conn-check Documentation

Release 1.1.0

Canonical

May 30, 2016

Contents:

CHAPTER 1

About conn-check

conn-check allows for checking connectivity with external services.

You can write a config file that defines services that you need to have access to, and conn-check will check connectivity with each.

It supports various types of services, all of which allow for basic network checks, but some allow for confirming credentials work also.

1.1 Configuration

The configuration is done via a yaml file. The file defines a list of checks to do:

```
- type: tcp
  host: localhost
  port: 80
- type: tls
  host: localhost
  port: 443
  disable_tls_verification: false
```

Each check defines a type, and then options as appropriate for that type.

For a step by step guide on configuring conn-check for your application see the tutorial.

1.2 Check Types

1.2.1 tcp

A simple tcp connectivity check.

host The host.

port The port.

timeout Optional connection timeout in seconds. Default: 10 (or value from `--connect-timeout`).

1.2.2 tls

A check that uses TLS (*ssl* is a deprecated alias for this type).

host The host.

port The port.

disable_tls_verification Optional flag to disable verification of TLS certs and handshake. Default: false.

timeout Optional connection timeout in seconds. Default: 10 (or value from `--connect-timeout`).

1.2.3 udp

Check that sending a specific UDP packet gets a specific response.

host The host.

port The port.

send The string to send.

expect The string to expect in the response.

timeout Optional connection timeout in seconds. Default: 10 (or value from `--connect-timeout`).

1.2.4 http

Check that a HTTP/HTTPS request succeeds (*https* also works).

url The URL to fetch.

method Optional HTTP method to use. Default: "GET".

expected_code Optional status code that defines success. Default: 200.

proxy_url Optional HTTP/HTTPS proxy URL to connect via, including protocol, if set proxy_{host,port} are ignored.

proxy_host Optional HTTP/HTTPS proxy to connect via.

proxy_port Optional port to use with `proxy_host`. Default: 8000.

headers: Optional headers to send, as a dict of key-values. Multiple values can be given as a list/tuple of lists/tuples, e.g.: `[('foo', 'bar'), ('foo', 'baz')]`

body: Optional raw request body string to send.

disable_tls_verification: Optional flag to disable verification of TLS certs and handshake. Default: false.

timeout Optional connection timeout in seconds. Default: 10 (or value from `--connect-timeout`).

allow_redirects Optional flag to Follow 30x redirects. Default: false.

params Optional dict of params to URL encode and pass in the querystring.

cookies Optional dict of cookies to pass in the request headers.

auth Optional basic HTTP auth credentials, as a tuple/list: `(username, password)`.

digest_auth Optional digest HTTP auth credentials, as a tuple/list: `(username, password)`.

1.2.5 amqp

Check that an AMQP server can be authenticated against.

host The host.

port The port.

username The username to authenticate with.

password The password to authenticate with.

use_tls Optional flag whether to connect with TLS. Default: true.

vhost Optional vhost name to connect to. Default '/'.

timeout Optional connection timeout in seconds. Default: 10 (or value from `--connect-timeout`).

1.2.6 postgres

Check that a PostgreSQL db can be authenticated against (*postgresql* also works).

host The host.

port The port.

username The username to authenticate with.

password The password to authenticate with.

database The database to connect to.

timeout Optional connection timeout in seconds. Default: 10 (or value from `--connect-timeout`).

1.2.7 redis

Check that a redis server is present, optionally checking authentication.

host The host.

port The port.

password Optional password to authenticatie with.

timeout Optional connection timeout in seconds. Default: 10 (or value from `--connect-timeout`).

1.2.8 memcache

Check that a memcached server is present (*memcached* also works).

host The host.

port The port.

timeout Optional connection timeout in seconds. Default: 10 (or value from `--connect-timeout`).

1.2.9 mongodb

Check that a MongoDB server is present (*mongo* also works).

host The host.

port Optional port. Default: 27017.

username Optional username to authenticate with.

password Optional password to authenticate with.

database Optional database name to connect to, if not set the `test` database will be used, if this database does not exist (or is not available to the user) you will need to provide a database name.

timeout Optional connection timeout in seconds. Default: 10 (or value from `--connect-timeout`).

1.2.10 smtp

Check that we can reach, authenticate with and send an email using an SMTP server.

Note 1: if this check succeeds an email is actually sent to the email defined in `to_address`, be careful how this is check is configured so it doesn't unintentionally spam anyone.

Note 2: only EHLO/HELO over a TLS connection is supported with the `use_tls` flag, this check cannot currently create new TLS connection using the STARTTLS Extension.

host The host.

port The port, normally 465 for TLS and 25 for plaintext.

username Username to authenticate with.

password Password to authenticate with.

from_address: Email address to send *from*.

to_address: Email address to send *to*.

message: Optional email body.

subject: Optional email subject.

helo_fallback: Optional flag that defines whether to fall back to `HELO` if the `EHLO` extended command set fails.

use_tls: Optional flag to enable TLS security on connection. Default: true.

timeout Optional connection timeout in seconds. Default: 10 (or value from `--connect-timeout`).

1.3 Tags

Every check type also supports a `tags` field, which is a list of tags that can be used with the `--include-tags` and `--exclude-tags` arguments to conn-check.

Example YAML:

```
- type: http
  url: http://google.com/
  tags:
    - external
```

To run just "external" checks:

```
conn-check --include-tags=external ...
```

To run all the checks *except* external:

```
conn-check --exclude-tags=external
```

1.4 Buffered/Ordered output

conn-check normally executes with output to `STDOUT` buffered so that the output can be ordered, with failed checks being printed first, grouping by destination etc.

If you'd rather see results as they available you can use the `-U`/`--unbuffered-output` option to disable buffering.

1.5 Generating firewall rules

conn-check includes the `conn-check-export-fw` utility which takes the same arguments as `conn-check` but runs using `--dry-run` mode and outputs a set of *egress* firewall rules in an easy to parse YAML format, for example:

```
# Generated from the conn-check demo.yaml file
egress:
- from_host: mydevmachine
  ports: [8080]
  protocol: udp
  to_host: localhost
- from_host: mydevmachine
  ports: [80, 443]
  protocol: tcp
  to_host: login.ubuntu.com
- from_host: mydevmachine
  ports: [6379, 11211]
  protocol: tcp
  to_host: 127.0.0.1
```

You can then use this output to generate your environments firewall rules (e.g. with *EC2 security groups*, *OpenStack Neutron*, *iptables* etc.).

`conn-check-convert-fw` is a utility that does just this, it accepts multiple firewall rule YAML files, merges/de-dupes them, and outputs commands for AWS, Openstack Neutron, OpenStack Nova (client), iptables, and ufw (mostly for testing purposes).

It is designed for this workflow:

- On each host you run conn-check from, you run `conn-check-export-fw` to generate a YAML file containing egress firewall rules.
- Each of these files is transfered to a host with the correct DNS entries for the egress hosts.
- On this host `conn-check-convert-fw` is run to generate a set of commands for your firewall.
- These commands are audited by a human / possibly merged with other rules, such as adding ingress rules, and then run to update your environment's firewall.

1.6 Building wheels

To allow for easier/more portable distribution of this tool you can build conn-check and all its dependencies as Python wheels:

```
make clean-wheels
make build-wheels
make build-wheels-extra EXTRA=amqp
make build-wheels-extra EXTRA=redis
```

The *build-wheels* make target will build conn-check and its base dependencies, but to include the optional extra dependencies for other checks such as amqp, redis or postgres you need to use the *build-wheels-extra* target with the *EXTRA* env value.

By default all the wheels will be placed in *./wheels*.

1.7 Automatically generating conn-check YAML configurations

The conn-check-configs package contains utilities/libraries for generating checks from existing application configurations and environments, e.g. from Django settings modules and Juju environments.

CHAPTER 2

Tutorial Part 1: Checking connections for a basic web app

2.1 Hello World

Suppose you have the basic webapp *HWaaS* (Hello World as a Service, naturally).

It returns a different translation of “Hello World” on every request, and accepts new translations via `POST` requests.

- The translations are stored in a *PostgreSQL* database.
- *memcached* is used to keep a cache of pre-rendered “Hello World” HTML pages.
- Optionally requests are sent to the Google Translate API to get an automatically translated version of the page in the user’s language if they push a certain button and a translation in their language isn’t available in the *PostgreSQL* DB.
- The *Squid* HTTP proxy is sat between it and the Translate API to cache requests (varied by language), to avoid hitting Google’s rate limiting.

2.2 Why use conn-check?

Our *HWaaS* example service depends on not only 3 internal services, but also a completely external service (the Google Translate API), and any number of issues from network routing, firewall configuration and bad service configuration to external outages could cause issues after a new deployment (or at any time really, but we’ll address that later in *Nagios*).

conn-check can verify connections to these dependencies using not just basic TCP/UDP connects, but also service specific ones, with authentication where needed, timeouts, and even permissions (e.g. can *user A* access *DB schema B*).

2.3 Yet another YAML file

conn-check is configured using a YAML file containing a list of checks to perform in parallel (by default, but this too is configurable with a CLI option).

Here’s an example file (it could be called `hwaas-cc.yaml`):

```
- type: postgresql
  host: gibson.hwaas.internal
  port: 5432
  username: hwaas
  password: 123456asdf
  database: hwaas_production
- type: memcached
  host: freeside.hwaas.internal
  port: 11211
- type: http
  url: https://www.googleapis.com/language/translate/v2?q=Hello%20World&target=de&source=en&key=BLAH
  proxy_host: countzero.hwaas.internal
  proxy_port: 8080
  expected_code: 200
```

2.4 Let's examine those checks..

2.4.1 PostgreSQL

```
- type: postgresql
  host: gibson.hwaas.internal
  port: 5432
  username: hwaas
  password: 123456asdf
  database: hwaas_production
```

type: This one doesn't require much explanation, except the fact that you can use either *postgresql'* or `postgres` (many checks have aliases), see the readme..

host, *port*: The host to connect to is always, understandably, required, but if not supplied the default psql port of `5432` will be used.

username, *password*: Auth details are required and important when used with…

…*database*: This is the psql schema to attempt to switch to use, and *username* has permission to access.

2.4.2 memcached

```
- type: memcached
  host: freeside.hwaas.internal
  port: 11211
```

type: `memcache` or `memcached` are valid, see the readme.

host, *port*: If port isn't supplied the memcached default `11211` is used instead.

2.4.3 HTTP

```
- type: http
  url: https://www.googleapis.com/language/translate/v2?q=Hello%20World&target=de&source=en&key=BLAH
  proxy_host: countzero.hwaas.internal
  proxy_port: 8080
  expected_code: 200
```

type: `http` or `https` are valid, see the readme.

url: As we're doing a simple GET to the Translate API I've included the `key` in the querystring, but you could also include auth defailts as HTTP headers using the `headers` check option.

proxy_host, *proxy_port*: We supply the host/port to our Squid proxy here, we could also use the `proxy_url` check option instead to define the proxy as a standard HTTP URL (makes it possible to define a HTTPS proxy).

expected_code: This is the status code we expect to get back from the service if the request was successful, anything other than `200` in this case will cause the check to fail.

2.5 Using conn-check with Nagios

conn-check output tries to stay as close as possible to the Nagios plugin guidelines so that it can be used as a regular Nagios check for more constant monitoring of your service deployment (not just ad-hoc at deploy time).

Example NRPE config files, assuming `conn-check` is system installed:

```
# /etc/nagios/nrpe.d/check_conn_check.cfg
command[conn_check]=/usr/bin/conn-check --max-timeout=10  --exclude-tags=no-nagios /var/conn-check/hw


# /var/lib/nagios/export/service__hwaas_conn_check.cfg
define service {
    use                                 active-service
    host_name                           hwaas-web1.internal
    service_description                 connection checks with conn-check
    check_command                       check_nrpe!conn_check
    servicegroups                       web,hwaas
}
```

A few arguments to note:

`--max-timeout=10`: This sets the global timeout to 10 seconds, which means it will error if the total time for all checks combined goes above 10s, which is the default max time allowed by Nagios for a plugin to run.

This way we still get all the individual check results back even if one of them went above the threshold.

`--exclude-tags=no-nagios`: Although optional, this allows you to exclude any check tagged with `no-nagios`, which is especially handy for checks to external/third-party services that you don't want to be hit constantly by Nagios.

For example if we didn't want Nagios to hit Google every few minutes:

```
- type: http
  url: https://www.googleapis.com/language/translate/v2?q=Hello%20World&target=de&source=en&key=BLAH
  proxy_host: countzero.hwaas.internal
  proxy_port: 8080
  expected_code: 200
  tags: [no-nagios]
```

CHAPTER 3

Tutorial Part 2: Auto-generating conn-check config for a Django app

3.1 Hello World (again)

Let's assume that you've actually created the `Hello World` service from part 1 as a Django app, and you think to yourself:

"Hang on, aren't all these connections I want conn-check to check for me already defined in my Django settings module?"

3.2 conn-check-configs

Yes, yes they are, and with the handy-dandy conn-check-configs package you can automatically generate conn-check config YAML from a range of standard Django settings values (in theory from other environments too, such as Juju, but for now just Django).

3.3 exempli gratia

Given the following `settings.py` in our *HWaaS* app:

```
INSTALLED_APPS = [
    'hwaas'
]
DATABASES = {
    'default': {
            'ENGINE': 'django.db.backends.postgresql_psycopg2',
            'HOST': 'gibson.hwass.internal',
            'NAME': 'hwaas_production',
            'PASSWORD': '123456asdf',
            'PORT': 11211,
            'USER': 'hwaas',
}
CACHES = {
    'default': {
        'LOCATION': 'freeside.hwaas.internal:11211',
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
    },
}
PROXY_HOST = 'countzero.hwaas.internal'
```

```
PROXY_PORT = 8080
TRANSLATE_API_KEY = 'BLAH'
```

We can create a `settings-to-conn-check.py` script with the least possible effort like so:

```
#!/usr/bin/env python
from conn_check_configs.django import run

if __name__ == '__main__':
    run()
```

This will output *postgresql* and *memcached* checks to similar our hand-written config:

```
$ chmod +x settings-to-conn-check.py
$ ./settings-to-conn-check.py -f cc-config.yaml -m hwaas.settings
$ cat cc-config.yaml
```

```
- type: postgresql
  database: hwaas_production
  host: gibson.hwaas.internal
  port: 5432
  username: hwaas
  password: 123456asdf
- type: memcached
  host: freeside.hwaas.internal
  port: 11211
```

3.4 Customising generated checks

In order to generate the checks we need for Squid / Google Translate API, we can add some custom callbacks:

```
#!/usr/bin/env python
from conn_check_configs.django import run, EXTRA_CHECK_MAKERS


def make_proxied_translate_check(settings, options):
    checks = []
    if settings['PROXY_HOST']:
        checks.append({
            'type': 'http',
            'url': 'https://www.googleapis.com/language/translate/v2?q='
                   'Hello%20World&target=de&source=en&key={}'.format(
                       settings['TRANSLATE_API_KEY']),
            'proxy_host': settings['PROXY_HOST'],
            'proxy_port': int(settings.get('PROXY_PORT', 8080)),
            'expected_code': 200,
        })
    return checks

EXTRA_CHECK_MAKERS.append(make_proxied_translate_check)


if __name__ == '__main__':
    run()
```

In the above we define a callable which takes 2 params, `settings` which is a wrapper around the Django settings module, and `options` which is an object containing the command line arguments that were passed to the script.

The `settings` module is not the direct settings module but a dict-like wrapper so that you can access the settings just a like a dict (using indices, `.get` method, etc.)

To ensure `make_proxied_translate_check` is collected and called by the main `run` function we add it to the `EXTRA_CHECK_MAKERS` list.

The above generates our required HTTP check:

```
- type: http
  url: https://www.googleapis.com/language/translate/v2?q=Hello%20World&target=de&source=en&key=BLAH
  proxy_host: countzero.hwaas.internal
  proxy_port: 8080
  expected_code: 200
```

3.5 A note on statsd checks

Getting more operational visibility on how *HWaaS* runs would be great, wouldn't it?

So let's add some metrics collection using StatsD, and as luck would have it we can get a lot for *nearly free* with the django-statsd, after adding it to our dependencies we update our `settings.py` to include:

```
INSTALLED_APPS = [
    'hwaas'
    'django_statsd',
]
MIDDLEWARE_CLASSES = [
    'django_statsd.middleware.GraphiteMiddleware',
]
STATSD_CLIENT = 'django_statsd.clients.normal'
STATSD_HOST = 'bigend.hwaas.internal'
STATSD_PORT = 10021
```

Note: You don't actually need the django-statsd app to have conn-check-configs generate statsd checks, only the use of `STATSD_HOST` and `STATSD_PORT` in your settings module matters.

Another run of our `settings-to-conn-check.py` script will result in the extra statsd check:

```
- type: udp
  host: bigend.hwaas.internal
  port: 10021
  send: conncheck.test:1|c
  expect:
```

As you can see this is just a generic UDP check that attempts to send an incremental counter metric to the statsd host.

Unfortunately the fire-and-forget nature of this use of statsd/UDP will not error in a number of common situations (the simplest being that statsd is not running on the target host, or even a routing issue along the way).

It will catch simple problems such as not being able to open up the local UDP port to send from, but that's usually not enough.

If you use a third-party implementation of statsd, such as txStatsD then you might have the ability to define a pair of health check strings, for example by changing the send and expect values in the `STATSD_CHECK` dict we can send and expect different strings:

```
#!/usr/bin/env python
from conn_check_configs.django import run, STATSD_CHECK

STATSD_CHECK['send'] = 'Hakuna'
```

```
STATSD_CHECK['expect'] = 'Matata'

if __name__ == '__main__':
    run()
```

Which generates this check:

```
- type: udp
  host: bigend.hwaas.internal
  port: 10021
  send: Hakuna
  expect: Matata
```

In the above we would configure our txStatD (for example) instance to respond to the string `Hakuna` with the string `Matata`, which would catch pretty much all the possible issues with contacting our metrics service.

CHAPTER 4

Tutorial Part 3: Adding conn-check to Juju deployed services

4.1 Juju

Juju is an open source service orientated framework and deployment toolset from Canonical, given conn-check is also by Canonical you might expect there is an easy yet flexible way to add conn-check to your Juju environment.

You'd be right…

4.2 Adding conn-check charm support to your apps charm

The conn-check charm is a subordinate charm that can be added alongside your applications charm, and will install/configure conn-check on your application units.

To enable support for the conn-check subordinate in your applications charm you need to implement the `conn-check-relation-changed` hook, e.g.:

```
#!/bin/bash
set -e
CONFIG_PATH=/var/conn-check.yaml

juju-log "Writing conn-check config to ${CONFIG_PATH}"
/path/to/hwaas/settings-to-conn-check.py -f ${CONFIG_PATH} -m hwaas.settings

# Ensure conn-check and nagios can both access the config file
chown conn-check:nagios ${CONFIG_PATH}
chmod 0660 ${CONFIG_PATH}

# Set the config path, we could also tell the conn-check charm
# to write the config file for us by setting the "config" option
# but this is deprecated in favour of writing the file ourselves
# and setting "config_path"
relation-set config_path="${CONFIG_PATH}"
```

You may note that we set the user to `conn-check` and the group to `nagios`, you can actually get away with just setting the group to `nagios` as this will give both conn-check and nagios access to the config file, but you might as well set the user anyway otherwise it's likely to be `root`.

You'll also need to tell Juju your charm provides the `conn-check` relation in your `metadata.yaml`:

```
provides:
    conn-check:
        interface: conn-check
        scope: container
```

When deploying conn-check with your service you then deploy the subordinate, relate it to your service (you can also optionally set it as a *Nagios* provider):

```
$ juju deploy cs:~ubuntuone-hackers/trusty/conn-check my-service-conn-check
$ juju set my-service-conn-check revision=108 # pin to the rev of conn-check you want to use
$ juju add-relation my-service my-service-conn-check
```

4.3 Nagios

The conn-check charm provides the `nrpe-external-master` relation which means it can act as a Nagios plugin executor, so if you have a Nagios master in your environment for monitoring then conn-check can be regularly run along with your other monitoring checks to ensure your environments connections are as you expect them to be.

To set this up you need to relate the deployed subordinate to your servie nrpe:

```
$ # assuming something like:
$ # juju deploy nagios nagios-master
$ # juju deploy nrpe my-service-nrpe
$ # juju add-relation my-service:monitors my-service-nrpe:monitors
$ juju add-relation my-service-conn-check my-service-nrpe
```

For more details on Juju and Nagios you can see this handy blog post.

4.4 Actions

To manually run conn-check on all units, or a single unit, you can use the supplied `run-check` and `run-nagios-check` actions:

```
$ # all checks on all units
$ juju run --service my-service-conn-check 'actions/run-check'
$ # all checks on just unit 0
$ juju run --service my-service-conn-check/0 'actions/run-check'
$ # nagios (not including no-nagios) checks on all units
$ juju run --service my-service-conn-check 'actions/run-nagios-check'
$ # nagios (not including no-nagios) checks on just unit 0
$ juju run --service my-service-conn-check/0 'actions/run-nagios-check'
```

Note: before Juju 1.21 there is a bug which prevents juju-run from working with subordinate charms, you can work around this with juju-ssh:

```
$ # all checks on just unit 0
$ juju ssh my-service-conn-check/0 'juju-run my-service-conn-check/0 actions/run-check'
```

CHAPTER 5

ChangeLog for conn-check

5.1 1.3.1 (2015-08-11)

- Added guards for port numbers and the HTTP checks expected_code to cast any given value to an int.

5.2 1.3.0 (2015-07-15)

- Added new conn-check-convert-fw tool to generate aws/neutron/nova/iptables rule commands from YAML exported by conn-check-export-fw.

5.3 1.2.0 (2015-06-19)

- Added new smtp check to test auth/sending with SMTP servers.

5.4 1.1.0 (2015-06-05)

- Added new conn-check-export-fw tool to export firewall egress rules in a YAML format.
- Refactored CLI command handling code to make it easier to extend/override.

5.5 1.0.18 (2015-04-13)

- Ensure pyOpenSSL is always used instead of the ssl modules, see https://urllib3.readthedocs.org/en/latest/security.html#pyopenssl.

5.6 1.0.17 (2015-04-08)

- Unpin python-requests for wider distribution (e.g. precise).

5.7 1.0.16 (2015-03-06)

- Add –include-tags and –exclude-tags args with support for the *tags* YAML check field.

5.8 1.0.15 (2015-02-24)

- Package manifest fixes for debian package release.

5.9 1.0.13 (26-11-2014)

- Output is not buffered and ordered, with FAILED checks first, skipped last, and each check grouped by {type}:{host/url}.
- TCP subchecks triggered by a HTTP check are prefixed as such.
- There is now a -U/–unbuffered-output option to disable buffered/ordered output and write out to STDOUT as soon as a result is collected.

5.10 1.0.12 (17-11-2014)

- Command aliasing refactored, and more aliases added.

5.11 1.0.11 (04-11-2014)

- Disabled 30x redirects in HTTP checks by default, fixing regression introduced by requests switch.
- Added python-requests specific options for proxy, param, cookie and auth control in HTTP checks.

5.12 1.0.10 (30-10-2014)

- Added a mongodb check type.

5.13 1.0.9 (23-10-2014)

- Added –max-timeout CLI option to restrict maximum execution time.
- Added connection timeouts to HTTP and PostgreSQL checks.
- Added –connect-timeout CLI option to set global connection timeout.
- Added timeout option to each individual check to override global connection timeout.

5.14 1.0.8 (22-10-2014)

- Switched to using txrequests for HTTP requests with better proxy support.
- Fixed UDP checks targetting host rather than IP if available.
- Fixed initial TCP check for HTTP checks targetting upstream instead of proxy.

5.15 1.0.7 (09-10-2014)

- Fixed HTTP proxy error in HTTP checks due to typo.

1.0.6 (06-10-2014)

- Fixed dependencies when installing from local dir.
- Made improvements to readme.

5.16 1.0.5 (03-10-2014)

- Added optional headers and body arguments to HTTP checks.

5.17 1.0.4 (29-09-2014)

- Added HTTP proxy support to http checks
- Fixed issue with loading duplicate SSL CA certificates, and added flag to load from a custom dir

5.18 1.0.3 (24-09-2014)

- Removed config_generators module to it's own package: conn-check-configs

5.19 1.0.2 (22-09-2014)

- Added a script to auto-generate conn-check config YAML from a Django settings module

5.20 1.0.1 (18-09-2014)

- Trivial release to fix setup.py tags

5.21 1.0.0 (18-09-2014)

- Initial release
- Broken free of UbuntuOne
- Nagios compatible output
- YAML configuration

Indices and tables

- genindex
- modindex
- search