Jmeter Operator Guide: Step-by-Step

Overview

- Install Apache Jmeter
- Ingest Dataset
- Configure Jmeter
- Run Jmeter against API
- Analyzing Results

This overview assumes the following:

1. Ingestion-service, API, and Simple-IDP are deployed and running on version 74
2. Operator has been delivered a "canonical" dataset relevant to the 74 release
3. Operator has successfully ingested "canonical" dataset as described above

Installing Jmeter

1. **Extract the omni-jmeter tarfile:**
    a. # cd $JMETER_DIR
    b. # tar -xvf omni-jmeter.tar
3. **Add the jmeter binaries to your path (a), or alias the jmeter executable (b)**
    a. export PATH="${PATH}:$JMETER_DIR/omni-jmeter/apache-jmeter-2.8/bin" or
    b. alias jmeter='$JMETER_DIR/omni-jmeter/apache-jmeter-2.8/bin/jmeter'

Configuring Jmeter

-Properties file: $JMETER_DIR/omni-jmeter/config/midgar.properties
-There are 3 properties in midgar.properties that you should expect to configure:

1. hostname
2. threads
3. loops

-The **hostname** property should point to your API node if you only have one API node, or your load balancer, if you are using a load-balancer.
-The "**threads**" property is used to generate simultaneous users. So "threads = 4" would mean each api call would be done pseudo-simultaneously by 4 different "users"
    * note: there is a limit to how many threads a particular instance of Jmeter can handle. see below for instructions on how to control multiple Jmeter servers from a single controller.
-The "**loops**" property is used to specify how many times Jmeter should iterate through the entire test case instruction suite.

```
hostname = api.changeme.org
port = 443
protocol = https
threads = 1
loops = 1
clientId = EGbI4LaLaL
clientSecret = iGdeAGCugi4VwZNtMJR062nNKjB7gRKUjSB0AcZqpn8Beeee
realmId = 45b02cb0-1bad-4606-a936-094331bd47fe
schoolId.1 = Daybreak%20Central%20High
sectionId.1 = 37
staffIdLogin.1 = badmin
staffIdPassword.1 = badmin1234
principalIdLogin.1 = stff-0000000033
principalIdPassword.1 = stff-00000000331234
teacherIdLogin.1 = cgray
teacherIdPassword.1 = cgray1234
studentFirstName.1 = Staci
studentLastName.1 = Crowell
```

If you wish to perform distributed Jmeter testing using multiple Jmeter servers to REAAALLLLY stress the system, please follow the step-by-step tutorial here:

http://jmeter.apache.org/usermanual/remote-test.html

*Note: This is probably not needed, you should be able to adequately load the system with one node running 10-20 threads. But to truly "stress" the system, especially if there are more than 2 API nodes, you might consider a distributed Jmeter approach if your Jmeter node starts dying.

Jmeter Realm Setup

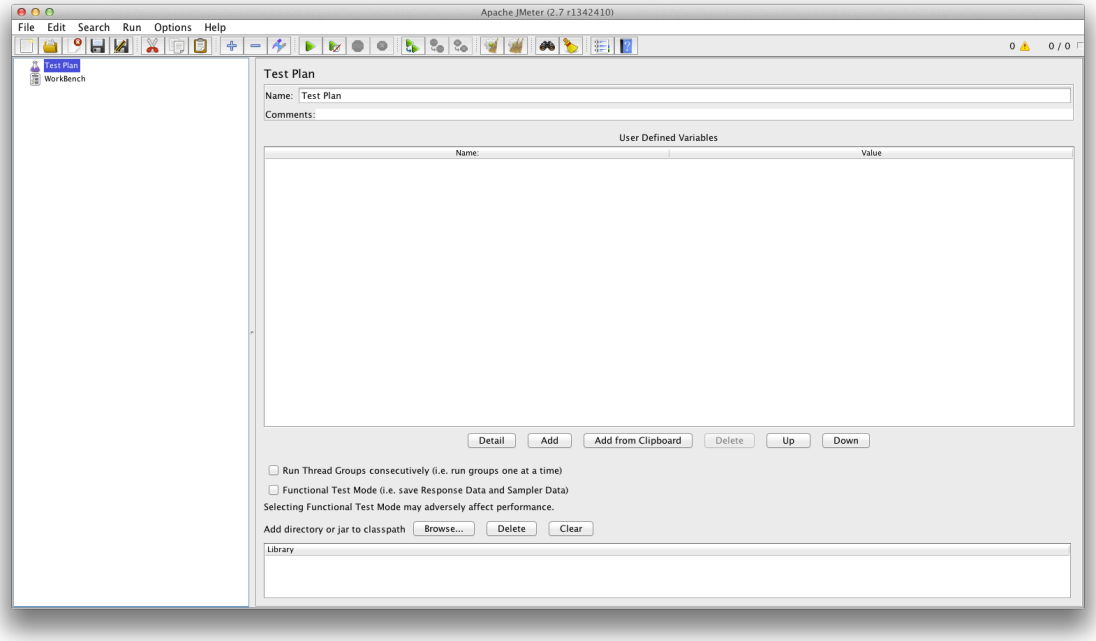
This is required to authorize the Jmeter application for use against the SLI data store

- On the Jmeter server machine:
  - cd $JMETER_DIR/config
  - run jmeter-realm-setup.sh, which will:
    - import the Jmeter application collection into mongo
    - Authorize the Jmeter application for that tenant
    - persist application data to the SLI db
- On the API server (or wherever acceptance tests are run):
  - cd sli/acceptance-tests
  - bundle install
  - bundle exec rake runSearchBulkExtract

Running Jmeter

Our recommendation for continuous load-testing is to use the "load-profile-test.jmx" Jmeter test case suite. You may, however, use a particular use-case-specific suite (listed below).

Jmeter Basic UI:



*Note: All commands assume your PWD is **$JMETER_DIR/omni-jmeter**

Load-specific Test Case Suite (Note: This will hit your API very hard, especially if threads is a relatively large number):

Jmeter GUI for Load Testing

```
jmeter -t load-profile-test.jmx -l logs/load-profile-test.jtl -q
config/midgar.properties
```

Use-case-specific Tests (documentation/descriptions below)

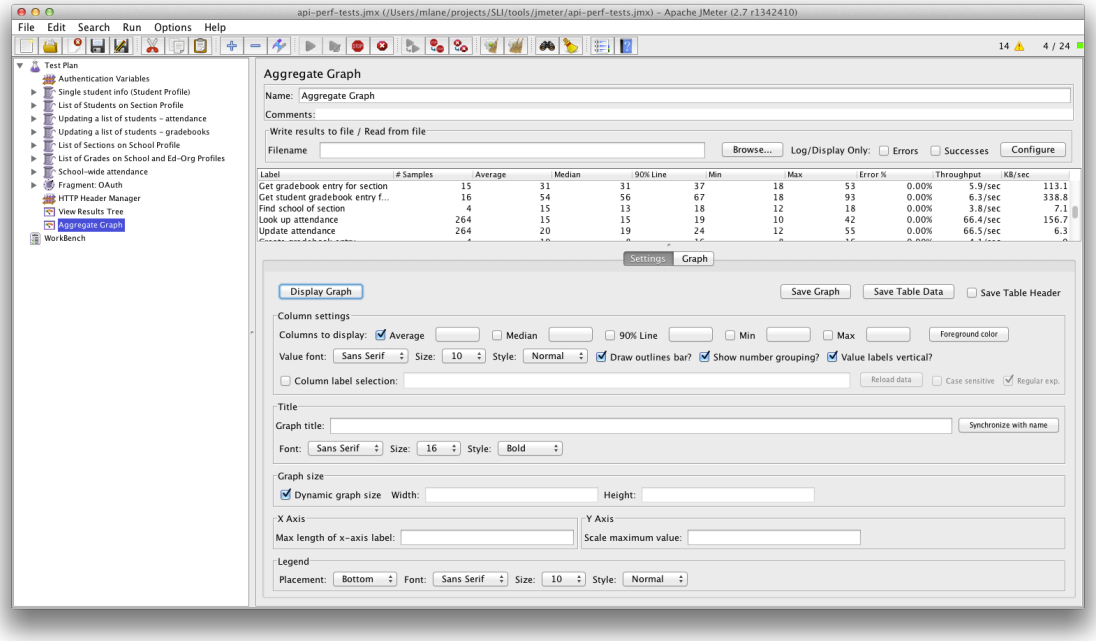
Jmeter GUI commands

```
List-attendance:
jmeter -t list-attendance.jmx -l logs/list-attendance.jtl -q config/midgar.properties

List-setions:
jmeter -t list-sections.jmx -l logs/list-sections.jtl -q config/midgar.properties

List-students:
jmeter -t list-students.jmx -l logs/list-students.jtl -q config/midgar.properties

List-grades:
jmeter -t list-grades.jmx -l logs/list-grades.jtl -q config/midgar.properties

Single-student:
jmeter -t single-student.jmx -l logs/single-student.jmx -q config/midgar.properties

Dashboards:
jmeter -t dashboard-list-students.jmx -l logs/dashboard-list-students.jtl -q
config/midgar.properties
jmeter -t dashboard-single-student.jmx -l logs/dashboard-single-student.jtl -q
config/midgar.properties
jmeter -t assessment_dashboard.jmx -l logs/assessment_dashboard.jtl -q
config/midgar.properties
```

Load and Run API use cases

- File -> Open
    - Navigate to **(SLI_HOME)/tools/jmeter/api-perf-tests.jmx**
- You can view the test on the left. The test plan consists of several use case scenarios.
    - In the Test Plan click 'View Results Tree' to open the results display.
        - Optional: Write results to file in the results display
- Start the Benchmark by pressing the geen play icon on the control bar.
    - Tests will start to run, they can be stopped by pressing the red stop icon.
    - You can determine that tests have completed when the stop icon is gray (disabled) and the play button is green (ready for next run)
- View results on the results tree page and the aggregate graph page
    - On the aggregate graph page you will see a table of metrics for each benchmark
    - Click display graph to see a visual representation of the benchmark times
    - All times are in milliseconds (1000 equals 1 second)

Analyzing Results



Use Case Identification

The POs have created a set of use cases API Use Cases. Our performance goals are optimized to support these use case.

Stephen Peha put together a list of API Load Test Cases based around scenarios like daily attendance and BOY enrollment.

API Performance Tests Mapped to Use Cases

JMeter Test	Description	Use Case Reference	Use Case Priority
list-sections.jmx	Given a school, show me all the sections in that school. Include course offerings, courses, and teachers.	"List of Sections on School Profile"	P1
single-student.jmx	Given a student, show me information about the sections they're enrolled in and their transcripts.	"Single student info (Student Profile)"	P1
update-attendance.jmx	Given a section, record attendance events for students in that section.	"Updating a list of students"	P1
update-gradebooks.jmx	Given a section, record a gradebook entry for every student in that section.	"Updating a list of students"	P1
list-students.jmx	Given a section, show me all the students in that section. Include the section gradebook, and all student gradebook entries for the section.	"List of Students on Section Profile"	P2

list-grades.jmx	Given a school or EdOrg, show me a list of grades for students in that school or EdOrg.	"List of Grades on School and Ed-Org Profiles"	P2
list-attendance.jmx	Given a school, show all attendance events for the school.	"Daily Absentee List"	P3
single student report	Given a teacher, show a single-student report in the dashboard.	Peha	
BOY Enrollment	Enroll the student body of an entire school at the beginning of the year.	Peha	

Rally Saga, Tags, and Defect Suite

The top level API Performance Sage in Rally is US3551. Each story is tagged with 'API Performance'.

Defect suite DS45 holds defects filed as a result of API Performance work.

A summary report page is available that captures all of this in a single Rally page.

Targets for Individual Use Cases

The set of API calls required to construct a single screen for each use case should take no more than 500 milliseconds (0.5 seconds), i.e. the time from invoking the API(s) on the client to the time a response is received from the server. This target derived from the target of displaying a page to the user in under 1 second (references: Response Times: The 3 Important Limits, Factors That Affect Usability).

500 ms of additional latency is allocated for (1) application processing those responses and constructing a page, (2) returning that page to the browser over the network, and (3) rending the page in the browser.

500 ms of overhead does not account for the "extraordinary" network latency or page sizes. Instead this is taken as a target for 90% of "normal" calls in a non-colocated environment (app client tier and server tier are not in the same cloud).

10 or fewer API calls required to serve data for each use case. This target may require the introduction of use-case specific end points.

Targets for Individual API Calls

The performance target for specific use cases (above) is the primary performance target for API. Secondarily, there is a performance target for individual API calls

- Individual entity reads: 50 milliseconds
- Multi-entity reads: 400 milliseconds

The configuration for these tests is the same as for the "Performance targets for use cases" above.

Targeted Number of Mongo Calls

The set of Mongo calls required for specific use cases (above) is an important consideration in overall API performance and scalability. The optimal case requires a single mongo call for each use case. Baseline measurements across all use cases have yet to be performed. Capturing these metrics will inform improvement stories going forward.

Targets for Concurrent Users - Normal and Peak Usage

The API and Ingestion Load Estimates google doc provides a rough estimate on nominal and peak load as the system grows. A concise summary

of this information is provided below for reference:

Scenario	Dataset Size	Concurrent Users	Pages per Second	API Calls per second
R1 Nominal	5M students	**TBD**	2	24
R1 Peak	5M students	**TBD**	14	168
R1.x Nominal	20M+ students	**TBD**	7	84
R1.x Peak	20M+ students	**TBD**	55	660

The performance target for concurrent users is for the reference server configuration to support the API calls for up to 55 pages/second, where the pages are a mix of the use cases described above.

Scalability

The system can scale horizontally to handle more concurrent users (more API calls per second) by adding more API servers and more Mongo shards.

The performance target for R1.0 is to be able to scale via horizontal scaling.

Where are we now

The following sections are the current results of various API performance and load tests. Previous results are archived elsewhere.

Use Case Transaction Times

Transactions for each individual use case is measured and captured systematically as part of a performance run. These results follow the transaction from the time the client make the API call, through the API, into mongo, and back to the client. These transaction times do not include the time for the user to log in.

Measuring these transaction times gives vertical slice snapshot of what is happening in the system during a particular use case.

Average Use Case Performance

December 12, 2012 - NTS JMeter API Performance Build #38

Environment: CI

Dataset: 150K

Some of the performance regression was addressed, but still see significant regression in the student-related use cases.

Use Case	API Calls	Total Average Time (ms)	Total 90% Time (ms)	Per API tax (10ms)	Estimated User Time (ms)
List of Attendances	1003	20238	22243	10030	30268
List of Grades	5	3611	3634	50	3661
List of Sections	602	7456	10462	6020	13476
List of Students	4	1109	1295	40	1149
Single Student	4	859	1011	40	899
Update Attendance	135	3441	3915	1350	4791
Update Gradebooks	70	2650	2877	700	3350

The "Per API tax" is an estimate of the additional time a real user will see based on the caller and callee being remote from each other instead of being on the same network (and same machine in this case).

See the JMeter Results Archive page for historic JMeter results and trends.

Roadmap

You can view the full API Performance roadmap on the API Performance Tracking Rally page.