Windows Installation

Before installing NetVizura make sure to set the time on your server correctly. Time change after the installation will invalidate the license!

Before installing NetVizura you will have to install: Oracle Java 7, Tomcat 7 or higher and PostgreSQL 9.3 or higher, in that order. The installation process has been tested on Windows Server 2008 R2 and Windows Server 2012 R2.

NetVizura Installation Steps

To install NetVizura on Windows follow these steps:

Step 1: Download and install Oracle Java 7 from Oracle official website www.oracle.com/technetwork/java/javase/downloads/index.html

Step 2: Download and install Tomcat 7+ as a service from Tomcat official website tomcat.apache.org. 32-bit/64-bit Windows Service Installer is available on the downloads page

- Make sure to install Tomcat as a service, otherwise NetVizura installation won't be able to complete successfully.
- Make sure you have exactly one version of Tomcat installed on your system, otherwise application might not work as expected.

Step 3: Download and install PostgreSQL 9.3+ from PostgreSQL official website http://www.postgresql.org/download/windows/

- While PostgreSQL installation you will be prompted for username and password, make sure that username is "postgres" and password is "postgres".
- Other values should stay the same (port 5432 and Default Locale).
- Make sure you have exactly one version of PostgreSQL installed on your system, otherwise application might not work as expected.

Step 4: Download NetVizura Windows Installer from NetVizura website and run installer with administrative privileges

Step 5: Follow the installation steps

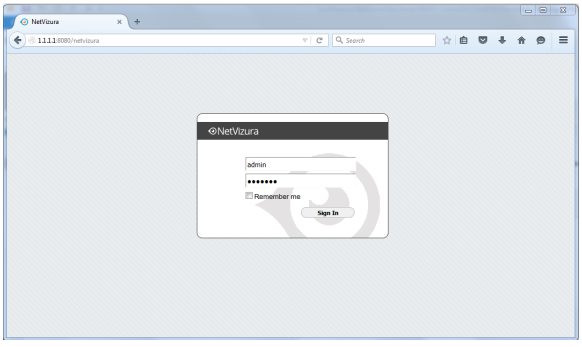
Step 6: Verify installation

Now you can go to NetVizura web interface http://serverip:8080/netvizura.

Default login credentials:

- Username: **admin**
- Password: **admin01**

For example, if your server IP is 1.1.1.1 then point your browser to http://1.1.1.1:8080/netvizura like in the screenshot below:



Post Install Steps

Tomcat Memory Allocation

After installation tweaking of configuration files is required in order to utilize the installed RAM to the fullest extent. The main consumers of RAM are operating system, PostgreSQL database and Tomcat. General rule for distributing memory is to split it in ratio 2:1 between PostgreSQL and Tomcat with 1 GB or more reserved for operating system. For instance:

Installed RAM	PostgreSQL	Tomcat	OS
4 GB	2 GB	1 GB	1 GB
16 GB	10 GB	5 GB	1 GB

During installation NetVizura automatically allocates memory for Tomcat process. The amount allocated to Tomcat process is calculated according to the formula:

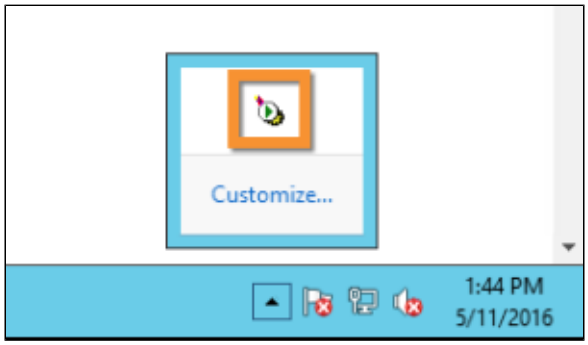
$(RAM_{total}$ - 1GB) / 3 but no less than 1GB.

For instance:

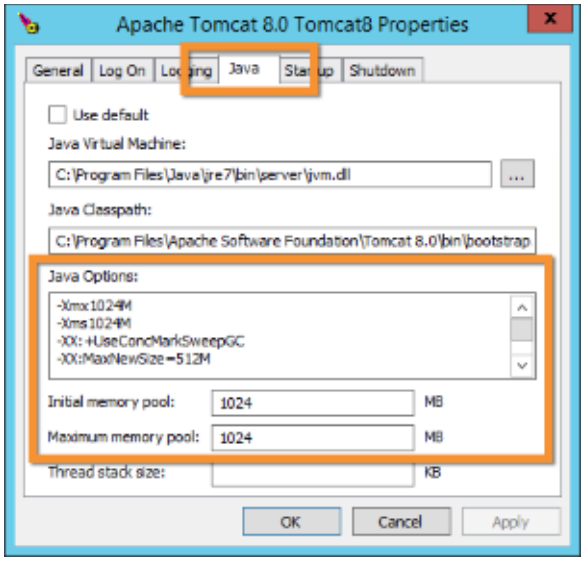
Total RAM	Tomcat
3 GB	1 GB
4 GB	1 GB
16 GB	5 GB

However, if you need to tweak Tomcat RAM allocation differently (the example for 2048MB):

1. Double click on Apache Tomcat Properties in system tray



2. In Java tab under Java options modify the `-Xmx` parameter to allocate additional memory to Tomcat. Additionally, set parameter `-Xms` to the same amount. Also set Initial memory pool and Maximum memory pool to the same amount. This should look like on picture below.



3. Back to the General tab, click Stop and Start to restart Tomcat.

Tweaking PostgreSQL

Tweaking PostgreSQL for best performance is a topic on which many books were written, but the following are some common sense suggestions. In general there are two groups of PostgreSQL tweaks that are helpful for NetVizura performance - "safe" and "unsafe" tweaks. "Safe" tweaks are those which can be applied in all cases. "Unsafe" tweaks trade reliability for performance. For the curious ones recommended reads (among countless others) are PostgreSQL Optimization Guide, PostgreSQL Tuning Guide, this article and this book.

In order to apply following tweaks edit file `postgresql.conf`, this file is usually located in PostgreSQL data folder. You will need to restart the PostgreSQL service after done editing. Almost all of the following parameters are commented with carron character (#). Although these tweaks are considered "safe" do take notice of the default values. Usually you can comment out the parameter that has been changed and PostgreSQL will revert to the default value.

PostgreSQL "safe" tweaks

In the following example it is assumed that 4 GB of RAM is allocated for PostgreSQL.

parameter	recommended value	comment
`max_connections`	30	NetVizura rarely uses more than 10 connections simultaneously, but it is good to have some reserve
`shared_buffers`	1024MB	the recommended amount is `RAM/4`
`effective_cache_size`	2048MB	the recommended amount is `RAM/2`, possibly even `RAM * 3/4`
`chekpoint_segments`	32	for write intensive apps (as NetVizura) it should be at least 16, with 32 as safe maximum
`checkpoint_completion_target`	0.9	
`default_statistics_target`	100	
`work_mem`	8MB - 12MB	The formula used is `max_connections*work_mem <= RAM/8`, but using a bit more is still "safe"

PostgreSQL "unsafe" tweaks

These optimizations are considered "unsafe" since they *could* in very rare cases lead to data loss and/or corruption. If your VM is properly backed up we would consider the following optimizations safe. The following bring huge performance boosts to DB write process.

parameter	recommended value	comment
`maitenance_work_mem`	32MB	speeds up DB self clean process, not really important
`wal_buffers`	16MB	
`full_page_writes`	off	
`fsync`	off	don't wait for HDD to finish previous *write* operation. This brings the most benefit, but is considered potentially the most unsafe of all. If there is OS or HDD failure in exact instant when PSQL issues write command to HDD, that data will be lost and the DB itself could be corrupted. On the other hand, DB can issue several magnitude more write commands in the same time period and consider all these done, thus improving write performance immensely.
`synchronous_commit`	off	similarly to "fsync" but less unsafe and with less benefit
`checkpoint_segments`	64	how much is cached in temp files before it is issued to *proper* DB files. Issuing big chunks of data for write rarely is usually better for performance than issuing small chunks often