TabJolt Installation Guide

TabJolt: A Tableau Server Point and Run Load Testing Tool

What is TabJolt?

TabJolt is a point and run load generator built on top of JMeter that is specifically designed for Tableau Server. It is available as a free download as-is from GitHub.

Unlike traditional load testing tools, TabJolt can drive load against your Tableau Server without you having to develop or maintain scripts. TabJolt is optimized for the Tableau presentation model and can load visualizations and interpret possible interactions during test execution.

TabJolt is a subset of our engineering load testing framework and we are making it available in hopes that it allows you to accelerate production go-live with Tableau Server, and also to help you with planning your specific on-site capacity needs.

Of course, running load tests does not replace understanding the Tableau Server architecture and following best practices for deployment during load testing. Treating Tableau Server as a black box for load testing is not recommended, and will probably yield unexpected results. For information about the Tableau Server architecture, see the Tableau Server Administration Guide.

When to use TabJolt

Here are the key questions TabJolt is designed to help you answer:

1. I want to deploy a brand new Tableau Server. How will Tableau Server scale on my hardware and with my workload?
2. I am moving from Tableau Server 8.x to version 9.0. Given my workbooks and hardware, how will Tableau Server 9.0 scale in my environment?
3. With guidance from Tableau, I want to tune my server configuration to suit my hardware, workbooks and environments. How do I measure and monitor the effects of configuration changes to select the best configuration?
4. I need to complete a load test as part of go-live testing for Tableau Server. How do I accelerate the time it takes to do a full load test?

Download package contents

When you download TabJolt, the download package includes this Installation Guide along with a READ.ME file that you should review. You will also find the following files:

File / Folder	What it does
READ.ME	Contains information about the project.
postgresDBSchemaPart1.sql	Creates the performance results repository for use with TabJolt.
postgresDBSchemaPart2.sql	Creates the schema for the performance results for use with TabJolt and necessary data.
Tabjolt Installation Guide	This document.
go.bat	Executes TabJolt tests
PerformanceViz.twb	A sample workbook which consumes performance data generated by TabJolt
/bin /config /lib /testplans	Contains all of the necessary binary and configuration files for TabJolt for Tableau Server 9.0-9.3. Tabjolt does not work with earlier versions of Tableau Server.

Prerequisites

To install and use TabJolt, you need a Windows machine with a minimum of two cores with 8 GB or more RAM. As a best practice, you should monitor this machine for CPU and memory to ensure that your test runs don't create a bottleneck on the load injector.

You must also have downloaded the complete package as described above.

JAVA

If you don't have a recent, working version of JAVA, you need to install the latest version (Tabjolt doesn't work with Java 7 or earlier.). We recommend you install the x64 bit version of JAVA, as TabJolt requests an allocation of 2GB RAM at start up. If you cannot use x64 bit JAVA see the trouble shooting section below for workarounds.

You can get the latest JAVA installation and instructions at this website:
http://java.com/en/download/manual.jsp

Tableau Desktop

If you already have Tableau Desktop 9.0 or later, you can simply use that. If you don't, you can download a free trial version from this website:
http://www.tableau.com/products/desktop

Postgres

TabJolt uses a Postgres database as its performance result store. To ensure that running your tests doesn't impact Tableau Server, do not use Tableau Server's Postgres instance. Instead, you should install a standalone version of Postgres on the same machine where you are installing TabJolt.

NOTE: During Postgres installation, make sure that the database superuser password is set to `testresults`

Download and install the latest Postgres from the following website:

http://www.enterprisedb.com/products-services-training/pgdownload#windows

If you are running Postgres over a Remote Desktop Session, the install splash screen might not appear and the installer might not run. If this happens, navigate to the installation file from an elevated command prompt and then run the installer from the command prompt.

In some cases, we found that the easiest way to install Postgres is to use the default install path—changing the defaults might prevent your Postgres installation from completing in some cases. For help on Postgres, you should review community posts on the Postgres forums.

Configure Postgres

To configure Postgres, you must execute the Postgres SQL scripts that you downloaded (described in the preceding table) in the order that is specified in their file name.

To execute the scripts:

1. Open the administration tool PGADMIN III, which is installed as part of the Postgres installation.
2. Select one of the databases to launch the SQL query execution UI from PGADMIN III.

   IMPORTANT: The next step creates a new database.

3. After you open the SQL query execution UI, open and run the *postgresDBSchemaPart1.sql* script.

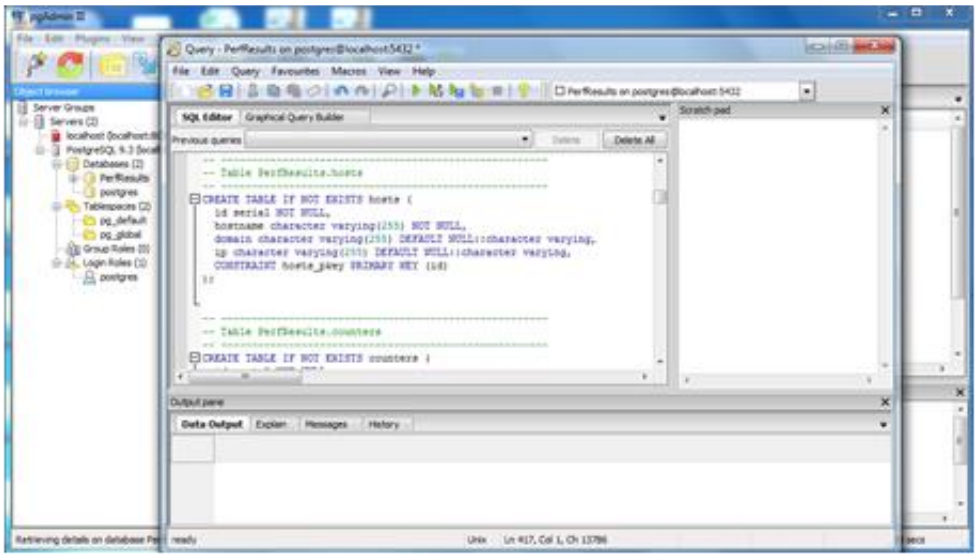


Figure 1: The PGADMIN III console.

Next, connect to the newly created database (PerfResults) and then run the second script, *postgresDBSchemaPart2.sql*, which creates the schema and necessary users to allow TabJolt to run.

Install and configure TabJolt

After the prerequisites are installed and working, unzip *tabjolt-master.zip* to c:\tabjolt. (If your install directory is in a different location, you should factor that in for all the following instructions).

To configure TabJolt, first, edit *c:\tabjolt\config\ServerTestConfig.yaml* and update the following field to point to your Tableau Server URL. Note that if your server is configured for SSL, the hostName will be https://yourhost.

```
hostName: http://yourhost
```

Important: Ensure there is a space between hostname: and the URL of the host. The example above will function correctly, while TabJolt will fail when it attempts to leverage the string below:

```
hostName:http://yourhost
```

Configure for metrics collection

TabJolt can collect both Windows performance metrics (perfmon) data and JMX data from Tableau Server very easily. If configured, metrics collection for the server occurs in the background during a load test. Although optional, metrics collection will help you identify performance bottlenecks that may happen.

You can configure TabJolt to collect performance counter information from all the workers, as well as from the injector (the injector machine is the machine where you run TabJolt from). It is important to monitor resource usage from the injector machine along with the workers to make sure the performance bottleneck is not from client machine.

The configuration settings that control what information to collect are set in *c:\tabjolt\config\dataretriever.config*. You only need to update the <hosts>…</hosts> section (found at the end of the file) to specify which counters are collected from a host.

```
<hosts>
    <host name="localhost">
        <applicableCounterGroups>
            <applicableCounterGroup>machineStatus</applicableCounterGroup>
            <applicableCounterGroup>tableauProcess</applicableCounterGroup>
            <!--enable the following section only after you jmx counter for
tableau-->
            <!--
            <applicableCounterGroup>vizqlserver</applicableCounterGroup>
            <applicableCounterGroup>dataserver</applicableCounterGroup>
            <applicableCounterGroup>vizportal</applicableCounterGroup>
            -->
        </applicableCounterGroups>
    </host>
</hosts>
```

To specify the host to collect metrics from, update the host element by changing <host name="localhost"> to <host name="yourhostname ">. Then update the <applicableCounterGroups>…</applicableCoutnerGroups> section within <host>…</host> to specify which performance counter you are collecting from that host (i.e., you will only need *machineStatus* counter groups for the injector machine but you will need additional counter groups for the workers running other Tableau processes.

If you need to collect performance counter information from multiple hosts, simply replicate the <host> …</host> section and update the host name for each additional host.

Windows performance Counter Group

By default, TabJolt collects the windows counter information remotely using the user account running TabJolt. If the user account doesn't have the proper permissions to collect windows counter information, you can impersonate a different user by setting an impersonation user as shown here:

```
<!--impersonation is only necessary when you need to collect windows perf
counters from another domain which your current run-as user doesn't have
permission to access-->
<!--
<settings>
    <impersonation userName="user" password="password" domain="domainName"/>
</settings>
-->
```

Here are the Windows performance counter groups that are collected:

Counter Group	Description
machineStatus	Collects machine level Memory, Processor, LogicalDisk, and Network Interface windows performance counters information
tableauProcess	Collects process level Memory and Processor windows performance counter information for the *backgrounder*, *dataserver*, *httpd*, *tabrepo*, *tabspawn*, *tabsvc*, *vizqlserver*, *wgserver*, and *vizportal* processes.

JMX Counter Group

JMX metrics give you better visibility into the performance of key Tableau Server processes under load. To activate JMX data collection, you must enable the feature on Tableau Server (by default it is disabled).

To enable collection of JMX metrics on Tableau Server, run the following tabadmin commands:

```
tabadmin stop

tabadmin set service.jmx_enabled true
```

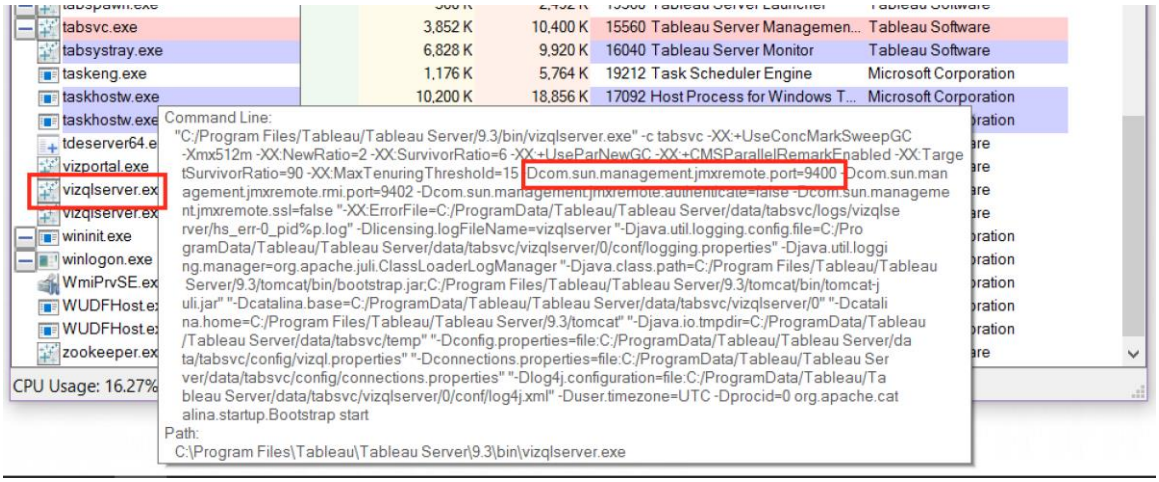
```
tabadmin configure

tabadmin start
```

Once you have enabled JMX counters on your Tableau server, uncomment <applicableCounterGroup> elements in the <host> section of *dataretriever.config* to start collecting information from the processes you are interested in. A quick test to check if the JMX counters are set properly is done using JConsole, a free java tool. In JConsole, connect to this URL:

service:jmx:rmi:///jndi/rmi://<*hostname*>:<*portnumber*>/jmxrmi

To get JMX port numbers, you can run Process Explorer on your Tableau Server (it can be downloaded from this link.). When launching Process Explorer, right click the program and **Run as Administrator** otherwise you will see "Error Opening Process" messages when browsing.

After clicking the **Process** column header to sort processes by name, mouse over each Tableau server component in the table below and look up the jmx port number (i.e. -Dcom.sun.management.jmxremote.port=**9400** for vizqlserver component in the following screenshot).



Record the JMX port for each of these processes:

Counter Group	Description
vizqlserver	Collects JMX counter information for Vizql process.
dataserver	Collects JMX counter information for data server process.

searchservice	Collects JMX counter information for search and browse process.
vizportal	Collects JMX counter information for application server process.

Note that your Tableau Server may run multiple copies of vizqlserver, dataserver, and potentially vizportal. Record the JMX port of each and every process. If you have deployed Tableau Server to multiple machines, repeat this process on each box.

Once you have the information in question, open *c:\tabjolt\config\dataretriever.config* and verify that the settings in the jmx/components section match your computer.

In the sample below, TabJolt will query:

- Two instances of vizqlserver (on ports 9400 & 9401)
- Two instances of dataserver (ports 10000 & 10001)
- A single instance of searchservice and vizportal

```
<jmx duration="300" outputFile="d:\jmxcounter.csv">
    <settings>
        <scheduler threadCount="5" />
        <connectorPoolConfiguration maxActive="5" testOnBorrow="true" testOnReturn="false" />
    </settings>
    <components>
        <component name="vizqlserver" serviceURL="service:jmx:rmi:///jndi/rmi://%s:9400/jmxrmi" />
        <component name="vizqlserver#1" serviceURL="service:jmx:rmi:///jndi/rmi://%s:9401/jmxrmi" />
        <component name="dataserver" serviceURL="service:jmx:rmi:///jndi/rmi://%s:10000/jmxrmi" />
        <component name="dataserver#1" serviceURL="service:jmx:rmi:///jndi/rmi://%s:10001/jmxrmi" />
        <!-- <component name="wgserver" serviceURL="service:jmx:rmi:///jndi/rmi://%s:8300/jmxrmi" /> -->
        <component name="searchservice" serviceURL="service:jmx:rmi:///jndi/rmi://%s:9004/jmxrmi" />
        <component name="vizportal" serviceURL="service:jmx:rmi:///jndi/rmi://%s:8900/jmxrmi" />
    </components>
```

Note that the wgserver process has been commented out in the file above. Beginning in Tableau 9.3, wgserver is disabled by default – therefore it makes no sense to attempt to monitor it.

Choose your workload

Workload on Tableau Server that is user facing is related to loading workbooks and interacting with visualizations within the workbook. In addition, there are background workloads, like extract refreshes and subscriptions. Often these background processes are set up to run in the middle of the night or off peak hours so they don't interfere with user loads.

First, pick the set of workbooks you want to test. To point TabJolt to your workbooks and visualizations, ensure the target workbook is published to Tableau Server.

Populate vizpool.csv

Go to *c:\tabjolt\config\vizpool.csv* and add the link to the target visualization in the *vizpool.csv* file. The best way to get this link is to manually navigate to it from the browser and copy the URL to the clipboard, and then paste it in the vizpool.csv file.

For example, the default site view URL in the browser looks like this:

```
http://localhost/#/views/WorldIndicators/GDPpercapita?:iid=1
```

Your vizpool.csv will contain this:

```
/views/WorldIndicators/GDPpercapita
```

As you can see, we removed the http://localhost and the "#".

The non-default site view URL in the browser looks like this:

```
http://localhost/#/site/newsite/views/WorldIndicators/GDPpercapita?:iid=1
```

The vizpool.csv in this case contains:

```
/site/newsite/views/WorldIndicators/GDPpercapita
```

You can now append a query parameter to the Viz Url to force a certain server behavior (You can find a list of query parameters here). Your vizpool.csv might look something like:

```
/site/newsite/views/WorldIndicators/GDPpercapita?:render=false&:refresh
```

IMPORTANT: Before you further scale your load tests, you must ensure that a single user test on this workbook performs within your expectations. If not, you should optimize the workbook by following best practices for workbook authoring.

Also, you may want TabJolt to auto-discover all the available views across all sites on the server. Instead of specifying each individual view to load, you can set the auto-discover mode to find them all. This is done by changing the auto-discover mode in *c:\tabjolt\config\PerfTestConfig.yaml* from "vizDataSource: csv" to "vizDataSource: web".

Choose your load mix

The workload mix that really matters for Tableau Server scale testing is not how many people are logged into the sever, but how many users are concurrently loading and interacting with visualizations.

To that end, we have provided the following three load mixes out of the box. Choose the one that best meets your needs.

- InteractVizLoadTest.jmx
- ViewVizLoadTest.jmx
- ViewInteractVizLoadTest.jmx

These load mixes are located in the **testplans** directory under the main install location. The underlying JMeter execution engine uses these files, so don't add to or change anything in them.

For the interaction mix, *InteractVizLoadTest.jmx*, TabJolt selects the URLs for the views (these views you provide in the *vizpool.csv* file, based on a uniform distribution). Next, it tries to load the view. After the viz is loaded, TabJolt checks whether the viz has any elements that allow interaction with the view (such as a slider bar, drop-down menu, and so on). If the view has interaction elements, TabJolt performs those interactions without requiring script development. If the view doesn't have any interactions, TabJolt selects marks on view.

The *ViewVizLoadTest.jmx* is a simple test and just loads the visualization without doing any interactions.

The *ViewInteractVizLoadTest.jmx* will do an interaction test for 50% of the text time and a view test with the other 50% of the test time.

You can specify the load mix you want on the command line using the `--t` command line parameter.

The command line parameter `--d` is the test run duration in seconds, and `--c` is the number of virtual users (clients) that you want TabJolt to use.

Set Think Time

You can use the think time settings to create a pause between a tests or between interaction operations (for an interact test case). Think time, in milliseconds, is set in the *c:\tabjolt\config\PerfTestConfig.yaml* with the following configuration entry:

```
# set think time to create a pause between tests
thinktimeBetweenTest: 0
# set think time to create a pause between view and interaction
thinktimeBeforeInteraction: 0
```

Note: Think time is reflected in the response time metrics you view in in the console and PerformanceViz.twb. For example, if the response time on a specific workbook is normally 2500ms and you add 1000ms of think time in PerfTestConfig.yaml, PerformanceViz.twb and the console will reflect a grand total of 3500ms for viz response time. The viz is still executing in 2500ms, of course – but think time makes it *appear* it is executing more slowly. Don't be fooled!

Manage user login

User login and logout requests are handled by the VizPortal component of Tableau Server and do not follow the execution path to VizQLServer, which services end user requests. TabJolt allows you to either leverage the built-in guest account, or specify a synthetic test user during test execution.

To use the built-in guest account, you must have a CORE server license, and you should have configured guest access on your server by checking the **Enable Guest account** option in your administration area on the Tableau Server General Settings page (shown below). You can then use TabJolt to stress or load-test Tableau Server using workbooks or visualizations that you have designed and developed for your users by using the built-in guest account.

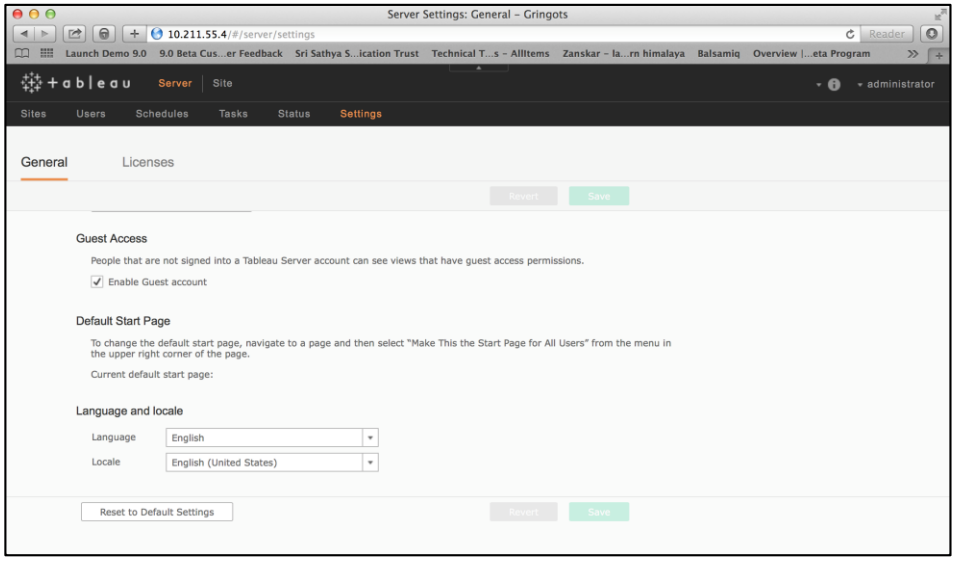


Figure 2: Tableau Server General Settings page, showing guest access.

However, if you want to use a test user to log in to server and then access the visualization, you must configure TabJolt to use one or more test user accounts. To provide test user names and passwords for login, you should edit *c:\tabjolt\config\userpool.csv:*

```
UserName,Password,Role
andy,andyP@ssword,ViewerOrInteractor
russell,russellP@ssword,ServerAdmin
julie,julieP@ssword,ViewerOrInteractor
```

In the file above, three user names (andy, russell, julie) and passwords (andyP@ssword, russellP@ssword, julieP@ssword) have been specified.

These users can exist in active directory or a local Tableau Server. The users must be valid, active, and have permissions on the vizzes you wish to test.

A **role** has also been set for each user. **ViewerOrInteractor** should be leveraged for "normal users", while **ServerAdmin** tells TabJolt that the particular user has admin permissions on the server. Users associated with the ServerAdmin role are used to access the Tableau Server metadata database when TabJolt is placed in "web mode" (see the topic Populate vizpool.csv for more information).

Important: If the Guest account is enabled on your Tableau Server, TabJolt will leverage it during testing regardless of whether there are users configured in userpool.csv. To force TabJolt to make use of your user list, set *forceLogin* to **true** in *PerfTestConfig.yaml:*

```
# If no, it will try to logon at
# the parameter is to override th
forceLogin: true
# Tabjolt now support login from
```

ServerTestConfig.yaml also contains a backwards-compatible set of properties which allows you to specify the credentials of a *single* user. The user you add here must be an administrator. Typically, this account is leveraged to collect viz names in your Tableau Server when using the vizDataSource type "web" or if *userpool.csv* is empty.

```
users:
    - !!com.tableausoftware.test.server.configuration.User
      name: russell
      password: russellP@ssword
```

If userpool.csv is empty, and you don't add a user name above, TabJolt tries to use the built-in guest account and fails if you don't have your guest access set up correctly.

Important: It is necessary to have a single space between the property/value pairs above. The following will fail:

```
users:
    - !!com.tableausoftware.test.server.configuration.User
      name:russell //bad!
      password:russellP@ssword //bad!
```

Run a TabJolt test

Now, you can start your scale test by using a Windows command prompt, navigating to *c:\tabjolt* and running the example command below for a short test. The command tells TabJolt to run the test for 240 seconds (using the **–d** parameter) and to run a single user client (**--c**). You can, of course, change these parameters.

```
go --t=testplans\InteractVizLoadTest.jmx --d=240 --c=1
```

The test will run and you will results displayed on the console.

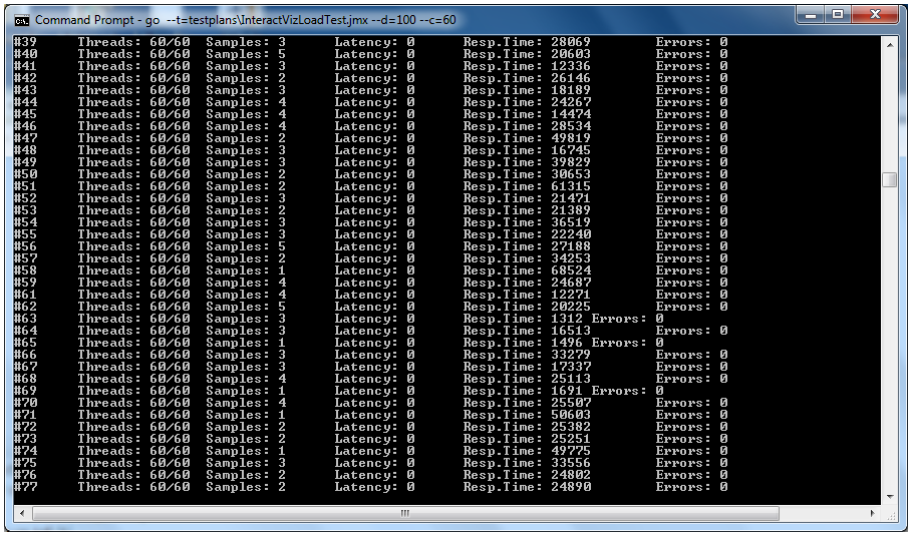


Figure 3: TabJolt running a test.

At the end of the run, on the command line, you will get the run ID, which you need to use as a filter when you analyze your data in Tableau Desktop.

You should explore all the options with the `go` command. If you expect to do a lot of runs, you should give your runs a useful description, by using the command **--r** followed by the description for the run.

Analyze results

After the run is finished, open the analysis workbook located at *c:\tabjolt\PerformanceViz.twb* using Tableau Desktop from the same computer. You must provide the user name and password for your TabJolt Postgres repository that you installed above (remember, this is not your Tableau Server repository).

You can view the test results and review each of the worksheets.

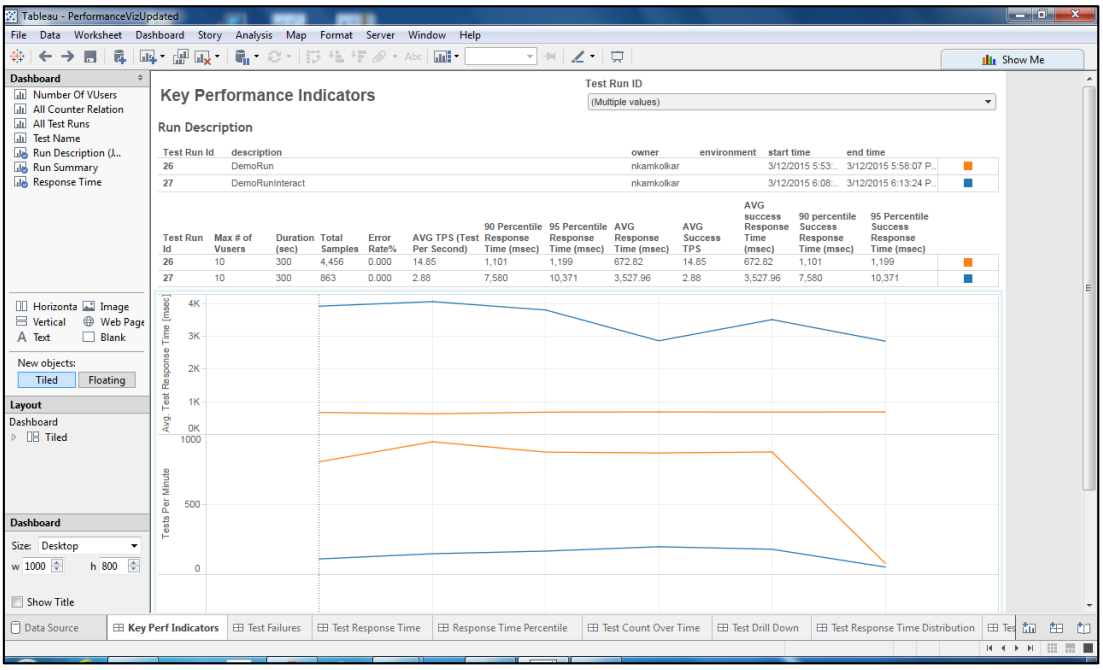


Figure 4: Tableau Desktop showing the results captured from a test run.

Analyzing your load test data is as exploratory as working with any data with Tableau. TabJolt has some key worksheets as starting point. Standard KPI metrics like response times, test cases per second (TPS), host metrics, and JMX metrics (if configured) appear in the workbooks automatically.

A test case is defined as either a "view" or an "interact" test case as described above. These parent test cases might have many child test steps to run. For example, as part of loading a new view, we might create a bootstrap session for the view, get a customized view, or perform operations after the load. These are subtests of the parent test case:

Test Case (Load View)

Boot Strap Request
Get Customized View
Perform Post Load Operations

When you look at these results, you can quickly find patterns and check how your workload is behaving under load.

Before proceeding with a larger scale test, best practice is to ensure that your workbook is optimized for a single user. If your workbook takes a very long time just for one user, you should either follow best practices on how to author workbooks for performance or request Tableau to help.

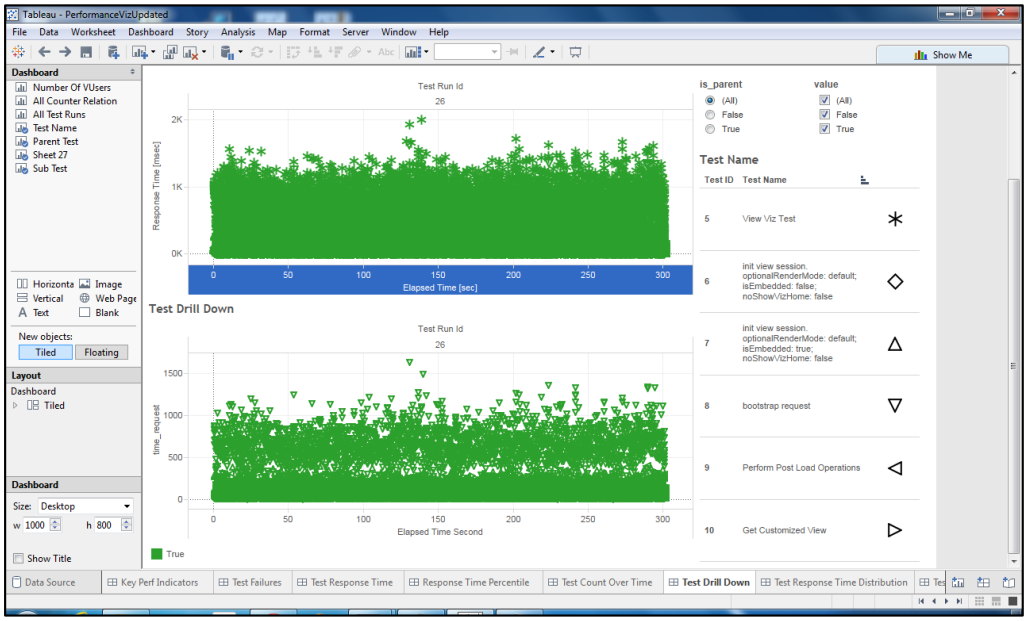


Figure 5: Tableau Desktop showing test drill down from a test run.

Customizing workload mix

The default workload mix (*InteractVizLoadTest.jmx*, *ViewVizLoadTest.jmx*, and *ViewInteractVizLoadTest.jmx*) has a constant load pattern which means that the number of threads stays the same throughout the run. Since TabJolt is built on top of JMeter, it is

possible to customize the workload mix to have other load patterns also supported by JMeter (stepup, RPS, and ultimate). There are few templates already in *c:\tabjolt\testplans\CustomizedTestPlan*. You can start by opening the test plan which has the load pattern that you want in the JMeter UI. Here the steps to make a custom workload mix:

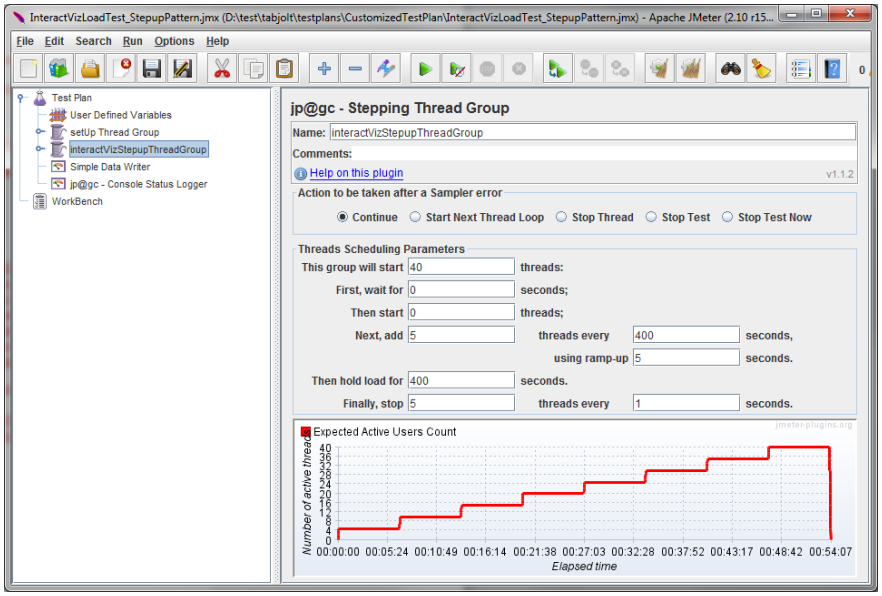
1. As an example, we use *c:\tabjolt\testplans\CustomizedTestPlan\InteractVizLoadTest_StepupPattern.jmx*.

2. Make a backup of the test plan.

3. To open JMeter UI, type in *bin\jmeter.bat* from the TabJolt root folder.

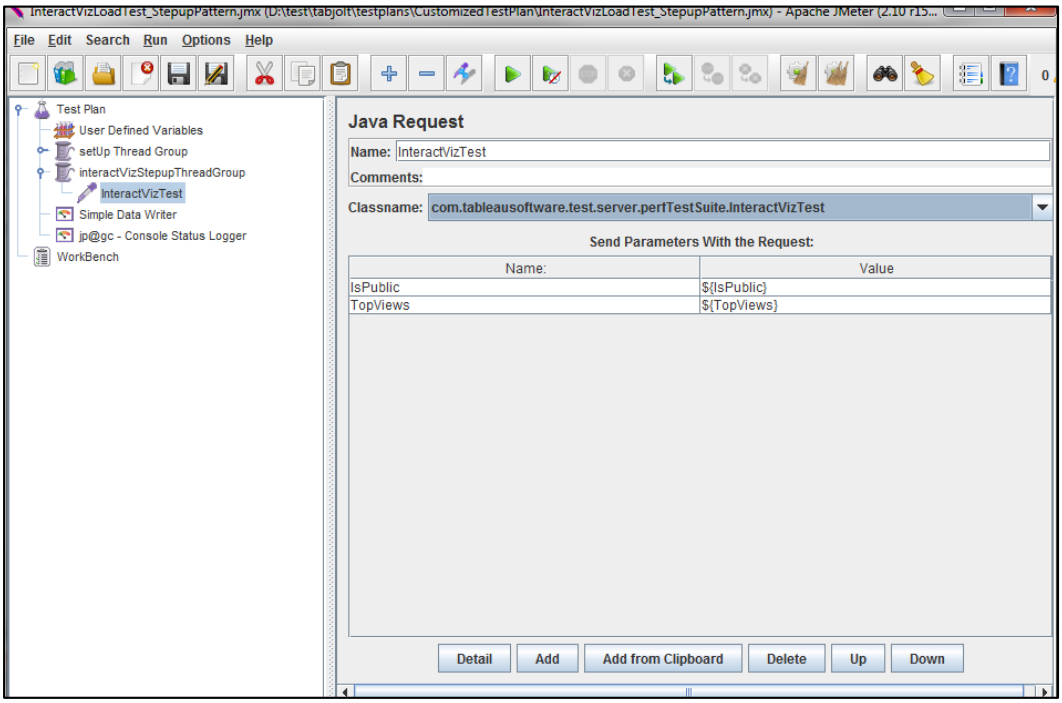
4. Once the Jmeter UI is opened, load the test plan *InteractVizLoadTest_StepupPattern.jmx* by selecting **File->Open** and entering the test plan path.

5. Click on the **interactVizStepupThreadGroup.**



6. You can then change the *stepup load pattern* to match you want, such as extending the run duration and the maximum number of threads. Note that the **--d** and **--t** parameters from the go command won't work anymore if you are using the customized test plan. However, you still want to set **--d** to have same time duration as the test plan so that TabJolt can connect performance counter information on the background with the same duration as the load generation.

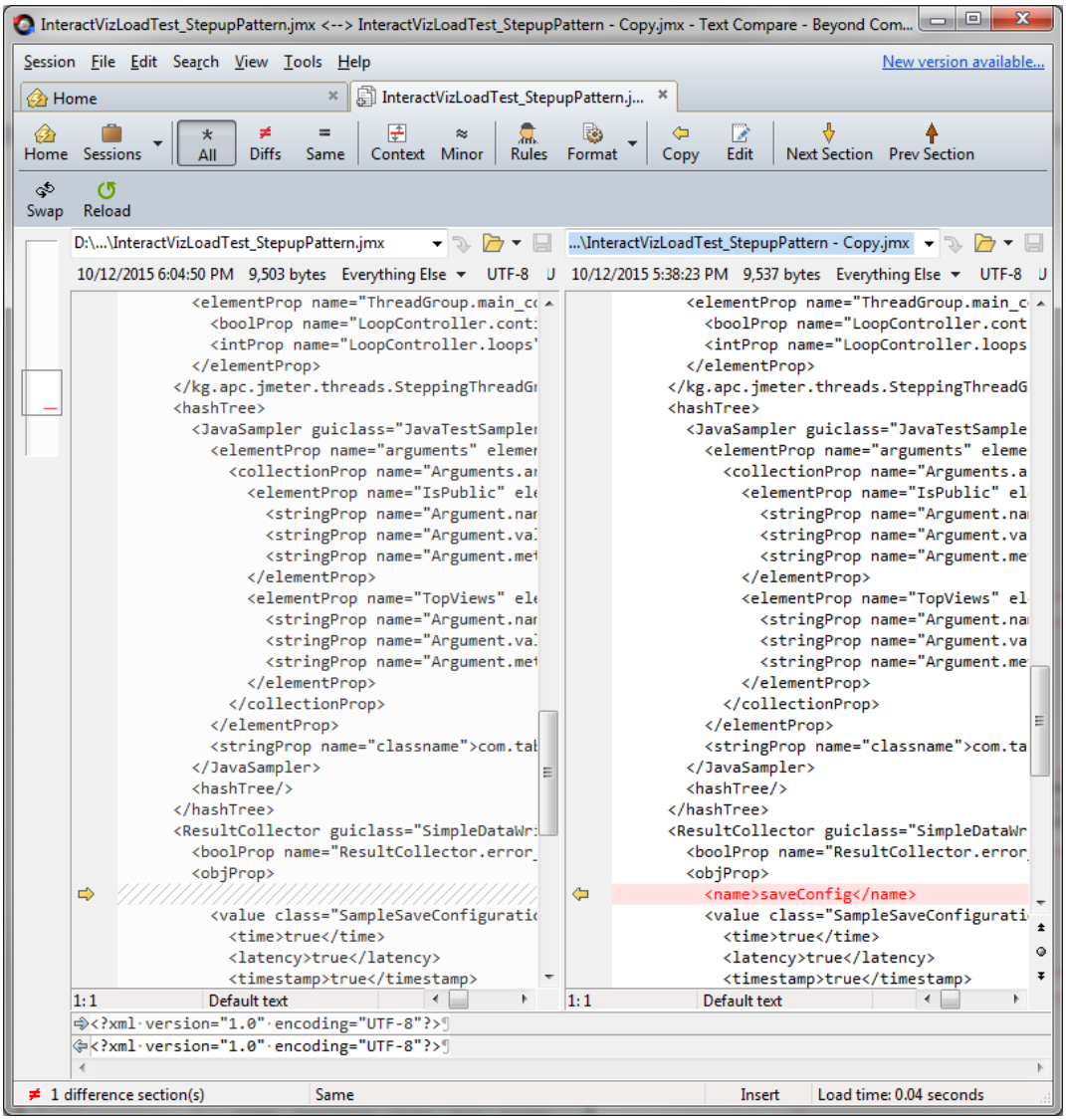
7. To change the test case from interact to view only, expand **interactVizStepupThreadGroup**, select **interactVizTest**, click on the **Classname** dropdown, and choose *com.tableausoftware.test.server.perfTestSuite.ViewVizTest*.



8. If you want to have both the view and interact tests running in the mix, replicate *interactVizStepThreadGroup*.

9. Once you are done, save the test plan.

10. There is a bug in JMeter which will give invalid results in the output file. You need to make a change the test plan to work around the bug. To fix this, load the saved test plan and the backup test plan in Beyond Compare, or a similar comparison utility, and find all the instances where <name>*saveConfig*</name> are missing. Copy over the entries from the backup file to the file which you have just saved. Then save the file again with the additional entries.



11. Now you can use the customized test plan in the go command :

```
go --t=testplans\CustomizedTestPlan\InteractVizLoadTest_StepupPattern.j
mx --d=60 --c=1.
```

Note that the **--d** and **--t** parameter from **go** command won't work anymore if you are using the customized test plan. However, you still want to set **--d** to have same time duration as the test plan, so that TabJolt can connect performance counter information on the background with the same duration as the load generation.

Known Issues and Troubleshooting

1. Is there source code for download?

   TabJolt uses Github as a repository for external distribution and you don't really need to download or edit source code to use TabJolt.

2. Why does TabJolt cause a heap allocation error at start up?

   By design, TabJolt tries to allocate 2GB RAM at startup. If you are NOT running x64 bit version of JAVA or you are running TabJolt on a system that has very low memory, you may run into an error or warning that says

   ```
   JavaHotSpot(TM) 64-Bit Server VM warning: ignoring option
   PermSize=64m; support was removed in 8.0.
   ```

   To solve this, you need to either move to an x64 bit JAVA installation, OR increase RAM in your system, OR update the heap requirement to suit your environment constraints with an aim to maximize the memory available to TabJolt. Set the HEAP range in *C:\tabjolt\bin\jmeter.bat*.

   ```
   set HEAP=-Xms512m -Xmx2048m
   ```

3. How do I turn on verbose logging?

   By default, TabJolt only logs fatal errors (to reduce the performance impact from logging itself). Sometimes, you may want to turn on verbose logging to troubleshoot a problem. To turn on verbose logging, open up the *c:\tabjolt\config\log4j.properties* file. Disable the first line by adding # in front of it and enable the second by removing # in front of it.

   ```
   #log4j.rootLogger=FATAL, stdout, logfile
   log4j.rootLogger=ALL, stdout, logfile
   ```

   The log is saved in *c:\tabjolt\logs\test.log*.

Additional Resources and Feedback

TabJolt Overview Video

Getting Started with TabJolt Video

Analyzing TabJolt Results Video

Tweet us @neeleshkamkol or @tableau

You can also post questions or comments in the forum for the Server Administration Community on Tableau.com.