Snap4City Sentinel

A tool to troubleshoot, monitor, perform test, gather logs from on and certificate your **Snap4City** distribution

User manual + developer notes

Version 0.7, 2024-07-30

Index

Installing and Starting Snap4Sentinel

To run Snap4Sentinel, you require Python 3.8 or greater. There are also a number of python packages which must be installed/enabled in your environment.

1. Subprocess: Run subprocesses
2. Flask: Make a web service
3. Requests: Perform http(s) requests with Python
4. Mysql.connector: Connect to the database (note that many similar packages to this exists, be careful on what you are downloading)
5. Json: Handle JSON documents
6. Os: Imports some OS operations, like folder creation
7. ReportLab: Generates pdf documents.
8. Smtplib: Handles SMTP protocol
9. Email: Sends email
10. Telegram: Handles Telegram bots
11. Asyncio: Handles concurrent programming
12. Apscheduler: Schedules operations (like cron)
13. Base64: Decodes and Encodes text
14. Random: Randomness for passwords
15. String: String operations
16. Traceback: Retrieves the stack for error troubleshooting
17. Urllib: Operations on urls
18. Datetime: Handles time and date
19. Waitress: Production server for Flask

You need to set up the database (the service is able to detect a database not properly set up). An automatic attempt of setting up can be attempted, but a copy of the database schema (and some initial data) is left in the installation folder anyway.
By default, Snap4Sentinel uses the same database as the one used inside Snap4City (does not interact with any schema except its own); this decreases the number of services required on the host but in turns makes it impossible for Snap4Sentinel to properly work if the Snap4City database cannot be reached for any reason (while trying to fire alerts anyway).

To start Snap4Sentinel, run

```
waitress-serve --port 4081 --call flask_app:create_app
```

in the console where Snap4Sentinel is installed. This command exposes Snap4Sentinel to the port 4081, which is expected to be the default one.
If you wish to start it as a background process, instead run

```
nohup waitress-serve --port 4081 --call flask_app:create_app >> demotec-log.txt 2>&1 &
```

Configuring Snap4Sentinel

To configure Snap4Sentinel, you must edit the file conf.json found in the installation path. A few fields are ready as is, but you may edit them according to your needs. You need to restart Snap4Sentinel to apply edits.

```
{
    "admin-log-length": 200,
    "default-log-length": 1000,
    "requests-timeout": 15000,
    "is-master": true,
    "telegram-api-token": "replaceme",
    "telegram-channel":0,
    "db-user": "root",
    "db-passwd": "admin",
    "db-host": "localhost",
    "db-port": 3306,
    "smtp-server": "change.me.com",
    "smtp-port":587,
    "sender-email": "change@at.me",
    "sender-email-password": "replacethispassword",
    "email-recipients": ["first_address@to.besent", "second_address@to.besent",
"more_addresses@to.besent"],
    "platform-url": "$#base-url#$"
}
```

- admin-log-length: How many items will be shown in the administrator logs. Full logs can be accessed in the database at any time.
- default-log-length: How many lines of logs will be extracted from the Docker logs.
- requests-timeout: How many milliseconds will the client wait before timing out requests.
- is-master: Determines if the instance of Snap4Sentinel is a master in its cluster.
- telegram-api-token: If you have a Telegram Bot, you will have access to its token, and you may use it to send Telegram messages as alerts.
- telegram-channel: The channel or chat where to send a message. You must edit this field if you want to send telegram alerts.
- db-user, db-password, db-host, db-port: These fields are populated from the Snap4City configuration, with the database shared with the platform. The password won't be "admin", but in any case, use a secure password.
- smtp-server: The provider of the email address you wish to send email alerts from.
- smtp-port: The port used to send emails. Usually 587.
- sender-email: The email address which will be sending the emails.
- sender-email-password: The password of the above mentioned email.
- email-recipients: The list of addresses which will receive the alerts.
- platform-url: The base url of your Snap4City distribution.

Log into Snap4Sentinel

By default, Snap4Sentinel is placed inside the Snap4City installation, behind the NGINX proxy server. It can be reached at http(s)://yoursnap4cityinstallation/sentinel, where *http(s)://yoursnap4cityinstallation* changes depending on your installation. To access the webpage, you have to insert the Snap4Sentinel credentials. The username for the administrator account is “admin” and the password is generated with the rest of the Snap4City installation. You can recover the original password from the file placeholder_used.txt. Any other user lacks administration permissions.

The accounts are memorised in the file .htpasswd of the proxy; the names are in clear text, and the relative passwords are hashed and not recoverable from there. You might edit the file to add other users or change their passwords; note however that the username “admin” is the only account with admin privileges. Your session ends, meaning you need to re enter your credentials, by closing your browser, by clicking the logout button under the menu “Show administrative actions”, or by getting a 401 Unauthorised error.

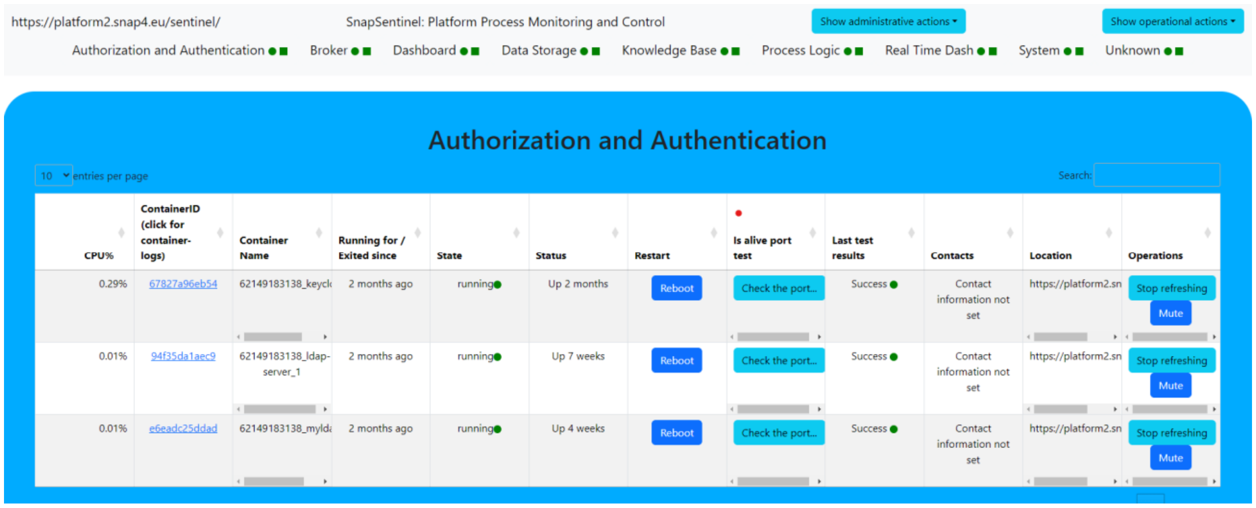
Muting components

It is possible to mute a component if the administrator chooses so. If that happens, the container having issues won’t cause to send Telegram notifications (unless another unmuted container would trigger a notification). Emails will be sent regardless. To mute a component, the administrator, through the main interface, selects the container then clicks on the Mute button; then, after picking a duration and confirming with a password, the component will be muted for the amount of time. Muting a container while it is already muted will reset the mute duration to the newest one. A container already muted is marked as muted by the mute button.

Supervising the platform with Snap4Sentinel

Basic Usage

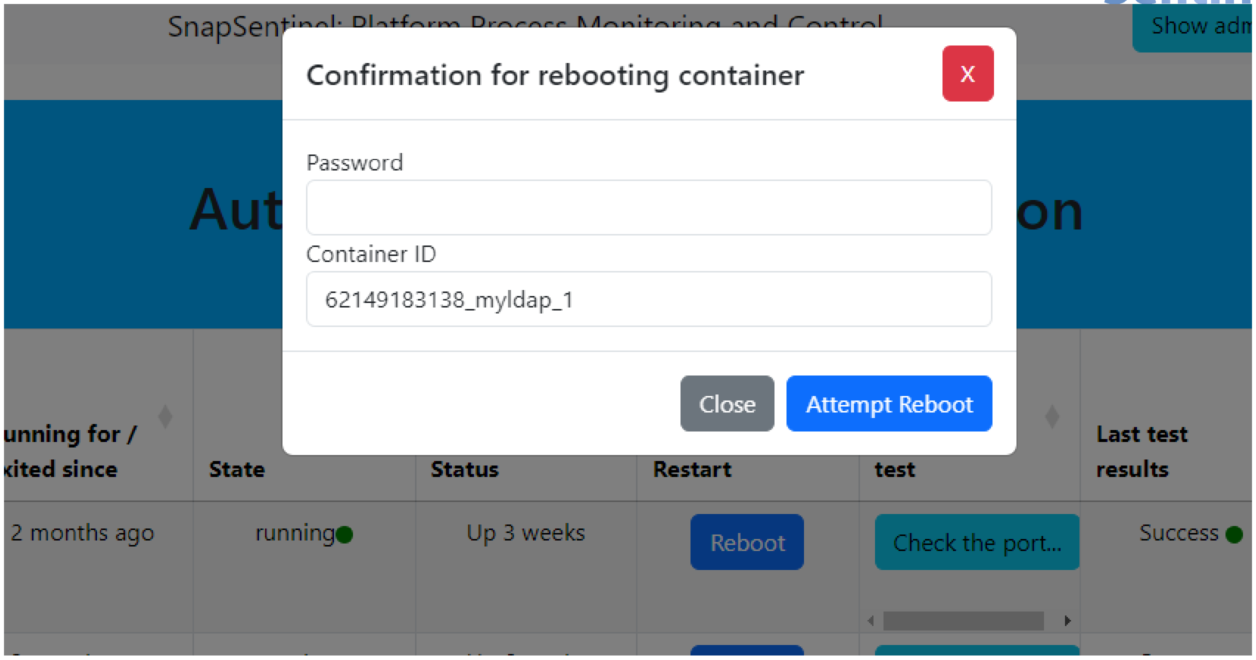
By visiting the main web page of Snap4Sentinel you can see the status of all the containers of the cluster, divided in multiple categories. These categories are defined in the database, and each container belongs to a given category with an entry in the database. Normally, the system provides an amount of information which is overwhelming to a novice user; by default, Snap4Sentinel reduces the data gathered by showing the most relevant columns. This can be toggled by accessing the menu "Show operational actions" on the top navbar and clicking on "Show additional columns" (The button changes its text to reflect the toggling operation).
The summary of each category of components, shown by accessing the "Toggle Categories" menu in the top bar, provides a short description of the current status. A green dot indicates that the category has all of its components in the intended execution status (usually running). A red dot indicates that at least one component is not in the intended execution status. There are also green and red squares; they will be explained in the "*Perform a "is alive" test*" category.


https://platform2.snap4.eu/sentinel/
SnapSentinel: Platform Process Monitoring and Control
Show administrative actions
Show operational actions
Authorization and Authentication ● ■
Broker ● ■
Dashboard ● ■
Data Storage ● ■
Knowledge Base ● ■
Process Logic ● ■
Real Time Dash ● ■
System ● ■
Unknown ● ■
Authorization and Authentication
10 entries per page
Search:

CPU%	ContainerID (click for container-logs)	Container Name	Running for / Exited since	State	Status	Restart	● Is alive port test	Last test results	Contacts	Location	Operations
0.29%	67827a96eb54	62149183138_keycl	2 months ago	running●	Up 2 months	Reboot	Check the port...	Success ●	Contact information not set	https://platform2.sn	Stop refreshing Mute
0.01%	94f35da1aec9	62149183138_ldap-server_1	2 months ago	running●	Up 7 weeks	Reboot	Check the port...	Success ●	Contact information not set	https://platform2.sn	Stop refreshing Mute
0.01%	e6eadc25ddad	62149183138_mylda	2 months ago	running●	Up 4 weeks	Reboot	Check the port...	Success ●	Contact information not set	https://platform2.sn	Stop refreshing Mute

Restarting a container

Each container, on its row on Snap4Sentinel, shows a button to restart the container. Given that restarting a container may result in service outage, a continuation popup will open to confirm the action. To further prevent accidental restarts, the operation must be completed by inserting the password of the user operating the instance of Snap4Sentinel. **Inserting the wrong password logs you out of the service immediately.** You can abort the operation by closing the window.



SnapSentinel: Platform Process Monitoring and Control

Confirmation for rebooting container

Password

Container ID

62149183138_myldap_1

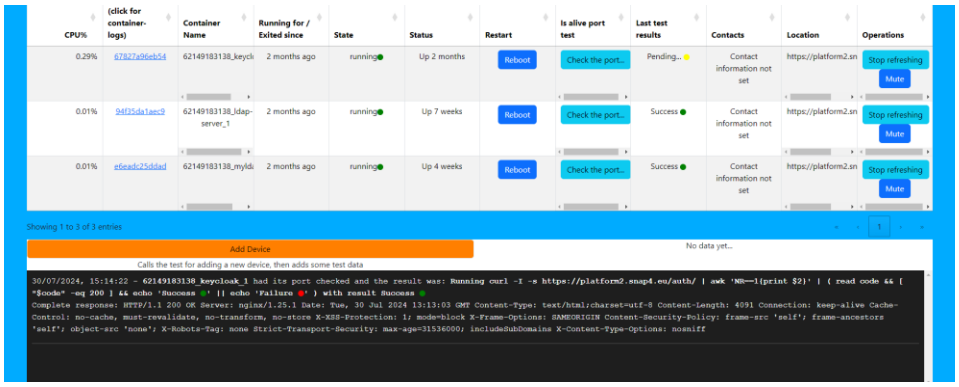
unning for / xited since	State	Status	Restart	test	Last test results
2 months ago	running●	Up 3 weeks	Reboot	Check the port...	Success ●

Assuming you wish to restart a container, there are a few notes to be aware of (besides being potentially responsible for service outages).

1. If you restart the proxy container, you will temporarily lose connection to the backend, as the proxy is what allows you to communicate with said backend.
2. If you restart the mysql database container, assuming it is where Snap4Sentinel holds its data, Snap4Sentinel will be temporarily unable to register events.
3. By restarting a container, there is a time during which the container isn't running. If that period of time matches the periodic backend health checks, it results in alerts being sent. On the client side, the component is temporarily set as "yellow", to indicate that the system isn't properly working at the current time, but it may resume doing so shortly.
4. It is possible for some containers to take a lot of time to be restarted. In such cases, the http(s) request to restart the container goes into timeout, but **that doesn't mean that the backend stopped trying to restart the container**. It is important to monitor the situation, but assuming there are no underlying issues, eventually the container will resume its running status. If there are issues, Snap4Sentinel will make you know about them.
5. The client is periodically updated to reflect the current state of the Snap4City installation, but a restart status isn't necessarily followed in real time; on the client, a component may be shown restarting even if the restart has already happened. Manually refreshing the containers shows the current status of the containers.

Perform a “is alive” test

Most components must provide a way to prove that they are working properly (those who don’t say that they can’t be tested). To do so, a button for each container is shown to perform such a test. The results of these tests usually result in either failure or success, along with a longer explanation of the command. The results of these tests are both shown at the end of the category of the component and in the associated column of the container. If a category has a test that resulted in a failure, its status will display a red square. Otherwise, a green square is displayed.



CPU%	(click for container-logs)	Container Name	Running for / Exited since	State	Status	Restart	Is alive port test	Last test results	Contacts	Location	Operations
0.29%	67827a96eb54	62149183138_keycl	2 months ago	running●	Up 2 months	Reboot	Check the port...	Pending... ●	Contact information not set	https://platform2.sn	Stop refreshing Mute
0.01%	94f35da1aec9	62149183138_ldap-server_1	2 months ago	running●	Up 7 weeks	Reboot	Check the port...	Success ●	Contact information not set	https://platform2.sn	Stop refreshing Mute
0.01%	e6eadc25ddad	62149183138_myld	2 months ago	running●	Up 4 weeks	Reboot	Check the port...	Success ●	Contact information not set	https://platform2.sn	Stop refreshing Mute

Showing 1 to 3 of 3 entries

Add Device

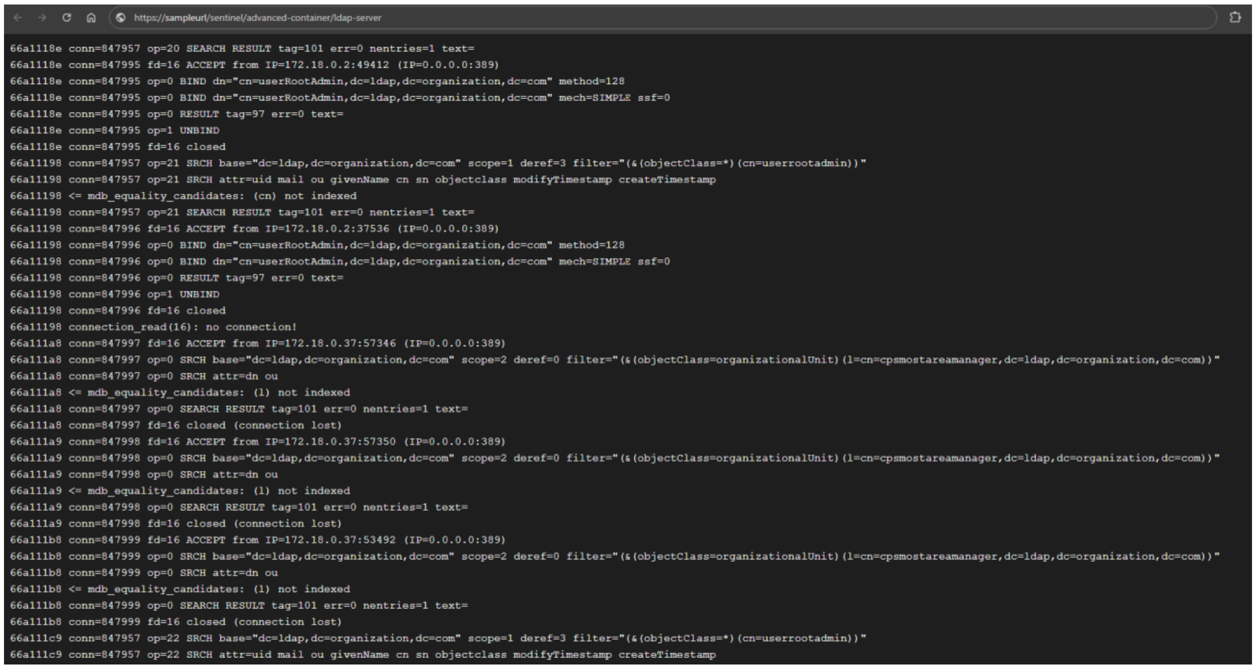
Calls the test for adding a new device, then adds some test data

No data yet...

```
30/07/2024, 15:14:22 - 62149183138_keycloak_1 had its port checked and the result was: Running curl -I -s https://platform2.snap4.eu/auth/ | awk 'NR==1{print $2}' | ( read code && [
"$code" -eq 200 ] && echo 'Success ●' || echo 'Failure ●' ) with result Success ●
Complete response: HTTP/1.1 200 OK Server: nginx/1.25.1 Date: Tue, 30 Jul 2024 13:13:03 GMT Content-Type: text/html;charset=utf-8 Content-Length: 4091 Connection: keep-alive Cache-
Control: no-cache, must-revalidate, no-transform, no-store X-XSS-Protection: 1; mode=block X-Frame-Options: SAMEORIGIN Content-Security-Policy: frame-src 'self'; frame-ancestors
'self'; object-src 'none'; X-Robots-Tag: none Strict-Transport-Security: max-age=31536000; includeSubDomains X-Content-Type-Options: nosniff
```

Check the docker logs of a container

In order to see the logs of a docker container, Snap4Sentinel exposes webpages to see the logs. By default it shows the last 1000 lines; you might increase or decrease this amount by editing the configuration file. To access these webpages, you only need to click on the containerID of the container you wish to inspect. The webpage doesn’t update itself, to see the newer result, reload the webpage. The following is an example of logs being shown.

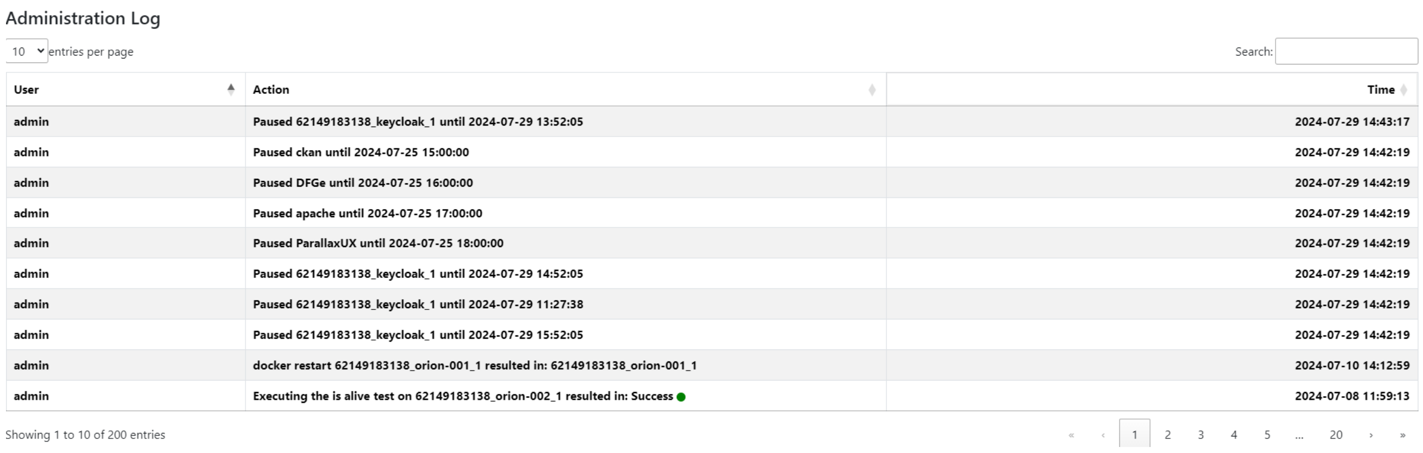


https://sampleurl/sentinel/advanced-container/ldap-server

```
66a1118e conn=847957 op=20 SEARCH RESULT tag=101 err=0 nentries=1 text=
66a1118e conn=847995 fd=16 ACCEPT from IP=172.18.0.2:49412 (IP=0.0.0.0:389)
66a1118e conn=847995 op=0 BIND dn="cn=userRootAdmin,dc=ldap,dc=organization,dc=com" method=128
66a1118e conn=847995 op=0 BIND dn="cn=userRootAdmin,dc=ldap,dc=organization,dc=com" mech=SIMPLE ssf=0
66a1118e conn=847995 op=0 RESULT tag=97 err=0 text=
66a1118e conn=847995 op=1 UNBIND
66a1118e conn=847995 fd=16 closed
66a11198 conn=847957 op=21 SRCH base="dc=ldap,dc=organization,dc=com" scope=1 deref=3 filter="(&(objectClass=*)(cn=userrootadmin))"
66a11198 conn=847957 op=21 SRCH attr=uid mail ou givenName cn sn objectclass modifyTimestamp createTimestamp
66a11198 <= mdb_equality_candidates: (cn) not indexed
66a11198 conn=847957 op=21 SEARCH RESULT tag=101 err=0 nentries=1 text=
66a11198 conn=847996 fd=16 ACCEPT from IP=172.18.0.2:37536 (IP=0.0.0.0:389)
66a11198 conn=847996 op=0 BIND dn="cn=userRootAdmin,dc=ldap,dc=organization,dc=com" method=128
66a11198 conn=847996 op=0 BIND dn="cn=userRootAdmin,dc=ldap,dc=organization,dc=com" mech=SIMPLE ssf=0
66a11198 conn=847996 op=0 RESULT tag=97 err=0 text=
66a11198 conn=847996 op=1 UNBIND
66a11198 conn=847996 fd=16 closed
66a11198 connection_read(16): no connection!
66a111a8 conn=847997 fd=16 ACCEPT from IP=172.18.0.37:57346 (IP=0.0.0.0:389)
66a111a8 conn=847997 op=0 SRCH base="dc=ldap,dc=organization,dc=com" scope=2 deref=0 filter="(&(objectClass=organizationalUnit)(l=cn=cpsmostareamanager,dc=ldap,dc=organization,dc=com))"
66a111a8 conn=847997 op=0 SRCH attr=dn ou
66a111a8 <= mdb_equality_candidates: (l) not indexed
66a111a8 conn=847997 op=0 SEARCH RESULT tag=101 err=0 nentries=1 text=
66a111a8 conn=847997 fd=16 closed (connection lost)
66a111a9 conn=847998 fd=16 ACCEPT from IP=172.18.0.37:57350 (IP=0.0.0.0:389)
66a111a9 conn=847998 op=0 SRCH base="dc=ldap,dc=organization,dc=com" scope=2 deref=0 filter="(&(objectClass=organizationalUnit)(l=cn=cpsmostareamanager,dc=ldap,dc=organization,dc=com))"
66a111a9 conn=847998 op=0 SRCH attr=dn ou
66a111a9 <= mdb_equality_candidates: (l) not indexed
66a111a9 conn=847998 op=0 SEARCH RESULT tag=101 err=0 nentries=1 text=
66a111a9 conn=847998 fd=16 closed (connection lost)
66a111b8 conn=847999 fd=16 ACCEPT from IP=172.18.0.37:53492 (IP=0.0.0.0:389)
66a111b8 conn=847999 op=0 SRCH base="dc=ldap,dc=organization,dc=com" scope=2 deref=0 filter="(&(objectClass=organizationalUnit)(l=cn=cpsmostareamanager,dc=ldap,dc=organization,dc=com))"
66a111b8 conn=847999 op=0 SRCH attr=dn ou
66a111b8 <= mdb_equality_candidates: (l) not indexed
66a111b8 conn=847999 op=0 SEARCH RESULT tag=101 err=0 nentries=1 text=
66a111b8 conn=847999 fd=16 closed (connection lost)
66a111c9 conn=847957 op=22 SRCH base="dc=ldap,dc=organization,dc=com" scope=1 deref=3 filter="(&(objectClass=*)(cn=userrootadmin))"
66a111c9 conn=847957 op=22 SRCH attr=uid mail ou givenName cn sn objectclass modifyTimestamp createTimestamp
```

Administration Logs

If you logged into Snap4Sentinel as the administrator (admin), at the end of the webpage (and upon logging in at its beginning, for a brief period of time) you are able to see a table showing the last operations performed with Snap4Sentinel. The table shows each user’s operations, including your own. It can provide help to you by troubleshooting scenarios where some users performed unexpected operations. The table merges the information of the database tables *test_ran* and *rebooting_containers*. By default it loads only the latest 200 operations (chronologically); this number can be increased by editing the configuration json.



Administration Log

10 entries per page Search:

User	Action	Time
admin	Paused 62149183138_keycloak_1 until 2024-07-29 13:52:05	2024-07-29 14:43:17
admin	Paused ckan until 2024-07-25 15:00:00	2024-07-29 14:42:19
admin	Paused DFGe until 2024-07-25 16:00:00	2024-07-29 14:42:19
admin	Paused apache until 2024-07-25 17:00:00	2024-07-29 14:42:19
admin	Paused ParallaxUX until 2024-07-25 18:00:00	2024-07-29 14:42:19
admin	Paused 62149183138_keycloak_1 until 2024-07-29 14:52:05	2024-07-29 14:42:19
admin	Paused 62149183138_keycloak_1 until 2024-07-29 11:27:38	2024-07-29 14:42:19
admin	Paused 62149183138_keycloak_1 until 2024-07-29 15:52:05	2024-07-29 14:42:19
admin	docker restart 62149183138_orion-001_1 resulted in: 62149183138_orion-001_1	2024-07-10 14:12:59
admin	Executing the is alive test on 62149183138_orion-002_1 resulted in: Success ●	2024-07-08 11:59:13

Showing 1 to 10 of 200 entries

« ‹ 1 2 3 4 5 … 20 › »

Generating a certification

A certification is the sum of all the files which define the current state of the Snap4City installation. The administrator of Snap4Sentinel may produce a certification by accessing the *Perform administrative actions* menu in the web interface (other users are unable to perform the operation). The result of the operation is an archive which holds the various configurations of all the relevant components of the Snap4City installation The archive is immutable and protected by a password, and the password is sent as part of the name of the file. The operation can be long and may result in a timeout of the connection; in such cases, you can circumvent the issue by extending the timeouts of the requests in the Nginx proxy (and then reloading it). Note that if a request has timed out it doesn’t mean the server won’t try to complete it; you just won’t receive the response.

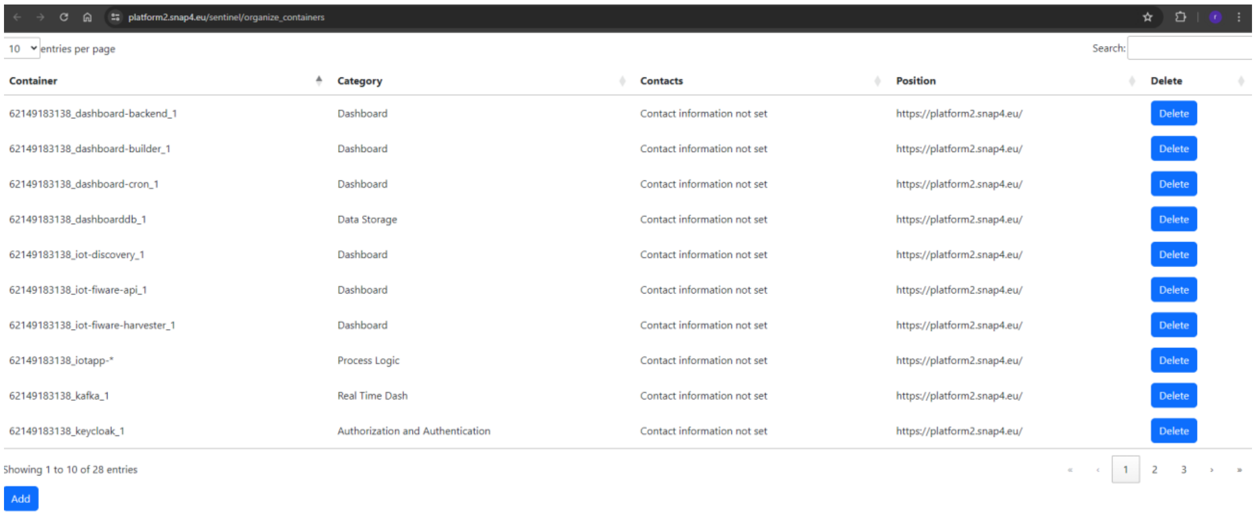
Generating Full Logs

Snap4Sentinel is able to generate multiple pdf reports with the information of all the containers of your Snap4City distribution, all of the lasts is alive for each container, all the additional tests and then a few additional log files (as long as each relevant host is exposing a Snap4Sentinel endpoint). Due to the amount of data the tool is exposed too, only the last 1000 lines of each individual log are generated. Even then, the amount of data might result in a timeout for the http(s) request. If you find this happening often, you might consider extending the timeout time inside NGINX to accommodate the generation of the archive holding the pdf(s). The downloaded archive is not protected by any password and consists of one pdf for each host holding a Snap4City installation, partial or complete.
Each individual pdf holds the information mentioned about, with a table of content at the beginning of the document. A document over 500 pages is common, so a big file is not of concern.
To generate the complete log, click on the button Generate full logs under the menu “Show administrative actions”, then for the download.
All of the previous generated logs are also left in folders with the respective creation date in the path of the Snap4Sentinel installation.

Adding new components components

You are able to add new containers to your Snap4Sentienl installation by using the Manage Containers interface, which can be reached in the Administration Operations submenu, as an administrator. With that interface you may add or remove new or old containers. For a new container you must provide all of the additional information, with the “Position” field indicating the url of the respective Snap4Sentinel local installation. Additions and deletions are effective immediately.

As for Version 0.6, to edit a container you must delete it first then recreate it updated. Any previous data is not lost, but won’t be associated with components or categories until a new container with the same name as the original is created.



platform2.snap4.eu/sentinel/organize_containers

10 entries per page

Search:

Container	Category	Contacts	Position	Delete
62149183138_dashboard-backend_1	Dashboard	Contact information not set	https://platform2.snap4.eu/	Delete
62149183138_dashboard-builder_1	Dashboard	Contact information not set	https://platform2.snap4.eu/	Delete
62149183138_dashboard-cron_1	Dashboard	Contact information not set	https://platform2.snap4.eu/	Delete
62149183138_dashboarddb_1	Data Storage	Contact information not set	https://platform2.snap4.eu/	Delete
62149183138_iot-discovery_1	Dashboard	Contact information not set	https://platform2.snap4.eu/	Delete
62149183138_iot-fiware-api_1	Dashboard	Contact information not set	https://platform2.snap4.eu/	Delete
62149183138_iot-fiware-harvester_1	Dashboard	Contact information not set	https://platform2.snap4.eu/	Delete
62149183138_iotapp-*	Process Logic	Contact information not set	https://platform2.snap4.eu/	Delete
62149183138_kafka_1	Real Time Dash	Contact information not set	https://platform2.snap4.eu/	Delete
62149183138_keycloak_1	Authorization and Authentication	Contact information not set	https://platform2.snap4.eu/	Delete

Showing 1 to 10 of 28 entries

The system is able to resolve names like ***some-prefix-**** to each instance of containers which begin with ***some-prefix-***; in short, multiple similarly named containers can be defined with a single entry. You need to assign the containers to an existing category (creating a new one if required, by adding it to the table *categories*).

On the nature of clusterification of Snap4Sentinel

In a Snap4City cluster, it is expected to have a local Snap4Sentinel service installed to gather the data for each host. One of these Snap4Sentinel instances is delegated as the master, and will be able to call the other instances of Snap4Sentinel to ask them about their local conditions. The master aggregates the results of the status requests and seamlessly shows them to the user with the web UI. To do so, each container entry in the database must describe to Snap4Sentinel where it is located by giving a host url to the master; this information is also shown in the table of the containers, by the column “Location”. The master’s host must be able to resolve such hostnames (an ip is also fine); the hostname is not required to be accessible from the internet. If the master’s host is holding some of the containers, it results in trying to ask itself the data it requires (the call isn’t recursive, and it won’t loop forever).

In short, the “clusterification” resolves to being a delegation of the functionalities to just each host for its own, then the master gathers the results. This implementation makes it trivial to add more hosts or move components around.

Backend endpoints

Snap4Sentinel exposes a number of endpoints to extract resources from the local status of the hosts. Here is a list and a brief explanation of these endpoints:

- /get_data_from_source: Have the server make a get request to the webpage found in the request argument “url_of_resource”, then return the results. Answers to a GET only.
- /get_complex_test_buttons: Returns the list of complex tests and their eventual parameters.
- /container_is_okay: Sets the summary status of the *category* in the POST argument to “green”. Does not prevent it from changing it in the future.
- /read_containers: Returns the status of the containers of that specific host. Answers to a GET only.
- /advanced_read_containers: Returns the status of the containers of the whole cluster by calling each /read_containers of the cluster. Answers to a GET only.
- /get_container_categories: Returns the content of the table *component_to_category*. Answers to a GET only.
- /run_test: Runs the “is alive” test of the container in the POST argument *container* and returns the result of the test.
- /run_test_complex: Runs a complex test named as the POST argument *test_name* and returns the result.
- /test_all_ports: Runs each “is alive test” and returns their results. Answers to a GET only.
- /deauthenticate: If it is called to a master of its cluster, it will deauthenticate the user. Otherwise, the operation fails. Answers to GET and POST.
- /reboot_container: Tries to restart a container named as the POST argument *id* which is found in the specific host. Fails if not authenticated. Answers to POST only.
- /reboot_container_advanced/<container>: Tries to restart a container named as <container> across the cluster. Fails if not authenticated. Answers to POST only.
- /tests_results: Returns the last test of each container, chronologically. Answers to a GET only.
- /load_db: Loads the database from a local file. Answers to POST only.
- /get_complex_tests: Same as /tests_results
- /container/<container_id>: Returns the logs of the local container <container_id>.
- /advanced-container/<container_id>: Returns the logs of the container <container_id> in the cluster.
- /get_summary_status: Returns the statuses of the categories from the table *summary_status*.
- /generate_clustered_pdf: Returns an archive containing all of the pdf reports of the cluster. Answers to a GET only.
- /generate_pdf: Returns a pdf report of the local containers and tests. Answers to a GET only.
- /certification: Generates a local certification and returns an archive. Answers to a GET only.
- /clustered_certification: Generates a clusterwide certification and returns an archive.

Answers to a GET only.

Configuring tests and “are alive”

Creating a “Is alive” test

“Are alive” tests are located in the table *tests_table*. To create or edit such a test, you need to know first which kind of answer do you expect from a healthy service and how can you get the container to give you said answer. After you can perform this operation with CLI commands, you must also implement the logic behind determining if a test was successful or not.

In the database, you can find preexisting examples of such tests. The first column is the ID, the second column determines which container belongs to the given test. The third column is a command whose result are printed in the web UI, and the fourth column is supposed to represent the result of the test without the logic to understand if it succeeded.

Creating a complex test

The database table *complex_test* stores the complex tests. A complex test is a generic code execution which involves multiple components or a non binary response. A complex test belongs to a category; such a relation is defined in the database table category test, where the column *test* is the id of the test and the column *category* is the id of the category.

In tha table *complex_test*, the first column is the id, the second column represents the text which is shown to the user as the button to run the test, the third column represents the bash command to be executed (note that given the likely complexity of the test, it is better to call a script instead of just having the commands in the database, for the sake of brevity), the 4th column represents optional from data which may be added to a test to take input from the user in the web interface. An example is as follows.

```
data_type_html:name_shown_to_the_user:letter_used_as_console_parameter
```

You may have multiple of these inputs by separating them with a semicolon (;). The fifth column represents the rgb colour of the button in the user web interface; the colour of the text is computed automatically. The sixth column is reminder text shown to the user to explain what the complex test does.

Firing alerts from the backend

The backend runs sanity checks on the containers, on startup and periodically. The containers which are going to be checked are found in the table *component_to_category*; by default is all of the Snap4City distribution and any other container is ignored. If a container is found not to be its expected status, the backend takes notice of that. Then, all “are alive” tests are executed, and the backend takes notice of each test that failed as well. Once the previous operations are completed, if any unexpected status is found or if any “is alive” test has failed, the backend sends an email to all the recipients listed in the configuration files, then it also sends the same alert with a Telegram Bot to a channel specified in the configuration.
At 08:00 AM and 08:00 PM an additional alert is sent, using the same channels as above, to signal that the platform is alive. Note that hours and time refer to the hosting machine, so they may be sent at a different time if you don’t share the timezone with the host (or +/-12 hours).

Starting the service with a crontab

To edit your crontab with the console, type “crontab -e” and then press enter. You may be asked which cli text editor you want to use; pick your favourite. Then, add the following lines (editing it first if you need to tweak it).

```
10 1 * * * cd /path/to/your/installation; PID=$(lsof -ti :4081); if [ ! -z "$PID" ]; then kill $PID; else echo "No process found on port 4081" >> sentinel-log.txt; fi; nohup waitress-serve --port 4081 --call flask_app:create_app >> sentinel-log.txt 2>&1 &
```

Explanation: each day, at 01:10 AM, check if the port 4081 is used by a process (port 4081 is the port used by default by Snap4Sentinel; change it accordingly to your installation). If there are no processes using the port, it means that Snap4Sentinel is not running, and it writes so in the log. If there is a process using the port, then that process is killed. In either case, the Snap4Sentinel process is started in the background, listening on port 4081, redirecting both stdout and stderr into the file sentinel-log.txt, which functions as a log.

```
*/5 * * * * cd /path/to/your/installation; PID=$(lsof -ti :4081); if [ ! -z "$PID" ]; then echo "Don't need to restart sentinel" >> sentinel-log.txt; else echo "No process found on port 4081"; nohup waitress-serve --port 4081 --call flask_app:create_app >> sentinel-log.txt 2>&1 &; fi;
```

Explanation: each time every 5 minutes, check if the port 4081 is used by a process (port 4081 is the port used by default by Snap4Sentinel; change it accordingly to your installation). If there are no processes using the port, it means that Snap4Sentinel is not running, writes so in the log then the Snap4Sentinel process is started in the background, listening on port 4081, redirecting both stdout and stderr into the file sentinel-log.txt, which functions as a log. If instead there is a process using the port, then it is logged that it is not necessary to turn the service on.