# Server part

1. The task of the server part is to receive HTTP requests and respond to them. In the process of work it is necessary to interact with

DBMS PostgreSQL. The DBMS may be physically located on another machine, so you must provide for such a deployment option

systems: several server applications can run on different physical servers and one DBMS on a single physical server.

1.1 The database stores information about clients, files for them (command /5/, point 3.3) and client-related events.

Each client has a number of inherent properties: Group Tag, Client Id, operating system, ip-address, geoip (country),

Importance, userdefined, when was registered, when was last online (command /0/, point 3.1).

Importance and userdefined are two integer numeric parameters (negative numbers, positive numbers and zero),

are installed or modified by other server applications or the server administrator/operator.

2. Work with the server is carried out through HTTP-requests.

They can be either GET or POST.

There can be as many reverse-proxies, load-balancer and DNATs between the server and the client as you want.

Thus, the only way to know the ip-address of the client that sent the request is the client-self-ip-address parameter

in the /0/ command (point 3.1). There are no other ways, will not be and cannot be in principle.

2.1 Each request contains three components in the URI, separated by "/"

``group-tag, client id, command code. /<group-tag>/<clientid>/<ccode>/,``

where group-tag is the group tag,

clientid - client id,

ccode - command code.

2.1.1 A group tag is an arbitrary string consisting of characters (a-z) and numbers (0-9). The parameter is not case-sensitive.

2.1.2 Client Id is a string consisting of two components separated by a dot.

The first part has the format <name>\_XYYYYYYY, where name is some name that can somehow identify the machine

(computer name or user name, depending on the type of operating system),

X - symbol indicating the type of system the client runs on (W - windows, L - linux, A - android, M - Mac OS),

YYYYYYYY - 3-7 digits containing major-version, minor-version and build of operating system if the system has any

(e.g. for 6.1 build 7600 it would be 617600).

The second part contains 32 random characters 0-9, A-F.

An example client id is ``QWERTY\_W617600.11223344556677889900AABBCCDDEEFF``.

The parameter is not case-sensitive.

2.1.3 The command code is a number from 0 to 999.

2.2 The fourth and subsequent components in the URI as well as the request and response body depend on the command code

2.3 The server response can be with the following HTTP codes: 200, 403, 404

# 3. Commands

## 3.1 /0/ - command loop initialization /1/.

Accepts 4 additional parameters:

\* system-version - the name of the operating system with version, service package and architecture (32, 64 bit)

\* client-version - client version, arbitrary number greater than 1000

\* client-self-ip-address - client's own external ip-address (ipv4 and ipv6) in string representation.

It is obtained by the client itself by means chosen at the discretion of the developer of this client.

\* token-string - a random string with length from 16 to 32 characters.

Can contain large and small English characters and numbers from 0 to 9. Case sensitive

\* devhash - hardware identifier. Arbitrary HEX string (characters 0-9, A-F) representing some 256-bit hash,

which depends on the hardware or operating system on which the client is running

Request format:

```/<group-tag>/<clientid>/0/<system-version>/<client-version>/<client-self-ip-address>/<devhash>/<token-string>/``

The server response is a file (content-type: binary) with the following content:

``/1/<group-tag>/<clientid>/<token-string>/<binary-content-length>/\r\n

<binary-content>\r\n

<server-sign> ``

token-string - the string copied from the GET request

binary-content-length - the length of the binary-content block in bytes written as a string

binary-content - block of binary data

server-sign is the server's digital signature, which includes the content of the response from the first byte to the second \r\n string translation (\r\n characters are included in the digital signature)

If the client is not in the database when you receive the command, it must be added to it, if it is, the information in the database

(group tag, ip-address, client version, operating system) must be updated.

File Selection.

The block of binary data in this case is the file extcfg, which is selected according to the same rules as in command /5/. If

Receiving the /0/ command means that the client is currently online. It is possible to receive several times in a row

commands /0/ from one client is a normal situation.

## 3.2 /1/ - issuing a command to the client.

It has only one additional parameter -

\* token-string (which will be duplicated in the response).

It is allowed to process this command only if the last command /0/ came from this client

no more than 24 hours ago (the time can be changed through the config file of the server application),

otherwise the server should respond with an HTTP 403.

The format of the response to this command:

``/<incode>/<group-tag>/<clientid>/<token-string>/<cmd-id>/\r\n

<command-params>\r\n

<server-sign> ``

\* incode - code of the issued command

\* token-string - the string copied from the /1/ command parameter from the GET request

\* cmd-id - unique identifier of issued command, random string with large and small English characters, numbers from 0 to 9 and symbols -,+,dot,#. Case sensitive.

\* command-params - any string with parameters (must not contain \r\n characters, as this indicates the end of the string)

\* server-sign is a digital signature of the server, which includes the content of the response from the first byte to the second \r\n string translation (\r\n characters are included in the digital signature)

## 3.3 /5/ - issuing the file to the client.

It has only one additional parameter - the filename. The file name is not case sensitive.

In response to this command, the server should output a file (content-type: binary) intended for this client.

The internal storage of the server application (database) must contain any number of files,

each file can have a filter on the following parameters:

group filter (defined by a list of like patterns separated by spaces),

group exclusion filter (defined by a list of like patterns separated by spaces, if there is at least one match in this list, then the file is considered not suitable for the client),

geoip,

operating system,

(beginning to end range),

userdefined (range start to end),

client id (strict match),

priority.

The priority is an arbitrary positive integer.

The group tag can be set not only by the strict name of the group, but also as a like-pattern from the SQL language.

The server can have several files with the same name, but with different filters.

If several files fit the client, the one with the higher priority value in the filter should be given

than the others. The base must not allow multiple files with the same priority.

If there is no file that meets the conditions of the client, the server should reply with an HTTP 404.

If a request comes from a client from whom /0/ has never been requested, then only the group tag should be checked

(as there is no other information in the database yet).

## 3.4 /10/ - getting a report from the client about the command he executed.

Query format for command /10/:

``/<group-tag>/<clientid>/10/<incode>/<cmd-id>/<result-code>/``

incode - the code of the executed command

cmd-id - the identifier of the executed command.

result-code - the result code of the executed command. A number from 0 to 999

When this command is received, the command that is in the queue at this client must be removed from the queue

No matter what result-code the client sends.

If there is no command in the queue with the specified code and ideentifier specified in the incode and cmd-id parameters (obligatory

check both values), then this is considered an abnormal situation must be recorded in some log or other

entity, which can then be viewed by the server administrator.

The response to this command is always the same HTTP 200, Content-type: text/plain, response body content "/1/"

## 3.5 /14/ - saving a key-value pair to the database.

To allow the administrator or operator to view these values.

Request format:

``/<group-tag>/<clientid>/14/<name>/<value>/0/``

\* name - key. An arbitrary string with English uppercase and lowercase letters, numbers, and the symbols point, "+" and "-".

\* value - value. Arbitrary string, all special characters must be urlencoded.

\* The third parameter is always zero.

The response to this command is always the same HTTP 200, Content-type: text/plain, response body content "/1/"

## 3.6 /15/ - Reading a key-value pair from the database.

``/<group-tag>/<clientid>/15/name``

Result:

HTTP 200 -- body: value

HTTP 204 -- value not found

## 3.7 /23/ - issuing a config to the client.

It has only one additional parameter - the version of the current client config. The version is a number from 0 to 2^32-1.

An example of a customer request:

``/<group-tag>/<clientid>/23/<current-config-version>/``

current-config-version - the current version of the config.

Example of a server response:

``/23/<group-tag>/<clientid>/<config-version>/<binary-content-length>/\r\n

<binary-content>\r\n

<server-sign> ``

binary-content - binary content of the config

config-version - config version in binary-content

binary-content-length - the length of the binary-content in bytes written as a string

The internal storage of the server application (database) must contain any number of configs.

Like a config file, it has an inherent parameter - version - an arbitrary numeric value that is set by the administrator or higher logic.

Each config can have a filter on the following parameters:

\* group tag (can be set as a like-pattern from the SQL language),

\* geoip,

\* operating system,

\* importance (range beginning to end),

\* userdefined (range start to end),

\* customer id (strict compliance).

The server can have several configs with the same version, but with different filters.

All configs with a version equal to or lower than the one specified by the client in the query are discarded.

The client can be given a config with a version strictly larger than the one he specified in the request.

If several configs fit the client at once, the one with a higher version than the others should be issued.

The base must be able to store configs with the same version.

If there is no file that meets the conditions of the client, the server should reply with an HTTP 404.

If a request comes from a client from which there has never been a request /0/ (or it came a long time ago), then the server responds immediately HTTP 403, without any checks.

## 3.8 /25/ - giving the client a link.

The command has no parameters.

An example of a customer request:

``/<group-tag>/<clientid>/25/<token-string>/``

token-string -identical to the parameter in /1/

Example of a server response:

``/25/<group-tag>/<clientid>/<token-string>/\r\n

<link>\r\n

<server-sign> ``

The internal storage of the server application (database) must contain an arbitrary number of references.

A link is an arbitrary string constructed according to URL rules.

Like files and configs, the link has an inherent parameter - the validity period in minutes, from the time it was added by the administrator or higher logic.

Each link can have a filter on the following parameters:

\* group tag (can be set as a like-pattern from the SQL language),

\* geoip,

\* operating system,

\* importance (range beginning to end),

\* userdefined (range start to end),

\* customer id (strict compliance).

The server may have several of the same links, but with different filters.

All expired links must be discarded when selecting links for the client.

If the client fits several links at once, it should be given the one that expires later than all the others (namely relevance, i.e. date-time of addition + expiration date in minutes).

If there are no links that satisfy the conditions of the client, the server should reply with an HTTP 404.

If a request comes from a client from which there has never been a request /0/ (or it came a long time ago), then the server responds immediately HTTP 403, without any checks.

# 3.9 /63/ - module data reception.

The command has the following parameters:

\* module name,

\* ctl to module,

\* the resulting ctl string,

\* auxiliary tag,

\* ctl\_OutData.

And the resulting ctl string, auxiliary tag, and OutData ctl are optional.

The parameter "ctl\_OutData" is a block of arbitrary binary data and is passed to the body via multipart/form-data, the parameter name inside multipart/form-data is "noname".

If ctl\_OutData is present in the request, the request is sent by POST.

An example of a customer request

``/63/<module name>/<ctl>/<ctl-result-string>/<aux-tag>/``

\* module name - the string consists only of English letters, the maximum length is 64 characters. It is assumed that there will be an index on this column in the table.

\* ctl - the string consists only of English letters and numbers, the maximum length is 64 characters.

\* ctl-result-string - a string in base64 format. The backend must save this string to the database in decoded form. The maximum length after decoding is 1024 characters

\* aux-tag - auxiliary tag, string consists only of English letters and numbers, maximum length is 128 characters. The field is necessary for easy search in the table, it is assumed that there will be an index on this column in the table.

\* ctl\_OutData - is passed in the body of POST request and contains a block of arbitrary binary data. The maximum size is 32 MB.

The server response is either always 200 - "/1/", or 403 if the client is not in the database or he sent /0/ too long ago

The server must store all the information received through the command /63/, namely:

\* date time,

\* clyntid,

\* ctl,

\* ctl-result-string,

\* aux-tag

\* ctl\_OutData.

All information should be stored in separate columns to allow sampling and filtering (of course, except ctl\_OutData, for which a column with a binary data type is sufficient).

## 3.10 /64/ - module event report.

The command has the following parameters:

\* module name,

\* event name,

\* event information,

\* auxiliary tag,

\* event data.

The event information, auxiliary tag, and data are optional.

The event information is a UTF-8 encoded string with a maximum length of 64 KB (bytes, not characters).

The data parameter is a block of arbitrary binary data and is passed to the body via multipart/form-data,

The name of the parameter inside multipart/form-data is "data" and the name of the parameter with information about the event is "info".

If a request contains data or information about an event, the request is sent by POST.

An example of a customer request

``/64/<module name>/<event-name>/<aux-tag>/``

module name - the string consists only of English letters, the maximum length is 64 characters. It is assumed that there will be an index on this column in the table.

event-name - the string consists only of English letters and numbers, the maximum length of 64 characters.

aux-tag - auxiliary tag, string consisting only of English letters and numbers, maximum length 128 characters.

The field is necessary for easy searching in the table, it is assumed that this column in the table will have an index.

The server response is either always 200 - "/1/", or 403 if the client is not in the database or he sent /0/ too long ago

The server must store all the information received through the /64/ command, namely:

\* date time,

\* clyntid,

\* event name,

\* info,

\* aux-tag

\* data.

All information should be stored in separate columns to allow sampling and filtering (of course, except data, for which a column with a binary data type is sufficient).

# 4. Each client must have its own command queue.

Commands are added to the queue by the operator or certain mechanisms of the server application.

Commands are removed from the queue only when the client sends a report on their execution with the /10/ command.

Until it reports in, the command will stand in the queue, EVEN if it results in the client requesting several

the same command. The command must be deleted from the queue upon receipt of the execution report, regardless of

which result-code the client has sent.

Each team has two attributes:

\* command code (number from 0 to 999)

\* The command parameter is an ANSI string of any length.

# 5. Server digital signature

which appears in the response of some commands is an ECDSA 256bit signature.

The current version is not used, instead it should be the string "1234567890"

# 6 DataPost.

- is a separate version of the server application that accepts ONLY the /60/ command and nothing else.

6.1 /60/ - not used in the current version of the system

# 7 IdleCommands functionality.

The server must provide some API (in the form of stored procedures) that will create "waiting commands".

The functionality of waiting commands implies that there is a group of commands that initially has no recipient clients.

Each group of waiting commands has the following parameters:

\* command code,

\* command parameter,

\* number of teams,

\* geoip filters (up to 10 countries)

\* operating system filter

\* group filter. The filter can be set as a like-pattern from the SQL language, or several like-patterns separated by spaces

\* group exclusion filter. The filter can be specified as a like pattern from the SQL language, or several like patterns separated by spaces. If the client group tag fits at least one pattern listed in this parameter, the waiting command is treated as NOT matching it. Consistency of this parameter with the previous parameter is ensured by the administrator.

\* importance filter (beginning and end of the range)

\* user defined filter (beginning and end of range).

Each incoming request /1/ client, if it has no unexecuted commands in the queue, is checked by the "waiting commands" groups filters, and those groups from which commands have been previously issued to this client must be excluded.

If such filters are found, the first filter is taken (if more than one is found), the command counter of this group is decreased by one, and then the command code and its parameter are added to the command queue of this client.

An important requirement: one client from each group of "waiting commands" can be given a command only once.

It is also necessary to provide functionality for managing groups of "waiting commands". Final functionality (stored procedures):

1. add a group

2. delete the group

3. get a list of groups (each group must have all the parameters in the list, plus the number of remaining teams in the group and the original number of teams in the group)

4. increase the number of commands (take care of atomicity of operation and conflict resolution when due to simultaneous operations of admin and backend the number became negative)

5. change the command parameter (but not the command code)

6. add country to filters (cannot be removed)

# 8 server management API.

The server on a separate port provides an API to control some server functions. The API port must be configured through the server config.

8.1 API access occurs

over the HTTPS protocol, regardless of the server port number. HTTP request can be either GET or POST. POST request is used if one of the API arguments is represented as a binary data block. Access control is performed by two parameters: apikey and apikeypass.

8.1.1 The format of the API request is as follows:

```/<apikey>/<apikeypass>/<function>?param\_name1=<param1>&param\_name2=<param2>.....```

If the function accepts some binary block of data, the request is made by POST, and the block of data is passed inside multipart/form-data with the name "bdata"

function - function name, the string is case-sensitive, can contain capital and small English letters and numbers

8.2 Each apikey has the following attributes:

a list of functions he is allowed to perform,

the range of ip-addresses from which he is allowed access.

The range is specified in CIDR format. For example, 1.2.3.4/24 - range from 1.2.3.0 to 1.2.3.255, 1.2.3.4/32 - single ip address, 1.2.3.4/0 - any ip address.

A list of all apikey and apikeypass with their settings (function list and ip range) must be specified through a special database table

8.2.1 The IP address of the request initiator is extracted from the "X-Forwarded-For" HTTP header. If this header is missing, the request should be discarded with an HTTP 403 code.

## 8.3 API Functions

### 8.3.1 GetGroupData - get group information.

Accepts one parameter: the time period in minutes.

The function should report activity by group for a specified period of time up to this point.

Data about the client are included in the report if he had activity in the specified period of time.

The group report is a text file. The format of the data by group is :

```<group> <client\_count> <first\_created>

\* group - group name

\* client\_count - the number of unique clients

\* first\_created - the time of the first registration of the earliest client in this group. Time format: epoch time.

Lines with group information are separated by a pair of "\r\n".

For example:

``qwerty1 151 1451278532

test 12 1358237562

test2 1005 1428237531

test111 100 1438257732``

The function if successful responds HTTP 200 c text file with group data, otherwise 403 (if parameters are not specified, wrong format, wrong apikey/apikeypass data or error somewhere).

### 8.3.2 UploadFile - uploading a file to the server.

Accepts parameters in the following order:

\* id - number (if a specific file is addressed for replacement)

\* file name (filename)

\* group -- group. (if not specified, "\*")

\* sys\_ver (if not specified, "\*")

\* country (if not specified, "\*")

\* client\_id (if not specified, 0)

\* importance\_low (if not specified, 0)

\* importance\_high (if not specified, 100)

\* userdefined\_low (if not specified, 0)

\* userdefined\_high (if not specified, 100)

\* priority (number)

\* bdata -- the contents of the file inside multipart/form-data

Before uploading a file to the database, the file priority is calculated as follows: calculate the maximum priority value in the table and add one.

The function answers HTTP 200 /1/ if successful, otherwise 403 (if parameters are not specified, wrong format, wrong apikey/apikeypass data or error somewhere).

### 8.3.3 UploadConfig - uploading config to the server.

\* id -- if you want to replace the link

\* version version of the config

\* group

\* country (if not specified, "\*")

\* client\_id (if not specified, 0)

\* importance\_low (if not specified, 0)

\* importance\_high (if not specified, 100)

\* userdefined\_low (if not specified, 0)

\* userdefined\_high (if not specified, 100)

\* sys\_ver

\* bdata -- the contents of the config inside multipart/form-data

The function answers HTTP 200 /1/ if successful, otherwise 403 (if parameters are not specified, wrong format, wrong apikey/apikeypass data or error somewhere).

### 8.3.4 UploadLink - uploading a link to a server.

\* id -- if you want to replace the link

\* expiry\_at -- unixtime (integer) - link expiry time

\* sys\_ver -- system version

\* group -- group

\* country -- country

\* client\_id (if not specified, 0)

\* importance\_low (if not specified, 0)

\* importance\_high (if not specified, 100)

\* userdefined\_low (if not specified, 0)

\* userdefined\_high (if not specified, 100)

\* bdata -- the full text of the link inside multipart/form-data

The function answers HTTP 200 /1/ if successful, otherwise 403 (if parameters are not specified, wrong format, wrong apikey/apikeypass data or error somewhere).

### 8.3.5 PushBack - adding a command to the client queue

\* cid - string client identifier. The part of the identifier that is after the dot. Например, 4D2873436FA2371F319DD55C147DC9B2

\* code - command code, number

\* param - command parameter, string. When passing through a link in GET, it can be in urlencode format

The function answers HTTP 200 /1/ if successful, if the cid does not exist - 404.

Otherwise 403 (if parameters are not specified, wrong format, wrong apikey/apikeypass data, or an error somewhere).

### 8.3.6 GetFilesList - get a list of all files with all parameters.

The function, if successful, responds with an HTTP 200 text/plain listing all the files that are in the file table.

One line - one file, the line separator is "\r\n". Intra-line parameter separator is tabulation (9).

Parameters follow in the following order: id, name, priority, group, geo, importance\_low, importance\_high, system, userdefined\_low, userdefined\_high.

Otherwise 403 (if parameters are not specified, wrong format, wrong apikey/apikeypass data, or an error somewhere).

### 8.3.7 DeleteFile - deleting a file

\* id - identifier of the file to be deleted

The function answers HTTP 200 /1/ if successful, if the id does not exist - 404. Otherwise 403 (if parameters are not specified, wrong format, wrong apikey/apikeypass data or error somewhere).

### 8.3.8 GetLastEventData - get the last data from the specified module and event

\* cid - string client identifier. The part of the identifier that is after the dot. Например, 4D2873436FA2371F319DD55C147DC9B2

\* module - module name

\* event - event name

Function in case of success replies HTTP 200 and octet-stream with content of data sent last time by specified module and event, if cid does not exist - 404, if there is no satisfying event (module, event) - 404, if there is event, but no data, then 204. Otherwise 403 (if parameters are not specified, wrong format, wrong apikey/apikeypass data or error somewhere).

### 8.3.9 GetOnline - get a list of all clients by last activity.

\* period - period of activity in seconds.

The function, if successful, responds with an HTTP 200 text/plain listing the string client IDs that were active during the specified time period.

One line - one identifier, the line separator is "\r\n". In the identifier specify only the part after the dot.

Otherwise 403 (if parameters are not specified, wrong format, wrong apikey/apikeypass data, or an error somewhere).

### 8.3.10 GetLastActivity - get the time of the last client activity

\* cid - string client identifier. The part of the identifier that is after the dot. Например, 4D2873436FA2371F319DD55C147DC9B2

The function in case of success responds with HTTP 200 and text/plain which contains EPOCH-time with last activity of the specified client as a string, if the cid does not exist - 404. Otherwise 403 (if parameters are not specified, wrong format, wrong apikey/apikeypass data or error somewhere).

### 8.4 All API activity must be logged.

The following information should be stored: date-time, apikey, search address, function. Store information only I successfully made queries, all the rest should be logged in the general text log of the server.

# 9 Client Log.

Each client should have its own activity log.

This activity log should contain information about all commands to the server except the /1/ command.

Information about all issued commands to the client should also be recorded.

If a command from commands\_idle groups has been issued, the commands\_idle group identifier must be stored.

The following information should be stored in the activity log:

\* date time

\* client\_id

\* command code, incoming or outgoing.

\* Additional information in the form of a string.

\*\* There will be no information for outgoing commands.

\*\* When you receive a command /10/, you must save the code of the executed command in this field.

\*\* If the command /1/ is not "idle", the code of the issued command must be stored in this field.

\*\* For commands\_idle this will be the identifier of the group.

\*\* For /63/ it will be the module name and ctl (with a space).

\*\* For /64/ it will be the module name and the event name and aux-tag (with a space).

\*\* For /23/ this will be the version of the issued config.

\*\* For /5/ this will be the id of the issued file.

\*\* For /25/ this will be the link text.

\*\* For /10/ this will be the identifier of the executed command (cmdid) and the result code.

# 10 Importance.

The Importance mechanism must be able to generate an integer value from 0 to 100 inclusive based on client related events.

## 10.1 Each event has Importance value modifiers; there are three: preplus, mul, postplus.

All are of type float with at least 4 decimal places and cannot take the value null and .

Preplus and postplus can take values from -100, up to and including 100, mul can take values from 0 to 100 inclusive.

The event also has a class and an additional parameter that depends on the class.

There can be any number of events with the same class, but no two events can have the same class and optional parameter.

For each client, the event can only work once.

After the event has triggered, the importance value is recalculated using the following formula:

newimportance = (oldimportance+preplus)\*mul+postplus

If newimportance turns out to be less than 0 or more than 100, it is assigned a value of 0 or 100, respectively.

The newimportance value is also rounded to an integer value using the standard rounding function before it is saved to the database.

Among other things, each client must have a parameter (flag) that prohibits the automatic change of the importance value.

By default, it's off.

The event list is global for all clients.

It should be changed through the RDBMS, not through configs.

When you delete an event from the list (table) with events, no recalculations take place.

The event list itself changes rarely (no more than once every 30 minutes), so it is allowed to cache the event list.

The regularity of the update must be set through the config.

## 10.2 In the current implementation, the server must support the following event classes

### 10.2.1 The class is "online". Command /0/ .

An additional parameter is the number of received commands /0/ . For example, the following events are specified:

id1 - online(1), preplus=xx, mul= xx, postplus=xx

....

id16 - online(4)), preplus=xx, mul= xx, postplus=xx

....

id23 - online(7)), preplus=xx, mul= xx, postplus=xx

Thus the first command received /0/ will trigger the event "id1",

at the second command received, the event "id1" will NOT work,

The third one won't work either,

and the fourth command received /0/ will trigger the event "id16",

at the fifth and sixth commands /0/ events "id1" and "id16" will not trigger,

the seventh command received /0/ will trigger event "id23".

If the parameter is 0 or null, it is treated as one.

### 10.2.2 The class is "age". Time after registration.

An additional parameter is the number of minutes since the first command /0/, which caused the client to register in the database.

The number of minutes is compared according to the operation "more or equal" (">=").

The event check only occurs when /0/ or /1/ is received.

For example, there are the following events:

id2- age(30), preplus=0, mul= 1.0, postplus=10

....

id15- age(90), preplus=0, mul= 0.2, postplus=0

Thus, when receiving the command /0/ or /1/ it turns out that 37 minutes have passed since the registration of the client, then the event "id2" is triggered.

Then if receiving /0/ or /1/ it turns out that 123 minutes have passed, then "id15" will be triggered.

As a result, the value of importance will be 2 (at the beginning importance is 0, then after "id2" it will be (0+0)\*1+10 = 10, and then after "id15" it will be (10+0)\*0.2+0=2)

It is also possible that at some point the /0/ or /1/ command was received and 201 minutes have passed since the client was registered,

then both events "id2" and "id15" will be triggered simultaneously. In this case we will get both events triggered at the same time, the same calculations and value importance = 2.

### 10.2.3 Class - "geo". Country.

An additional parameter is the name of the country, a string. The event check is performed only when the /0/ command is received.

### 10.2.4 The class is "devhash\_dup". There is another client that has exactly the same devhash.

The optional parameter is not used.

The event is only checked when the /0/ command is received. The event is triggered in case if when receiving /0/ command and saving it to the table it turns out that the exact same devhash is present in another client.

### 10.2.5 Class - "command\_complete". Successful execution of the outgoing command.

An additional parameter is the number of commands executed.

The event check is performed only when a command /10/ is received.

If the parameter is 0 or null, it is treated as one.

### 10.2.6 The class is "geo\_change".

Change country. There are no additional parameters.

The event check is only performed when the /0/ command is received.

## 10.3 For each triggered event in the client log (item 9) there must be a record with the ID of the triggered event

class online - individual counter

age class - global, time is counted from the time specified in the created\_at column

geo class - no counter needed

devhash\_dup class - no counter needed

command\_complete class - individual counter

## 11 Mechanism for auto addition of commands to the client

Implement a mechanism for adding commands to the client automatically when receiving event reports (command 64).

The auto-add trigger must respond to the module name, event name and info field, and when they occur, the server must add a command with the specified code and parameter.

The info field must be checked by a regular expression.

So, if the info field contains "qrrr45werty", then checking "r45" should give a positive result, while checking "^r45" should give a negative result.

Among other things, the trigger must contain an integer parameter that limits the minimum frequency of triggering of this trigger on each particular client.

The parameter is set in seconds.

If it is 0, there is no limit.

For example, if it is 300 seconds, then if this client triggered 180 seconds ago, then the command does not need to be placed in the queue.

The auto addition list must be managed through a table.

The table can be cached, the regularity of cache refreshing should be set through config in seconds.