



Sizing Servers Lab
www.sizingservers.be

An introduction to stress testing with vApus

SIZING SERVERS LAB
DIETER VANDROEMME
2018-02

Table of contents

| | |
|--|----|
| About Sizing Servers Lab..... | 1 |
| About the author..... | 2 |
| Prior knowledge | 3 |
| Introduction | 4 |
| How Sizing Servers Lab explains stress testing to our (future) customers | 5 |
| vApus, the stress test framework | 7 |
| Terminology | 9 |
| Installation | 10 |
| Building a basic stress test | 11 |
| Create a .vass | 11 |
| Add a connection..... | 12 |
| Add a scenario | 14 |
| Add a stress test | 16 |
| Capture the scenario..... | 17 |
| Test the ... test | 24 |
| Install monitors | 26 |
| Add monitors to the test..... | 26 |
| Executing the test and storing test / monitor results | 28 |
| Analyzing results | 30 |
| Next steps | 33 |

About Sizing Servers Lab

Sizing Servers Lab is an independent applied IT research lab at Howest University College, affiliated with the IT bachelor degree NMCT. The lab was founded in 2003 and is recognized by the Belgian government as an official research lab since 2007.

Our purpose is to offer best practices and directly applicable results to companies in Flanders, Belgium and internationally by researching cutting-edge server technologies. To accomplish this goal, we combine the three strongest factors of our lab:

A profound knowledge of hardware and software interfaces and optimization technics: virtualization, power management, etc.

Feedback from professionals in the field by publishing our results on popular IT websites (Anandtech.com, ZDNet, etc.)

A close cooperation with the engineers of market leaders as Intel, AMD, VMware, Facebook, etc.



NEW MEDIA &
COMMUNICATION
TECHNOLOGY

www.nmct.be



www.howest.be/en



About the author

The author of this document and the developer of stress testing tool vApus Dieter Vandroemme:

Intern at Sizing Servers Lab in 2007.

Graduated with great distinction from Howest University College, technical bachelor degree NMCT.

Sizing Servers Lab employee since 2007.

Developed stress testing tool vApus.

The author, Dieter Vandroemme, is a software developer mainly experienced with in C# (.NET Framework and Core), VB6, VB.NET, Java and a bit of PHP, JavaScript (jQuery) and C++.

He develops the stress testing tool, vApus, and monitors to measure hardware and software metrics, in the context of Sizing Servers Lab's research.

Dieter has 7 years of experience with (mostly web application) performance testing for and Recommending adjustments based on these test results.

Furthermore, he coordinates projects for Sizing Servers Lab and Howest University College with different stakeholders: students, lectors and companies.

He occasionally teaches various technical subjects at Howest or to companies, e.g. performance testing.

www.linkedin.com/in/dieterandroemme



Prior knowledge

To be able to execute a stress test, basic computer science knowledge is required: You need to have a basic understanding about CPUs, RAM, DISKs and NICs.

Furthermore, one has to understand how web sites work (TCP, HTTP, HTML, CSS, JS), have a fair level of understanding (relational) databases, Linux and Windows and have a notion about *virtualization*.

Some experience in (Object-Oriented) programming is necessary, or at least you need to be able to understand programming logic.

When stress testing a server application, we recommend that the owner of the application-to-test is available, a technical person if possible; and all other technical stakeholders as well, for instance sys admins.



Introduction

“

***Stress testing** (sometimes called **torture testing**) is a form of deliberately intense or thorough testing used to determine the stability of a given system or entity. It involves testing beyond normal operational capacity, often to a breaking point, in order to observe the results. (Wikipedia)*

Stress testing is one of many available forms to test a system. Just like load testing (testing the stability of a system with a defined load over a period of time) it falls under the performance testing category.

vApus (virtualized application unique stress testing) is a framework that allows us to performance test all applications communicating over a server socket, such as web apps, relational databases, custom APIs, etc.

The aim is to stay as close as possible to how real users accesses an application: Multiple users follow their own path through an application.

With vApus it is entirely possible to simulate hundreds of real users accessing an application all at once using only one test client.

The framework mainly focusses on testing web applications, as certain features are specifically implemented to facilitate this.

This overview deals with building and executing a test while monitoring and analyzing results.



How Sizing Servers Lab explains stress testing to our (future) customers

This is best explained using an excerpt of Sizing Servers Lab's web site (sizingservers.be/about-us/performance):

“

Tailor-Made Advice & Recommendations Based On Your (Web)Services' Performance

*The performance-testing goal of Sizing Servers Lab is to offer companies insight in **how much stress your current services can handle** & an **overview of the bottlenecks** of your current services. Based on these stress-testing results, Sizing Servers Lab offers you **tailor-made advice** on how to optimize your services for your expected amount of users. Sizing Servers Lab will even **recommend hardware & software** for your specific case. info@sizingservers.be*

Stress-Testing Your Current (Web)Services & Recommending Adjustments Based On Your Needs

How A Sizing Servers Lab Stress-Test Works

1. Defining Your Users' Actions

*Together with our client we **determine the different successive user actions** and **average think time** between each action that will occur in everyday reality. After defining these, we **simulate this reality** with a minimum amount of simulated users on the clients' hardware setup.*

[NedBox](http://www.nedbox.be) is a free website to practice Dutch through television clips and newspaper articles. This website was developed by University Leuven and industry partners, including Televic Education (<http://www.televic-education.com/en/>). Sizing Servers Lab stress tested this case.

Example of 1 successive user action on NedBox: a student surfs to the NedBox website, selects a clip and watches the clip. We simulated this reality by letting a minimum amount of simulated students perform the successive user action on the clients hardware setup.



2. Baseline Stress Test Of Your Service

Next, we **gradually increase the number of user actions per second and measure the effect of each increase on response times and throughput, while monitoring and correlating this with the resource-usage** (CPU, disk, memory and network usage). At Sizing Servers Lab, we always monitor this with our in-house developed tool vApus. Because vApus was designed as a research tool, it's very customizable and applicable on a multitude of desired platforms and scenarios. The result of this baseline stress test is a **clear bottleneck overview** of your service.

In the NedBox case, we gradually increased the number of times a simulated student surfed to the NedBox website per second, the number of times the student selected a clip per second and the number of times a student watched a clip per second. We measured the effect of these increases on response times, throughput and resource-usage.

3. Advice & Recommendations

Based on the bottleneck overview from the baseline stress test, we're able to offer you **advice on how to optimize your IT or web service** and which **hardware & software** will work out best for your case. We test these recommended hardware-software combinations with new stress tests and offer a comparative overview of all suggested combinations.

Looking at the Nedbox baseline stress test results, we pinpointed the main NedBox bottlenecks. For this specific case we advised some modifications at infrastructure and application level. In the end report we described how to convert these advices into reality.



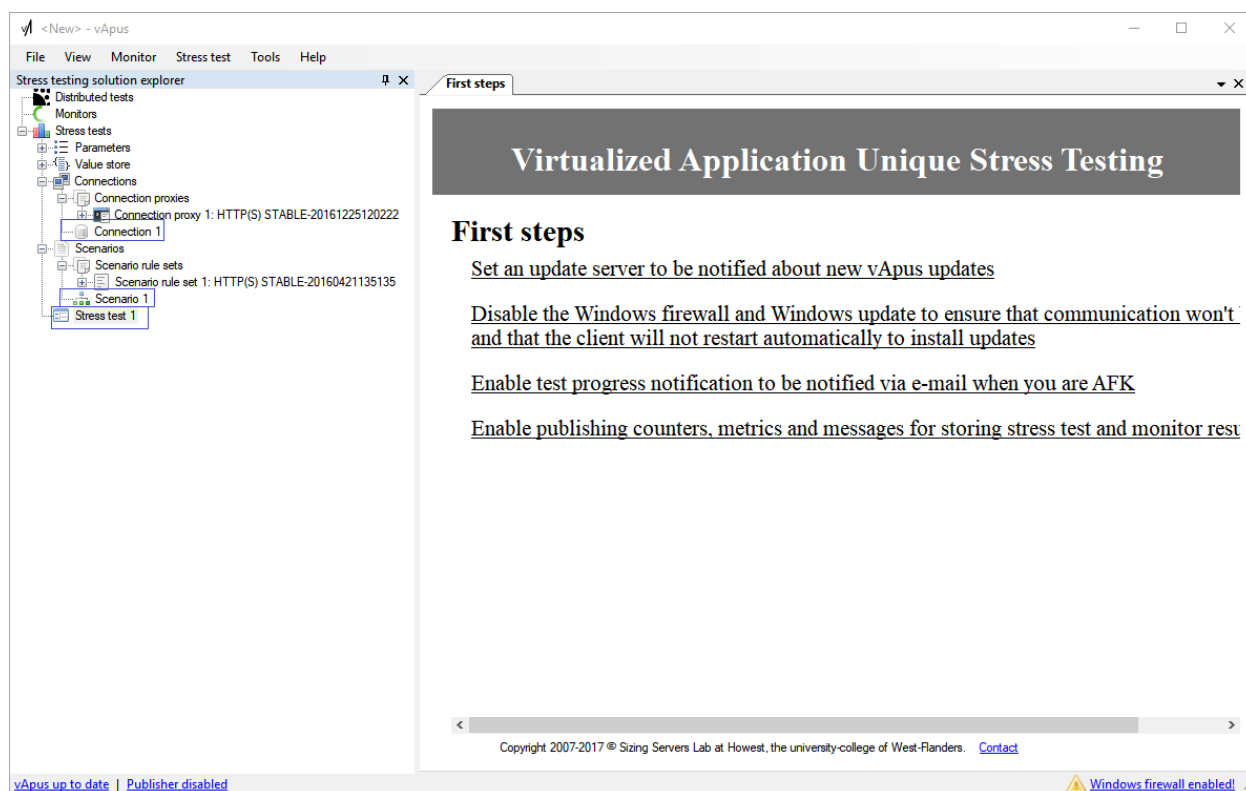
vApus, the stress test framework

vApus allows us to test a wide variety of server applications, from databases to web applications to exotic APIs.

Basically, if the application communicates via a server socket, vApus can test it. vApus is in fact a framework, meaning that it is very customizable.

vApus consists out of a few building blocks that can be chosen and adapted freely to one's needs. In order to execute a stress test you need at least the following 3 parts:

- A **connection** for communicating to the server application. Measuring response times and some error handling happen here as well.
- A **scenario** holds all requests to be sent to the application by a simulated user. Multiple scenarios can be used within the same test to simulate different types of users. For instance, admins and non-admins.
- A **stress test** bundles a connection and scenarios + configuration.



It is possible and advised that while stress testing, the server-infrastructure is **monitored** using vApus. By doing this, bottlenecks can be detected and resolved. For instance, it takes a long time for a user to see the home page of a website. We see that the database server's CPU is completely saturated. Maybe a SQL query is too heavy, or maybe we just do not have enough CPU power.



Say one test client does not suffice to stress the server(s): add another. A stress test's workload can be distributed over multiple test clients within a **distributed test**. A master vAplus instance orchestrates slave instances doing the actual work.

The same distributed test allows us to test different server applications as well. This to see how the underlying server infrastructure reacts while all hosted applications are accessed at the same time.

Requests can be **parameterized** with all sorts of values like user-unique credentials. Filling in parameters does not happen while a test is running so the test client does not get stressed.

The **value store** serves as *stress test time parameter*. One can store and retrieve data like HTTP cookies and other values retrieved from responses to be used in further requests.



Terminology

Before we begin to build our own stress test, we first have to go over some Sizing Servers Lab terminology.

vApus is an in-house developed framework to stress test and monitor different applications. The power of vApus lies in its flexibility to adjust the framework to the required platform.

A **User action** is a sequence of **requests** sent by vApus, initiated by a **simulated / virtual user**.

An action can be defined as for instance, “all needed requests to Login a web application”.

An action is complete if all requests received a response. Next, the virtual user will proceed with the next action. In this test set-up we can define an action at best with a **page or a “click”** on a website.

A **Scenario** is a collection of successive user actions. A stress test can contain one or more of these. Each simulated user executes all user actions in its appointed scenario.

Concurrency or concurrent users determines the number of user actions executed per second. While testing a website the concurrency can be best interpreted as “**number of pages per second**”. In the Sizing Servers Lab tests we linearly increase the concurrency. By doing this we gradually increase the stress on the server(s) (ramp-up testing).

The number of **runs** defines the number of times a test with a certain concurrency is (re)done, this to smooth out average concurrency results.

Response time translates itself best as “user experience”. The response time determines the average time it took for a simulated user to receive a full page. This is measured from the moment the request is sent until the last byte is received. **A lower response time is always better.**

Throughput gives a general idea of the server(s) performance, but doesn’t translate itself directly as user experience. The throughput determines the quantity of responses per second a server can process and send. **A higher throughput is always better.**

Think time can be defined as the average time between actions: the average time a user stays on the page before clicking through to next page. Usually we assume an average think time of 1 second (900-1100ms), consequently we speak about “pages per second”.

Ramp-up time is used to gradually increase the number of users over a set period of time (e.g. 20 seconds) until the concurrency we want is reached. This way we spread the requests to the server application to have a more realistic test.



Installation

De vApus suite consists out of vApus itself + a few hardware monitor agents: like vApus-wmi-net to monitor Windows, vApus-proc to monitor Linux, etc.

vApus has to be installed on Windows 10 or Windows Server 2012 R2 or newer in order to make sure response times get measured correctly (see *...Running vApus virtualised.pdf*).

The vApus client is preferably a dedicated (virtual) machine that has sufficient CPU power (≥ 4 x modern CPU cores), enough RAM (≥ 8 GB) and a *wired* network connection (\geq gigabit). For this introduction, a simple laptop suffices.

vApus requires that \geq .NET 4.7 is installed (www.microsoft.com/en-us/download/details.aspx?id=55170).

vApus-proc requires \geq Java 6, vApus-wmi-net requires \geq .Net 4.5. Other monitor agent specific prerequisites can be found in their readme's.

vApus stores stress test- and monitor results in a MySQL database. A reachable machine needs to have \geq MySQL GA 5.7.

Install Fiddler (www.telerik.com/download/fiddler) for debugging web application scenarios.

Use Excel 2016 to review exported results. Copy *personal.xlsm* from vApus' folder to Office's *XLSTART* folder (most likely *C:\Program Files (x86)\Microsoft Office\Office16\XLSTART*).

This .xlsm contains macros to generate charts from monitor results.



Building a basic stress test

As previously said, vApus allows us to test a broad range of server applications.

For this introduction we will focus us on testing web applications: vApus is used mostly to test this type of applications and is adapted towards this.

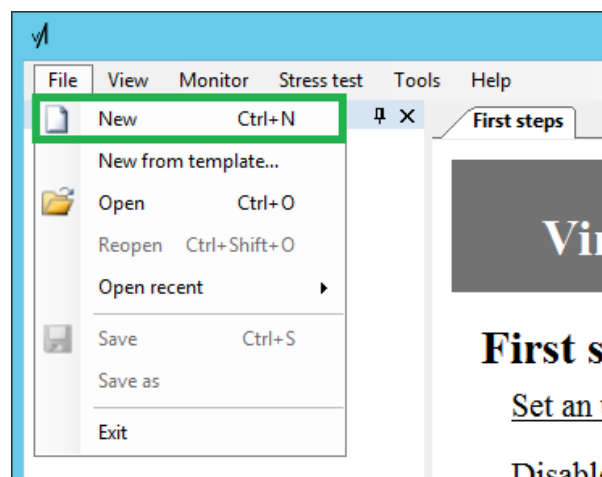
In the following tutorial we will build a stress test for your favorite website: howest.be.

Create a .vass

After installing vApus you open it by double-clicking the vApus.exe under c:/Program Files/Sizing Servers/vApus v2.

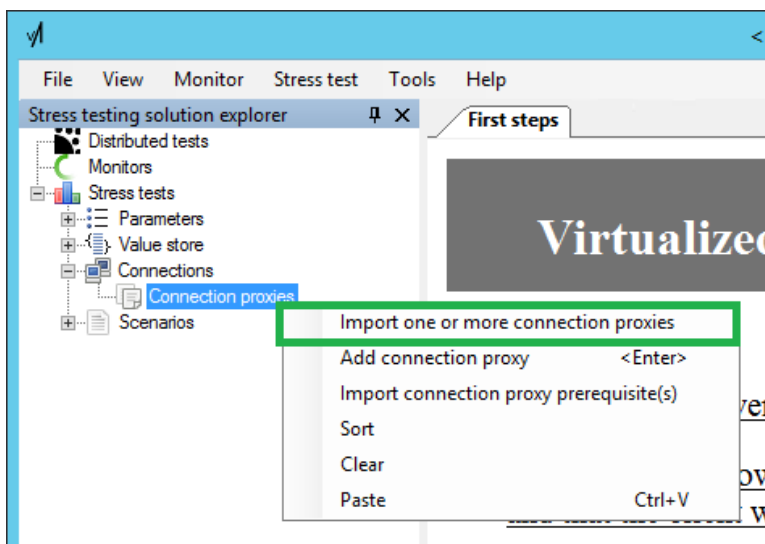
Create a new vApus stress testing solution (.vass) by clicking File > New in the main menu.

Save this .vass immediately; choose a good filename for it.

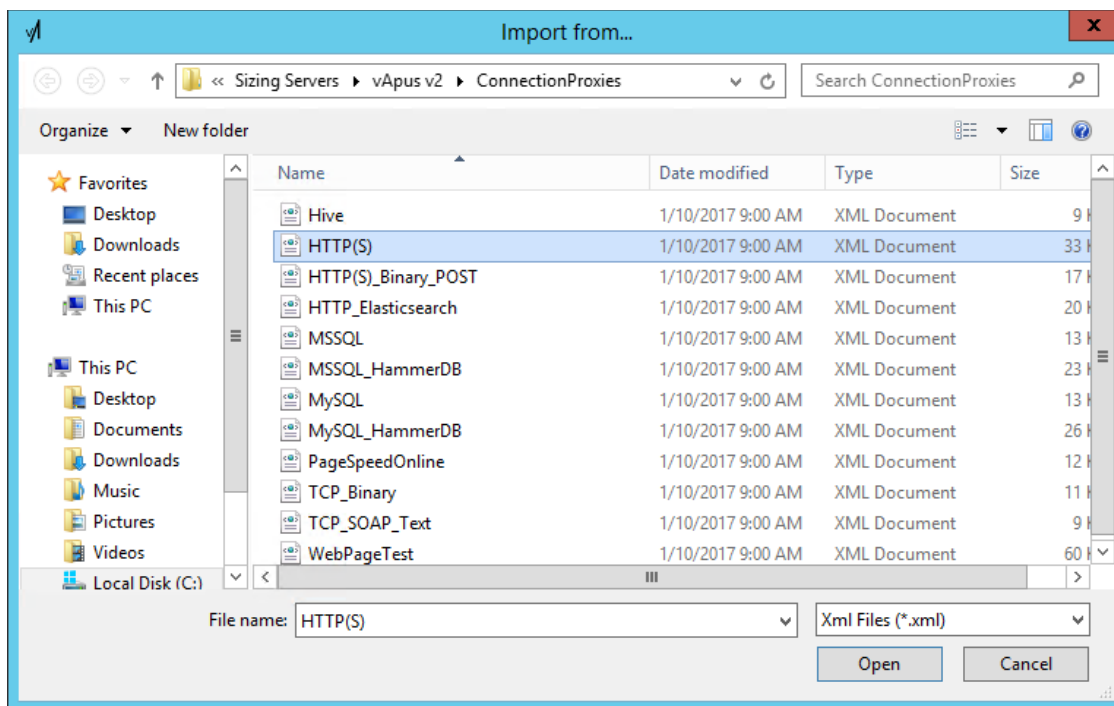


Add a connection

Open the *Connections* node, right click *Import one or more connection proxies*.



Import *HTTP(S).xml*. Herein lies the power of vApus: a connection proxy contains the code that provides the connection and the communication to the application + time measurements of requests.



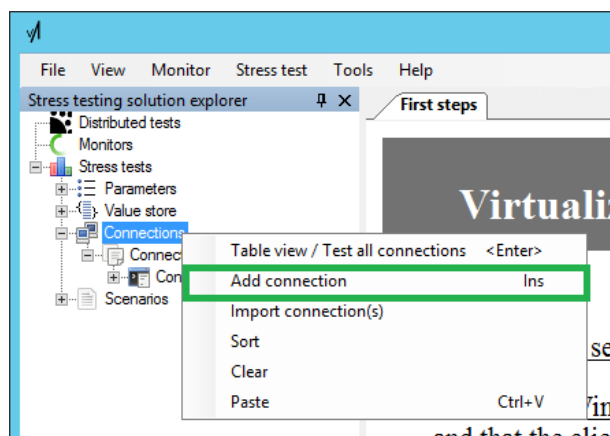
A connection proxy code is highly customizable. It allows us to do *anything*: it takes different parts of requests in a scenario to build and send for instance a HTTP web request. (See *Add a scenario*).

In almost all cases, you need to edit the *Connection Proxy Code* (.NET C#) so client side sessions are managed correctly, amongst others.

This requires some programming skills and a fair bit of reverse-engineering experience. Sizing Servers Lab uses Fiddler most of the time to deduce how a web application works by looking at requests and responses. All this is completely out of scope of this introduction.

Since *Howest.be* test does not require login ins, handling web store baskets, etc. the test will work out without customizing the connection proxy.

Right-click *Connections*, click *Add connection*, give the new connection a name and press the *ENTER* key.

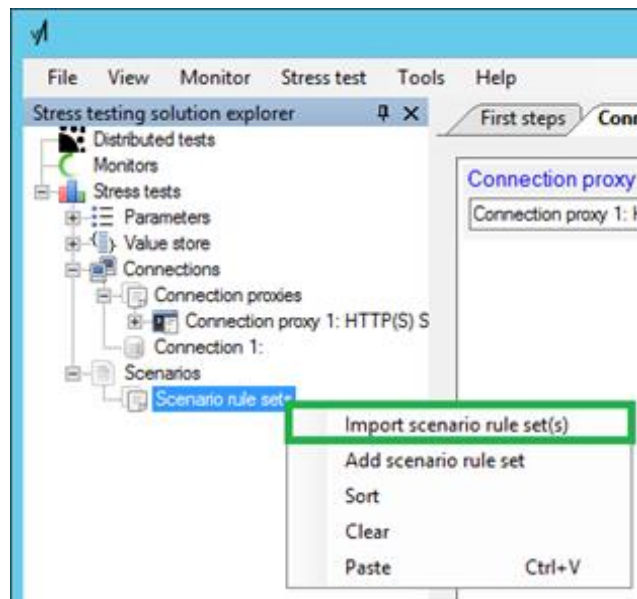


Double click on your new connection, enter the hostname **www.howest.be** the field *Host*, fill in **https** in Protocol and **443** in Port. Click on *Test connection* and OK.

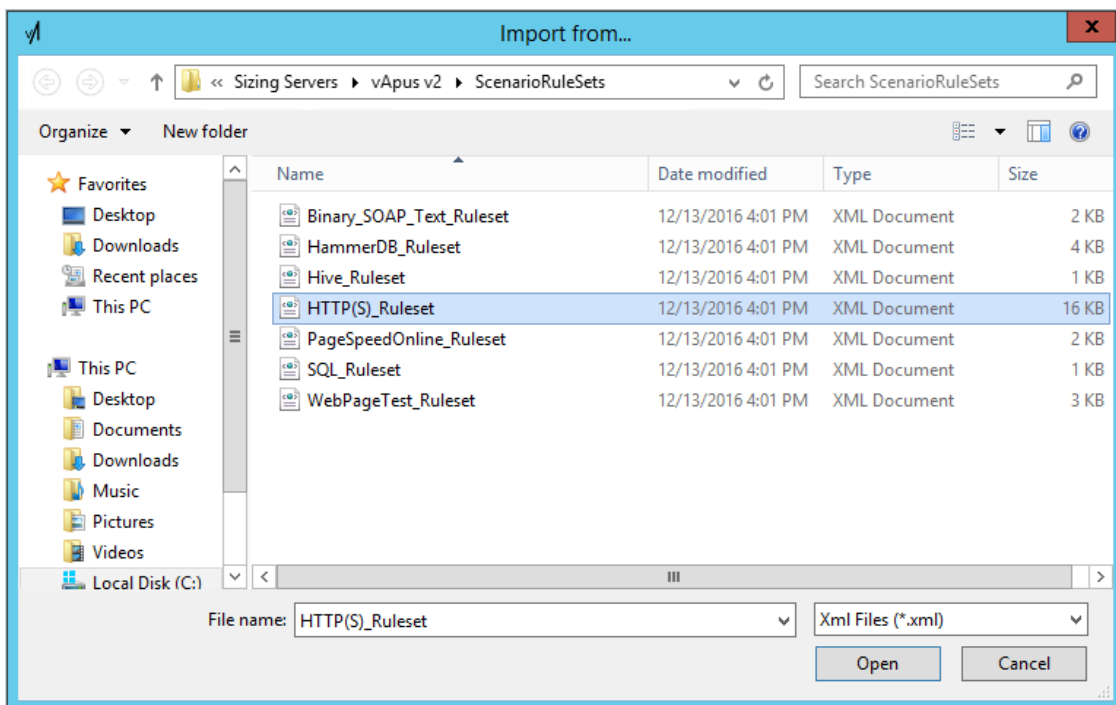
 A screenshot of the vApus connection configuration dialog box. It has fields for 'Host', 'Protocol', and 'Port'. The 'Host' field is empty, 'Protocol' is set to 'http', and 'Port' is set to '80'. Below the fields are buttons for 'Test connection', 'Trace Route', '? hops', '? Roundtrip time', and 'Idle'. A text box above the 'Host' field provides examples and instructions: 'Provide the hostname, an IPv4 address, or an IPv6 address enclosed in "[]". Examples: www.coolomain.com, 192.168.2.3 or [2001:db8:85a3:8d3:1319:8a2e:370:7348]. This value is overiden with the log's value when #MULTICONNECTION is used.'

Add a scenario

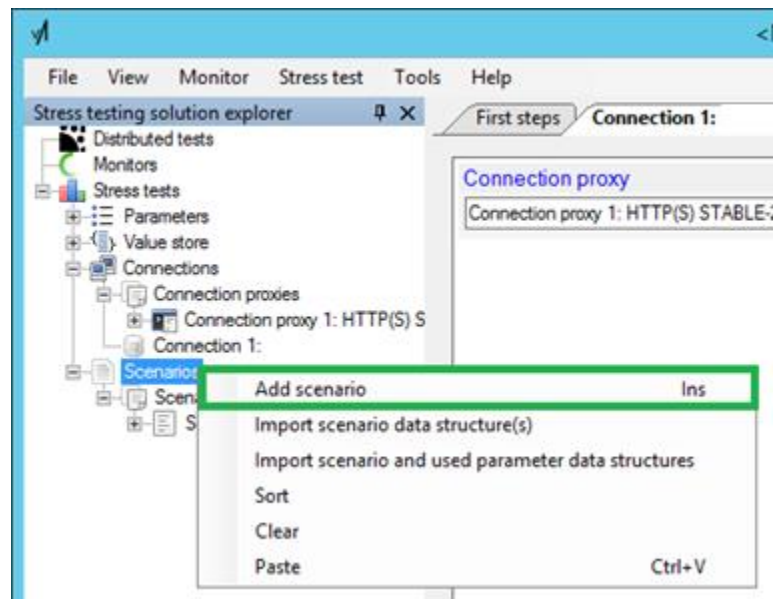
Open up Scenarios, right-click *Scenario rule sets*, click *Import scenario rule set(s)*.



Import *HTTP(S)_Ruleset.xml*: this ruleset describes the different parts of a request, e.g. the *URL*, *POST* or *GET*, ...

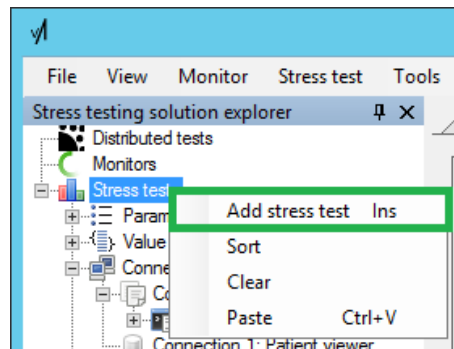


Right click *Scenarios*, click *Add scenario*, give the new scenario a name and press *ENTER*.



Add a stress test

Right-click *Stress tests*, Click *Add stress test* and give the new stress test a name.



Double-click the stress test and review the configuration settings. At this moment we leave this as it is.

 A screenshot of the 'Configure' tab for a stress test. The tab has three sub-tabs: 'Configure', 'Stress test', and 'Detailed results'. The 'Configure' sub-tab is active. It contains several sections:

- Description:** A text area with a placeholder 'The description for this test that will be stored in the results database for easy finding.'
- Tags:** A text input field.
- Connection:** A dropdown menu showing 'Connection 1:'.
- Scenarios:** A text input field showing '[Scenario 1: : 1]'.
- Monitors:** A text input field.
- Concurrencies:** A text input field showing '5, 5, 10, 25, 50, 100'.
- Runs:** A text input field showing '1'.
- Shuffle:** A text input field showing 'True'.
- Action distribution:** A text input field showing 'False'.
- Maximum number of user actions:** A text input field showing '0'.
- Monitor before:** A text input field showing '0'.
- Monitor after:** A text input field showing '0'.
- Show/Hide advanced settings:** A link.

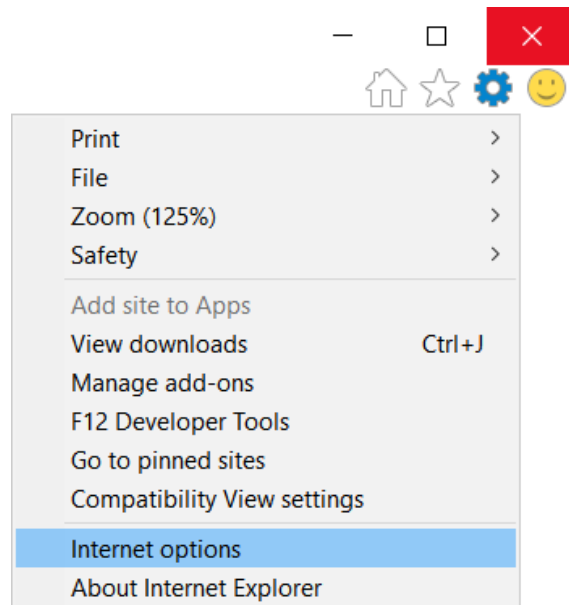
Capture the scenario

Internet explorer is used for capturing the scenarios because:

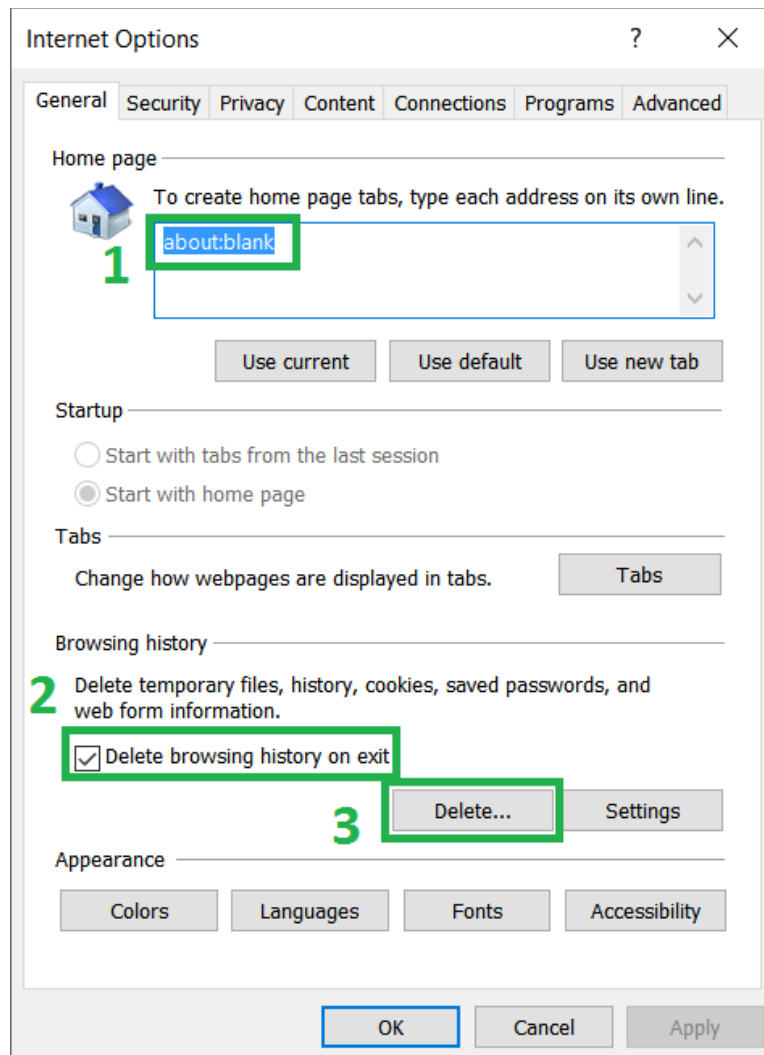
- It is a browser you do not use for anything, so it is clean (no plugins)
- Internet explorer aligns very well with .NET. Both are from Microsoft. vApus is built using .NET.

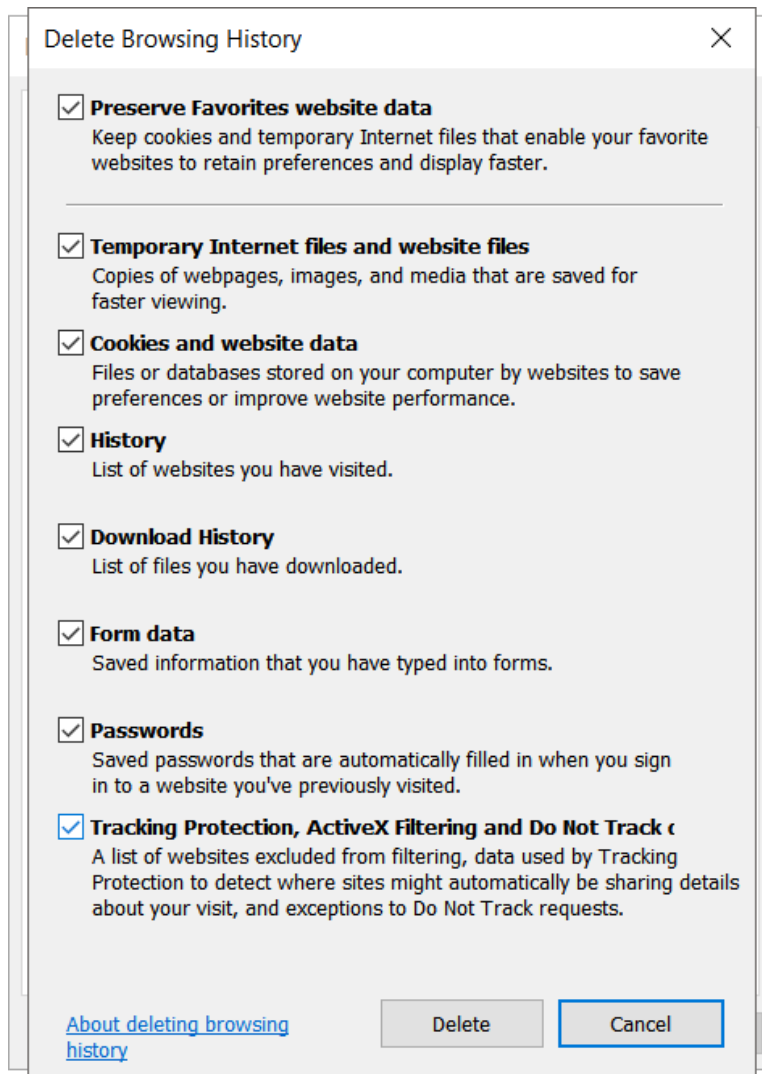
We first need to make some changes, since we do not want too many useless requests in our captured scenario. This will come clear later on.

Start **Internet Explorer**; open up *Internet Options*.



Fill in the options as follows:

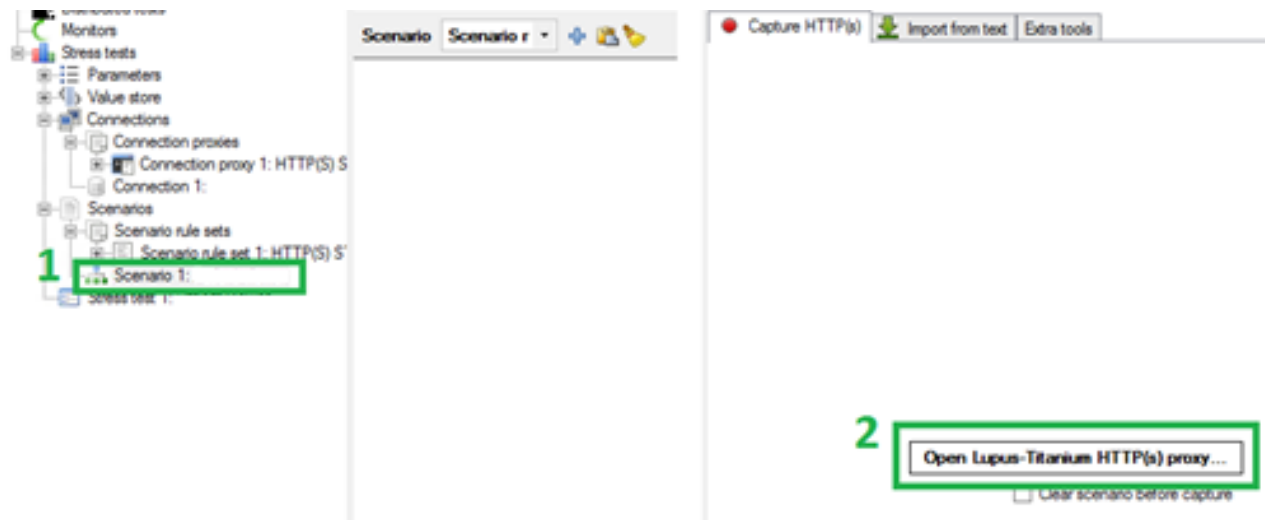




Close all browsers and anything else that generates HTTP traffic.

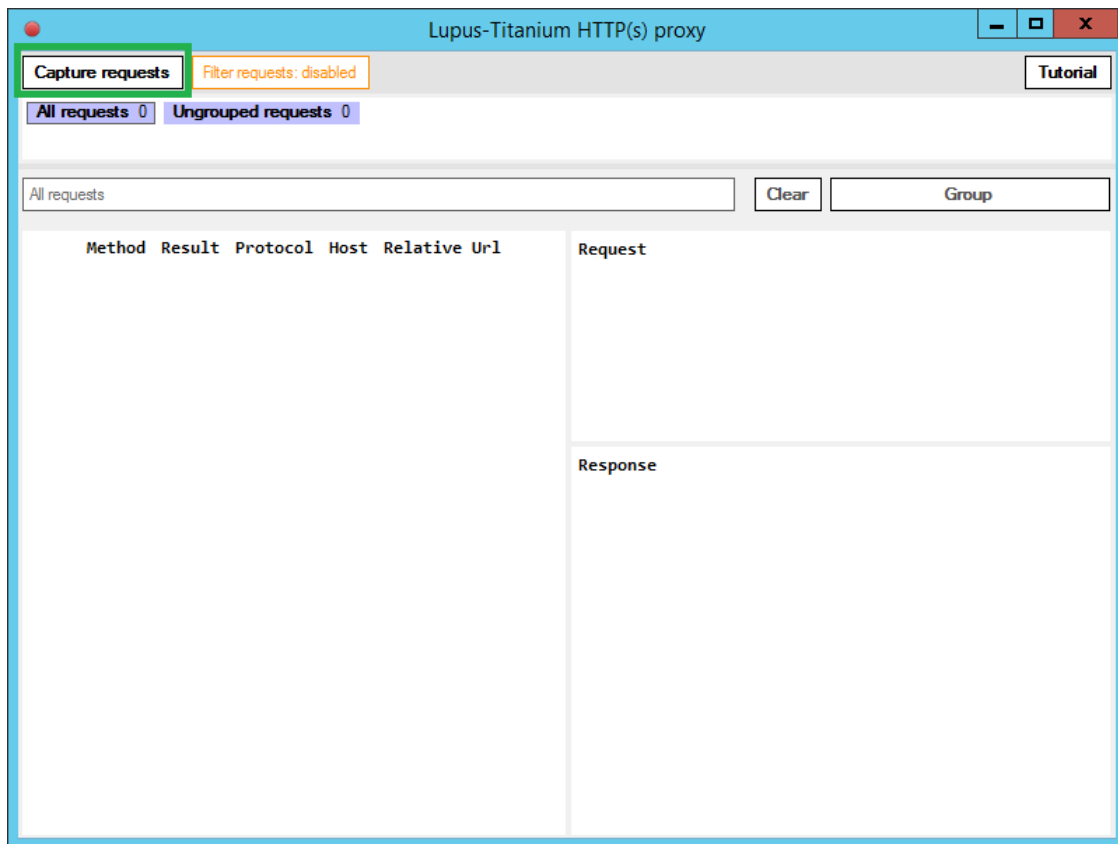
Double-click your scenario in vApus,

Click *Open Lupus-Titanium HTTP(s) proxy...*; *Lupus-Titanium* is a HTTP proxy, meaning all HTTP traffic is rerouted through *Lupus-Titanium*. This allows us to capture HTTP traffic.



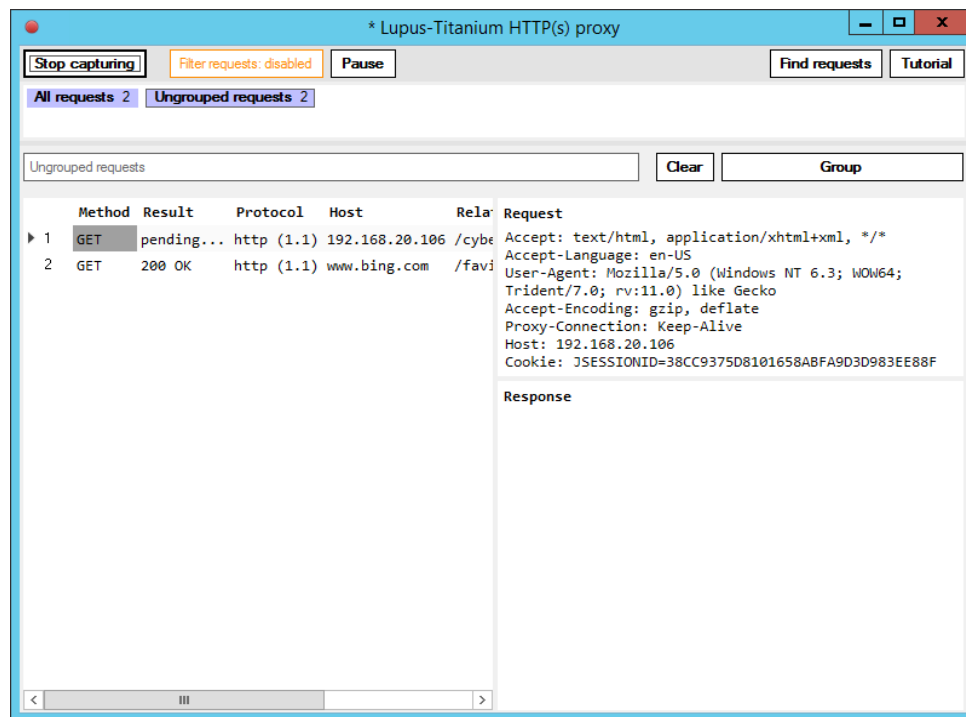
Click *Capture requests*.

Install the SSL certificate when asked.



Start Internet Explorer.

Surf to <https://www.howest.be/en>.



Wait until the landing page of Howest.be is fully loaded.

Return to *Lupus-Titanium*, click *Group* to create a new user action. Give it the name *Landing page*.

Go back to Internet Explorer and click *Info days 2018*.



Wait until the page is fully loaded and group the requests in a user action. Give it the name *Info days*.

Close *Lupus-Titanium* and close *Internet Explorer*.

Go back to vApus. Review the contents of the 2 user actions in the scenario. If there is a third user action (auto-group ungrouped requests), hover over it and click the *delete icon*.



All requests at the right side of the window are shown in a grid. Notice that the different parts of the requests (as defined in the Scenario Rule Set) are split over different columns.

Requests [61]

Structured Plain text

| | *Request method | *Protocol | *Host | *Port | *Relative Url |
|-----|-----------------|-----------|---------------------------|-------|------------------|
| ▶ 1 | GET | https | 5-edge-chat.messenger.com | 443 | /pull |
| 2 | OPTIONS | https | 0.docs.google.com | 443 | /spreadsheets/u/ |
| 3 | GET | https | 0.docs.google.com | 443 | /spreadsheets/u/ |
| 4 | OPTIONS | https | 0.docs.google.com | 443 | /document/d/1NXT |
| 5 | GET | https | 0.docs.google.com | 443 | /document/d/1NXT |
| 6 | POST | https | play.google.com | 443 | /log |
| 7 | GET | https | www.howest.be | 443 | /en |
| 8 | GET | https | www.howest.be | 443 | /sites/default/f |

These parts are used in a connection proxy (see Add a connection) to correctly construct HTTP web request to send to the application-to-test, in this case *Howest.be*.

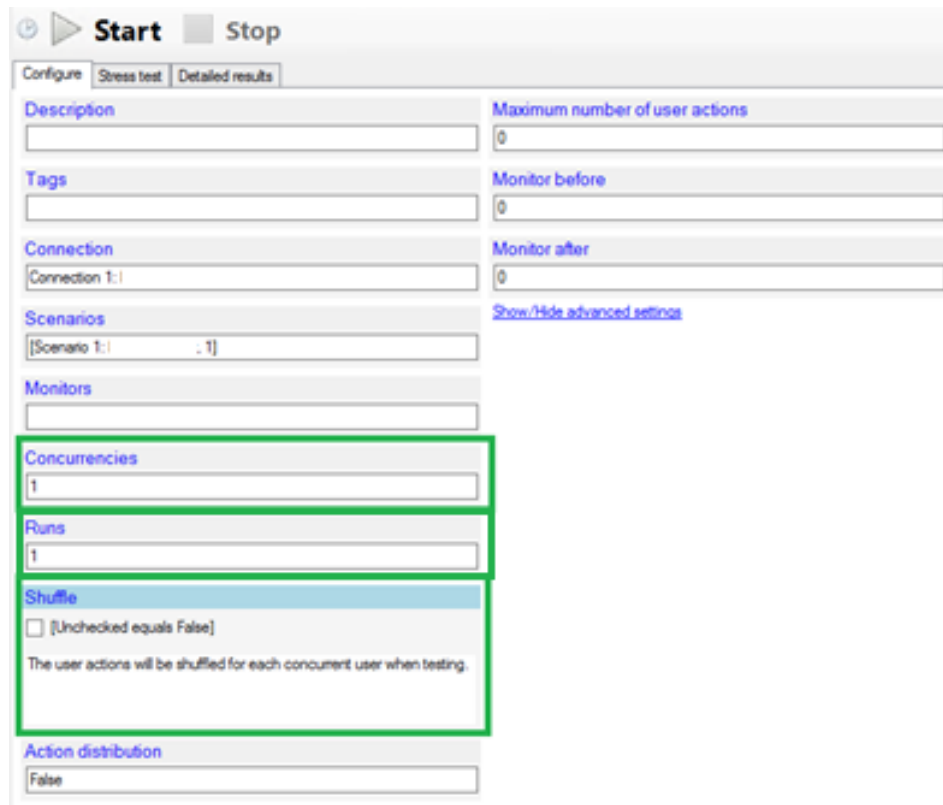
This *request grid* has more or less the same functionality as Excel. Delete all rows that do not have www.howest.be for a host in both user actions.

We do this because external calls are not useful if we want to measure the performance for a specific application.

Test the ... test

Double-click the stress test, click on *Concurrencies* > *Edit ...*, replace the text with a **1**, and click *OK*.

Set *Runs* to **1** and *Shuffle* to **false**.



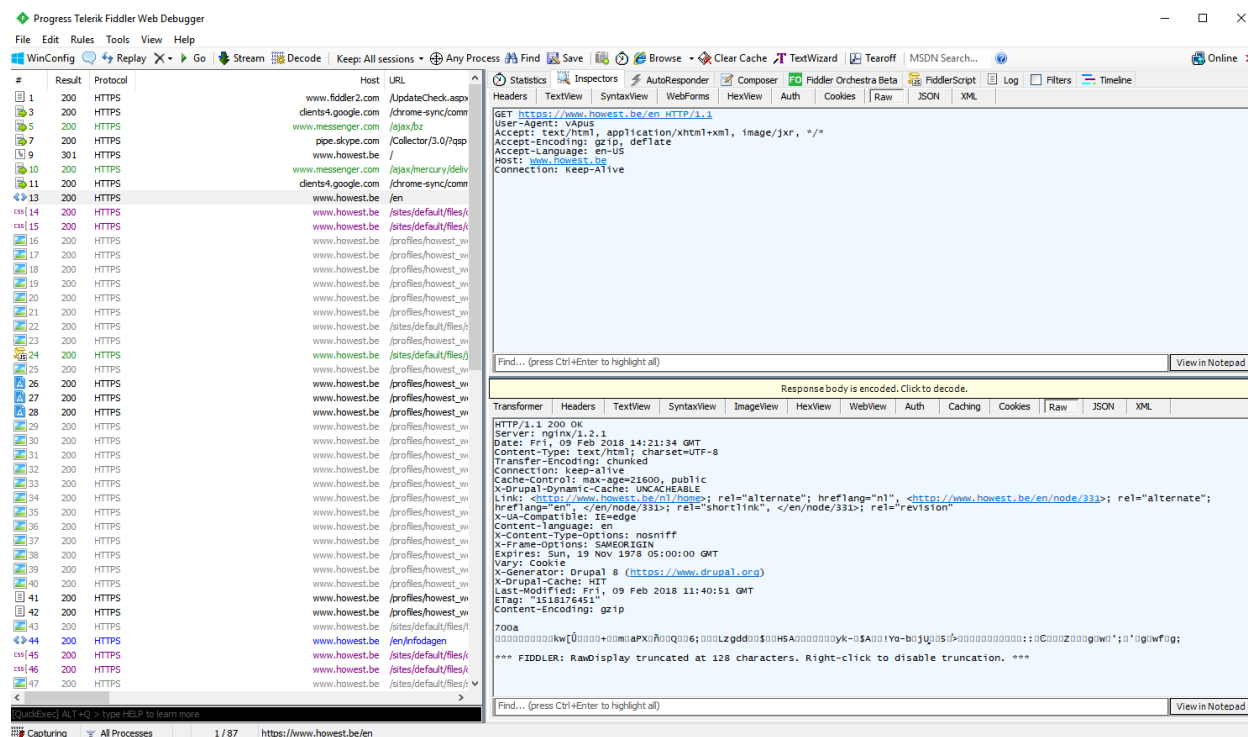
The screenshot shows the Fiddler stress test configuration window. The 'Start' button is highlighted with a green box. The 'Concurrencies' field is set to 1, 'Runs' is set to 1, and 'Shuffle' is unchecked. The 'Action distribution' field is set to False. The 'Start' button is highlighted with a green box.

Start Fiddler, a proxy you installed earlier, so we can follow the test progress. You could use *Lupus-Titanium* for this as well. *Fiddler* has a few handy features *Lupus-Titanium* lacks.

Start the stress test, ignore the *blue popup*, just click *OK*.



Fiddler shows you all web requests + responses that pass through your network card. It is easy in Fiddler to do a text search, for instance a value from a response that you need to fill in, in a following request using a customized connection proxy (out of the scope of this introduction).



If everything goes well you should have no errors.

Fast results

Drill down to **Concurrencies** Test started at 2018-02-09 15:21:30 . ran 7 s **and finished at 2018-02-09 15:21:37.883** ☒ Readable **Save displayed results...**

| Started at | Time left | Measured time | Concurrency | Requests processed | Errors | Throughput (responses / s) | User actions / s | Avg. response time (ms) | Max. response time (ms) | Avg. delay (ms) |
|-------------------------|-----------|---------------|-------------|--------------------|--------|----------------------------|------------------|-------------------------|-------------------------|-----------------|
| 2018-02-09 15:21:30.196 | -- | 8 s | 1 | 37 / 37 | 0 | 10.7 | 0.58 | 36 | 160 | 57 |

Now we are ready to add monitors to the test.



Install monitors

When stress testing you should monitor hardware counters (CPU, RAM, DISK, NIC) for each machine in the server infrastructure.

vApus is able to link monitor metrics (like increased CPU usage) to the number of concurrent users that put load on the server application.

The two most important monitor agents are **vApus-wmi**, to monitor Windows, and **vApus-proc**, to monitor Linux.

A monitor agent has to be installed and executed on the machines-to-monitor. Here are the steps to do this:

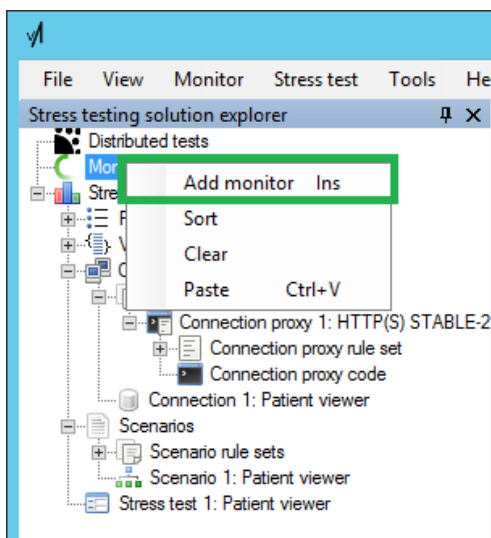
- Put the agent.zip file on the server instance (e.g. a VM) and unzip it.
- Make sure that \geq Java 6 is installed on Linux machines, \geq .Net 4.5 on Windows machines.
- Execute the vApus-proc agent using Terminal (`java -jar vApus-proc.jar`) / Execute `vApus-wmi-net.exe`.

For the sake of convenience, we will run **vApus-wmi-net.exe** on the same computer that runs vApus and pretend that it is running server-side.

Add monitors to the test

Go back to vApus.

In vApus, on the left: right-click on Monitors, click on Add monitor and give it a name.



Double-click on the monitor, choose *WMI Agent* as monitor source, enter the IP address of the same machine that runs vApus.

The screenshot shows a software interface for configuring a monitor. The title bar includes tabs for 'First steps', 'Scenario 1: Patient viewer', 'Connection Proxy Code (HTTP(S...))', and 'Monitor 1: WEB2 System'. Below the tabs are 'Start' and 'Stop' buttons. The 'Monitor' tab is selected, displaying a 'Monitor source' dropdown menu with 'WMI Agent' chosen. To the right, under 'Monitor source parameters', there are two input fields: 'Host Name or IP address' containing '192.168.X.X' and 'Port' containing '5556'. At the bottom of the window are two buttons: 'Get counters' and 'Configuration'.

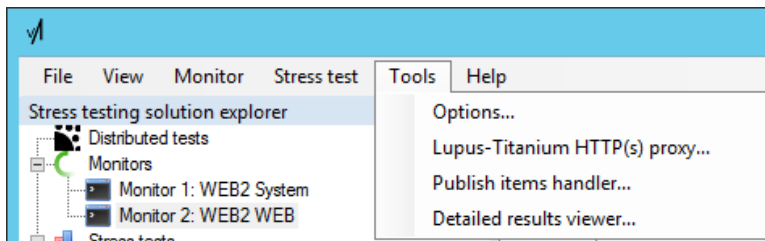
Click on *Get counters* and click on *Use defaults*; also, click on *Configuration* and save the contents of the window that appears somewhere.

Open the stress test and add the new monitor to the configuration.

Executing the test and storing test / monitor results

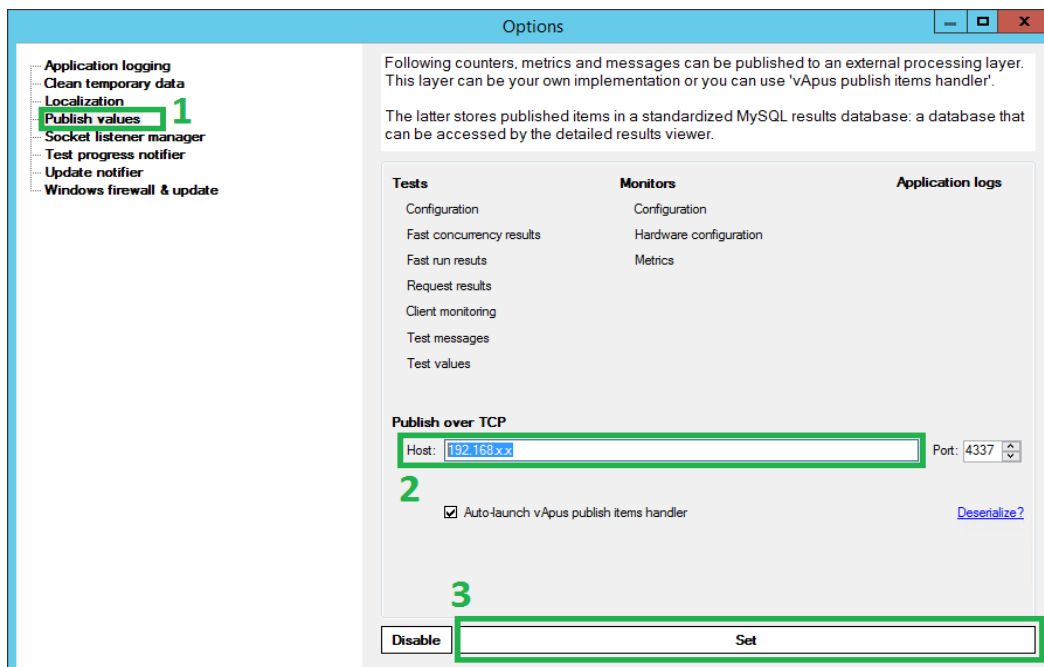
First and foremost, we must ensure that our stress test results can be stored somewhere. So make sure that a MySQL server is available for this.

In vApus: click on *Tools> Options...* and choose *Publish values*.



vApus makes all useful events available over a TCP socket; You can write a handler yourself to capture it for example a dashboard on a web page.

Enter the IP of the test client as Host, the *Port* must be **4337**, uncheck *Autolaunch vApus publish items handler* and click *Set*.



In the window that appears, fill in how the handler should connect to MySQL (port 3306).

After filling in everything disappears the window to the background.

Close the *Options window*.

Now the stress test must be correctly configured and started:

Double click on the stress test.

Fill in the description and tags so that the results can easily be found later.

Select the available monitors.

We choose *Concurrencies: 5, 10, 15; Runs: 2; Monitor before and after: 1* minute.

Click Show / Hide advanced settings, set *Initial minimum delay* to **0** and *Initial maximum delay* to **20000** ms: this will ensure that the application will not be stressed every second with the same requests, but that these will be spread out.

| Configure Stress test Detailed results | | |
|--|--|---|
| Description | Minimum delay | Simplified fast results |
| Fiddler | 900 | False |
| Tags | Maximum delay | Show/hide advanced settings |
| deter, load test | 1100 | |
| Connection | Shuffle | |
| Connection 1: Patient viewer | False | |
| Scenarios | Action distribution | |
| [Scenario 1: Patient viewer, 1] | False | |
| Monitors | Maximum number of user actions | |
| Monitor 1: WEB2 System, Monitor 2: WEB2 WEB | 0 | |
| Concurrencies | Monitor before | |
| 5, 10, 15 | 0 | |
| Runs | Monitor after | |
| 2 | 0 | |
| Initial minimum delay | [Browser] Use parallel execution of requests | |
| 0 | False | |
| Initial maximum delay | [Browser] Maximum persistent connections | |
| 20000 | 0 | |
| <small>The maximum delay in milliseconds before the execution of the first requests per user. This is not used in result calculations, but rather to spread the requests at the start of the test.</small> | [Browser] Persistent connections per hostname | |
| | 6 | |

Make sure that Fiddler is no longer running and start the test.

Wait until the test is finished, wait another minute until the monitor stops.

Small side note: if there are requests that are added, disappeared or changed in the application itself, the test must be built again.

You can also run this test via CMD eg **vApus.exe test.vass -s 1**. (vApus.exe -h to see all options)

Analyzing results

When executing a case for a Sizing Servers Lab client, a detailed report is written containing an extensive analysis of test results. For internal cases Sizing Servers Lab uses the same results, but results are not always stored in the form of such a report.

In *Sizing_Servers_-_Example_report.pdf* you see the same terminology as is used here. Take your time to read this example report.

The most important charts are *Response time vs Throughput*, *Errors vs Throughput* and, not in the case of many user actions / not in this report, *Top 5 User Actions*.

Notice that the report does not contain *vApus agent monitor metrics*. The client does not always agree to installing own software on their servers. Sizing Servers Lab adapts.

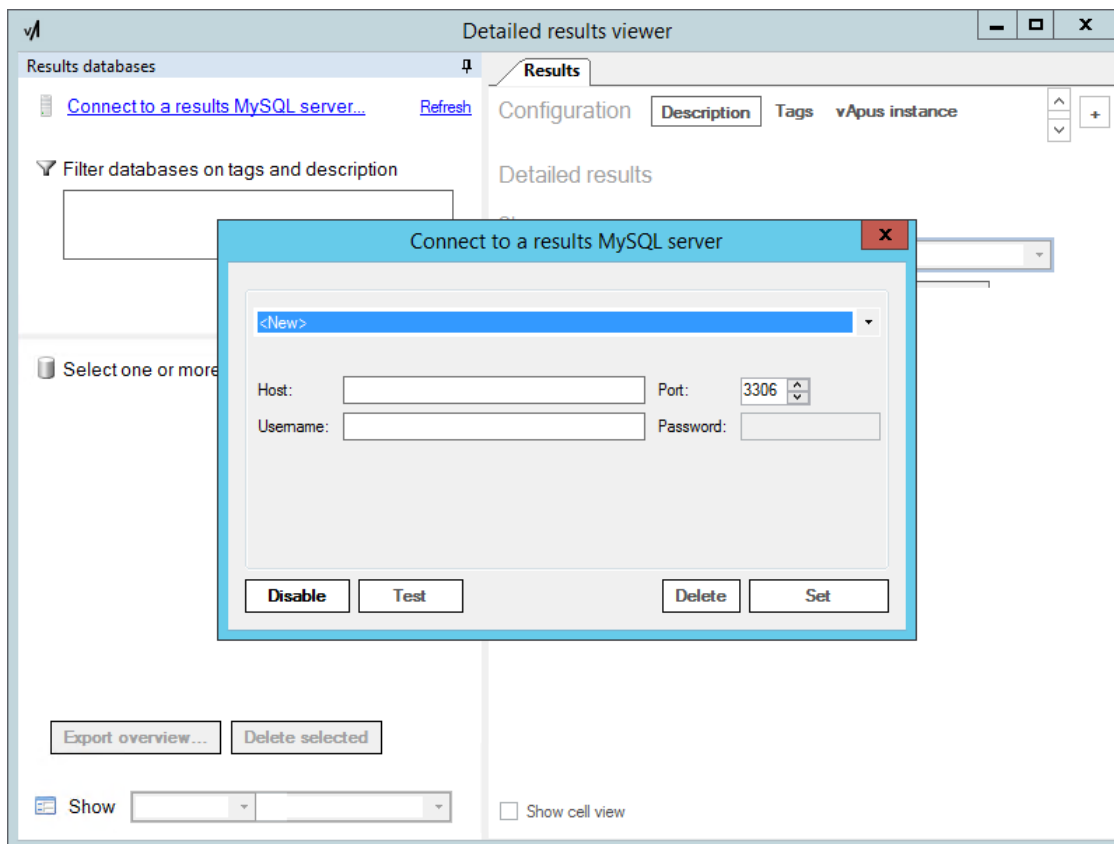
Reviewing the report will help you understand the next steps in this introduction

Go back to your stress test in vApus.

Via the tab *Detailed results* or via *Tools> Detailed results viewer...* we can consult our test results and export them to an *Excel spreadsheet*.



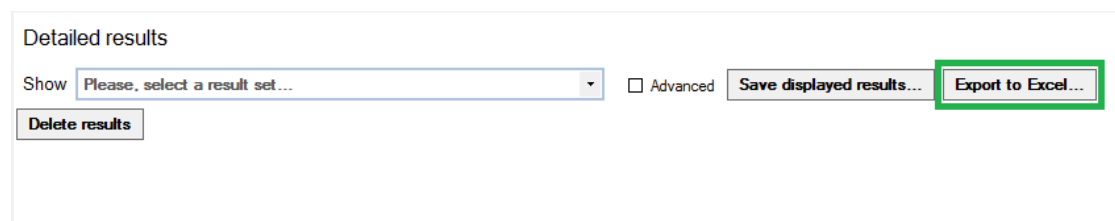
First, we have to set up the connection to the MySQL server. Enter the same settings you entered in the *Publish items handler* before.



If all is well, we can select the results database and calculate various result sets.

Instead, we export results directly to Excel:

Click the *Export to Excel...* button



Make sure that everything except Specialized is indicated and click on Export to Excel ...

Unzip the zip file, you should see one Excel file with the stress test results and one Excel file with the monitor metrics.

Notice that the *Stress Test* spreadsheet contains the same type of charts that are used in the example report you reviewed.

The monitor spread sheets do not yet contain charts. We must generate these using an *Excel macro*.

Open the *monitor spreadsheet* in Excel.

Remove the Network columns where you do not see a network load so you only have 1 Send and 1 Receive column.

Via *View> Macros* choose to run the *WMI_charts_make* macro.

Once the macro is finished, you still have to enter the RAM that the monitored servers, in bytes, e.g.: = $16 * 1024 * 1024 * 1024$.

Remember: in this case we pretended that the vApus client was the server / ran the monitor on the vApus client.

At this point it should be clear to you that writing a PDF report of the results is not an automatic process.

This requires the necessary insights into how the tested application works. Every application is different, so every report is custom tailored to the client's needs.

Besides analyzing vApus results, other insights are often necessary. If it concerns a web application, we can do a PageSpeed and YSlow analysis using e.g. gtmetrix.com.

Knowing all this, read the example report again, thoroughly.



Next steps

This introduction is just as small overview of the vApus basics. . We did not talk about:

- Customizing connections and reverse-engineering server application communication
- Parameterizing requests in a scenario, so e.g. a simulated users logs into a web site with a unique user name
- Distributed testing one or multiple applications using a vApus master-slave setup
- Technology specific monitors, hardware and software

And, maybe even more importantly, we did not talk about the methodologies to test, monitor and analyze that Sizing Servers Lab fine-tuned over years of stress testing experience.

For specific questions, you can try to e-mail info@sizingservers.be. Keep in mind that support is not guaranteed, since active development is halted.

info@sizingservers.be

