PROGRAMMING

HTTP streaming of command output in Python Flask

ON 2014-03-04 • (15 COMMENTS)

I needed an endpoint that streamed the output of an external program to the remote client. In this article I describe how I did it and discuss a few issues I encountered. Note that if you just want to stream events back to the browser, I'll also cover that. An external command is just what I needed, and is the more difficult case.

A simple stream

The program below is a simple Flask server. To run it you need to `pip install flask shelljob`. Save it to a file `server.py` and then run `python server.py`.

```
1 import flask
2 from shelljob import proc
3
4 app = flask.Flask(__name__)
5
6 @app.route( '/stream' )
7 def stream():
8     g = proc.Group()
9     p = g.run( [ "bash", "-c", "for ((i=0;i<100;i=i+1)); do echo $i; sleep 1; d
```

```
    def read_process():
        while g.is_pending():
            lines = g.readlines()
            for proc, line in lines:
                yield line

    return flask.Response( read_process(), mimetype= 'text/plain' )

if __name__ == "__main__":
    app.run(debug=True)
```

Once running you can issue a curl request to see that it's streaming: curl http://127.0.0.1:5000/stream (http://127.0.0.1:5000/stream) . This assumes it started on port 5000 , just check the server output.

The streamed data in this example is a simple bash loop that generates a sequence of number and pauses in between. You'll probably want to put some useful command in its place.

The Group class is part of the shelljob (https://pypi.python.org/pypi/shelljob) package I wrote before. It takes care of the ugly streaming of data from a subprocess in Python. (I've not yet tested/ported it to Python 3. Let me know if you need that.) Here I'm just reading the data from the process and yielding it to the stream. If you're already familiar with yield that bit should be clear. If not, then I suggest you read up on yield as its too much to cover here.

Timeout?

Since you're a responsible server programmer you probably have a question about timeouts. At the moment that above loop is hopeful that the subprocess

timeout. I have taken care that readlines itself has a default timeout of two seconds and will simply return an empty list after that time. This gives you a chance to put in your own premature termination conditions.

Here is a modified read_process function that prints an interval message every 5 seconds (very roughly).

```
1  def read_process():
2      trigger_time = time.time() + 5
3      while g.is_pending():
4          lines = g.readlines()
5          for proc, line in lines:
6              yield line
7
8          now = time.time()
9          if now > trigger_time:
10             yield "*** Interval"
11             trigger_time = now + 5
```

Does that block my server?

Running the server directly will cause the loop to block the server. If you do another curl request in a second console you won't get any response. Kill the first request and then suddenly you'll start getting data. This isn't very satisfying.

Chances are you already have a solution for your server, like gunicorn or uwsgi. In case you don't I'll go over how to do it with gunicorn. First pip install gunicorn eventlet. Now don't run the server directly, instead use the command gunicorn -k eventlet

different port (usually 8000).

Unfortunately, if you do a curl request now, it won't work. It's something I don't yet understand. It only applies to external subprocess calls; if you stream internally generated data everything is fine. The fix is to add a call to monkey_patch at the top of the server code, after the other imports.

```
1 import flask
2 from shelljob import proc
3
4 import eventlet
5 eventlet.monkey_patch()
6
7 app = flask.Flask(__name__)
```

Good. Now you can issue a bunch of curl requests in multiple consoles and all of them will stream the results.

> I would be appreciative if somebody could explain the reason why I need monkey_patch here. It feels like a defect somewhere, possibility compatiblity between gunicorn and shelljob. I'd like to find the proper fix.

Now my Python process doesn't work

It seems to work until you call a Python subprocess. Now it appears to block for a long-time, possibly until

— or timeout.

This happens because Python is trying to be *clever*. If the output of the command is not a console it goes into a buffered output mode. So instead of flushing on every line of output it accumulates a lot more text before anything is actually written.

To get around this the python process can be launched with a `-u` parameter. This turns off the buffering.

I made a simple external script to test this, called `slow.py`.

```
1 import time
2
3 for i in range(0,100):
4     print(i)
5     time.sleep(1)
```

In the server I changed the process I run to `p = g.run( [ "python", "-u", "slow.py" ] )`. A curl request now reports the lines as we hoped. To see the broken behaviour just remove the `-u` part.

Stream events to a browser

The above can be combined with an `EventSource` in a browser (at least those that support it). First create the following `page.html` file.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
```

```
5     var source = new EventSource( '/stream' );
6     source.onmessage = function(event) {
7         document.getElementById("output").innerHTML += event.data + "
8     }
9     </script>
10 </head>
11 <body>
12     <h1>Output</h1>
13     <div id="output"></div>
14 </body>
15 </html>
```

A few lines in the server must be modified. The `yield` line needs to format a message response. This is merely a `data:` header and two line-feeds. The response must be marked as `text/event-stream` to tell the browser it's a stream. Finally we need an endpoint to serve the HTML page — don't load the file directly in the browser as cross-domain restrictions will break it. The below is the modified server.

```
1  import flask
2  from shelljob import proc
3
4  import eventlet
5  eventlet.monkey_patch()
6
7  app = flask.Flask(__name__)
8
9  @app.route( '/stream' )
10 def stream():
11     g = proc.Group()
12     p = g.run( [ "bash", "-c", "for ((i=0;i<100;i=i+1)); do echo $i; sleep 1; d
13
14     def read_process():
15         while g.is_pending():
16             lines = g.readlines()
17             for proc, line in lines:
18                 yield "data:" + line + "\n\n"
19
```

```
21
22 @app.route('/page')
23 def get_page():
24     return flask.send_file('page.html')
25
26 if __name__ == "__main__":
27     app.run(debug=True)
```

Navigate to http://127.0.0.1:8000/page (http://127.0.0.1:8000/page) in your browser and the stream should start appearing. If you have the gunicorn stuff setup correctly you should be able to open multiple tabs. Each load of this page will have its own stream.

Also interesting, EventSource restarts the connection when it is done. Once you reach the final number it will just start counting again. Try reducing the bash loop to see this effect. I don't know much else about event sources, so you're on your own from here.

Per your question:

Reply

"I would be appreciative if somebody could explain the reason why I need monkey_patch here. It feels like a defect somewhere, possibility compatiblity between gunicorn and shelljob. I'd like to find the proper fix."

monkey_patch does things like change paths to library calls, for example if you were using gevents, time.sleep(), it would be changed to gevent.sleep().
I think underneath monkey_patch deals with blocking versus nonblocking library calls.
time.sleep() inside is a simple loop, gevent.sleep() syncs with the master thread.

> I thought the combination of Gunicron with Flask would automatically do the monkey_patch however. I assume this since the HTTP layer is patched automatically.

In my limited experience, use of monkey_patch is arbitrary. Not sure why

Reply

I tried substituting -k gevent for -k eventlet but curl only responded in one terminal. When I then monkey_patched gevent, curl did not respond in any terminal.

> I'm not entirely happy with the Gunicorn stack. From experiece I know a lot of combinations of workers and configuration just don't work correctly.

Dear mororay,

Reply

Firstly, thanks for your script

My issue is: I want to input variable to your stream() function, when I tried to do that, I always received error "stream() takes exactly 1 argument (0 given)"

In my opinion, I think I will edit your page.html or input via url, but I don't know how to do that.

Would you please point me some key? Thanks for your support.

suker200

About this:

Reply

> It feels like a defect somewhere, possibility compatiblity between gunicorn and shelljob. I'd like to find the proper fix.

coding mainly), intermediate files, and subprocess. Exact same problem seems to happen – first curl gets streaming output, second curl hangs until first finishes

Hi! Reply
I would like to have stream output inline, preferably inside some html tag and with blazing fast speed of the output update. Do you think something like this can be made with flask ?

> You'll want to loook at the section on "Stream events to the browser". This is an effective way to get the browser to update as data comes back.
>
> You could also create your own HTTP reader in JS and continually add it to the text/html of an element. I don't know for sure how the buffering affects this though.

Thanks. I implemented this yesterday evening using socketIO. It all works smoothly so now I have to figure out how you can stop and resume stream where you left off but that is another story.

Thanks for your great work!I followed your instructions and implements have a problem here,when i nav to localhost:5000/page,the webpage keeps updating all the time,that is,it keeps print 0~100 as if the for loop is running all the time.However,when nav to localhost:5000/stream,it seems the loop only exec once.where am wrong ? Reply

> The browser sees this as a permanent stream and expects a long-term connection. If it loses the connection it will simply connect again. In our simple case our server will just print the numbers again.

If i want run only one command (infinity) and send same data to multiple how i do? Reply

> That will be a bit more setup. You'll need one thread (or process) that reads from the command and publishes to a 1:N queue. Each receiver then listens to that queue and writes to the output stream.
>
> Since Python in this configuration could be multitasking you'll actually need an interprocess queue for this. I'm not sure which ones are easy to use from Python.

I have no idea how to do this, I asked some people but it seems to be something complicated https://stackoverflow.com/questions/45361520/flask-and-subprocess-popen-send-same-data-to-multiple-users/45363418 (https://stackoverflow.com/questions/45361520/flask-and-subprocess-popen-send-same-data-to-multiple-users/45363418)

Hi! Thanks for your script. But I try to change the command, and it give a Reply
while I open localhost:8000/page
b'tail: invalid number of lines: \xe2\x80\x98r\xe2\x80\x99\n'