
Flask-Shell2HTTP

Release 1.9.1

Eshaan Bansal

Apr 29, 2022

CONTENTS

1	Quickstart	3
1.1	Quick Start	3
1.2	Examples	5
1.3	Configuration	5
2	API Reference	7
2.1	API Reference	7
3	Indices and tables	9
	Python Module Index	11
	Index	13

A minimalist [Flask](#) extension that serves as a RESTful/HTTP wrapper for python's subprocess API.

- **Convert any command-line tool into a REST API service.**
- Execute shell commands asynchronously and safely via flask's endpoints.
- Designed for binary to binary/HTTP communication, development, prototyping, remote control and more.

Use Cases:

- Set a script that runs on a succesful POST request to an endpoint of your choice.
- Map a base command to an endpoint and pass dynamic arguments to it.
- Can also process multiple uploaded files in one command.
- This is useful for internal docker-to-docker communications if you have different binaries distributed in micro-containers.
- You can define a callback function/ use signals to listen for process completion.
- You can also apply View Decorators to the exposed endpoint.

Note: This extension is primarily meant for executing long-running shell commands/scripts (like nmap, code-analysis' tools) in background from an HTTP request and getting the result at a later time.

QUICKSTART

Get started at [Quick Start](#). There are also more detailed [Examples](#) that shows different use-cases for this package.

1.1 Quick Start

1.1.1 Dependencies

- Python: >=v3.6
- Flask
- Flask-Executor

1.1.2 Installation

```
$ pip install flask flask_shell2http
```

1.1.3 Example Program

Create a file called `app.py`.

```
from flask import Flask
from flask_executor import Executor
from flask_shell2http import Shell2HTTP

# Flask application instance
app = Flask(__name__)

executor = Executor(app)
shell2http = Shell2HTTP(app=app, executor=executor, base_url_prefix="/commands/")

def my_callback_fn(context, future):
    # optional user-defined callback function
    print(context, future.result())

shell2http.register_command(endpoint="saythis", command_name="echo", callback_fn=my_
↪callback_fn, decorators=[])
```

Run the application server with, `$ flask run -p 4000`.

With <10 lines of code, we successfully mapped the shell command `echo` to the endpoint `/commands/saythis`.

1.1.4 Making HTTP calls

This section demonstrates how we can now call/ execute commands over HTTP that we just mapped in the *example* above.

```
$ curl -X POST -H 'Content-Type: application/json' -d '{"args": ["Hello", "World!"]}' \
↳ http://localhost:4000/commands/saythis
```

```
# You can also add a timeout if you want, default value is 3600 seconds
data = {"args": ["Hello", "World!"], "timeout": 60, "force_unique_key": False}
resp = requests.post("http://localhost:4000/commands/saythis", json=data)
print("Result:", resp.json())
```

returns JSON,

```
{
  "key": "ddbe0a94",
  "result_url": "http://localhost:4000/commands/saythis?key=ddbe0a94&wait=false",
  "status": "running"
}
```

Then using this key you can query for the result or just by going to the `result_url`,

```
$ curl http://localhost:4000/commands/saythis?key=ddbe0a94&wait=true # wait=true so we
↳ don't need to poll
```

Returns result in JSON,

```
{
  "report": "Hello World!\n",
  "key": "ddbe0a94",
  "start_time": 1593019807.7754705,
  "end_time": 1593019807.782958,
  "process_time": 0.00748753547668457,
  "returncode": 0,
  "error": null,
}
```

1.1.5 Bonus

You can also define callback functions or use signals for reactive programming. There may be cases where the process doesn't print result to standard output but to a file/database. In such cases, you may want to intercept the future object and update its result attribute. I request you to take a look at *Examples.md* for such use-cases.

1.2 Examples

I have created some example python scripts to demonstrate various use-cases. These include extension setup as well as making test HTTP calls with python's `requests` module.

- `run_script.py`: Execute a script on a succesful POST request to an endpoint.
- `basic.py`: Map a base command to an endpoint and pass dynamic arguments to it. Can also pass in a timeout.
- `multiple_files.py`: Upload multiple files for a single command.
- `with_callback.py`: Define a callback function that executes on command/process completion.
- `with_signals.py`: Using `Flask Signals` as callback function.
- `with_decorators.py`: Shows how to apply `View Decorators` to the exposed endpoint. Useful in case you wish to apply authentication, caching, etc. to the endpoint.
- `custom_save_fn.py`: There may be cases where the process doesn't print result to standard output but to a file/database. This example shows how to pass additional context to the callback function, intercept the future object after completion and update it's result attribute before it's ready to be consumed.
- `deletion.py`: Example demonstrating how to request cancellation/deletion of an already running job.

1.3 Configuration

1.3.1 POST Request Options

One can read `post-request-schema.json` to see and understand the various *optional* tweaks which can be done when making requests to the API.

There are many *example programs* with client requests given which demonstrate these different behaviours.

1.3.2 Logging Configuration

This extension logs messages of different severity INFO, DEBUG, ERROR using the python's inbuilt `logging` module.

There are no default handlers or stream defined for the logger so it's upto the user to define them.

Here's a snippet of code that shows how you can access this extension's logger object and add a custom handler to it.

```
# python's inbuilt logging module
import logging
# get the flask_shell2http logger
logger = logging.getLogger("flask_shell2http")
# create new handler
handler = logging.StreamHandler(sys.stdout)
logger.addHandler(handler)
# log messages of severity DEBUG or lower to the console
logger.setLevel(logging.DEBUG) # this is really important!
```

Please consult the Flask's official docs on `extension logs` for more details.

API REFERENCE

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

class flask_shell2http.base_entrypoint.**Shell2HTTP**(*app=None, executor:*
Optional[flask_executor.executor.Executor] =
None, base_url_prefix: str = '/')

Flask-Shell2HTTP base entrypoint class. The only public API available to users.

app

Flask application instance.

executor

Flask-Executor instance

base_url_prefix

base prefix to apply to endpoints. Defaults to “/”.

Type str

Example:

```
app = Flask(__name__)
executor = Executor(app)
shell2http = Shell2HTTP(app=app, executor=executor, base_url_prefix="/tasks/")
```

get_registered_commands() → OrderedDict[str, str]

Most of the time you won't need this since Flask provides a `Flask.url_map` attribute.

Returns OrderedDict[uri, command] i.e. mapping of registered commands and their URLs.

init_app(*app, executor: flask_executor.executor.Executor*) → None

For use with Flask's [Application Factory](#) method.

Example:

```
executor = Executor()
shell2http = Shell2HTTP(base_url_prefix="/commands/")
app = Flask(__name__)
executor.init_app(app)
shell2http.init_app(app=app, executor=executor)
```

register_command(*endpoint: str, command_name: str, callback_fn: Optional[Callable[[Dict, concurrent.futures._base.Future], Any]] = None, decorators: List = []*) → None

Function to map a shell command to an endpoint.

Parameters

- **endpoint** (*str*) –
 - your command would live here: `{base_url_prefix}/{endpoint}`
- **command_name** (*str*) –
 - The base command which can be executed from the given endpoint.
 - If `command_name='echo'`, then all arguments passed to this endpoint will be appended to `echo`.

For example, if you pass `{ "args": ["Hello", "World"] }` in POST request, it gets converted to `echo Hello World`.
- **callback_fn** (*Callable[[Dict, Future], Any]*) –
 - **An optional function that is invoked when a requested process** to this endpoint completes execution.
 - **This is added as a** `concurrent.Future.add_done_callback(fn=callback_fn)`
 - The same callback function may be used for multiple commands.
 - if request JSON contains a `callback_context` attr, it will be passed as the first argument to this function.
- **decorators** (*List[Callable]*) –
 - A List of view decorators to apply to the endpoint.
 - *New in version v1.5.0*

Examples:

```
def my_callback_fn(context: dict, future: Future) -> None:
    print(future.result(), context)

shell2http.register_command(endpoint="echo", command_name="echo")
shell2http.register_command(
    endpoint="myawesomescript",
    command_name="./fuxsocy.py",
    callback_fn=my_callback_fn,
    decorators=[],
)
```

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

f

`flask_shell2http.base_entrpoint`, [7](#)

INDEX

A

`app` (*flask_shell2http.base_entrypoint.Shell2HTTP*
attribute), 7

B

`base_url_prefix` (*flask_shell2http.base_entrypoint.Shell2HTTP*
attribute), 7

E

`executor` (*flask_shell2http.base_entrypoint.Shell2HTTP*
attribute), 7

F

`flask_shell2http.base_entrypoint`
module, 7

G

`get_registered_commands()`
(*flask_shell2http.base_entrypoint.Shell2HTTP*
method), 7

I

`init_app()` (*flask_shell2http.base_entrypoint.Shell2HTTP*
method), 7

M

module
 flask_shell2http.base_entrypoint, 7

R

`register_command()` (*flask_shell2http.base_entrypoint.Shell2HTTP*
method), 7

S

`Shell2HTTP` (*class in flask_shell2http.base_entrypoint*),
7