this? Scripted cURL (UNIX shell) is resource intensive on the load generator. Used REST so eventually loads can be applied to competitive offerings using the same utility and makes testing ForgeRock services much easier and flexible. Looked at Jmeter and gatling.io. Very good utilities but the learning curve to achieve the desired results seemed high for both. Plus needed a utility that helped craft a profile of the platform not just stress the platform.

LoadGen uses REST to interact with the services (OpenAM, OpenDJ, OpenIDM) and uses json for configuration and testing parameters.

Job example ($LOADGEN_HOME/config):

```
{
"name": "my job 0",
"service-environment": "one 4xcore 7GB Ubuntu 14.04 VM in Azure Apache Tomcat 7.59, JDK8, OpenAM 12.0.0 with 200,000 users",
"service-location-port": "http://openam.example.com:8080",
"ADMINUSERID": "adminuser",
"ADMINUSERPASSWORD": "password",
"load-generator": "one 2xcore Ubuntu 14.04 VM in Azure on same vnet at OpenAM server (example.com)",
"threads" :20,
"threadgroupsize" : 2,
"threadinterval" : 10000,
"upperbounds" : 10000,
"workloads": [
        {
        "read-user-object" : 1000
        },
        {
        "query-user-objects" : 100
        },
        {
        "authenticate-read-logout-user-object" : 0
        },
        {
        "create-and-update-user-object": 20
        }
]
}
```

The value after each workload such as “read-user-object” is the number of iterations that the workload should be performed. The **`threads`** value is the number of Java threads that will be created and executed. Each thread will run the same workloads in the same order. Workloads are intermixed based on the quantity of the various workloads. If there are 10 workloads of type A and 2 workloads of type B then Workload A will be executed 5 times, followed by 1 Workload B, then 5 more Workload A and 1 Workload B.

The **`threadgroupsize`** and **`threadinterval`** help regulate the initial spawning of threads. Helps with creating a ramped up load and sustainable load. Set to zero to turn off and have all threads start at the same time.

Some attributes/values can be defined at all three levels - jobs, workloads, tasks – such as: service-location, service-port, adminuserid, adminuserpassword, randomfile, etc. Precedence: tasks override workloads; workloads override jobs.

workload example ($LOADGEN_HOME/workloads):

```
{
"name": "authenticate-read-logout-user-object",
"persist" : [tokenId, ...]
"tasks": [
        {
        "authenticate-openam-user-object": 400
        },
        {
        “sleep” : 10
        },
        {
        "read-openam-user-object" : 400
        },
        {
        "logout-openam-user-object" : 100
        }
]
}
```

The value after each task is the millisecond threshold that the task must complete in (except for “sleep”). If the task is not completed in under the specified ms then it is counted as “exceeded”. If the task is “sleep” than the value is added to the millisecond value specified in the sleep's “url-payload”. Using a ramped up load generation (using threadgroupsize and threadinterval) can skew certain aspects of the results. For example total successful operations should be considered invalid when a ramped up load is utilized.

task examples; three are listed below. Essentially the data for a operational cURL command. ($LOADGEN_HOME/tasks):

```
{
"name": "authenticate-openam-user-object",
"service-location-port" : "http://openam0.example.com:8080",
"request": "POST",
"header": {
"X-OpenAM-Username": "user.$RANDOMVALUE",
"X-OpenAM-Password": "password",
"Content-Type": "application/json"
},
"url-endpoint": "/openam/json/",
"url-payload": "authenticate",
"data-payload": "{}"
}
-------------------------
{
"name": "create-opendj-user-object",
"request": "PUT",
"header": {
"X-OpenIDM-Username": "$ADMINUSERID",
"X-OpenIDM-Password": "$ADMINUSERPASSWORD",
"Content-Type": "application/json",
"If-None-Match": "*"
},
"url-endpoint" : "/users/",
"url-payload": "user.new.$RANDOMVALUE",
"data-payload": {
"_id": "user.$RANDOMVALUE",
"contactInformation": {
"telephoneNumber": "+1 408 555 1212",
"emailAddress": "user.new.$RANDOMVALUE@example.com"
},
"name": {
"familyName": "New.$RANDOMVALUE",
"givenName": "User"
},
"displayName": "User New $RANDOMVALUE"
```

```
}
}
--------------------------
{
"name": "sleep",
"service-location-port" : "http://localhost:8080",
"header": {
"Content-Type": "application/json"
},
"url-endpoint": "$SLEEP",
"url-payload": 100
}
```

Output from a job with no ramp up:

Job Started with 4 threads on Fri Jun 26 11:58:36 PDT 2015...and completed Fri Jun 26 11:59:43 PDT 2015

Operation	TxTotal	AccmTime	Thrshold	TxPass	PassTime	TxExced	ExcdTime	TxFail	Failtime	CbdPsOps	ThrdOps	Avrms/op	Success	Exceed	Fail
read-opendj-user-object	40000	211.088s	400ms	39991	211.023s	0	0.000s	9	0.065s	758.041/s	189.510/s	5.277ms/op	99.98%	0.00%	0.02%
query-opendj-user-objects	4000	34.814s	400ms	3999	34.809s	0	0.000s	1	0.005s	459.536/s	114.884/s	8.704ms/op	99.97%	0.00%	0.03%
create-opendj-user-object	40	1.803s	400ms	40	1.803s	0	0.000s	0	0.000s	88.741/s	22.185/s	45.075ms/op	100.00%	0.00%	0.00%
update-opendj-new-user-object	40	1.032s	200ms	38	0.596s	2	0.436s	0	0.000s	255.034/s	63.758/s	15.684ms/op	95.00%	5.00%	0.00%
read-opendj-new-user-object	80	1.277s	200ms	76	1.277s	0	0.000s	4	0.000s	238.058/s	59.514/s	16.803ms/op	95.00%	0.00%	5.00%

```
Job lapsed time = 66399ms
Passed Accum time = 249508ms
Total Threads = 4
Thread group size = 0
Launch interval = 1000ms
Transaction goal = 44160
Transction actual = 44144
Successful ops/s = 707.697
Ops/s during job = 664.829
Success rate = 99.96%
Total sleep sec = 12.179
Based on following workloads:
Workload: read-user-object with 40000 loops; Percent load = 90.33%
        Tasks used:
        read-opendj-user-object
Workload: query-user-objects with 4000 loops; Percent load = 9.03%
        Tasks used:
        query-opendj-user-objects
Workload: create-and-update-user-object with 40 loops; Percent load = 0.63%
        Tasks used:
        create-opendj-user-object
        sleep
        update-opendj-new-user-object
        sleep
        read-opendj-new-user-object
        sleep
        read-opendj-new-user-object
Target environment(s): mac laptop running two OpenDJ 2.6.2 instance in vbox, replicated, behind haproxy
Load generator: mac laptop running JDK 8
```

But when a job “ramps up” then the combined pass operations (CbdPsOps) as well as “`Successful ops/s =`” are skewed and is invalid.

Job Started with 32 threads on Fri Jun 26 17:33:06 UTC 2015...and completed Fri Jun 26 19:09:41 UTC 2015

Operation	TxTotal	AccmTime	Thrshold	TxPass	PassTime	TxExced	ExcdTime	TxFail	Failtime	**CbdPsOps**	ThrdOps	Avrms/op	Success	Exceed	Fail
authenticate-openam-user-object	8000	195.228s	200ms	7932	181.388s	68	13.840s	0	0.000s	1399.343/s	43.729/s	22.868ms/op	99.15%	0.85%	0.00%
validate-openam-user-session	16000	42.684s	100ms	15861	42.376s	3	0.308s	136	0.000s	11977.346/s	374.292/s	2.672ms/op	99.13%	0.02%	0.85%
logout-openam-user-object	8000	25.430s	100ms	7927	25.223s	2	0.207s	71	0.000s	10056.854/s	314.277/s	3.182ms/op	99.09%	0.03%	0.89%

Job lapsed time = 5795471ms
Passed Accum time = 248987ms
Total Threads = 32
Thread group size = 1
Launch interval = 10000ms
Transaction goal = 32000
Transction actual = 31720
Successful ops/s = 4076.679
Ops/s during job = 5.473
Success rate = 99.13%
Total sleep sec = 174473.266
Based on following workloads:
Workload: authenticate-read-logout-user-object with 8000 loops; Percent load = 100.00%
        Tasks used:
        authenticate-openam-user-object
        validate-openam-user-session
        sleep
        validate-openam-user-session
        sleep
        logout-openam-user-object
Target environment(s): one 4xcore 7GB Ubuntu 14.04 VM in Azure Apache Tomcat 7.59, JDK8, OpenAM 12.0.0 with 200,000 users
Load generator: one 2xcore Ubuntu 14.04 VM in Azure on same vnet at OpenAM server (example.com)

Some preliminary findings:

Increased thread count quickly reduces the transaction rate per thread. While two threads may scale up against a two core system two threads do NOT see double the performance over one thread. In fact OpenAM, with a simple authentication/validation workload did not scale to 4 cores as well as hoped.

For example a 4 core OpenAM instance with 16 threads:

```
===============
Job Started with 16 threads on Thu Jun 25 15:43:06 UTC 2015...and completed Thu Jun 25 15:55:17 UTC 2015
Operation                          TxTotal  AccmTime  Thrshold    TxPass  PassTime   TxExced  ExcdTime    TxFail  Failtime    CbdPsOps     ThrdOps      Avrms/op   Success    Exceed      Fail
authenticate-openam-user-object     16000011130.598s     200ms    15967911065.405s       321   65.193s         0    0.000s   230.888/s    14.430/s   69.298ms/op    99.80%     0.20%     0.00%
logout-openam-user-object           160000  545.667s     100ms    159639  541.514s        40    4.153s       321    0.000s  4716.820/s   294.801/s    3.392ms/op    99.77%     0.03%     0.20%
Job lapsed time =        731276ms
Passed Accum time =      11606919ms
Total Threads =          16
Transaction goal =       320000
Transction actual =      319318
Successful ops/s =       440.176
Ops/s during job =       436.659
Success rate =           99.79%
Total sleep sec =        0.000
Based on following workloads:
        Workload: authenticate-read-logout-user-object with 160000 loops; Percent load = 100.00%
        Tasks used:
                authenticate-openam-user-object
                logout-openam-user-object
Target environment(s): one 4xcore 7GB Ubuntu 14.04 VM in Azure Apache Tomcat 7.59, JDK8, OpenAM 12.0.0 with 200,000 users
Load generator: one 2xcore Ubuntu 14.04 VM in Azure on same vnet at OpenAM server (example.com)
=================
```

And with a 8 core OpenAM instance with 16 threads

```
=================

Job Started with 16 threads on Thu Jun 25 19:54:52 UTC 2015...and completed Thu Jun 25 20:02:00 UTC 2015
Operation                          TxTotal  AccmTime  Thrshold    TxPass  PassTime   TxExced  ExcdTime    TxFail  Failtime    CbdPsOps     ThrdOps      Avrms/op   Success    Exceed      Fail
authenticate-openam-user-object     160000 6389.917s     200ms    159632 6313.170s       368   76.747s         0    0.000s   404.569/s    25.286/s   39.548ms/op    99.77%     0.23%     0.00%
logout-openam-user-object           160000  443.919s     100ms    159632  443.919s         0    0.000s       368    0.000s  5753.554/s   359.597/s    2.781ms/op    99.77%     0.00%     0.23%
Job lapsed time =        428219ms
Passed Accum time =      6757089ms
Total Threads =          16
Transaction goal =       320000
Transction actual =      319264
Successful ops/s =       755.980
Ops/s during job =       745.562
Success rate =           99.77%
Total sleep sec =        0.000
Based on following workloads:
        Workload: authenticate-read-logout-user-object with 160000 loops; Percent load = 100.00%
        Tasks used:
                authenticate-openam-user-object
                logout-openam-user-object
Target environment(s): one 8xcore 14GB Ubuntu 14.04 VM in Azure Apache Tomcat 7.59, JDK8, OpenAM 12.0.0 with 200,000 users
Load generator: one 2xcore Ubuntu 14.04 VM in Azure on same vnet at OpenAM server (example.com)

=================
```

It will be very important to understand the load breakout. Does the potential workload include many threads (1,000 – 100,000) that are somewhat concurrent against a service endpoint or a concentrated set of threads (100 or less)? For example mobile devices hitting an endpoint directly would constitute a very high thread count which may be best served by many small (2 core) OpenAM servers. Large quantities of threads seem to cause as much stress as the type of workload they represent. It seems to be just as important to understand the workload as how the workload is delivered.

When a system started to stress the number of operations exceeding threshold started to rise quickly and “Successful ops/s” started to drop. Thresholds can be adjusted higher but understanding how quickly a task can be responded to is very important. For example OpenDJ reads can occur in under 100ms very easily (from the load generators standpoint). Adjusting the threshold to something higher may result in a higher transaction rate but in fact if OpenDJ reads start taking longer than 50ms (from the load generators standpoint) the service is in fact starting to show stress. OpenDJ updates on the other hand can exceed 400ms regularly but the service is not stressed. Best way to establish a baseline threshold (one where the service is not stressing and consistently delivering expected performance) is to run one thread against a service with at least two cores, plenty of memory and is properly tuned,

for an extended amount of time (one hour). This “unstressed” service should be handle to respond consistently. The “Avrms/ops” data reflects the average time, in milliseconds, the load generator “waited” for the service to complete the transaction.

Generally, when a customer asks for sizing information, they want to put in place a system that will handle the average load without undue stress. This way if a spike occurs there is headroom for the service to handle the spike without impacting performance (the end user's experience for example).

With simple tests is was next to impossible to drive the service to 100% busy from a CPU standpoint (seemed true for both OpenDj and OpenAM with the very basic testing thus far).

A multi-core load generator was able to push more load against the service than multiple load generating instances (separate Vms). For example a 4 core load generator can kick out more transactions than two 2 core load generators.

Additional capability that is needed:

1. Website/REST endpoint to provide sleep as well as OpenAM agent
2. Random reads and injection from files for user ids and data
3. Attributes definable at jobs, workloads, and tasks levels with said precedence.
4. Website to build and run tests. Test as a service or trial or download war from backstage (need to authenticate)
5. Fix PATCH for OpenDJ and OpenIDM
6. Workload needs to specify if value needs to be persisted across tasks by specifying the attribute name returned by the a task operation i.e. tokenId
7. Define wildcards:

```
WILDCARDS : [
        {
        name : $RANDOMVALUE,
        type : generated
        },
        {
        name : $TOKENID,
        type : retained
        },
        {
        name : $FILE-A,
        type : file,
        source : path to file
        },
        {
        name : $OBJECT-O,
        type : rest-endpoint,
        source : url to object
        }
]
}
```

```
Sample output as of October 19th 2016:
rmfaller-forgerock-mac:LoadGen rmfaller$ java -jar ./dist/LoadGen.jar --config ./jobs/jobs-openam.json
Job Started with 4 threads on Wed Oct 19 15:37:24 PDT 2016...and completed Wed Oct 19 15:37:25 PDT 2016
Operation                          TxTotal  AccmTime  Thrshold    TxPass  PassTime   TxExced  ExcdTime    TxFail  Failtime    CbdPsOps     ThrdOps      Avrms/op   Success    Exceed      Fail
authenticate-openam-admin               40    0.816s     400ms        40    0.816s         0    0.000s         0    0.000s   196.078/s    49.020/s   20.400ms/op   100.00%     0.00%     0.00%
create-openam-user-object-post          40    1.015s     800ms        40    1.015s         0    0.000s         0    0.000s   157.635/s    39.409/s   25.375ms/op   100.00%     0.00%     0.00%
logout-openam-admin                     40    0.440s     100ms        40    0.440s         0    0.000s         0    0.000s   363.636/s    90.909/s   11.000ms/op   100.00%     0.00%     0.00%
Job lapsed time =       617ms
Passed Accum time =     2271ms
Total Threads =         4
Thread group size =     0
Launch interval =       0ms
Transaction goal =      120
Transaction actual =    120
Successful ops/s =      211.361
Ops/s during job =      194.489
Success rate =          100.00%
Total sleep sec =       0.000
Based on following workloads:
        Workload: adminuser-authenticate-createuser-logout with 40 loops; Percent load = 100.00%
        Tasks used:
                authenticate-openam-admin
                create-openam-user-object-post
                logout-openam-admin
Target environment(s): mac laptop running vbox hosting ubuntu (2 cores, 8GB) running an OpenAM 13.5 instance loaded w/2000 objects
Load generator: mac laptop running JDK 8
```

Operation	from the ./tasks JSON formatted REST call
TxTotal	number of actual tasks executed (threads * workload count {as specified in the ./jobs description file})
AccmTime	accumulated time in seconds to execute all Operations (successful and unsuccessful)
Thrshold	maximum time in milliseconds allowed for the Operation to complete successfully and specified in the ./workloads description file
TxPass	number of Operations that passed successfully (did not timeout or incur any other type of error)
PassTime	accumulated time in seconds to execute all successful Operations
TxExced	number of Operations that exceeded the specified Threshold
ExcdTime	accumulated time in seconds spent on Operations that eventually exceeded the Threshold
TxFail	number of Operations that failed for issues outside of exceeding the specified Threshold
Failtime	accumulated time in seconds spent on Operations that failed for issues outside of exceeding the specified Threshold
CbPsOps	calculated number of successful Operations that could be executed by all threads in a second
ThrdOps	calculated number of successful Operations per thread that could be executed in a second
Avrms/op	average number of milliseconds needed to execute one Operation
Success	percentage of successful Operations (this value degrades further if following Operations are dependent on the success of previous Operations)
Exceed	percentage of Operations that exceeded the threshold
Fail	percnetage of Operations that failed for reasons other than exceeding the threshold

Job lapsed time	clock time in milliseconds it took for the job to complete
Passed Accum time	total accumulated time of all successful Operations
Total Threads	as specified in the ./jobs file
Thread group size	used to spread load over time; of the total number of Threads how many threads to start together

Job lapsed time	clock time in milliseconds it took for the job to complete
Launch interval	time in milliseconds to wait before executing a thread group
Transaction goal	total number of transactions that will hopefully complete without exceeding the thresholds or an error
Transaction actual	reality
Successful ops/s	For the entire clock time of the job the number of Operations per second that completed successfully
Ops/s during job	calculated average Operations per second based on completion time of each Operation
Success rate	overall total success rate of all Operations for the job
Total sleep sec	if specified the total time in seconds that threads spent sleeping