

Selenium Grid

Documentation & Research

Version	Status	Author	Date
0.1	<i>Initial draft</i>	Juan Krzemien	04/24/2015

Table of Contents

Introduction to Selenium Grid	3
What is Selenium Grid?	3
When to use it	3
How It Works	5
Required hardware per node	7
Master Node	8
Slave Nodes	8
Number of nodes	10
Other optimizations to be done	10
Frequently Asked Questions	11
I need a Selenium Grid right now, I don't care about all this technical crap	11
How do I setup Selenium Hub and Nodes to start at boot time on a Windows machine?	11
How can I configure my service to be able to access Windows GUI/Desktop?	11
How do we kill orphan processes left over before any new runs in Selenium Grid?	13
When we test the application with Selenium Grid, we get nondeterministic results	14
Machines/VMs given by the organization run Windows Server 2012 OS. What's the version of IE there? Can it be downgraded? Updated?	14
How do I control how many tests can a node run in parallel?	15
I'm getting a cryptic error message from Selenium Hub	16
Can you give some tips for running tests against a Selenium Grid?	16
Building a laboratory for testing with Selenium Grid	17
Grid's Hub config file	18
Grid's Node config file	19
Grid's Hub Batch file (Windows)	20
Grid's Node Batch file (Windows)	20
Gathered Metrics	21
Hub Specific Metrics	22
Node 1 Specific Metrics	24
Node 2 Specific Metrics	26

Introduction to Selenium Grid

Organizations are adopting virtualization and cloud-based technologies to reduce costs and increase efficiency. Virtualization and cloud-based infrastructure can also be used to scale the test automation by reducing investment in physical hardware needed to set up the test environment. Using these technologies, web applications can be tested on a variety of browser and operating system combinations. Selenium WebDriver has unmatched support for testing applications in virtual environment, executing tests in parallel, reducing costs, and increasing speed and coverage. This document will cover recipes to configure and execute Selenium WebDriver tests for parallel or distributed execution.

Running tests in parallel requires two things:

- an infrastructure to distribute the tests
- Framework and/or test runner that will run these tests in parallel in the given infrastructure.

What is Selenium Grid?

Selenium-Grid allows you run your tests on different machines against different browsers in parallel. That is, running multiple tests at the same time against different machines running different browsers and operating systems. Essentially, Selenium-Grid support distributed test execution.

When to use it

Generally speaking, there are two reasons why you might want to use Selenium-Grid.

- To run your tests against multiple browsers, multiple versions of browser, and browsers running on different operating systems.
- To reduce the time it takes for the test suite to complete a test pass.

Selenium-Grid is used to speed up the execution of a test pass by using multiple machines to run tests in parallel. For example, if you have a suite of 100 tests, but you set up Selenium-Grid to support 4 different machines (VMs or separate physical machines) to run those tests, your test suite will complete in (roughly) one-fourth the time as it would if you ran your tests sequentially on a single machine. For large test suites, and long-running test suite such as those performing large amounts of data-validation, this can be a significant time-saver. Some test suites can take hours to run.

Another reason to boost the time spent running the suite is to shorten the turnaround time for test results after developers check-in code for the AUT. Increasingly software teams practicing Agile software development want test feedback as immediately as possible as opposed to wait overnight for an overnight test pass.

Selenium-Grid is also used to support running tests against multiple runtime environments, specifically, against different browsers at the same time.

For example, a 'grid' of virtual machines can be setup with each supporting a different browser that the application to be tested must support. So, machine 1 has Internet Explorer 8, machine 2, Internet Explorer 9, machine 3 the latest Chrome, and machine 4 the latest Firefox. When the test suite is run, Selenium-Grid receives each test-browser combination and assigns each test to run against its required browser. In addition, one can have a grid of all the same browser, type and version.

For instance, one could have a grid of 4 machines each running 3 instances of Firefox, allowing for a 'server-farm' (in a sense) of available Firefox instances. When the suite runs, each test is passed to Selenium-Grid which assigns the test to the next available Firefox instance. In this manner one gets test pass where conceivably 12 tests are all running at the same time in parallel, significantly reducing the time required to complete a test pass.

Selenium-Grid is very flexible. These two examples can be combined to allow multiple instances of each browser type and version. A configuration such as this would provide both, parallel execution for fast test pass completion and support for multiple browser types and versions simultaneously.

The selenium-server-standalone package includes the Hub, WebDriver, and Selenium RC needed to run the grid.

How It Works

Selenium Grid builds on the traditional Selenium setup, taking advantage of the following properties:

- The Selenium test, the application under test, and the remote control/browser pair do not have to be co-located. They communicate through HTTP, so they can all live on different machines.
- The Selenium tests and the web application under test are obviously specific to a particular project. Nevertheless, neither the Selenium remote control nor the browser is tied to a specific application. As a matter of fact, they provide a capacity that can easily be shared by multiple applications and multiple projects.

Consequently, if only we could build a distributed grid of Selenium Remote Controls, we could easily share it across builds, applications, projects - even potentially across organizations. Of course we would also need to address the scalability issues as described earlier when covering the traditional Selenium setup. This is why we need a component in charge of:

- Allocating a Selenium Remote Control to a specific test (transparently)
- Limiting the number of concurrent test runs on each Remote Control
- Shielding the tests from the actual grid infrastructure

Selenium Grid calls this component the Selenium Hub.

The Hub exposes an external interface that is exactly the same as the one of a traditional Remote Control. This means that a test suite can transparently target a regular Remote Control or a Selenium Hub with no code change. It just needs to target a different IP address. This is important as it shields the tests from the grid infrastructure (which you can scale transparently).

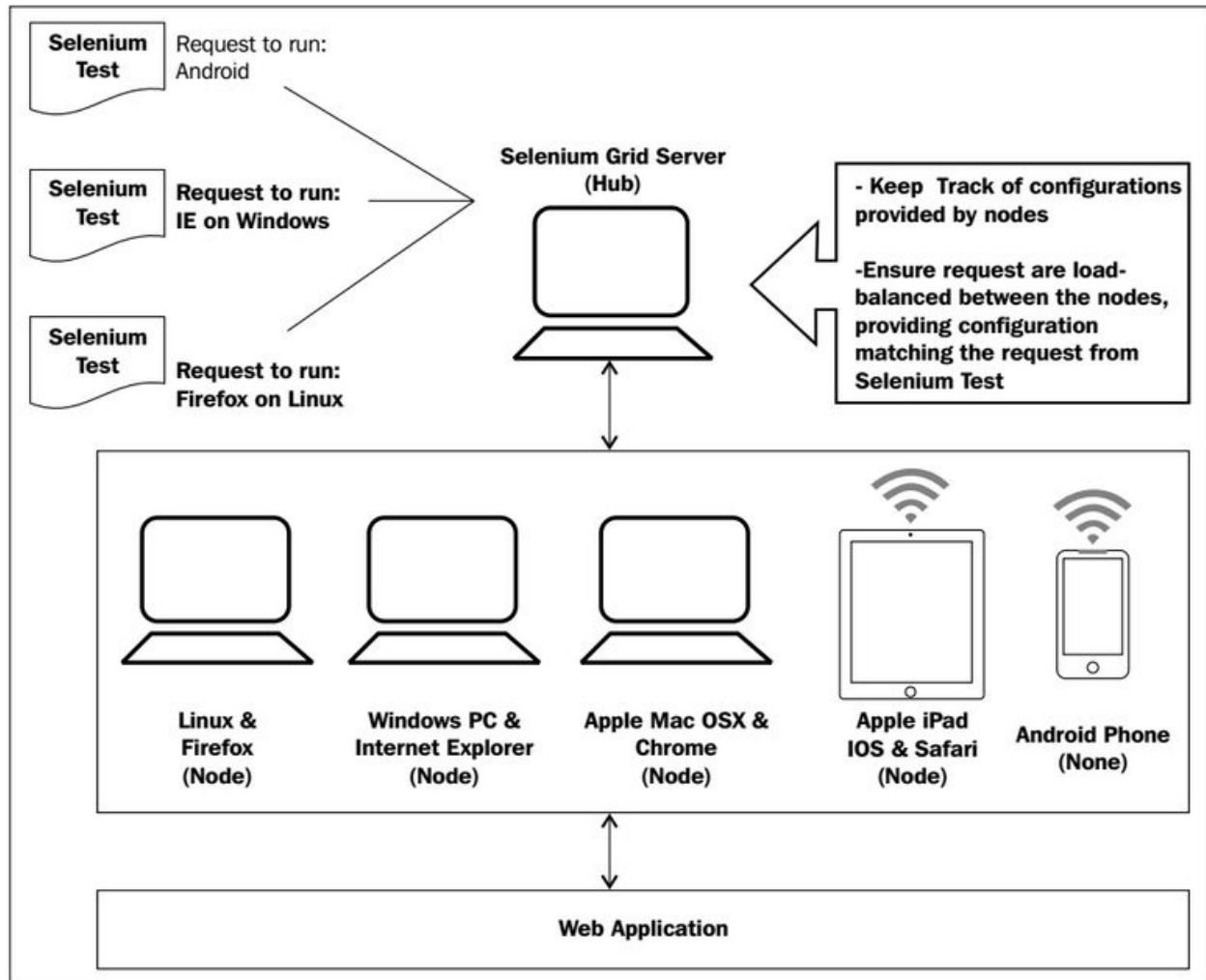
This also makes the developer's life easier. The same test can be run locally on a developer machine, or run on a heavy duty distributed grid as part of a build – without ever changing a line of code.

The Hub allocates Selenium Remote Controls to each test. The Hub is also in charge of routing the Selenese requests from the tests to the appropriate Remote Control as well as keeping track of testing sessions.

When a new test starts, the Hub puts its first request on hold if there is no available Remote Control in the grid providing the appropriate capabilities. As soon as a suitable Remote Control becomes available, the Hub will serve the request. For the whole time, the tests do not have to be aware of what is happening within the grid; it is just waiting for an HTTP response to come back.



Following you can find a diagram of the Selenium Grid architecture having capabilities to run tests on (but not limited to) Linux, Windows and Mac environments. Mobile platforms, such as iOS and Android, are supported as well using Appium as integration framework.



Required hardware per node

There are a couple of notions to take into account before giving away hardware requirements and amount of nodes for a Selenium Grid, since it heavily depends on the type of tests being run and the type of usage you are going to give to the Grid.

As I've mention before, there is a difference between running tests on multiple runtime environments & browsers at the same time to test for homogeneity, than to intend to use the Grid to minimize tests run time while focusing on a single browser.

Sadly, IE browser does not scale well vertically as it does horizontally. Meaning that if many instances of IE are stacked up in a single machine/node, instabilities will be noticed sooner than later. In IE's case it is better to keep a low number of instances per node, and to have as many nodes as possible.

For Firefox and (specially) Chrome browsers that is not the case. They scale well vertically to the point of being able to have up to 10 instances of a browser running at the same time in a low end machine hardware with 512 MB of RAM.

Master Node

The requirements for the Hub node are minimal after its initialization (almost no CPU usage and about 20 MB of RAM). Upon client requests, the load grows a little bit, mostly due to creation and processing of network handlers by the servlets. Running test suites with 10 threads (10 simultaneous users/consumers) against the hub during the experiments did not raise the RAM usage above 55 MB of RAM. CPU usage did not raise above 5%, and only during the connections exchange period.

The test runner, which is downloaded from the VCS along with the tests or as a dependency by the build system, generally consumes very little memory (around 20 MB) but to run tests concurrently it spawns a lot of threads on which it runs the tests. It consumes all the CPU cycles available if the amount of threads is high enough. For large suite runs, take into account that the Test Runner listener is gathering the results of the test runs in an XML, dumped at the end of the run. This can make it consume more memory than it usually does.

To summarize:

- Any single core CPU would suffice for it. If the CPU can be a dual-core it would be better, to allow Hub and Test Runner to run independently from each other
- 1 GB of RAM should suffice to cover the requirements expressed by Hub + Test Runner. If large suites are going to be ran, I would raise the RAM amount to 2 GB.

Slave Nodes

They are only supposed to have desired browsers installed and the Selenium Grid slave node up and running.

The Grid node consumes about 10% of CPU and around 75 MB of RAM when under load of 10 concurrent users.

Know that despite Firefox consumes a little bit more memory than Chrome, Firefox does not need an extra Selenium server instance to handle it as IE and Chrome do, since it is embedded.

With a single core CPU and 512 MB of RAM the node can spawn up to: 2 IE instances **or** up to 10 Chrome instances **or** up to 10 Firefox instances before crashing or having erratic behaviours. Heavy swapping took place during this test runs, meaning that this extreme case should be avoided and that extra memory should be considered.

Having a dual-core CPU makes the tests run more “freely” given that each IE instance can benefit with a dedicated core, and having extra RAM does not force the system into a swapping state so easily.

Therefore, the recommended specs for a Slave node would be:

- A dual-core CPU
- Between 1 and 1.5 GB of RAM for any desired browser combination.

Number of nodes

As explained earlier in this chapter, IE browser does not scale well vertically as it does horizontally:

- 1 slave running 2 IE instances would reduce the time run time by (roughly) half.
- 2 slaves running 2 IE instances each would reduce the time run time by (roughly) fourth.
- 3 slaves running 2 IE instances each would reduce the time run time by (roughly) eighth.

Progression would be as follows (roughly):

$$\frac{T_0}{b^n} \Rightarrow T_1$$

Where:

- T_0 is total suite time not parallelized
- b is number of browsers instances
- n number of nodes (slaves) in Grid
- T_1 is new total time (parallelized)

Other optimizations to be done

We need to take into account also that some tests are not possible to be run in parallel with others. Consider, for example, a test that needs special permissions to be granted/modified for the logged in user in order to test the required scenario. A different test (running in parallel) could log into the application only to find that the section on which it needs to work on is disabled/hidden because of its current permissions (defined by the other test).

Currently, we have a “category” for these tests called “Sequential” that forces the tests runner to run them sequentially after all others tests are done running in parallel. Further optimization could be done, since as the minimalistic example above shows that tests need to be grouped together by functionality, permission levels, access to resources or simply by not reusing the same user in every test.

Frequently Asked Questions

I need a Selenium Grid right now, I don't care about all this technical crap

Just download and execute [Selenium Grid Extras](#). Follow the onscreen instructions. Hakuna matata...

How do I setup Selenium Hub and Nodes to start at boot time on a Windows machine?

Use a simple program called [NSSM](#) (The Non-Sucking Service Manager, actual name, not my fault :P) to install (register) and remove (unregister) programs as Windows services.

Download it and, from a Command Console (cmd.exe), run:

- Install a service: `nssm.exe install`
- Remove a service: `nssm.exe remove <service name>`

When creating a new service, I suggest using a single word name (without spaces) such as "SeleniumHub" so it is easy to invoke `nssm.exe remove SeleniumHub` in case you screw something up while defining the service settings.

You can, for example, use the Selenium [Hub/Node](#) batch files (defined below) as starting commands.

Warning

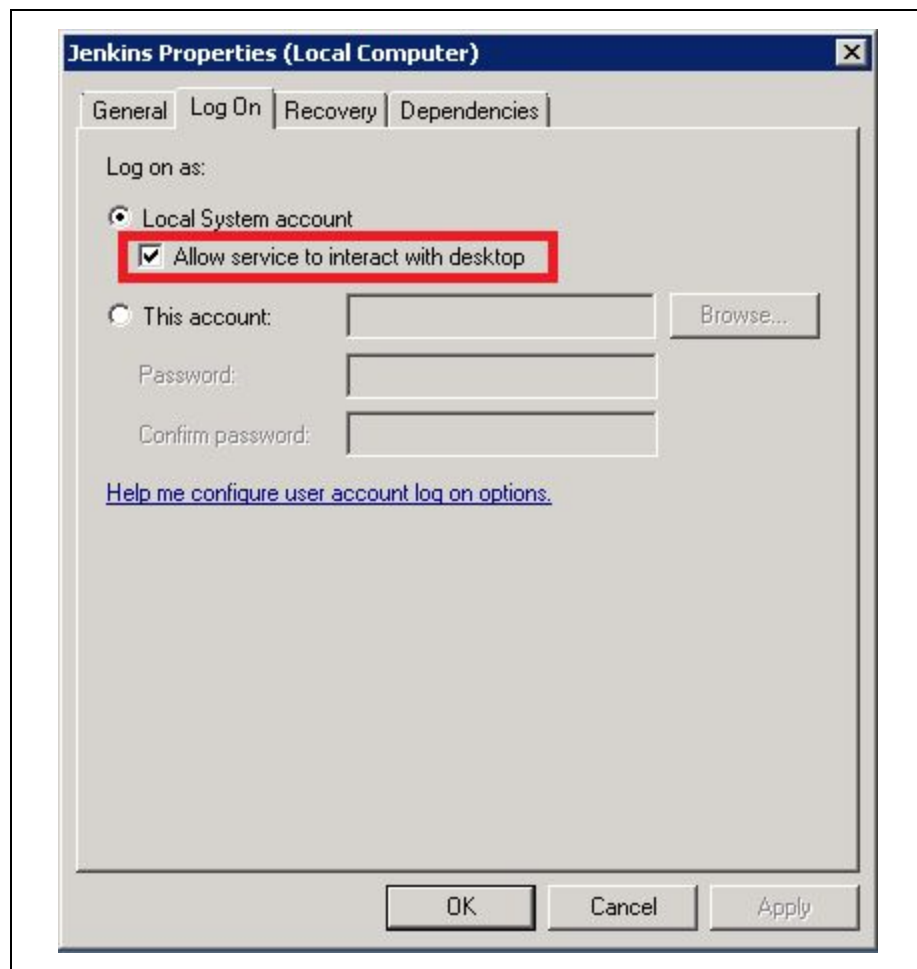
Take into account that programs/scripts linked to Windows Services must not exit immediately (under 1500 ms) or NSSM will think the Service startup failed and you won't be able to start the Service.

This means, remove the **START /B** (move to background) command if you are planning on using the batch scripts defined below as a Windows Service. Windows Services run in background either way so it really does not matter.

How can I configure my service to be able to access Windows GUI/Desktop?


Non-console based windows applications (such as internet browsers) require access to the GUI to be able to perform their UI paints, open/close/handle windows, etc.

By default, Windows does not allow a service to interact against the UI, but this can be changed. Microsoft provides a way for services to interact against a virtual GUI dedicated just for services.

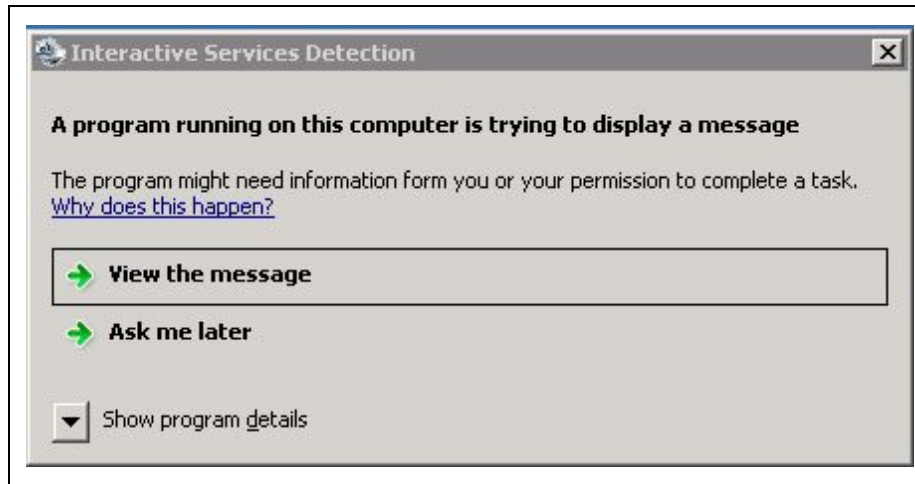


Service allows to interact with a virtual desktop to launch GUI based browsers

Please make sure the "Interactive Services Detection" service is STARTED. By default the startup type is set to Manual, you may like to set it to Automatic as well.

Name	Description	Status	Startup Type
 Interactive Services Detection	Enables user notification of ...		Manual

Every time there is “activity” in this virtual screen (as in a service is rendering something) , you should see a dialog informing that a program is running behind scenes rendering into the virtual screen. This is normal and expected.



Services interacting with the virtual desktop will raise a dialog like this one

As noted in the screenshots, you can use this to setup Jenkins as a service and be able to run automated WebDriver based tests in it without Selenium Grid.

How do we kill orphan processes left over before any new runs in Selenium Grid?

Short answer is: you shouldn't. Selenium Grid includes a "browser reaper feature" and tries its best to keep each node as tidy as possible. The hub will cycle between the slave nodes on a configurable timely basis and attempt to close any session unlinked browser.

If even then you find issues with leftover browsers, there are still plenty of solutions for this:

- First thing would be to implement a more resilient configuration for the hub/nodes that conform the Grid. Avoid node dying and not coming back to life due to OutOfMemory exceptions. Use mechanisms to respawn the Java hub/client node. One option is *JARSpawner*. This is a standalone java application that can continuously monitor a another JVM and if the JVM crashes or exits it can re-spawn it again.
- Avoid OutOfMemory exceptions from really long runs (after days) by recycling your nodes periodically. Selenium Grid is highly extendable, so one or more proxy/servlets can be added to it upon initialization to deal with any kind of issue that we may be having. This proxy when injected into the Grid, would starts counting unique test sessions. After "n" test sessions, the proxy unhooks the node gracefully from the grid and self terminates gracefully.
- **Recommended** - Another option would be to deploy [Selenium Grid Extras](#), which is a repackaged version of Selenium with many enhancements, such as:
 - Explicitly kill web browser after each test
 - Modify settings for Internet Explorer
 - Periodically restart Grid Nodes
 - Restart the OS after several test runs

When we test the application with Selenium Grid, we get nondeterministic results

Most likely some tests are timing out in a non-deterministic manner because the CPU or Network is over-utilized. Monitor CPU and Network activity on all the machines involved. Once you find the bottleneck launch fewer processes. For instance, if the load average is way higher than the number of CPUs on the machine running the remote controls, cut the number of remote controls you launch by two until you get to a sustainable machine load.

Make sure you spend some time figuring out the optimal number of concurrent test runners and remote controls to run in parallel on each machine, before deploying a Selenium Grid infrastructure your organization is going to depend on.

Machines/VMs given by the organization run Windows Server 2012 OS. What's the version of IE there? Can it be downgraded? Updated?

Windows Server 2012 comes with Internet Explorer 10 and does not support earlier versions.

You can only install IE9 in the OS versions mentioned in the list below.

- Windows Vista 32-bit with Service Pack 2 (SP2) or higher
- Windows Vista 64-bit with Service Pack 2 (SP2) or higher
- Windows 7 32-bit or higher
- Windows 7 64-bit or higher
- Windows Server 2008 32-bit with Service Pack 2 (SP2) or higher
- Windows Server 2008 64-bit with Service Pack 2 (SP2) or higher
- Windows Server 2008 R2 64-bit

In order to have an earlier version of IE running on a Windows Server 2012 machine you would need to virtualize a guest Windows 7 with it.

Portabilization of an Internet Explorer version is not a solution given that it is **illegal** to re-package versions of IE.

How do I control how many tests can a node run in parallel?

MaxInstances represents how many instances of same version of a browser can be run on a given node.

For example, if I have the following configuration:

```
{
  "browserName": "firefox",
  "maxInstances": 5,
  "seleniumProtocol": "WebDriver"
},
{
  "browserName": "internet explorer",
  "maxInstances": 5,
  "seleniumProtocol": "WebDriver"
}
```

Then I can run 5 instances of Firefox as well as 5 instances of IE at the same time in a remote machine. So a user can run a total of 10 instances of different browsers (FF & IE) in parallel.

MaxSession represents how many browsers (*any browser and any version*) can be run in parallel at a time in the node. This overrides the *MaxInstances* settings and can restrict the number of browser instances that can run in parallel.

For above example, when `maxSession=1` it forces Grid to never have more than 1 browser running at any given moment.

With `MaxSession=2` you can have 2 Firefox tests at the same time, or 1 Internet Explorer and 1 Firefox test), irrespective of what *MaxInstances* you have defined.

I'm getting a cryptic error message from Selenium Hub

Please refer to the following table for the error condition and its possible solution:

Reason	Cause/fix
TIMEOUT	The session timed out because the client did not access it within the timeout. If the client has been somehow suspended, this may happen when it wakes up
BROWSER_TIMEOUT	The node timed out the browser because it was hanging for too long (parameter browserTimeout)
ORPHAN	A client waiting in queue has given up once it was offered a new session
CLIENT_STOPPED_SESSION	The session was stopped using an ordinary call to stop/quit on the client. Why are you using it again??
CLIENT_GONE	The client process (<i>your</i> code) appears to have died or otherwise not responded to our requests, intermittent network issues may also cause
FORWARDING_TO_NODE_FAILED	The hub was unable to forward to the node. Out of memory errors/node stability issues or network problems
CREATIONFAILED	The node failed to create the browser. This can typically happen when there are environmental/configuration problems on the node. Try using the node directly to track problem.
PROXY_REREGISTRATION	The session has been discarded because the node has re-registered on the grid (in mid-test)

Can you give some tips for running tests against a Selenium Grid?

- If your tests are running in parallel, make sure that each thread deallocates its webdriver resource independently of any other tests running on other threads.
- Starting 1 browser per thread at the start of the test-run and deallocating all browsers at the end is **not** a good idea. If one test-case decides to consume abnormal amounts of time you may get timeouts on all the other tests because they're waiting for the slow test. Also, browsers consume more RAM the longer it is traversing a web application.

Building a laboratory for testing with Selenium Grid

In order to backup any suggestions made in this document, some demo exercises were run in a sample laboratory that I improvised. The idea was to run real tests under the same conditions that we will face if the enterprise proposed in this document comes into reality, but data gathered from this experiment is all I could collect when at the time of writing this document (I also have life, you know?).

In order to create the laboratory on which the tests were made, the following hardware was used:

Windows edition	
Windows 8 Pro	
© 2012 Microsoft Corporation. All rights reserved.	
Get more features with a new edition of Windows	
System	
Rating:	6.7 Windows Experience Index
Processor:	Intel(R) Core(TM) i7-4770K CPU @ 3.50GHz 3.50 GHz
Installed memory (RAM):	16.0 GB (15.4 GB usable)
System type:	64-bit Operating System, x64-based processor
Pen and Touch:	No Pen or Touch Input is available for this Display

This single machine was the host for another 3 VirtualBox based Windows 7 guest VMs, freely distributed for 30 days by Microsoft from its [virtualization site](#).

This VM comes with a Windows 7 Enterprise edition and Internet Explorer version 9. Other combinations of domestic Windows versions/IE can be found there.

This downloaded VM was imported into VirtualBox with its default settings. Required software to carry on with the tests was installed on it.

Once the VM was as desired, I made a snapshot of it to be able to reproduce the laboratory in the future (without limiting its usage to 30 days) and cloned it two more times to create a total of 3 VMs running on a single host machine.

Hardware specifications were as follows:

- Single core CPU
- 1024 MB of RAM
- NIC in bridged mode, unrestricted access to host's NIC.

Additional software installed:

- JRE 8 (Selenium Grid is a Java based software)
- Firefox browser
- Chrome browser
- Single shared folder with host machine containing:
 - Selenium standalone JAR file version **3.0.0-beta2**
 - IEDriverServer (always use latest version, 32 bits)
 - ChromeDriver (always use latest version)
 - GeckoDriver (only if you are using Selenium 3.0+, you don't need this driver for 2.x. Always use latest version)
 - Selenium-Grid Hub JSON config file
 - Selenium-Grid Node JSON config file
 - A BATCH script file to launch Selenium Hub with some default arguments
 - A BATCH script file to launch Selenium Node node with some default arguments

Grid's Hub config file

```
{
  "host": null,
  "port": 4444,
  "newSessionWaitTimeout": -1,
  "servlets" : [],
  "prioritizer": null,
  "capabilityMatcher": "org.openqa.grid.internal.utils.DefaultCapabilityMatcher",
  "throwOnCapabilityNotPresent": true,
  "nodePolling": 5000,
  "cleanUpCycle": 5000,
  "timeout": 30000,
  "browserTimeout": 90000,
  "maxSession": 10,
  "jettyMaxThreads":-1
}
```

Grid's Node config file

```
{
  "capabilities":
  [
    {
      "browserName": "firefox",
      "maxInstances": 5,
      "seleniumProtocol": "WebDriver"
    },
    {
      "browserName": "chrome",
      "maxInstances": 5,
      "seleniumProtocol": "WebDriver"
    },
    {
      "browserName": "internet explorer",
      "maxInstances": 1,
      "seleniumProtocol": "WebDriver"
    }
  ],
  "proxy": "org.openqa.grid.selenium.proxy.DefaultRemoteProxy",
  "maxSession": 5,
  "port": 5555,
  "register": true,
  "registerCycle": 5000,
  "hub": "http://<MASTER HUB IP>:4444",
  "nodeStatusCheckTimeout": 5000,
  "nodePolling": 5000,
  "role": "node",
  "unregisterIfStillDownAfter": 60000,
  "downPollingLimit": 2,
  "debug": false,
  "servlets" : [],
  "withoutServlets": [],
  "custom": {}
}
```

Grid's Hub Batch file (Windows)

```
@ECHO OFF

CD <directory with selenium server, config files and drivers>

START /B "Selenium Hub" "%JAVA_HOME%\bin\java.exe" -jar
c:\Automation\selenium-server-standalone-3.0.0-beta2.jar -port 4444 -role hub
-hubConfig c:\Automation\selenium-hub.json > c:\Automation\selenium-hub.log
```

Grid's Node Batch file (Windows)

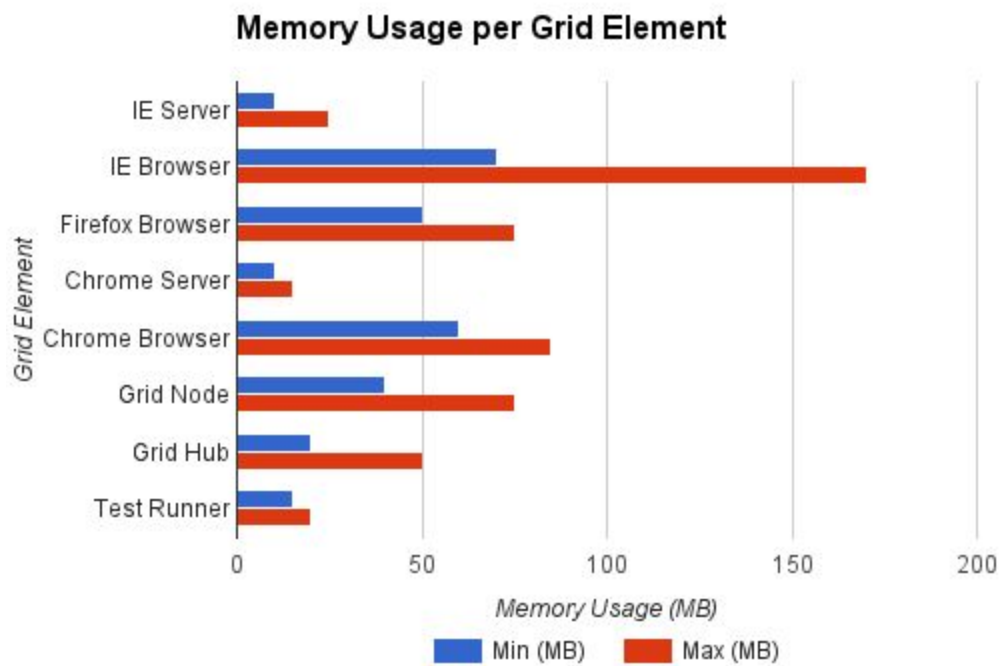
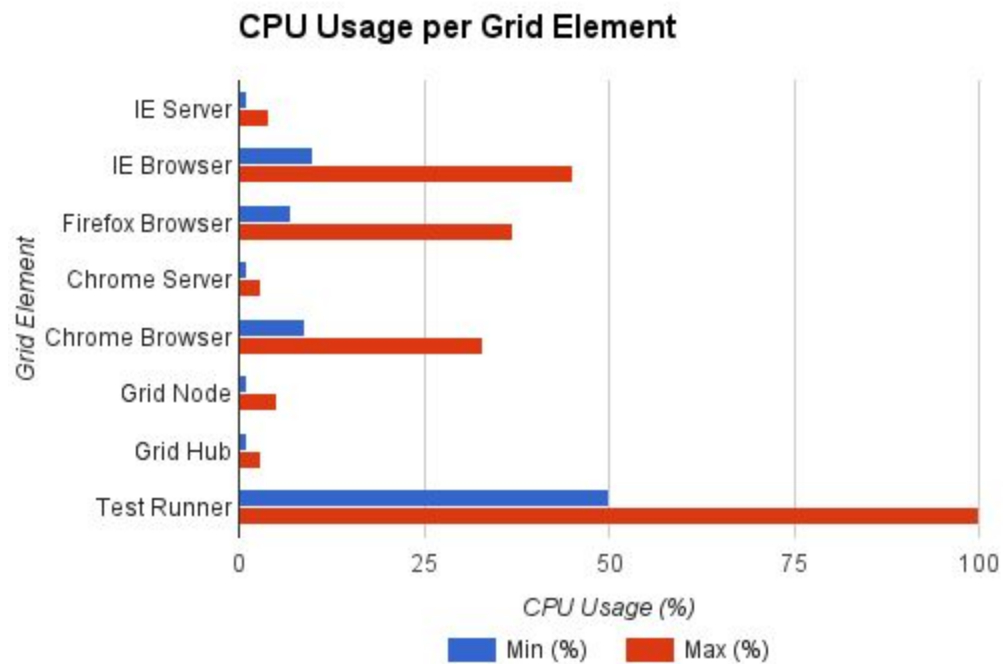
```
@ECHO OFF

CD <directory with selenium server, config files and drivers>

for /f "delims=[] tokens=2" %a in ('ping %computername% -4 -n 1 ^| findstr "["]')
do (set thisip=%a)

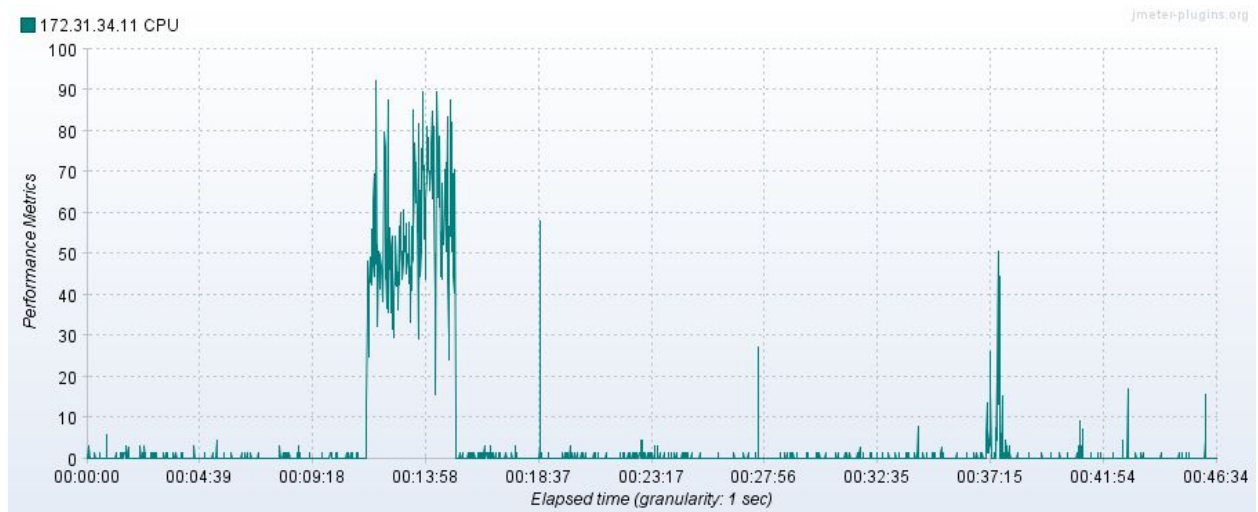
START /B "Selenium Node" "%JAVA_HOME%\bin\java.exe"
-Dwebdriver.gecko.driver=c:\Automation\geckodriver.exe
-Dwebdriver.chrome.driver=c:\Automation\chromedriver.exe
-Dwebdriver.ie.driver=c:\Automation\IEDriverServer.exe -jar
selenium-server-standalone-3.0.0-beta2.jar -port 5555 -role node -hub
http://192.168.100.55:4444/hub/register -nodeConfig selenium-node.json >
selenium-node.log
```

Gathered Metrics



Hub Specific Metrics

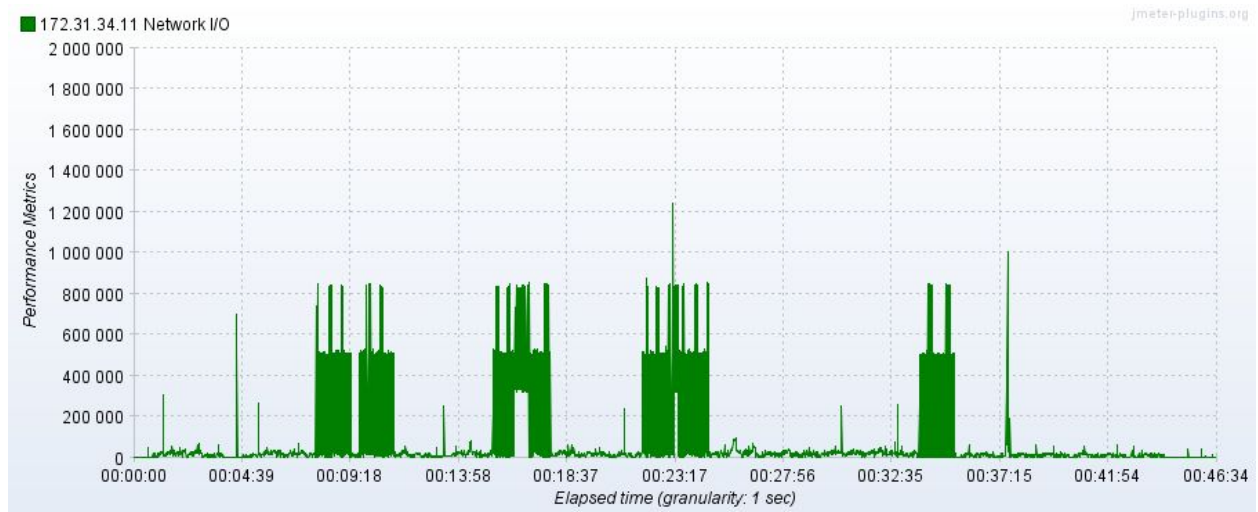
CPU - Single Core VM



Memory - 1024 MB RAM



Network - Bridged VM NIC

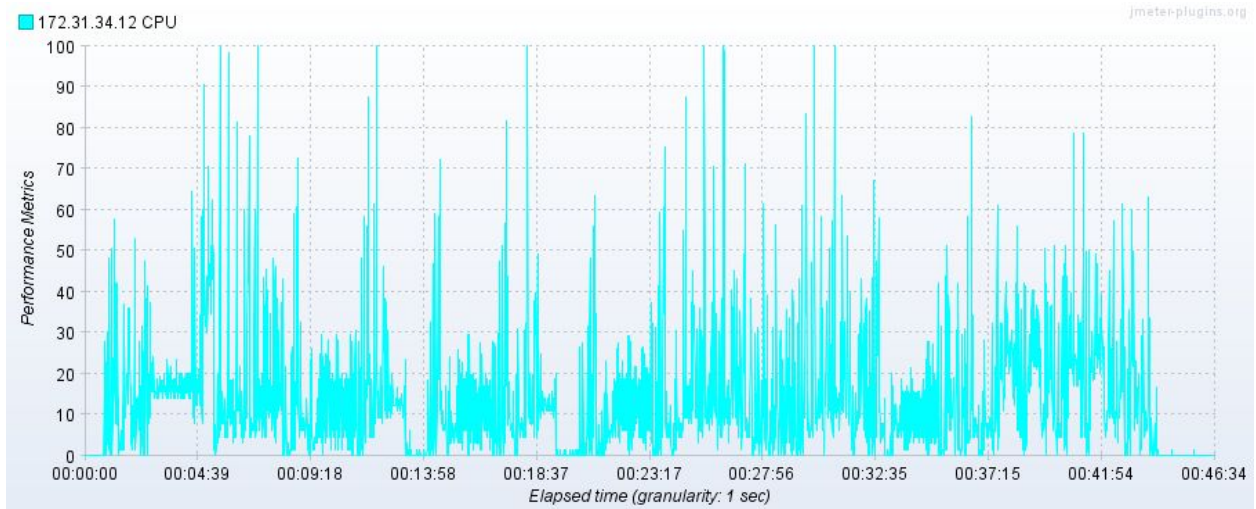


Swap - VFS

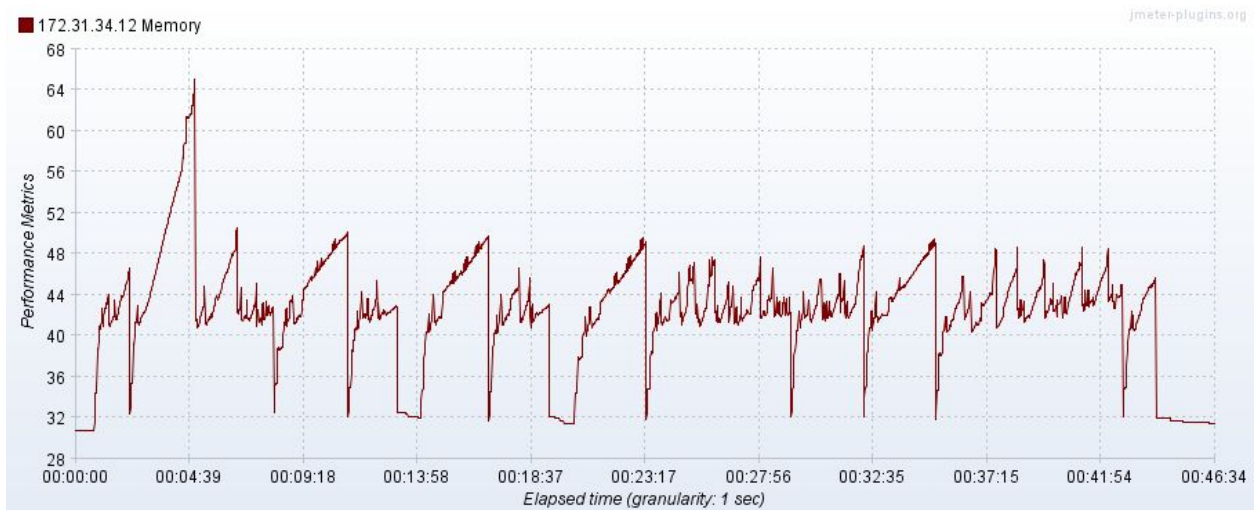


Node 1 Specific Metrics

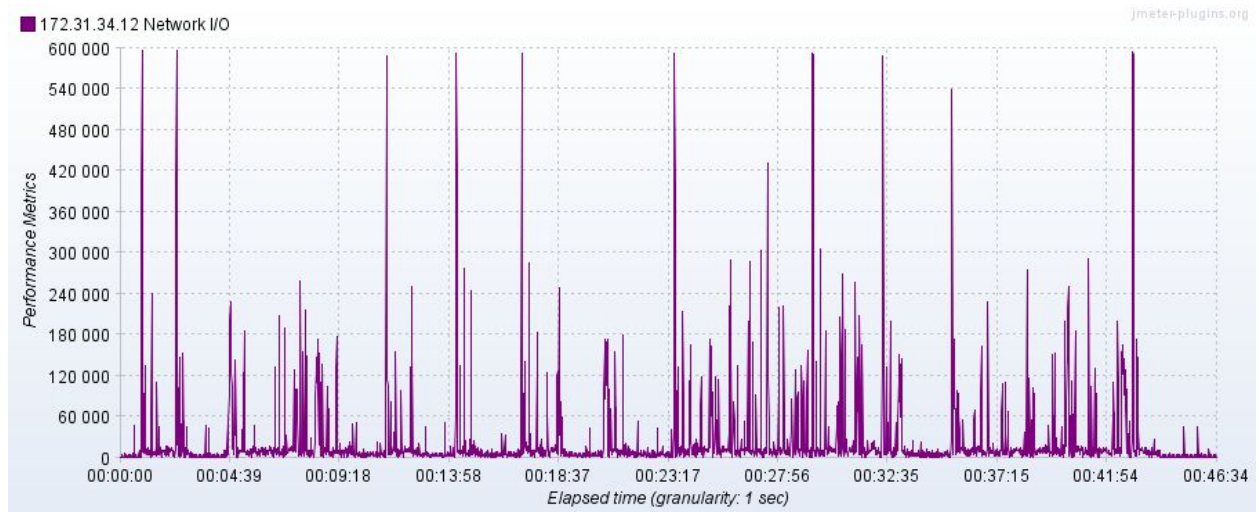
CPU - Single Core VM



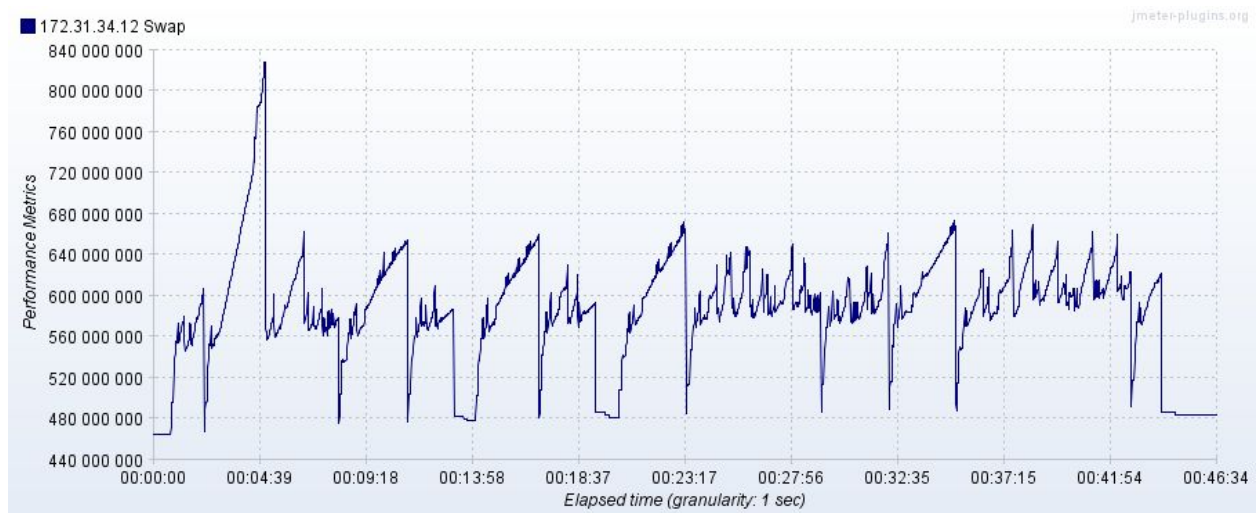
Memory - 1024 MB RAM



Network - Bridged VM NIC

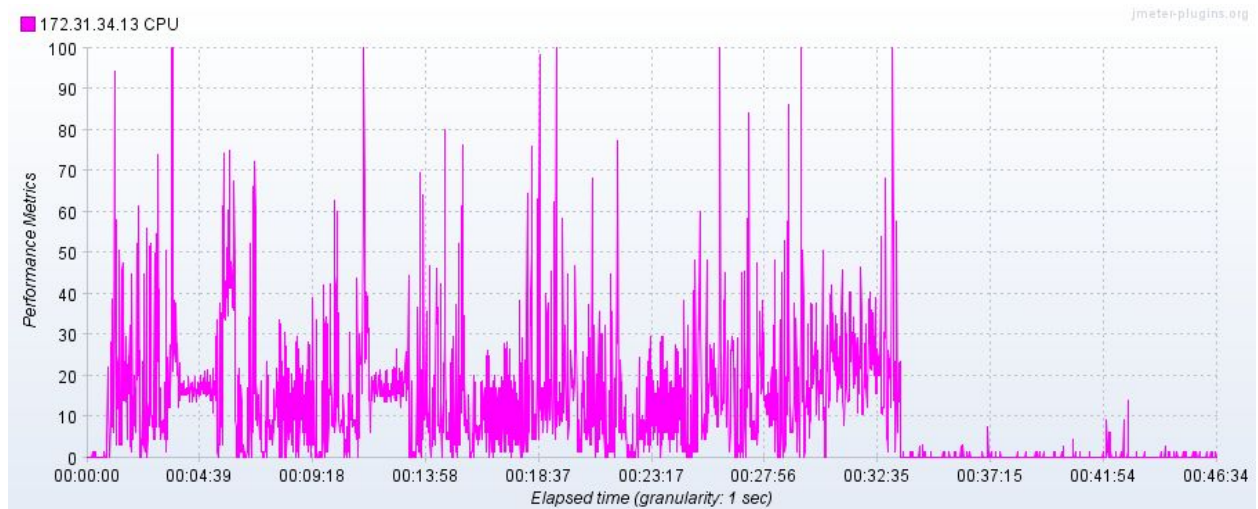


Swap -VFS

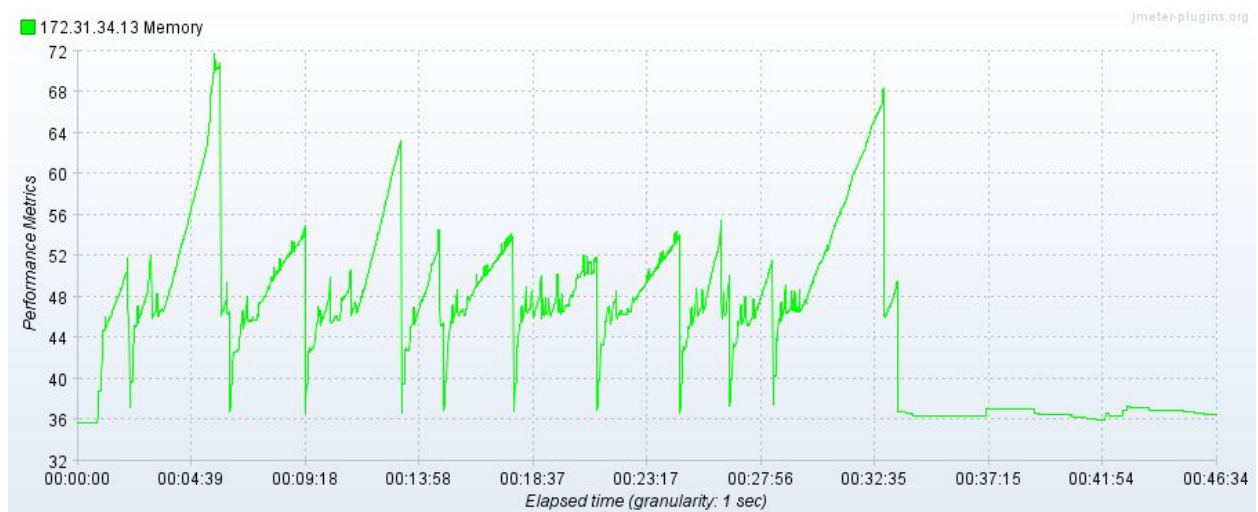


Node 2 Specific Metrics

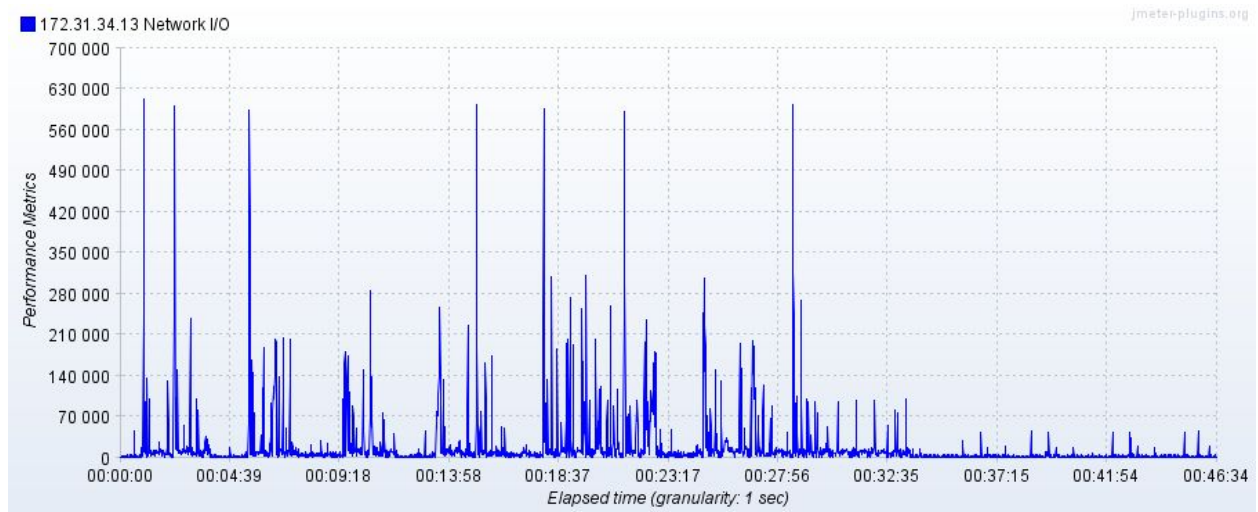
CPU - Single Core VM



Memory - 1024 MB RAM



Network - Bridged VM NIC



Swap - VFS

