Execution of the service

To run the app, use the following command in the CLI in the folder of this readme -> `waitress-serve --port 4080 --call flask_app:create_app`

You might change the port to something else if you want, it doesn't matter as long as you don't pick something else in use

The current db on this machine is listening on 3344
you might also change it; but don't use the standard 3306; the snap4city database container already uses it and there will be a conflict

You need to change the port in the flask_app.py file, there is one single instance

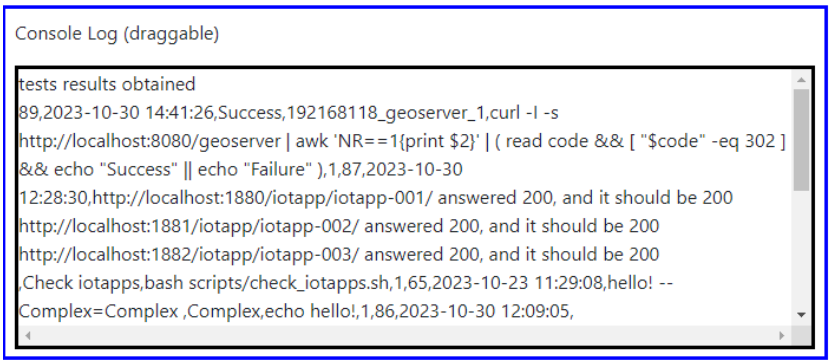
They all say Success or Failure

For Kafka you will need to install a package -> `apt-get install kafkacat`; you might need admin privileges to install it

If this doesn't run be sure to update Python (3.9 minimum), Flask, json, waitress and mysql-connector-python for python (use apt-get and pip)

You will also need docker (not sure which one is enough but 20.whatever wasn't good and 23 was) (again, use apt-get)

```
debian@debian:~/checker-app$ waitress-serve --port 4080 --call flask_app:create_app
INFO:waitress:Serving on http://0.0.0.0:4080
```



Disk Data I/0	CPU%	ContainerID	Image	Mem%	Mem Used / Mem Max	Container Name	Networl
16.9MB / 7.33MB	0.68%	33a8aab0e853	kartoza/geoserver:2.20.0	5.53%	1.737GiB / 31.41GiB	192168118_geoserver_1	55.1MB ,
99.3MB / 74.1MB	19.12%	c676420eda97	apache/nifi:1.16.2	3.47%	1.091GiB / 31.41GiB	192168118_nifi_1	59.5MB , 10.1kB

Using the service

The service opens the 4080 port (or whichever you picked if you changed it earlier) and you can reach it by going to http://localhost:4080/

The top part shows a console which outputs logs (which you can drag around) and a checkbox allowing for automatic refreshes (once every 15 seconds). (2)

Logs are the sum of the standard out and the standard error; errors are shown in red. (1)

The timing of the updates isn't synchronised to toggling the checkbox so any change might take less than 15 seconds.

Then you will see the list of the containers running on the machine; the list is taken in real time and lists all containers running, even those which have nothing to do with Snap4City

Of course if no functionality is set for a given container, then the button will not work even if present

You might click a container ID (third column) to see the logs of the chosen container; it will open a new tab in the browser (3)

The log window doesn't update; if you need to see the newest logs, just refresh the webpage

The last columns of the table are the button for restarting the container, the button for checking if a container is alive/healthy and the latter shows the result of the previous operation (if any) (4,5,6)

After this table, there is an addition table which shows a list of operations which are activated by clicking the relative button

Some operations require additional data; said data can be inserted in the input fields below the button.

The results of the operations are shown on the second column; it follows the same format of the logs.

Stop containers refresh 2

Console Log (draggable) 1

```
tests results obtained
89,2023-10-30 14:41:26,Success,192168118_geoserver_1,curl -I -s
http://localhost:8080/geoserver | awk 'NR==1{print $2}' | ( read code && [ "$code" -eq 302 ]
&& echo "Success" || echo "Failure" ),1,87,2023-10-30
12:28:30,http://localhost:1880/iotapp/iotapp-001/ answered 200, and it should be 200
http://localhost:1881/iotapp/iotapp-002/ answered 200, and it should be 200
http://localhost:1882/iotapp/iotapp-003/ answered 200, and it should be 200
,Check iotapps,bash scripts/check_iotapps.sh,1,65,2023-10-23 11:29:08,hello! --
Complex=Complex ,Complex,echo hello!,1,86,2023-10-30 12:09:05,
```

Disk Data I/0	CPU%	3 ContainerID	Image	Mem%	Mem Used / Mem Max	Container Name	Network I/O	Ports	Running for / Exited since	Size	Status	4 Reboot	5 Is alive test	6 Last tests results
16.9MB / 7.33MB	0.68%	33a8aab0e853	kartoza/geoserver:2.20.0	5.53%	1.737GiB / 31.41GiB	192168118_geoserver_1	55.1MB / 0B	8443/tcp, 0.0.0.0:8600->8080/tcp, :::8600->8080/tcp	3 weeks ago	0B	running	Reboot	Run tests...	Success
99.3MB / 74.1MB	19.12%	c676420eda97	apache/nifi:1.16.2	3.47%	1.091GiB / 31.41GiB	192168118_nifi_1	59.5MB / 10.1kB	8000/tcp, 8080/tcp, 0.0.0.0:1030->1030/tcp, :::1030-	3 weeks ago	0B	running	Reboot	Run tests...	No tests found

14.4MB / 3.71GB	0.36%	bd204fb05836	mongo:3.6	0.43%	137MiB / 31.41GiB	192168118_mongo-001_1	96.7MB / 43.4MB	0.0.0.0:27017->27017/tcp, :::27017->27017/tcp	6 weeks ago	0B	running	Reboot	Run tests...	No tests found

Run command	Command Result
Run setup.sh Calls setup.sh of Snap4City	
Run post-setup.sh Calls post-setup.sh of Snap4City	
Read parameters This tests simply ensures that passing parameters works username age fullname	
Check iotapps Check all iotapps, no matter how many they are	http://localhost:1880/iotapp/iotapp-001/ answered 200, and it should be 200 http://localhost:1881/iotapp/iotapp-002/ answered 200, and it should be 200 http://localhost:1882/iotapp/iotapp-003/ answered 200, and it should be 200

Editing, adding tests

There are 2 kind of tests: on a container (will appear next to the container) and a complex tests

The first kind is supposed to be a ping, in order to see if a given container is alive/healthy

The second kind can be generic bash code

To write a test, go add an entry in the database; to `tests_table` for a container test or add an entry to `complex_tests` for the other kind

For the first table, the first column is the index and you can leave it blank

The second column holds the name of the container and that is the method used to send a certain command; a container can be forced to have a given name by telling it which is it in the docker-compose or the console command; if you are using the latest version (2023/10/30) all the containers are already forced to have a certain name

The third column is the command ran in the console; note that the console depends on the machine on which you run this tool, it could be Windows, *nix or whatever. Write commands accordingly

For the second table, the first column is once again an index

The second column is the name of the command; such a name will appear on the button of the user interface. The name is unique and the database enforces this

The third column is the command ran; the above disclaimer on the platform still applies

The fourth column is an optional declaration of special parameters; as a test might require data to be asked by the user, an interface must be provided. the structure is `data_type_html:name_shown_to_the_user:letter_used_as_console_parameter`

You might require more than one extra parameter; you may put several of these strings separated by a semicolon `;`, no spaces. use only one letter in the third block

The fifth column is the colour of the button in the interface and exists only for UI reasons; it defaults to white

The sixth column provides an additional explanation should one be needed

A seventh column is generated (and thus doesn't exist in the table) as a high contrast colour compared to the colour of the button. The generation of the colour depends on a function inside the database

```
1 • SELECT * FROM checker.tests_table;
```

id	container_name	command
2	opensearch-dashboards	curl -I -s http://localhost:5601/ | awk 'NR==1{print $2}' \| (read code && ["$code" -eq 302] && echo "Success" || echo "Failure")
3	proxy	curl -I -s http://localhost/dashboardSmartCity/ | awk 'NR==1{print $2}' \| (read code && ["$code" -eq 302] && echo "Success" || echo "Failure")
4	certbot	echo Not meant to be running nor tested
5	dashboard-backend	echo Not meant to be tested
6	dashboard-builder	curl -I -s http://localhost/dashboardSmartCity/ | awk 'NR==1{print $2}' \| (read code && ["$code" -eq 302] && echo "Success" || echo "Failure")
7	dashboard-cron	echo Not meant to be tested
8	dashboarddb	if nc -z localhost 3306; then echo "Success"; else echo "Failure"; fi
9	geoserver	curl -I -s http://localhost:8080/geoserver | awk 'NR==1{print $2}' \| (read code && ["$code" -eq 302] && echo "Success" || echo "Failure")
10	geoserver-db	if nc -z localhost 5432; then echo "Success"; else echo "Failure"; fi
11	heatmap-api	echo Not meant to be tested
12	heatmap2geosrv	echo Not meant to be tested
14	iot-fiware-api	curl -I -s http://localhost/iot-directory/ | awk 'NR==1{print $2}' \| (read code && ["$code" -eq 200] && echo "Success" || echo "Failure")
15	iot-fiware-harvester	echo Not meant to be tested
16	iotapp-001	curl -I -s http://localhost:1880/iotapp/iotapp-001/ | awk 'NR==1{print $2}' \| (read code && ["$code" -eq 200] && echo "Success" || echo "Failure")
17	iotapp-002	curl -I -s http://localhost:1881/iotapp/iotapp-002/ | awk 'NR==1{print $2}' \| (read code && ["$code" -eq 200] && echo "Success" || echo "Failure")

```
1 • SELECT * FROM checker.complex_tests;
```

id	name_of_test	command	extraparameters	button_color	explanation
1	Run setup.sh	bash scripts/run-setup.sh	NULL	#63c8e9	Calls setup.sh of Snap4City
2	Run post-setup.sh	bash scripts/run-post-setup.sh	NULL	#ffffff	Calls post-setup.sh of Snap4City
3	Read parameters	bash scripts/read_params.sh	string:username:u;number:age:a;string:fullnam...	#000000	This tests simply ensures that passing paramete...
4	Check iotapps	bash scripts/check_iotapps.sh	NULL	#20ff14	Check all iotapps, no matter how many they are
5	Add Device	bash scripts/add_device.sh	string:username:u;password:password:p	#ff7f00	Calls the test for adding a new device, then ad...
NULL	NULL	NULL	NULL	NULL	NULL