# Zabbix Agent Simulator

*To developing the extensions, to testing them, to verify the configuration, to teaching, to demo and to the glory of Zabbix !*

## Credits.

I wish to thank the authors of the software called SNMPSIM.

http://snmpsim.sourceforge.net/

The ones who shows me the way.  Also, the authors of the Zabbix monitoring platform. Without you, this software wouldn't exists at all

http://www.zabbix.com

*Thanks you for all your help and encouragement*

# Preface

What is the Zabbix Agent Simulator. Well, as the name probably suggesting, it is  a piece if software which simulating the Zabbix Agent. Your first question probably will be: “Why on Earth you wanna do this ?” There are simple answer on this question and an answer with “multiple choices of answers”. I will start with simple one:

Zabbix Agent Simulator intended to be used instead regular Zabbix passive Agent in the situations, where use of the real Zabbix Agent is not practical or simply not desirable.

**What is “non-practical” ?**

“Non-practical” means, that the use of the regular Zabbix Agent which is coming with Zabbix will not bring desirable results. For example: when you are developing your Triggers and Actions, you test host rarely having the thresholds that you are programming. Many times, I saw how Zabbix Administrators deployed configurations, which were not fully tested.

**What is “not desirable” ?**

“Non desirable” means that you are following the design and development solution which is flawed in one or several ways and continue to use this development method will bring you into a troubles later on. For example: many times, I saw when people were tried to test there configurations on live installations. Indeed, this is a very bad practice and it must be avoided at all cost.

So, what is the “long answer” on what is the Zabbix Agent Simulator and the scope of use for this software tool

- Configurations and monitoring architecture testing.
  Zabbix Agent Simulator can generate you the values for the metrics exactly to your specifications and scenarios. You can simulate any conditions on your host or Agent monitored application, without bothering the production environment and you can perform full test of your Zabbix configuration behavior in all possible situations.

- Post-mortem analysis.
  If your production configuration doesn't work, you can take the values and “re-play” them on your development and test Zabbix installations. This will permit you to see what is wrong with your production configuration and how to fix that.

# Preface

So, what is the “long answer” on what is the Zabbix Agent Simulator and the scope of use for this software tool

- Zabbix architecture and capabilities  demonstration.
- Safe simulation of the massive infrastructure on your virtual hosts for your development and test “sandboxes”
- Use during training and in other forms of the lab and educational environment. Safe simulation of the catastrophic failures of your infrastructure for educational and training purposes. I say: “let them play with this simulator, not with the real things”.

*And any other situations similar and not similar to those.*

Why not just use the zabbix_agentd with added UnixParameters or loadable modules which will generating us sweet randomness or read REDIS or whatever ?

Because zabbix_agentd does not permit us to re-define embedded keys, such as vfs.fs.size, net.if.in and others. And in my view, it is a very bad idea to use one keys during the tests, and then change them for production. In fact, this is invitation of serious potential problems with configuration.


That's why we must have something, which will act exactly like a real Agent, without being one.


You will be needed a REDIS Server if you want to run ZAS Agent. You can make ZAS running without REDIS, but for the sanity and for the future, where ZAS will be even more closely tied with REDIS, I say, this is a mandatory requirement. I will not be covering on how to obtain and install REDIS server, since this topic is outside of the scope of our discussion.

ZAS Agent is a Python application and do require that you have Python2 of version 2.6+ and older installed on your host. Now-days, Python is a standard staple on most Unix-like operating systems, but do check if you have one and what it's version.

```
[root@class1 ~]# python --version
Python 2.6.6
```

ZAS Agent requires some of the standard and 3-rd party Python modules to function. To check, if your environment is ready, please run the script check_python_packages.py located in subdirectory “install”

```
[root@class1 install]# python check_python_packages.py
Module 'os'.................. OK
Module 'sys'................. OK
Module 'ConfigParser'...... OK
Module 'argparse'.......... OK
Module 'struct'............ OK
Module 'multiprocessing'... OK
Module 'socket'............ OK
Module 'time'.............. OK
Module 'logging'........... OK
Module 'redis'............. OK
Module 'numpy'............. OK
Module 'fnmatch'........... OK
Module 're'................ OK
Module 'signal'............ OK
Module 'daemonize'......... OK
```

Here, we will review some of the required packages, which usually are not the part of the standard Python library. The easiest way to install those packages is with help of the handy python-pip tool. If this tool is not included in your OS distribution, you can download and install it from here:

https://pip.pypa.io/en/stable/installing/

| Package name | Homepage | Description |
|---|---|---|
| redis | https://github.com/andymccurdy/redis-py | The Python interface to the Redis key-value store. |
| numpy | http://www.numpy.org/ | NumPy is the fundamental package for scientific computing with Python. |
| daemonize | http://daemonize.readthedocs.org/en/latest/?badge=latest | Library for writing system daemons in Python. |

Change working directory to the package source directory and execute

python setup.py build

```
[root@class1 zas_agent]# python setup.py build
running build
running build_scripts
changing mode of build/scripts-2.6/zas_agent.py from 770 to 775
```

If there is no error messages, move to the next step ...

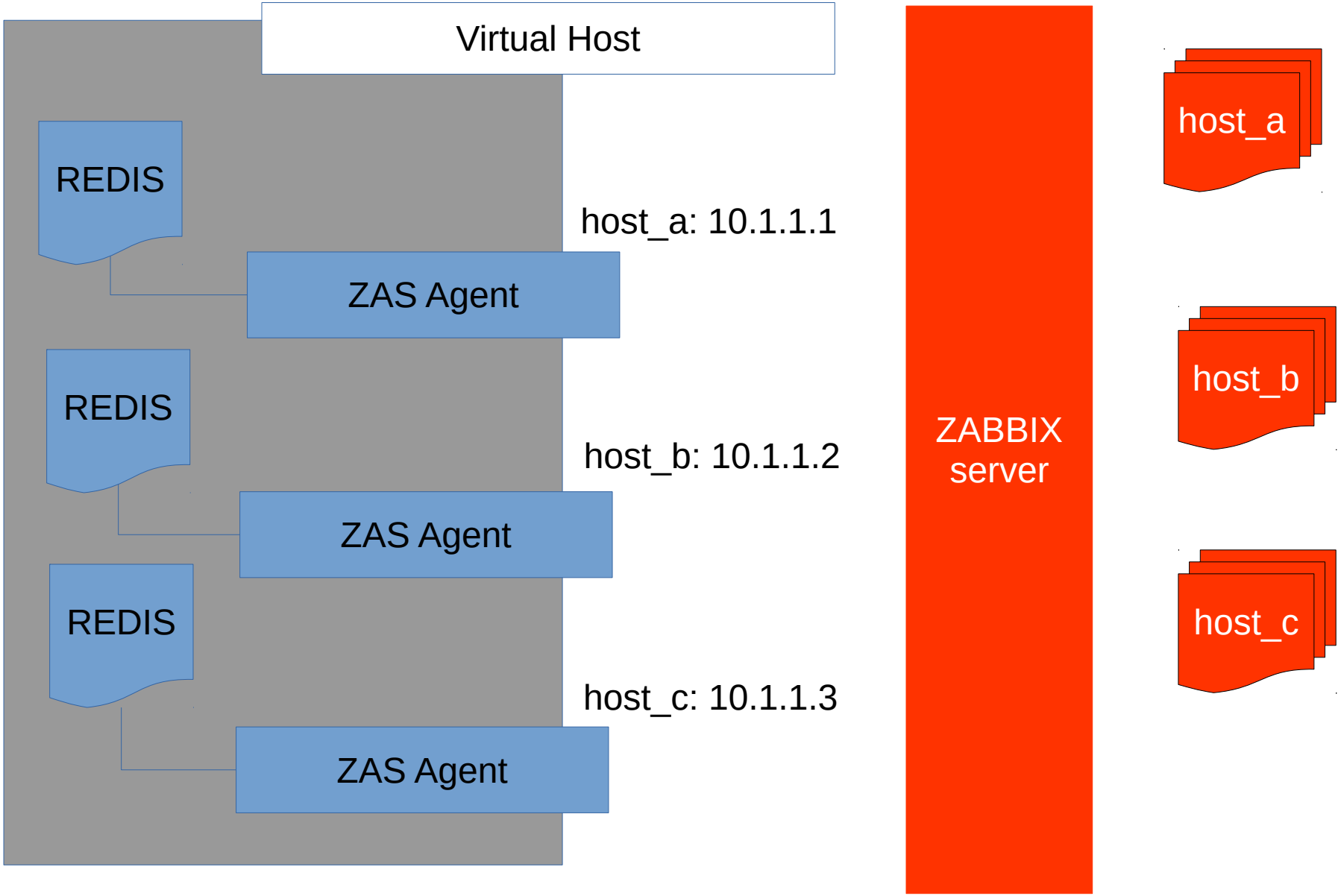
Run python setup.py install

```
[root@class1 zas_agent]# python setup.py install
running install
running bdist_egg
running egg_info
equirements to zas_agent.egg-info/requires.txt
g zas_agent.egg-info/PKG-INFO
level names to zas_agent.egg-info/top_level.txt
ncy_links to zas_agent.egg-info/dependency_links.txt
t file 'zas_agent.egg-info/SOURCES.txt'
t file 'zas_agent.egg-info/SOURCES.txt'
ary code to build/bdist.linux-x86_64/egg
install_lib
warning: install_lib: 'build/lib' does not exist -- no Python modules to install
installing package data to build/bdist.linux-x86_64/egg
running install_data
creating build/bdist.linux-x86_64/egg
copying doc/zas_agent.py.1 -> /usr/share/man/man1
creating build/bdist.linux-x86_64/egg/EGG-INFO
installing scripts to build/bdist.linux-x86_64/egg/EGG-INFO/scripts
running install_scripts
running build_scripts
changing mode of build/scripts-2.6/zas_agent.py from 770 to 775
creating build/bdist.linux-x86_64/egg/EGG-INFO/scripts
copying build/scripts-2.6/zas_agent.py -> build/bdist.linux-x86_64/egg/EGG-INFO/scripts
changing mode of build/bdist.linux-x86_64/egg/EGG-INFO/scripts/zas_agent.py to 775
copying zas_agent.egg-info/PKG-INFO -> build/bdist.linux-x86_64/egg/EGG-INFO
copying zas_agent.egg-info/SOURCES.txt -> build/bdist.linux-x86_64/egg/EGG-INFO
copying zas_agent.egg-info/dependency_links.txt -> build/bdist.linux-x86_64/egg/EGG-INFO
copying zas_agent.egg-info/requires.txt -> build/bdist.linux-x86_64/egg/EGG-INFO
copying zas_agent.egg-info/top_level.txt -> build/bdist.linux-x86_64/egg/EGG-INFO
zip_safe flag not set; analyzing archive contents...
creating 'dist/zas_agent-0.1.1-py2.6.egg' and adding 'build/bdist.linux-x86_64/egg' to it
removing 'build/bdist.linux-x86_64/egg' (and everything under it)
Processing zas_agent-0.1.1-py2.6.egg
removing '/usr/lib/python2.6/site-packages/zas_agent-0.1.1-py2.6.egg' (and everything under it)
creating /usr/lib/python2.6/site-packages/zas_agent-0.1.1-py2.6.egg
Extracting zas_agent-0.1.1-py2.6.egg to /usr/lib/python2.6/site-packages
zas-agent 0.1.1 is already the active version in easy-install.pth
Installing zas_agent.py script to /usr/bin

Installed /usr/lib/python2.6/site-packages/zas_agent-0.1.1-py2.6.egg
Processing dependencies for zas-agent==0.1.1
Searching for redis==2.0.0
Best match: redis 2.0.0
Adding redis 2.0.0 to easy-install.pth file

Using /usr/lib/python2.6/site-packages
Searching for numpy==1.4.1
Best match: numpy 1.4.1
Adding numpy 1.4.1 to easy-install.pth file

Using /usr/lib64/python2.6/site-packages
Searching for daemonize==2.4.2
Best match: daemonize 2.4.2
Processing daemonize-2.4.2-py2.6.egg
daemonize 2.4.2 is already the active version in easy-install.pth

Using /usr/lib/python2.6/site-packages/daemonize-2.4.2-py2.6.egg
Finished processing dependencies for zas-agent==0.1.1
```

setup.py script will install the following files:

| Filename | Path | Description |
|---|---|---|
| zas_agent.py | /usr/bin or /usr/local/bin | Main executable file of the ZAS Agent |
| network.scenario | /etc/zas | Sample scenario you can use as a guideline |
| zas_scenario.cfg | /etc | Default ZAS Agent configuration file |
| zas_agent.py.1 | /usr/share/man/man1 | Manpage for the main ZAS program file. |
| PDF documentation | /usr/share/zas_agent | PDF documentation |

# ZAS IN YOUR INFRASTRUCTURE



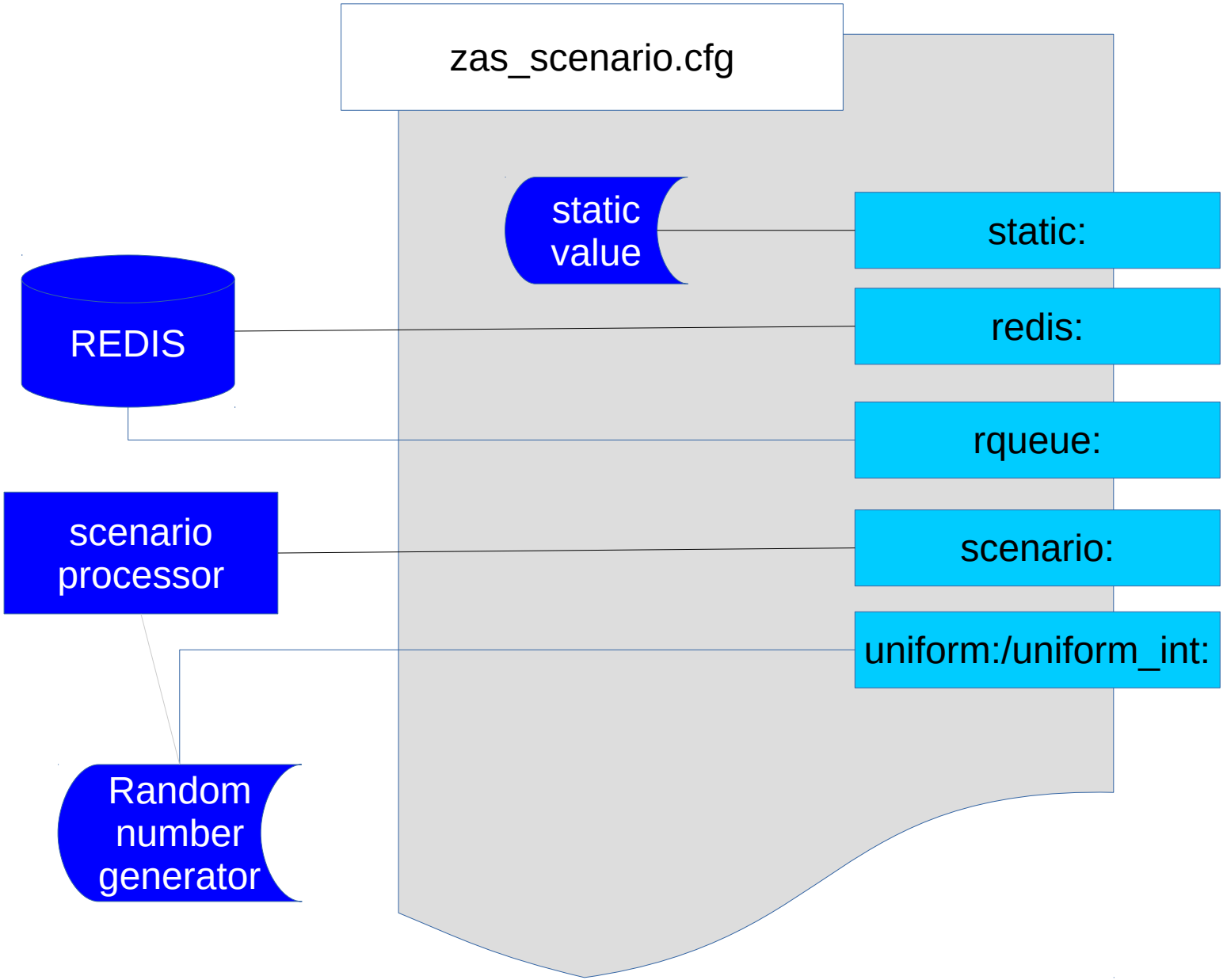
# ZAS DATA FLOW

Zabbix Agent Simulator while mimicking the real Zabbix passive Agent, will return the values for the requested keys, according to the Agent configuration. The values are ether static, generated according to some scenarios or obtainable from external sources (Redis datastore). Unlike real Agent, Zabbix Agent Simulator returns only values for the keys, specified in ZAS configuration.

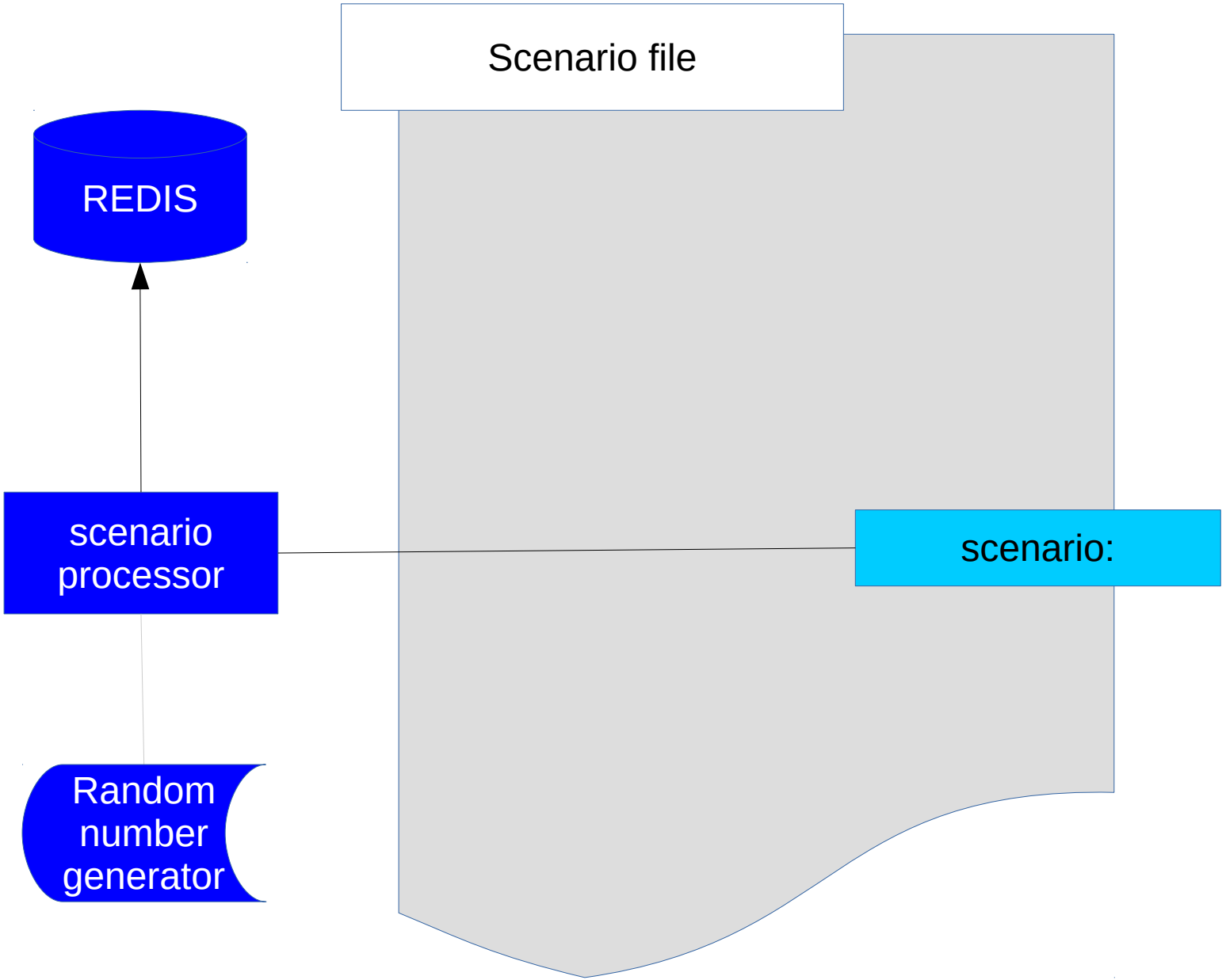
**Here is the value types, which can be used as return value with ZAS**

| Value type | Description |
|---|---|
| static: | The simplest value type, returning the static value defined in ZAS configuration. |
| uniform:<br>uniform_int: | Return the uniform random number within the range |
| redis: | Searches Redis database #0 for the key matched the requested metric key and return stored value |
| rqueue: | Searches Redis database #1 for the key matched the requested metric key, which refers the list and returns last item from this list |
| scenario: | |

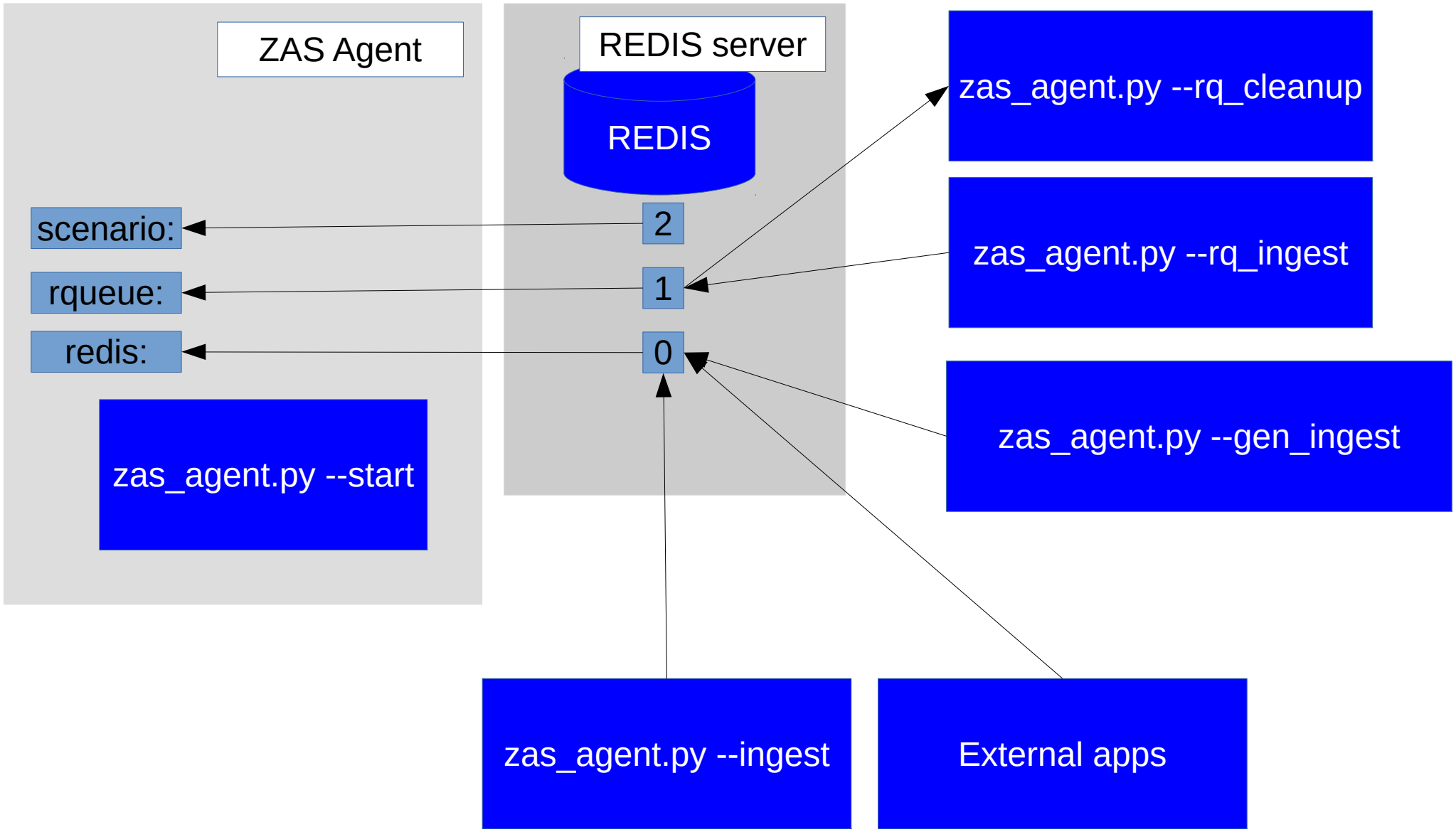
# ZAS Data flow



# ZAS Data flow



# ZAS Data flow



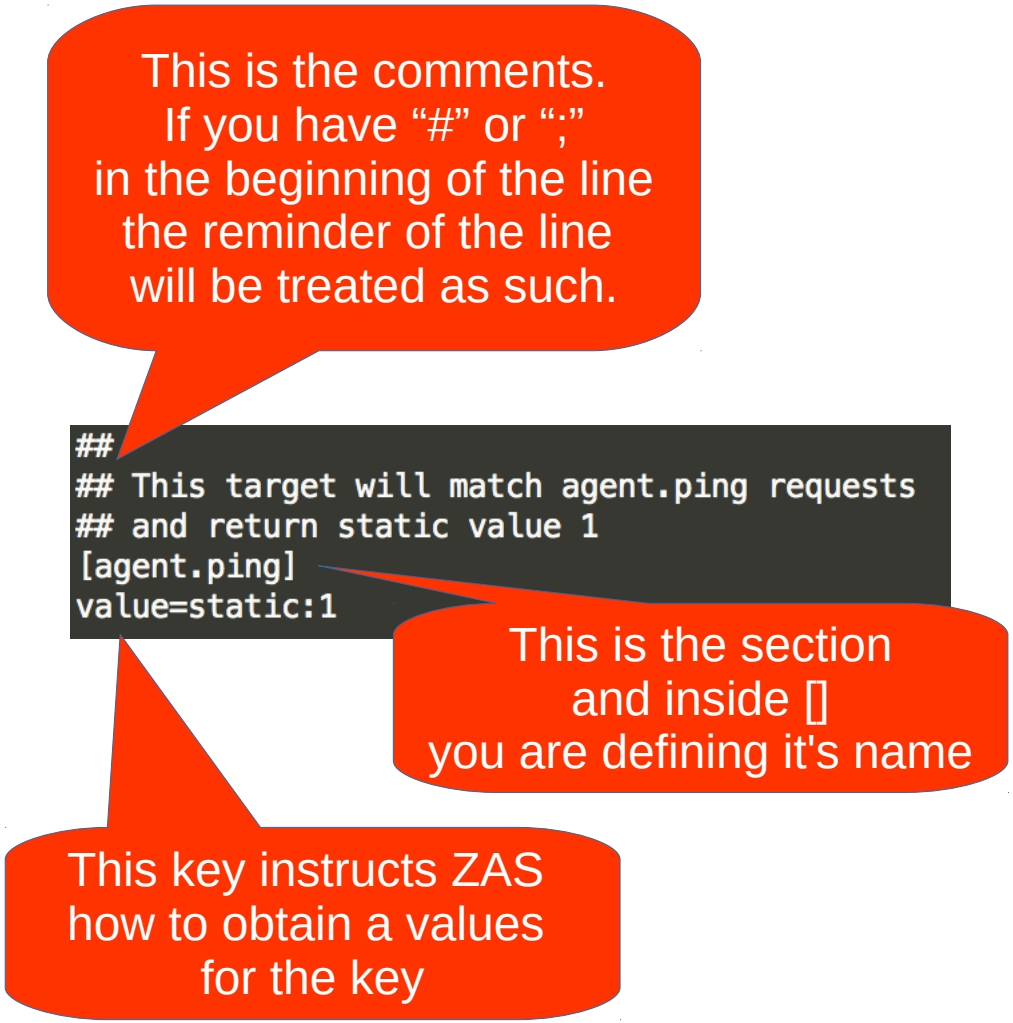
# ZAS CONFIGURATION

The first step of the ZAS Agent configuration is to configure agent itself. There are several ways on how the agent can obtain the data for the metric: through ether static value allocation, random number generation, reading the data from REDIS server, reading the data from the queues on the REDIS server or through direct processing of the scenarios. How exactly you will configure your simulator, which keys will be using which value sources, will be depend on your needs and a situations.

| static: | uniform: uniform_int: | redis: | rqueue: | scenario: |
|---|---|---|---|---|
| You shall use static: values if you do not plan to change those values “on the fly”. The good examples for the static: are the agent.version, agent.uname and the others of the same nature | If you do not care about what's your values are look like, and you do want them to be uniformly and randomly distributed within specified range, in that case, you can use uniform(_int): as your value source. | You can use redis: as the source for your values, if you are planning to ingest data into your simulator from external processes or from zas_agent.py running in --ingest mode. | You can use rqueue: as the values source, in the situations where you have a pre-recorded or pre-generated sets of values. | Acting like uniform: “on steroids”, scenario: offers realistically looking but randomly generated values. |

# ZAS CONFIGURATION

The main configuration file for the ZAS Agent do have a name zas_scenario.cfg and default location of this file is /etc . You can override defaults by passing –scenario option to the zas_agent.py.




The format of the file is familiar to anyone, who ever saw .ini configuration files. If you are Python developer, this is the format which is handled by the ConfigParser module.

This configuration file consist of the sections and a keys and a values, defined in those sections. Some key-value pairs do have a meaning only for the certain value sources.

Each section name may be exact match to the name of the Zabbix metric. Also, section name might be free-form description string.

The most important key within the section is value= . This key defines how ZAS Agent obtains the values for the specific key.

# ZAS CONFIGURATION

When ZAS Agent receives the request for metric from Zabbix Server or Zabbix Proxy, first, it is searching the scenario file and matches requested metric with values source. So, how the ZAS Agent discovers the metric key and what is the process of matching ?

```
##
## This target will match agent.ping requests
## and return static value 1
[agent.ping]
value=static:1
```

The request for the metric “agent.ping” will be matched to that section

There are two possible ways to do that. First, through section name which must be matched exactly to the requested metric name

If exact match in section names is not found, ZAS Agent loops through all sections and reads the value of the key “match=”

```
##
## Scenario calculations
##
[all system.cpu.util]
match=system.cpu.util*
value=scenario:
scenario={"min":0,"max":20,"type":"float","variation_min":10,"variation_max":10}
```

# ZAS CONFIGURATION

If exact match in section names is not found, ZAS Agent loops through all sections and reads the value of the key “match=” and using this value as regular expression tested against this metric. If requested metric key matches this regular expression, then this section will be used for further processing of the values for this key. There are two types of matching performed during each test. First, Unix shell like matches and then POSIX regexp matches. If you are the Python developer, first, I am using Python module fnmatch, then re. Of course, during those checks, the section names do not poses the special meanings, like in the first pass and you can use free-form descriptive strings as a section names.

```
##
## Scenario calculations
##
[all system.cpu.util]
match=system.cpu.util*
value=scenario:
scenario={"mi    0,"max":20,"type":"float","variation_mi
```

This is shell-like regular expression and it will match to all system.cpu.util[…] metrics requests

This is regexp regular expression and it will match to all vfs.fs.size[...,free] metrics requests

```
##
## This target      tch all vfs.fs.size[*,free] requests
## and request      from REDIS
##
[filesystem.s ze free]
match=vfs.fs.size\[(.?),free\]
value=redis:
```


Please note, ZAS Agent is not full-blown Zabbix Agent and you shall not use it as replacement for such. ZAS Agent will respond only with values for the keys to which match is found in the scenario file. If you will try to request the key, for which match is not found, ZAS Agent will simply close the connection.


Please take a look at the value of the key “value=”

The format for this value is:

**metric source:optional parameter**

The first and the simplest value source is static:

As the name implies, when requested, it is returning static value, passed as optional parameter to the value of the key “value=”

The format to this value source is **static:value**

For the metric “agent.ping”

```
#
#  is target will match agent.ping requests
#  and return static value 1
[agent.ping]
value=static:1
```

Value source is static:

And the returned static value will be “1”

The next pair of the sources uses internal mechanism of generating simple uniform random numbers within specified range.

The name of those sources are uniform: and uniform_int:. The difference between them, that the uniform: returns float and uniform_int: returns integer. The parameter to the value source is comma delimited two numbers. First is lower end of the range and second is higher end of the range, whithin which we will generate a uniform random number.
The format to this value source is:
**uniform:low end,high end** or **uniform_int:low end,high end**

```
[filesystems.size pfree]
match=vfs.fs.size\[(.?),pfree\]
value=uniform:1,100
```

For the all requests of vfs.fs.size[…,pfree]

Generate float uniform random number in a range between 1 and 100

For the all requests of application.connections

Generate integer uniform random number in a range between 1 and 100

```
[application.connections]
value=uniform_int:1,100
```

The first “external source” (means the source which requires component other than ZAS Agent) is redis:

The idea behind this value source is very simple: ZAS Agent will connect to REDIS server DB=0 and request the value of the key matched with requested metric. If value is found, it will be returned. If not, connection will be closed. REDIS server IP address or FQDN and the port can be passed through command-line options –redis_host and --redis_port
The format to this value source is: **redis:**

```
##
## This target will match all vfs.fs.size[*,free] requests
## and request data from REDIS
##
[filesystem.size free]
match=vfs.fs.size\[(.?),free\]
value=redis:
```

In this example, for all metric requests vfs.fs.size[…,free] the values will be requested from the REDIS server

Another “external source” is rqueue:

The difference between redis: and rqueue: in in how the values are stored in the REDIS server. For the redis:, the single value is stored and readed. For rqueue:, values are stored and requested from the queue. If queue is empty, ZAS Agent closes the connection. If you familiar with REDIS, redis: uses SET/GET and rqueue: uses LPUSH/LINDEX/RPOP. In order to separate rqueue: from redis:, REDIS uses DB=1 for rqueue:.  REDIS server IP address or FQDN and the port can be passed through command-line options –redis_host and --redis_port
The format to this value source is: **rqueue:**

```
##
## This target will match all vfs.fs.size[*,used] requests
## and request data from REDIS lists
##
[filesystem.size used]
match=vfs.fs.size\[(.?),used\]
value=rqueue:
```

In this example, for all metric requests vfs.fs.size[…,used] the values will be requested from the REDIS server queues


Every time, when you are requesting the value from rqueue: FIFO source, you will be getting the first value in the queue, and the value stays in the queue. ZAS Agent do not pop and remove the values from the queue authomatically. In order to move to the next value, you do need to run

zas_agent.py –rq_cleanup

This command will go through all keys in the queue with delay between passes –ttl seconds and remove the first element of the queue


The behavior of the

zas_agent.py –rq_cleanup

when the length of the queue is 1 will be dictated by the option: –rq_cleanup_full

By default, cleanup code will keep the last element of the queue indefinitely, so your queue will always contain some data. If you pass this option, cleanup code will empty the queue, when the time come.

The last (but not least) value source is scenario:

The scenario: value source is random generator “on steroids”. Unlike “uniform:”, scenario: generates next value taking previous value into account. The previous value is stored on REDIS server DB=2. The scenario itself is JSON dictionary, where the keys and the values defines how the next value will be generated.

The format to this value source is: **scenario:{scenario}**

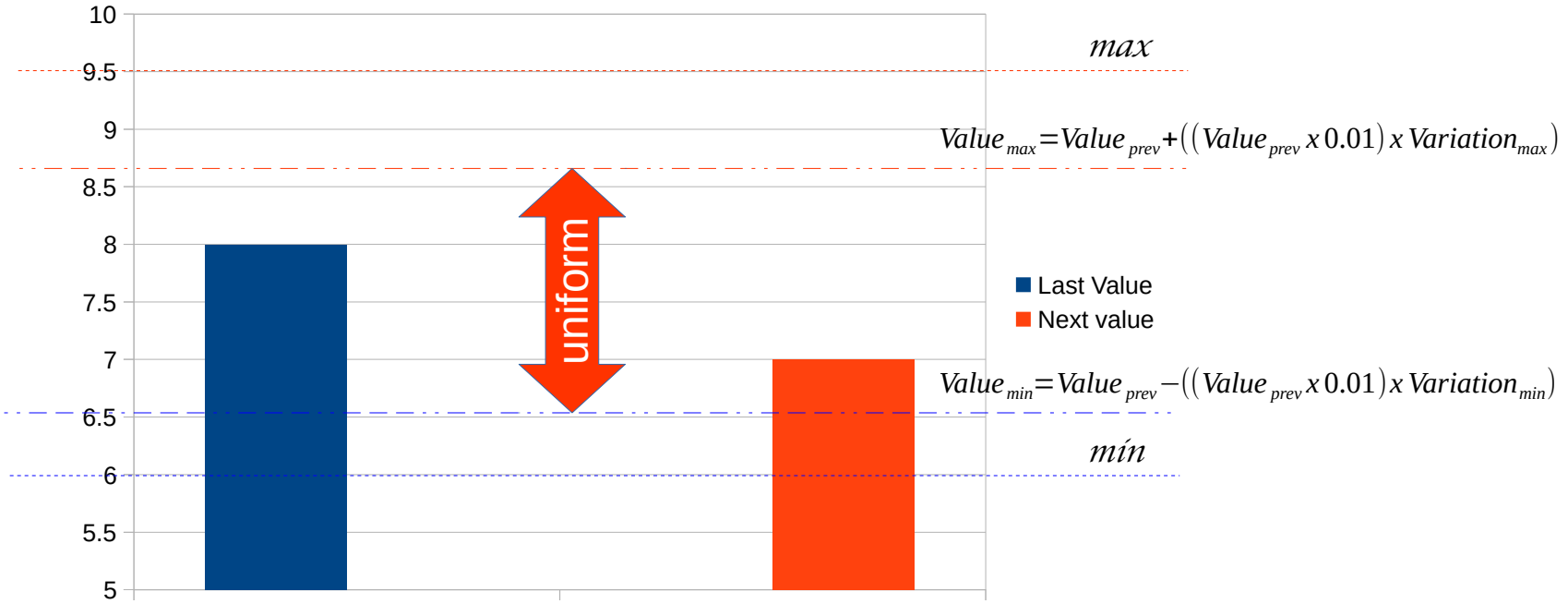
```
##
## Scenario calculations
##
[all system.cpu.util]
match=system.cpu.util*
value=scenario:
scenario={"min":0,"max":20,"type":"float","variation_min":10,"variation_max":10}
```

In this example, for all metric requests system.cpu.util[…] the values will be generated according specified scenario

Now, let's review the scenario keys and how they are impacting the value generation

| Scenario key | Description |
|---|---|
| min | Lower end of the range for generated values |
| max | Higher end of the range for generated values |
| type | Type of generated values. Possible options are “float” or “int” for the float or integer numbers respectfully |
| variation_min | When scenario generates next value, it will calculate “variation_min” percent between current and minimal value, |
| variation_max | and then calculates “variation_max” percent between current and maximum value. The next value will be generated within the range defined by this calculated values. |

Let's review in more details on how the next value in the scenario is generated

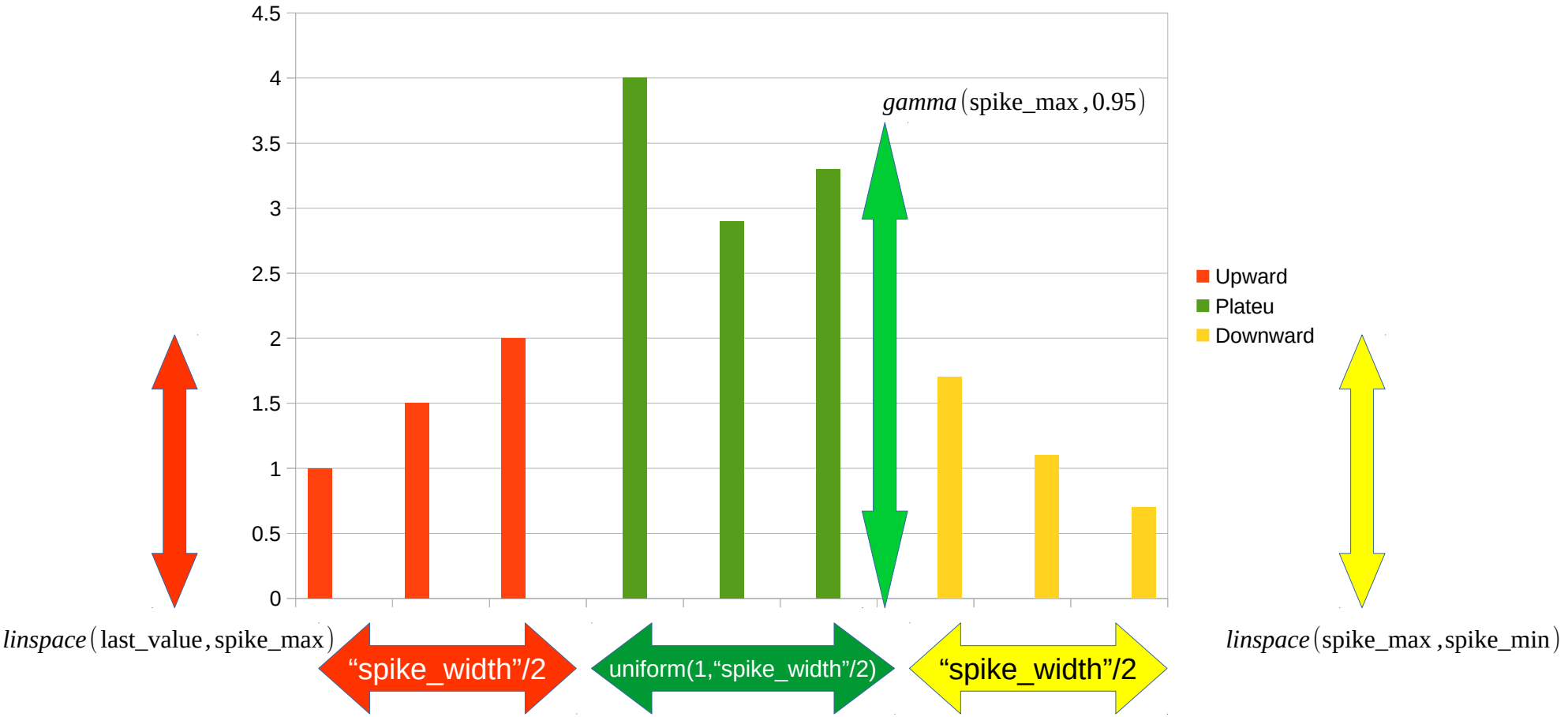


# ZAS CONFIGURATION

In addition to the values generation, scenario: supports generation of the spikes in the values. The process of generating “spikes”, controlled by following scenario parameters:

| Scenario key | Description |
|---|---|
| spike_barrier | Every time, when scenario: generates the new value, ZAS Agent are “throwing the dice” to see if it is a time to generate a new spike. The random number is generated between 1 and 100. If generated number is less than “spike_barrier”, and there is no spike currently generated, the scenario: switches from normal mode, to spike generation mode. So, higher the number for the “spike_barrier”, more spikes will be generated. |
| spike_width | When the scenario: goes to “spike mode” it will generate the spike, consists of “spike_width” values. Smaller number of “spike_width” means sharper spike. |

Let's review process of spike generation in more detail.




$$spike\_max = uniform(\text{current value}, maximum)$$

$$spike\_min = uniform(spike\_max, minimum)$$

# FEEDING SIMULATOR

So far, we learned how to make ZAS Agent retrieve or generate values. In some applications, just value generation would be enough. But there are three ways of data feeding, besides just update the values in the REDIS server using your own program.

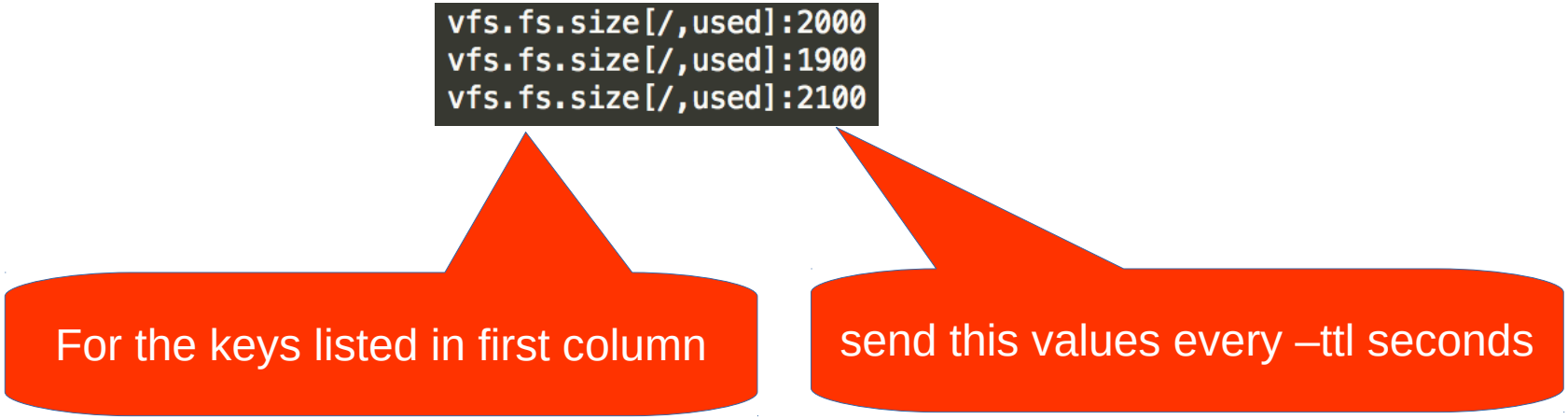
| --ingest | --rq_ingest | --gen_ingest |
|---|---|---|
| Ingest data from “ingest file” into REDIS server to the same database (DB=0) which is used by redis: value source.. | Push data from “ingest file”into REDIS queues in the same database (DB=1), used by rqueue: value source. | Parse “ingest scenario” file, generate values same way as scenario: value source and populates REDIS server, the same database (DB=0) used by redis: value source.<br><br>The path to the file containing ingest scenario is passed through argument --ingest_scenario |

Except –gen_ingest, you are passing the filename serving as “ingest file” for –ingest and –rq_inges using command-line argument –ingest_file. Next question is: what is the format of “ingest file” and “ingest scenario” ? Stay with us.

The format for “ingest file” used by –ingest and –rq_ingest is pretty simple and consists of two field, separated by “:”

**Metric key : Value**

Let see the example:



The format for “ingest scenario” file used by –gen_ingest the name of which passed through –gen_ingest argument is pretty similar to “ingest file” but instead of Value, we are passing scenario.

**Metric key : Scenario**

Let see the example:

```
# Comment
net.if.in[eth0]:{"min":0,"max":100000,"type":"int","variation_rnd":1,"spike_barrier":3,"spike_width":6}
net.if.in[eth1]:{"min":0,"max":100000,"type":"int","variation_min":10,"variation_max":25}
net.if.out[eth1]:{"min":0,"max":100000,"type":"int","variation_min":5,"variation_max":5}
net.if.out[eth0]:{"min":0,"max":100000,"type":"int","variation_min":5,"variation_max":5,"spike_barrier":10,"spike_width":6}
```

For the keys listed in first column

send this values every –ttl seconds

Please note. Those scenarios are the same JSON objects we discussed earlier. Another note, --gen_ingest will run those scenarios every –ttl seconds and will re-open and re-read scenario file during each pass.

# TODO

There are plans for the future development for the ZAS Agent. It shall be more modular, more extendable. The current scenarios and data generation processing are not perfect. I may incorporate the CLIPS for a more intelligent behavior. Making ZAS to mimic an active agent also would be nice thing to do. If you have any suggestions or you wan to contact an author with any related question, please feel free to do so. If you feel like participating in development of the ZAS Agent, your help will be always appretiated by community.

***vladimir.ulogov@zabbix.com***

# FINI

This is it. Now you know enough, so you can fill your test Zabbix with sweet random data and custom scenarios. Please make note, the first official release is 0.1.1 and it is very alpha code. But working code it is. The code is freely available under GPLv2 license and you will receive your copy of this license when you check this code out of the GitHub.

And here is, the not so secret location of the ZAS Agent source code.

https://github.com/vulogov/zas_agent