InfraCheck Documentation

Release 2

Wolnosciowiec Team

Feb 20, 2020

Contents:

HTTP healthcheck endpoint + shell healthcheck runner. Simple, easy to setup, easy to understand. Works perfectly with Docker.

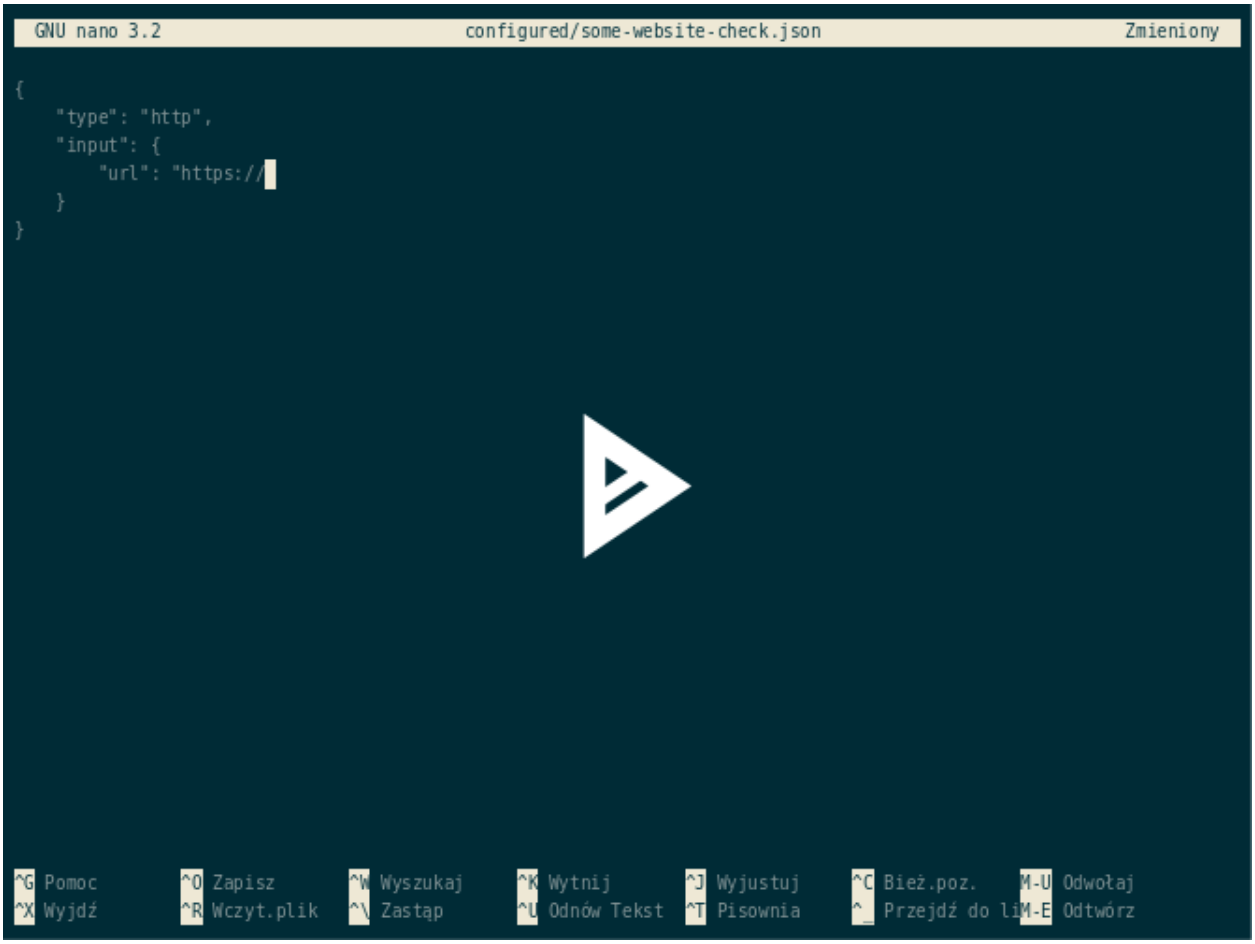

```
{
    "checks": {
        "disk-space": {
            "ident": "disk-space=True",
            "output": "There is 350.8GB disk space at '/', nothing to worry about,␣
↪defined minimum is 15GB\n",
            "status": true
        },
        "docker-health": {
            "ident": "docker-health=True",
            "output": "Docker daemon reports that there is no 'unhealthy' service␣
↪running in '' space\n",
            "status": true
        },
        "minio": {
            "ident": "minio=True",
            "output": "",
            "status": true
        },
        "replication-running": {
            "ident": "replication-running=True",
            "output": "Replica seems to be in good state\n",
            "status": true
        },
        "storage-synchronization": {
            "ident": "storage-synchronization=True",
            "output": "Storage synchronization looks fine\n",
            "status": true
        }
    },
    "global_status": true
}
```

CHAPTER 1

Quick start

To monitor applications and the infrastructure parts you need to configure **checks**. A configured check is a json file that defines a method name (script to be used) and the input parameters. Each check is executed when your external monitoring software invokes the HTTP endpoint, or when you execute the shell command.

Infracheck can work as a HTTP endpoint responding with JSON, or as a console command.



1.1 1. Requirements

You need to install all requirements manually if you decide not to use a docker container.

Requirements:

- Python 3.6+
- OpenSSH Client
- sshpass
- mysql-client
- postgresql-client
- docker client
- curl

1.2 2. Structure

You need to create a **project structure** from following template:

```
- checks/
    - http
    - smtp
    - port
- configured/
    - redis
    - duckduckgo_http
    - smtp_is_alive
```

In **checks** there should be scripts that will take parameters as environment variables, process and give results. For simpler cases you may not need to define any scripts, just configure pre-defined ones.

configured should contain your actual use cases, for example "duckduckgo_http" from above example could use "http" check with url "https://duckduckgo.com" as a parameter.

1.3 3. Configuring a first check

Let's assume that we need to check if a page contains given keyword, and does not contain another defined one. Following check will use **curl** to fetch page content.

Test cases:

- If page will not load, then THE CHECK RETURNS FAILURE
- If page contains "Server error", then THE CHECK RETURNS FAILURE
- If page will not contain keyword "iwa", then THE CHECK RETURNS FAILURE
- If page loads properly and contains "iwa" keyword, then THE CHECK RETURNS SUCCESS

```
{
    "type": "http",
    "input": {
        "url": "http://iwa-ait.org",
        "expect_keyword": "iwa",
        "not_expect_keyword": "Server error"
    }
}
```

Hint: You can pass environment variables in parameters - see: *Templating* section.

1.4 4. Running checks

With Docker

You can use a ready-to-use docker image **wolnosciowiec/infracheck** or **wolnosciowiec/infracheck:armhf** for 32 bit ARM. The image will by default expose a HTTP endpoint.

```
sudo docker run --name infracheck -p 8000:8000 -v $(pwd):/data -d --rm wolnosciowiec/
↪infracheck

# now test it
curl http://localhost:8000
```

List of supported environment variables:

- CHECK_INTERVAL="*/1 * * * *"
- WAIT_TIME=0
- LAZY=false

Without Docker

```
git clone https://github.com/riotkit-org/infracheck
cd infracheck
make install

# run checks in the shell
infracheck --directory=/your-project-directory-path-there

# run a webserver
infracheck --directory=/your-project-directory-path-there --server --server-port=7422␣
↪--lazy

# set up a scheduled checking
echo "*/1 * * * * infracheck --directory=/your-project-directory-path-there --force" >
↪> /etc/crontabs/root
```

Using PIP

```
sudo pip install infracheck

# run checks in the shell
infracheck --directory=/your-project-directory-path-there

# run a webserver
infracheck --directory=/your-project-directory-path-there --server --server-port=7422

# set up a scheduled checking
echo "*/1 * * * * infracheck --directory=/your-project-directory-path-there --force" >
↪> /etc/crontabs/root
```

CHAPTER 2

Hooks

After each execution of your checks there is a possibility to execute some commands.

Example:

```
{
    "type": "disk-space",
    "input": {
        "dir": "/",
        "min_req_space": "6"
    },
    "hooks": {
        "on_each_up": [
            "rm -f /tmp/maintenance.html"
        ],
        "on_each_down": [
            "echo \"Site under maintenance\" > /tmp/maintenance.html"
        ]
    }
}
```

Example above will delete a */tmp/maintenance.html* file when disk space will be at acceptable level. If there will be no enough disk space, then "Site under maintenance" will be written to the /tmp/maintenance.html With this practical example you can add a rule to your NGINX/Apache gateway to show a maintenance page, when a file is present.

CHAPTER 3

Predefined check types reference

Infracheck comes by default with some standard checks, there is a list of them:

3.1 http

Performs a HTTP call using curl.

Example:

```
{
    "type": "http",
    "input": {
        "url": "http://iwa-ait.org",
        "expect_keyword": "iwa",
        "not_expect_keyword": "Server error"
    }
}
```

Parameters:

- url
- expect_keyword
- not_expect_keyword

3.2 dir-present

Checks whenever a directory exists.

Parameters:

- dir

3.3 file-present

Checks if file is present.

Parameters:

- file_path

3.4 docker-health

Checks if containers are healthy.

Parameters:

- docker_env_name (it's a prefix, to check only containers that names begins with this - idea of docker-compose)

3.5 port-open

Checks if the port is open.

Parameters:

- po_host
- po_port (in seconds)
- po_timeout (in seconds)

3.6 replication-running

Checks if the MySQL replication is in good state. Works with Docker only.

Parameters:

- container
- mysql_root_password

3.7 free-ram

Monitors RAM memory usage to notify that a maximum percent of memory was used.

Parameters:

- max_ram_percentage (in percents eg. 80)

3.8 domain-expiration

Check if the domain is close to expiration date or if it is already expired.

Notice: Multiple usage of this check can cause a "request limit exceeded" error to happen

Suggestion: If you check multiple domains, then separate domains checking from regular health checks and set CHECK_INTERVAL (docker) to once a day, and WAIT_TIME=300 for non-docker installations - in crontab set a check with –force once a day

Parameters:

- domain (domain name)
- alert_days_before (number of days before expiration date to start alerting)

3.9 disk-space

Monitors disk space.

Parameters:

- min_req_space (in gigabytes)
- dir (path)

3.10 ovh-expiration

Checks if a VPS is not expired. Grab credentials at https://api.ovh.com/createToken/index.cgi

Required privileges on OVH API: "GET /vps*"

Parameters:

- endpoint (ex. ovh-eu)
- app_key
- app_secret
- app_consumer_key
- service_name (ex. somevps.ovh.net)
- days_to_alert (ex. 30 for 30 days)

3.11 ssh-fingerprint

Verifies if remote host fingerprint matches. Helps detecting man-in-the-middle and server takeover attacks.

Parameters:

- expected_fingerprint (example: zsp.net.pl ssh-rsa SOMESOMESOMESOMESOMEKEYHERE)
- method (default: rsa)
- host (example: zsp.net.pl)
- port (example: 22)

3.12 ssh-files-checksum

Calls remote process using SSH and expects: the listed files and checksums will be matching

Parameters:

- user (default: root)
- host
- port (default: 22)
- private_key
- password
- ssh_bin (default: ssh)
- sshpass_bin (default: sshpass)
- ssh_opts (example: -o StrictHostKeyChecking=no)
- known_hosts_file (default: ~/.ssh/known_hosts)
- command (default: uname -a)
- timeout: (default: 15, unit: seconds)
- method (default: sha256sum)
- expects (json dict, example: {"/usr/bin/bahub": "d6e85b50756a08e24c1d46f07b68e288c9e7e565fd662a15baca214f576c34be"})

3.13 ssh-command

Calls remote process using SSH and expects: exit code, keywords in the output

Parameters:

- user (default: root)
- host
- port (default: 22)
- private_key
- password
- ssh_bin (default: ssh)
- sshpass_bin (default: sshpass)
- ssh_opts (example: -o StrictHostKeyChecking=no)
- known_hosts_file (default: ~/.ssh/known_hosts)
- command (default: uname -a)
- timeout: (default: 15, unit: seconds)
- expected_keywords (Keywords expected to be in stdout/stderr. Separated by ";")
- unexpected_keywords (Keywords not expected to be present in stdout/stderr. Separated by ";")
- expected_exit_code (default: 0)

3.14 reminder

Reminds about the recurring date. Example: To extend validity of your hosting account

Parameters:

- ref_date (example: 2019-05-01 for a 1th of May 2019)
- each (values: week; month; year, default: year)
- alert_days_before (default: 5, the health check will be red when there will be 5 days before)

3.15 load-average-auto

Checks if the load average is not more than 100%

Parameters:

- maximum_above (unit: processor cores, default: 0.5 - half of a core)
- timing (default: 15. The load average time: 1, 5, 15)

3.16 load-average

Checks if the load average is not below specified number

Parameters:

- max_load (unit: processor cores, example: 5.0, default: 1)
- timing (default: 15. The load average time: 1, 5, 15)

3.17 swap-usage-max-percent

Defines maximum percentage of allowed swap usage

Parameters:

- max_allowed_percentage (default: 0.0)

3.18 postgres

Uses *pg_isready* tool to verify if PostgreSQL is up and ready to connect.

Parameters:

- pg_host (hostname or socket path, defaults to "localhost" which will use local unix socket, use IP address eg. 127.0.0.1 to connect via tcp)
- pg_port (port, defaults to 5432)
- pg_db_name (database name to connect to, defaults to "postgres")
- pg_user (username, defaults to "postgres")
- pg_conn_timeout (defaults to 15 which means 15 seconds)

3.19 postgres-primary-streaming-status

Verifies if local PostgreSQL instance is currently serving WALs to a specified replica. The SQL command that is validated: *select * from pg_stat_replication;*

Parameters:

- pg_host (hostname or socket path, defaults to "localhost" which will use local unix socket, use IP address eg. 127.0.0.1 to connect via tcp)
- pg_port (port, defaults to 5432)
- pg_db_name (database name to connect to, defaults to "postgres")
- pg_user (username, defaults to "postgres")
- pg_password
- pg_conn_timeout (defaults to 15 which means 15 seconds)
- expected_status (defaults to "streaming")
- expected_replication_user: Expected user that will be used for replication connection (defaults to "replication")

3.20 postgres-replica-status

Checks if local PostgreSQL server acts as a replication server, by validating the list of active wal receivers. The SQL command that is validated: *select * from pg_stat_wal_receiver;*

Parameters:

- pg_host (hostname or socket path, defaults to "localhost" which will use local unix socket, use IP address eg. 127.0.0.1 to connect via tcp)
- pg_port (port, defaults to 5432)
- pg_db_name (database name to connect to, defaults to "postgres")
- pg_user (username, defaults to "postgres")
- pg_password
- pg_conn_timeout (defaults to 15 which means 15 seconds)
- expected_status (defaults to "streaming")
- expected_replication_user: Expected user that will be used for replication connection (defaults to "replication")

CHAPTER 4

Templating

In order to increase the security there is a *simple templating* mechanism that allows to inject variables into parameters you define that are passed to the checks.

Example:

```
{
    "type": "ssh-command",
    "input": {
        "user": "thesecurityman",
        "host": "iwa-ait.org",
        "port": 6200,
        "password": "${ENV.IWA_SECURITY_MAN_PASSWD}",
        "command": "/usr/bin/some-security-check --is-secure",
        "expected_exit_code": 0,
        "timeout": 30
    }
}
```

4.1 Reference table

Pattern	Example	Description
${ENV.*}	${ENV.USER}	Injects an environment variable from the host
${checkName}	http	Name of the currently executed check
${date}	2019-10-31T07:53:45.380307	Current date and time

4.2 Example strategy of deploying passwords with Docker Compose and Ansible

1. Encrypt your passwords with ansible-vault

2. Decrypt them during deployment into *.env* on target machine for docker-compose
3. In docker-compose service definition pass variable explicitly from the *.env* file

```
environment:
    # variables in checks
    - IWA_SECURITY_MAN_PASSWD=${IWA_SECURITY_MAN_PASSWD}
```

CHAPTER 5

Writing custom checks

Infracheck provides very basic scripts for health checking, you may probably want to write your own. It's really simple.

1. "check" scripts are in **"checks" directory** of your project structure, here you can add a **new check script**
2. Your script needs to take **uppercase environment variables as input**
3. It is considered a good practice to validate environment variables presence in scripts
4. **Your script needs to return a valid exit code when:**
    - Any of environment variables is missing or has invalid value
    - The check fails
    - The check success

That's all!

A few examples:

```
#!/bin/bash

#
# Directory presence check
#
# @author Krzysztof Wesołowski
# @url https://iwa-ait.org
#

if [[ ! "${DIR}" ]]; then
    echo "DIR parameter is missing"
    exit 1
fi

if [[ ! -d "${DIR}" ]]; then
    echo "Failed asserting that directory at '${DIR}' is present"
```

(continues on next page)

(continued from previous page)

```
    exit 1
fi

echo "'${DIR}' directory is present"
exit 0
```

```
#!/usr/bin/env python3

"""
<sphinx>
load-average
------------

Checks if the load average is not below specified number

Parameters:

- max_load (unit: processor cores, example: 5.0, default: 1)
- timing (default: 15. The load average time: 1, 5, 15)
</sphinx>
"""

import os
import sys
import inspect

path = os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe()))) + '/./../'
sys.path.insert(0, path)

from infracheck.infracheck.checklib.loadavg import BaseLoadAverageCheck


class LoadAverageAuto(BaseLoadAverageCheck):
    def main(self, timing: str, max_load: float):
        current_load_average = self.get_load_average(timing)

        if current_load_average > max_load:
            return False, "Load {:.2f} exceeds allowed max. load of {:.2f}. Current load: {:s}".format(
                current_load_average, max_load, self.get_complete_avg()
            )

        return True, "Load at level of {:.2f} is ok, current load: {:s}".format(
            current_load_average, self.get_complete_avg())


if __name__ == '__main__':
    app = LoadAverageAuto()
    status, message = app.main(
        timing=os.getenv('TIMING', '15'),
        max_load=float(os.getenv('MAX_LOAD', 1))
    )

    print(message)
    sys.exit(0 if status else 1)

```

CHAPTER 6

Cache and freshness

It can be harmful to the server to run all checks on each HTTP endpoint call, so a cronjob in background is writing results, and HTTP endpoint is reading.

How often really the checks are performing depends on your configuration, how often you execute **infracheck –force**

6.1 –force

The *–force* parameter means that the application will write checks results to a cache.

When this flag is not specified, then application will read the data from the cache.

6.2 Docker

If you use an official docker image, then you can set CHECK_INTERVAL to a crontab-like syntax interval eg. **CHECK_INTERVAL=00 23 * * *** to check once a day (good for domains whois check).

6.3 Limits

Some checks could call external APIs, those can have limits. A good example is a *domain-expiration* check which is using whois. It is recommended to run a separate infracheck instance with less frequent checking, eg. once a day - see CHECK_INTERVAL in docker, and crontab in standalone installation.

You can also use *–wait* switch to set waiting in seconds between single checks (in docker it is *WAIT_TIME* variable)

6.4 –lazy

When running a ex. HTTP endpoint without *–force*, then application is only reading results of previously executed checks that are usually executed in background using cron. **Without lazy** the checks that were not executed yet will be showing "Not ready yet" and a **status equals to false**. If you really need to avoid such case, **then you can allow running a check on-demand by enabling** *–lazy* **flag** (LAZY=true in docker).

CHAPTER 7

From authors

Project was started as a part of RiotKit initiative, for the needs of grassroot organizations such as:

- Fighting for better working conditions syndicalist (International Workers Association for example)
- Tenants rights organizations
- Various grassroot organizations that are helping people to organize themselves without authority

RiotKit Collective