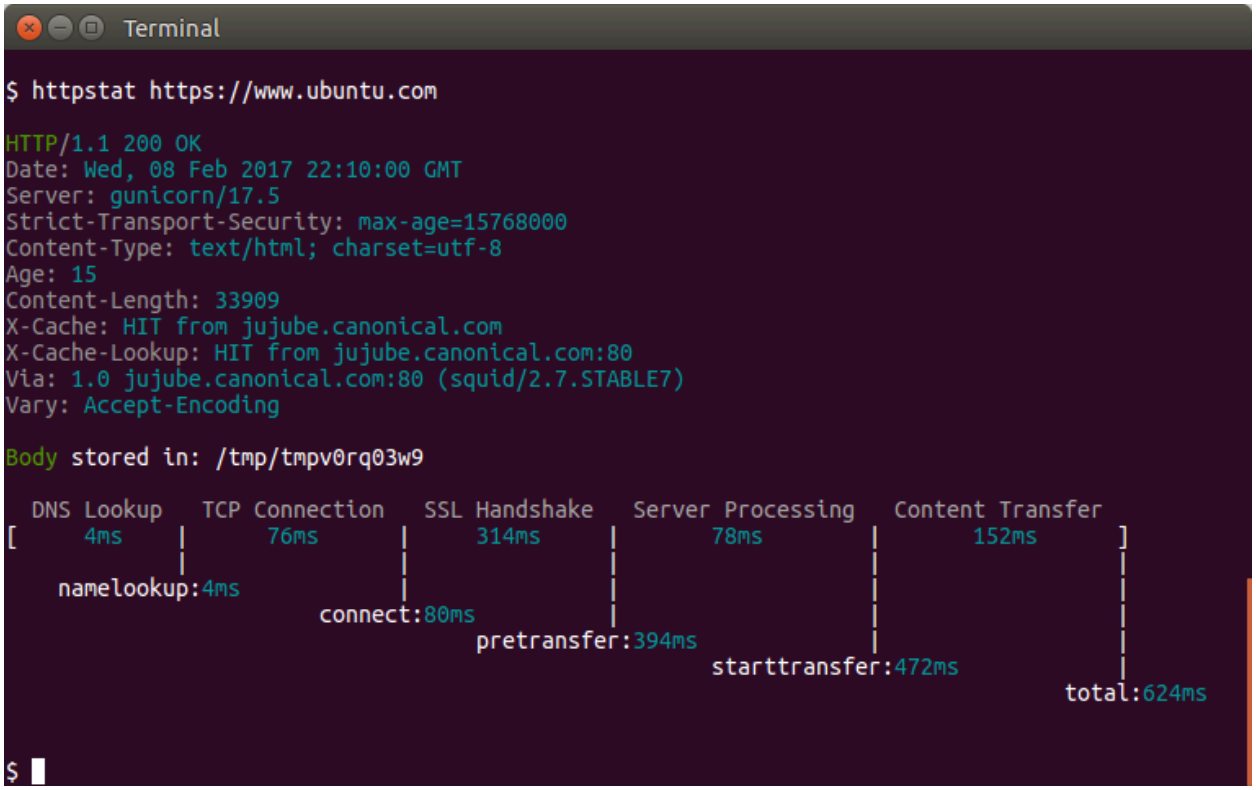
Snap a python application

Summary	In this codelab, you'll create a snap of a command-line utility for website benchmarking, called ***httpstat***. This utility is written in Python and requires only access to the network to perform its task.
URL	snap-python-app
Category	snapcraft
Environment	snapcraft, usage, build, beginner, python
Status	Draft
Feedback Link	https://github.com/ubuntu/codelabs/issues
Author	simos
Tags	python,network,interface
Difficulty	2
Published	2017-02-14

Overview

Duration: 1:00

In this tutorial we are going to snap a network utility called [httpstat](), written in Python. `httpstat` visualizes curl statistics in a way of beauty and clarity.



The screenshot above shows the output of `httpstat` when we run it on https://www.ubuntu.com. `httpstat` shows how much time it took for each individual stage of the page transfer, in this case is:

1. DNS Lookup
2. TCP Connection

3. SSL Handshake
4. Server Processing
5. Content Transfer

A system administrator would need to control those times in an effort to reduce them as much as possible. For example, they could switch to a different server location, get a faster server or optimize the server software so that it delivers the content much faster. In this specific example, we actually learn that our connection was served by a reverse proxy (shown as *jujube.canonical.com*), which guarantees a very short time in server processing when the web page content remains the same.

What you'll learn

- How to package a python project with snapcraft
- How to pull source code from a specific git tag
- We'll finally review how to make its strictly confined.

What you'll need

- Ubuntu 16.04 or newer
- Basic knowledge of Linux command line
    - Familiarize yourself first with snaps by following first these two introductory tutorials: "basic snap usage" and "create your first snap"

How will you use this tutorial?

- Only read through it
- Read it and complete the exercises

What is your current level of experience working with snap?

- Novice
- Intermediate
- Proficient

Getting familiar with the project

Duration: 2:00

This project is a **Python** application. It has a very simple project layout with an installation and test script.

Introspect the code

Simply run:

```
$ git clone https://github.com/reorx/httpstat
$ cd httpstat
$ ls
[…]
httpstat.py
setup.py
```

The two interesting files in that directory are:

- `httpstat.py` corresponding to the main (and only!) source file, containing the code. You can execute it directly once it's made executable.
- `setup.py`, which is a standard python installation script file. Running setup will build and install the project on the target host. The good news is that snapcraft has full support for projects following this best python practice.

You will note that this project has no `requirements.txt`, meaning that there is no non standard library module used and that needs to be pulled in at build time. If that were the case, snapcraft would have handle this for you, pulling the correct dependencies via pip, itself.

Do not hesitate to open the files to get familiar with them. Once this is done, you can now remove that `httpstat` directory.

Crafting a working snap in devmode

Duration: 9:00

Note that we know what we are going to snap and how it's structured. Let's start getting serious and work on it!

Scaffolding

We are going to create the snapcraft project in another directory. We will pull the main source code directly from github each time we build the snap.
Let's use `snapcraft` to scaffold an initial configuration file and then build on that to create the final snap for `httpstat`.

```
$ mkdir httpstat-snap
$ cd httpstat-snap
$ snapcraft init
Created snap/snapcraft.yaml.
Edit the file to your liking or run `snapcraft` to get started
```

You are now ready to go with our familiar `snap/snapcraft.yaml` file.

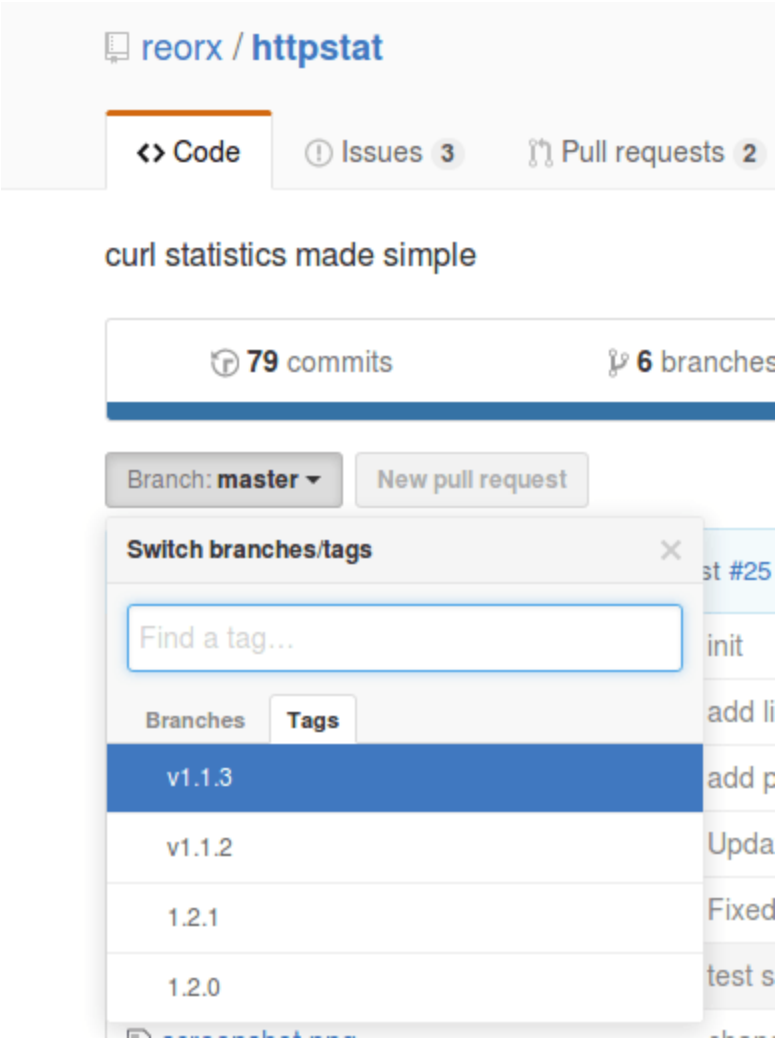
Main snapcraft.yaml metadata

Here is the first part of the httpstat `snapcraft.yaml`. Let's add to it some metadata concerning our snap and the project. Change your file to match the following:

```
name: httpstat
version: '1.1.3'
summary: Curl statistics made simple
description: |
    httpstat is a utility that analyses show fast is a website
    when you are trying to connect to it.
    This utility is particularly useful to Web administrators.
grade: stable
confinement: devmode
```

First, for the a name, we use `httpstat`.

Second, we select a version. Instead of using the latest development version that might not work or might happen to be broken momentarily, you can pick and choose the stable branch or the latest tag.



The tag `v1.1.3` will do!

Third and fourth, we add a summary and a description from text we got from the `httpstat` github page.

Fifth, we select a `grade`. That would be either devel (for development) or stable. This snap is going to land in the stable channel in the store.

Sixth, we select the `confinement` of the snap. As for other snaps, we are going to let it first in `devmode` before turning it to `strict` confinement.

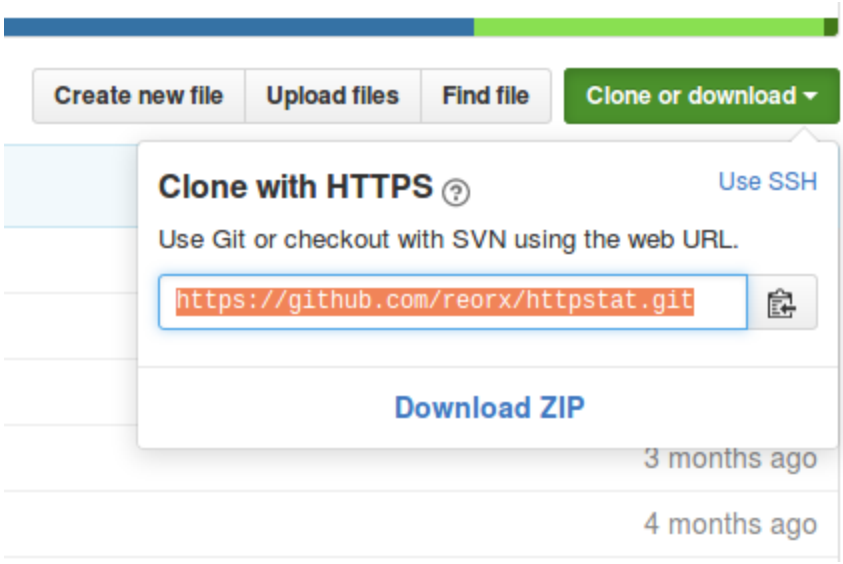
A python part

This simple application is going to be made of a single part, referencing the github source repository. We know as well that we want to pull a certain revision (a git tab) and the project is made in python.

This translates into:

```
parts:
  httpstat:
    source: https://github.com/reorx/httpstat.git
    source-tag: v1.1.3
    plugin: python
```

And that's it! In the `parts` section we provide instructions as to how to process this `httpstat` target.



We first specify the git URL for the source (note that it ends with .git) and the tag. The git URL appears when we click on the `Clone or download` green button at github. More information on the source is available via the `snapcraft source` command.

We then specify that we want to use the `plugin: python`, which is a plugin that performs `python setup.py build` and `python setup.py install`. The default behavior is to build a python3 program. The `python-version:` element can be specified to set the python version to 2. As we discussed previously, this project doesn't have a requirements.txt file, and so, no dependency will be pulled in and referenced via pip by snapcraft. Otherwise, this would have been done conveniently for you, in a relocatable fashion! Note that `snapcraft help python` will give you way more information in the available information of the plugin:

```
$ snapcraft help python
The python plugin can be used for python 2 or 3 based parts.

It can be used for python projects where you would want to do:

    - import python modules with a requirements.txt
    - build a python project that has a setup.py
    - install packages straight from pip

This plugin uses the common plugin keywords as well as those for "sources".
For more information check the 'plugins' topic for the former and the
'sources' topic for the latter.

Additionally, this plugin uses the following plugin-specific keywords:

    - requirements:
      (string)
      Path to a requirements.txt file
    - constraints:
      (string)
      Path to a constraints file
    - process-dependency-links:
      (bool; default: false)
      Enable the processing of dependency links.
    - python-packages:
      (list)
      A list of dependencies to get from PyPi
    - python-version:
      (string; default: python3)
      The python version to use. Valid options are: python2 and python3
```

Even if we know we didn't expose any command or service yet, let's try to build it to ensure our part definition is correct:

```
$ snapcraft prime
Preparing to pull httpstat
Hit http://archive.ubuntu.com/ubuntu xenial InRelease
Get:1 http://archive.canonical.com xenial InRelease [11.5 kB]
[...]
Fetched 6208 kB in 0s (0 B/s)
Pulling httpstat
Cloning in '<...>/parts/httpstat/src'...
remote: Counting objects: 251, done.
remote: Total 251 (delta 0), reused 0 (delta 0), pack-reused 251
Réception d'objets: 100% (251/251), 330.03 KiB | 0 bytes/s, fait.
Résolution des deltas: 100% (138/138), fait.
Vérification de la connectivité... fait.
Note: checking out '0f0e653309982178302ec1d5023bda2de047a72d'.
```

```
You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

Collecting pip
  Using cached pip-9.0.1-py2.py3-none-any.whl
  Saved ./parts/httpstat/packages/pip-9.0.1-py2.py3-none-any.whl
Collecting setuptools
  Downloading setuptools-34.2.0-py2.py3-none-any.whl (389kB)
    100% |████████████████████████████████| 399kB 2.2MB/s
  Saved ./parts/httpstat/packages/setuptools-34.2.0-py2.py3-none-any.whl
[...]
Installing collected packages: pip, six, appdirs, pyparsing, packaging,
setuptools, wheel
Successfully installed appdirs packaging pip-8.1.1 pyparsing-2.0.3
setuptools-20.7.0 six-1.10.0 wheel-0.29.0
pip download --disable-pip-version-check --dest
<...>/parts/httpstat/packages .
Processing <...>/step3/parts/httpstat/src
  Link is a directory, ignoring download_dir
Successfully downloaded httpstat
Preparing to build httpstat
Building httpstat
Collecting pip
  Saved /tmp/tmpmj_yh4e1/pip-9.0.1-py2.py3-none-any.whl
Collecting setuptools
  Saved /tmp/tmpmj_yh4e1/setuptools-34.2.0-py2.py3-none-any.whl
[...]
Successfully installed appdirs-1.4.0 packaging-16.8 pip-9.0.1
pyparsing-2.1.10 setuptools-34.2.0 six-1.10.0 wheel-0.29.0
pip wheel --disable-pip-version-check --no-index --find-links
<...>/step3/parts/httpstat/packages --wheel-dir /tmp/tmpah1cbkq9 .
Processing <...>/step3/parts/httpstat/build
Building wheels for collected packages: httpstat
  Running setup.py bdist_wheel for httpstat ... done
  Stored in directory: /tmp/tmpah1cbkq9
Successfully built httpstat
DEPRECATION: The default format will switch to columns in the future. You
can use --format=(legacy|columns) (or define a format=(legacy|columns) in
your pip.conf under the [list] section) to disable this warning.
pip install --user --no-compile --disable-pip-version-check --no-index
--find-links <...>/step3/parts/httpstat/packages httpstat --no-deps
--upgrade
Collecting httpstat
Installing collected packages: httpstat
Successfully installed httpstat-1.1.3
Staging httpstat
```

```
Priming httpstat
```

Success! (Well, if you don't have any typo in your yaml file ;)). Reading the log, you can see that the python plugin did some heavy lifting for you:

- Installing pip, setuptools, and a lot of other tools (from the distribution and pip)
- Putting the python3 binaries and standard library as part of the snap
- Running `setup.py` which installs the `httpstat` binary. This one finally ends up as `prime/bin/httpstat`.

> **Note:**
>
> Do not hesitate to look at the content of the `prime/` directory. This is what will be exactly in your final snap. You will really appreciate all the work snapcraft did for you there!

Adding an httpstat command

Ok, our httpstat project is being built, but we are missing something: a command to run it! Let's expose one.
Add in your `snapcraft.yaml`, between the top metadata and the `parts:` paragraph:

```
apps:
  httpstat:
    command: httpstat
```

We specify that the users will be running an executable named `httpstat` via the `command:` argument. We name that command `httpstat` itself.

That's it? Yes it is and it is high time to issue a build:

```
$ snapcraft prime
Skipping pull httpstat (already ran)
Skipping build httpstat (already ran)
Skipping stage httpstat (already ran)
Skipping prime httpstat (already ran)
```

And done! Let's ship it. Oh wait! Testing? Hum maybe… but let's add a new section for this.

> **Note:**
>
> You will notice in the generated `command-httpstat.wrapper` file from the `prime/` directory that snapcraft python plugin helped you in exporting `PYTHONUSERBASE` and `PYTHONHOME` to reference your local python installation. Nifty!

Polishing our snap (or rather, making it work ;))

Ok, let's see what this is getting us, trying the snap on our system:

```
$ snap try --devmode prime/
httpstat 1.1.3 mounted from <...>/prime
$ httpstat --help
Usage: httpstat URL [CURL_OPTIONS]
       httpstat -h | --help
       httpstat --version

Arguments:
  URL     url to request, could be with or without `http(s)://` prefix

Options:
  CURL_OPTIONS  any curl supported options, except for -w -D -o -S -s,
                which are already used internally.
  -h --help     show this screen.
  --version     show version.

Environments:
  HTTPSTAT_SHOW_BODY    By default httpstat will write response body
                            in a tempfile, but you can let it print out by
setting
                        this variable to `true`.
  HTTPSTAT_SHOW_SPEED   set to `true` to show download and upload speed.
```

Looks great! Let's try now to get some statistics on https://www.ubuntu.com:

```
$ httpstat https://www.ubuntu.com
Traceback (most recent call last):
  File "/snap/httpstat/x1/bin/httpstat", line 11, in <module>
    sys.exit(main())
    File  "/snap/httpstat/x1/lib/python3.5/site-packages/httpstat.py",  line
155, in main
                p    =    subprocess.Popen(cmd,    stdout=subprocess.PIPE,
stderr=subprocess.PIPE, env=cmd_env)
   File  "/snap/httpstat/x1/usr/lib/python3.5/subprocess.py",  line  947,  in
__init__
```

```
    restore_signals, start_new_session)
   File "/snap/httpstat/x1/usr/lib/python3.5/subprocess.py", line 1551, in
_execute_child
    raise child_exception_type(errno_num, err_msg)
FileNotFoundError: [Errno 2] No such file or directory: 'curl'
```

Ah, not so good… The error helps us here: **No such file or directory: 'curl'**. Indeed, curl is a binary, not a python module, and we don't ship it as part of our snap. `curl` is available in the Ubuntu `apt` repositories, therefore we can use the `stage-packages` functionality in order to reuse this available binary package:

```
parts:
  httpstat:
    [...]
    stage-packages: [curl]
```

Here, we only add one package to the stage-packages list: `curl`.

> **Important**: The error messages may be long and perhaps confusing. It is important to skim the whole error message for the interesting information. In the above case, we learn that the error produced a traceback (a trace of the instructions from start to the point of the error). At the end of the traceback, we get the important hint: `FileNotFoundError: [Errno 2] No such file or directory: 'curl'` It could not find `curl`, one of the most common utility that downloads content from websites. Every snap is considered empty unless we explicitly add commands like curl, which is needed here.

Let's quickly rebuild (after cleaning as we changed one part definition), reinstall, cross fingers, and run it:

```
$ snapcraft clean
Cleaning up priming area
Cleaning up staging area
Cleaning up parts directory
$ snapcraft prime
[...]
$ snap try --devmode prime/
httpstat 1.1.3 mounted from <...>/prime
$ httpstat https://www.ubuntu.com
HTTP/1.1 200 OK
Date: Thu, 16 Feb 2017 14:44:40 GMT
Server: gunicorn/17.5
Strict-Transport-Security: max-age=15768000
Content-Type: text/html; charset=utf-8
Age: 173
```

```
Content-Length: 34074
X-Cache: HIT from privet.canonical.com
X-Cache-Lookup: HIT from privet.canonical.com:80
Via: 1.0 privet.canonical.com:80 (squid/2.7.STABLE7)
Vary: Accept-Encoding

Body stored in: /tmp/tmp3m7quqre

  DNS Lookup   TCP Connection   SSL Handshake   Server Processing   Content
Transfer
[     61ms     |        17ms        |      230ms      |        20ms         |
38ms       ]
              |                  |                 |                       |
|
     namelookup:61ms              |                 |                       |
|
                              connect:78ms           |                       |
|
                                                 pretransfer:308ms          |
|
                                                                 starttransfer:328ms
|

total:366ms
```

Success! Before shipping our snap, to be able to publish in the stable channel, we need to turn confinement on though. But that’s for the next step, we have worked enough here!

Let's confine everything!

Duration: 4:00

Ok, the title may be a little bit catchy. Let's start by confining this snap alone :)

Lost or starting from here?

Check or download here to see what your current directory should look like.

Testing our current snap without modifying it

In other tutorials, we directly edited snapcraft.yaml and replaced `confinement: devmode` by `strict`, rebuilt the snap and so on. We will need to do that for the final snap. However, we can bypass this by now by installing our devmode snap with strict confinement.

How to do this? Quite easily:

```
$ snap try --jailmode prime/
httpstat 1.1.3 mounted from <...>/prime
```

jailmode forces any snap declaring needing devmode to be fully confined.

Let's give this a shot:

```
$ httpstat https://www.ubuntu.com
> curl -w <output-format> -D <tempfile> -o <tempfile> -s -S
https://www.ubuntu.com
curl error: curl: (6) Could not resolve host: www.ubuntu.com
```

We are almost there, but as you saw, this didn't work! Indeed, the snap does not have access to the Internet anymore because of the `strict` confinement. We need to declare that type of access and only that (for example, the `httpstat` would still not have any access to the files in our home directory).

Important: The error message was `curl error: curl: (6) Couldn't resolve host 'www.ubuntu.com'`. When trying to make an Internet connection, an app needs to first *resolve* the name of the server into the IP address. When there is no access to the Internet, the `resolve` (the first step towards a network connection) will not work, thus this error. A snap with `confinement: strict` does not have any access to the Internet, unless explicitly allowed.

Enabling access to network

To allow networking access to a snap, we need to specify that networking is OK for this snap. `network` is an ***interface*** in snaps and it is one of the many supported interfaces for snaps. There is the notion of `plugs` (provider of resource) and `slots` (consumer of resource). For most cases, like this one here, we need a plug, a plug for network.

Let's edit our `snapcraft.yaml` and declare for our `httpstat` command this access:

```
apps:
  httpstat:
    command: httpstat
    plugs: [network]
```

While we are at it and before we rebuild our snap, let's change the `confinement:` line to set it as `strict`.

Once we add these two lines, we reach the final version of snapcraft.yaml for httpstat.

```
confinement: strict
```

and let's build it all:

```
$ snapcraft
Skipping pull httpstat (already ran)
Skipping build httpstat (already ran)
Skipping stage httpstat (already ran)
Skipping prime httpstat (already ran)
Snapping 'httpstat' -
Snapped httpstat_1.1.3_amd64.snap
```

Final testing

It's time to install the final snap product, confined, and retest it (remember of `--dangerous` as we are installing a local snap, and not one signed from the store):

```
$ snap install --dangerous httpstat_1.1.3_amd64.snap
httpstat 1.1.3 installed
$ httpstat https://www.ubuntu.com

HTTP/1.1 200 OK
```

```
Date: Thu, 16 Feb 2017 15:27:22 GMT
Server: gunicorn/17.5
Strict-Transport-Security: max-age=15768000
Content-Type: text/html; charset=utf-8
Content-Length: 34222
X-Cache: HIT from jujube.canonical.com
X-Cache-Lookup: HIT from jujube.canonical.com:80
Via: 1.0 jujube.canonical.com:80 (squid/2.7.STABLE7)
Vary: Accept-Encoding

Body stored in: /tmp/tmpvnlglnz3

  DNS Lookup   TCP Connection   SSL Handshake   Server Processing   Content
Transfer
[     12ms     |        25ms        |      240ms      |        21ms         |
40ms       ]
               |                  |                 |                       |
|
     namelookup:12ms              |                 |                       |
|
                             connect:37ms           |                       |
|
                                          pretransfer:277ms                |
|
                                                              starttransfer:298ms
|

total:338ms
```

And that’s it! The snap, with strict confinement, works!

That's all folks!

Duration: 1:00

And here we go, that was quite easy isn't it?

Final code

Your final code directory should now look like this. Do not hesitate to download and build your snap from it if you only read it through!

You should by now have a complete python snap ready to be uploaded to the store, fully confined, only having access to the network.
You now know how to snap a python application, the available options and handy features it gives to you to make snapping it a breeze! You also have some notions on how to reference external programs, branching them at specific tags. Finally, looking at the output of our program, we were able to understand the secure interfaces it requires. This allowed us to turn on confinement.

Next steps

- Now that you have package an application, you can learn how to snap a service, a more iterative process on debugging confinement issues and more, on our unique build a nodejs service tutorial. Chuck Norris is also starring it, how to resist? :)
- Learn some more advanced techniques on how to use your snap system looking for our others codelabs!
- Join the snapcraft.io community on the snapcraft forum.

Further readings

- You can check online the python plugin reference.
- Common source options are used to define where to branch your code from.
- Snapcraft.io user interface documentation will give you some basics about slots, plugs and interfaces.
- An interface reference explains and list existing interface, if they auto-connect or not and more.
- Some basic notions on debugging a snap.
- Snapcraft syntax reference, covering various available options like the daemon ones.
- Check how you can contact us and the broader community.