**Recipe: Imagesize STP**

This tutorial will walk you through the process of building your very first STP. By the end of this tutorial, you will have an STP that returns the image size of provided images.

**Prerequisites**

* Basic Python knowledge
* Python3
* Python Virtual Environment (pip)
* Network connectivity to [ott-lhv3-lnx.cisco.com](http://ott-lhv3-lnx.cisco.com/)

**STEP 1: Build a Web Server**

Python provides a variety of modules that makes building an HTTP server a snap. In this example we’ll use Flask. Flask is a microframework for Python, which makes getting a Web Server up and running very easy.

To setup

$ pip install Flask

Next open a file called run\_api.py and paste for the following:

**from** flask **import** Flask, request

app **=** Flask(\_\_name\_\_)

@app.route("/runTest", methods**=**['GET', 'POST'])

**def** **runTest**():

**if** request**.**method **==** 'GET':

**return** "Hello World!"

**if** \_\_name\_\_ **==** "\_\_main\_\_":

app**.**run(host**=**'0.0.0.0', port**=**8005)

You should see:

<16> :: python run\_api.py

\* Running on <http://0.0.0.0:8005/> **(**Press CTRL+C to quit**)**

Now open a browser and navigate to http://<yourserver>:8005/runTest

Let’s do the exact same thing but use a handy little utility called curl. curl is a tool to transfer data from or to a server, using a host of different protocols, one of which is HTTP. We’ll be using CURL throughout this tutorial so give it a try.

$ curl -i [http://127.0.0.1:8005/runTest](http://127.0.0.1:8005/runTestHTTP/1.0" \t "_blank)

[HTTP/1.0](http://127.0.0.1:8005/runTestHTTP/1.0" \t "_blank) 200 OK

Content-Type: text/html; charset**=**utf-8

Content-Length: 12

Server: Werkzeug/0.11.10 Python/3.4.1

Date: Thu, 03 Nov 2016 19:54:29 GMT

Hello World!

**Tip**

If you have having troubles connecting to your HTTP server, be sure to disable any proxy you might have set. In bash, you can use the command

$ unset http\_proxy

$ unset HTTP\_PROXY

Congratulations, you now have a working web server. Making good progress, egh?!

**STEP 2: Implement the Incoming API**

Let’s process the Lighthouse incoming runTest API call.

**from** flask **import** Flask, request

app **=** Flask(\_\_name\_\_)

api\_ver **=** 2

v2\_required\_fields **=** ['version', 'caller\_name', 'caller\_context', 'stp',

'branch', 'target', 'refpoint', 'reason', 'image\_paths']

@app.route("/runTest", methods**=**['GET', 'POST'])

**def** **runTest**():

**if** request**.**method **==** 'GET':

**return** "Hello World!"

*# HTTP POST processing starts here*

**try**:

data **=** request**.**get\_json()

**except**:

**return** 'Unable to extract JSON data from message. Please

check JSON format', 400

*# Validate all required fields are present*

missing\_fields **=** set(v2\_required\_fields) **-** set(data**.**keys())

**if** len(missing\_fields) **!=** 0:

**return** 'Missing fields: %s' **%** missing\_fields, 400

**return** str(data)

**if** \_\_name\_\_ **==** "\_\_main\_\_":

app**.**run(host**=**'0.0.0.0', port**=**8005)

Now let’s test this using CURL. We will need to construct a CURL command with some data in it.

$ curl -d '{"version": 2, "caller\_name": "demo", "caller\_context": 1234, "stp": "tutorialSTP", "branch": "polaris\_dev", "target": "edison\_super", "refpoint": 1234, "image\_paths": {"bin": "ott-mcp-nfs1:/local\_nfs/polaris\_lite/saqib/prod/images/edison\_super/polaris\_dev/375/polaris\_dev-37506-cat3k\_caa-universalk9.bin"}}' -H 'Content-Type: application/json' <http://127.0.0.1:8005/runTest>

**{**'version': 2, 'target': 'edison\_super', 'caller\_context': 1234, 'caller\_name': 'demo', 'refpoint': 1234, 'stp': 'tutorialSTP', 'image\_paths': **{**'bin': 'ott-mcp-nfs1:/local\_nfs/polaris\_lite/saqib/prod/images/edison\_super/polaris\_dev/375/polaris\_dev-37506-cat3k\_caa-universalk9.bin'**}**, 'branch': 'polaris\_dev'**}**

Excellent, we learned how to extract parameters from the body of an HTTP POST message. We also learned how to construct a curl command to test our STP application. Next we learn how to build an image size test.

**STEP 3: Build an Imagesize Test**

HTTP requests need to be responded to quickly however execution of tests are slow, think milliseconds vs. minutes to hours. So its best to split the application into multiple concurrent proceses or threads and communicate between them through Queues. Don’t stress, this sounds more complicated than it really is.

When the application starts, it will create a requests Queue and start an imagesize thread. The job of the imagesize thread is to monitor the requests queue and if an item is found then it will start an imagesize\_test. The main processing loop handles the Flask application which is listening for HTTP POST runTest requests from Lighthouse. For now we will simply print the file size in bytes to stdout.

**Note**

A note about the Queue. Lighthouse will throttle requests to the STP based on the number testbeds and max runs per day parameters specified during STP registration. So the queuing used here is simply to communicate between the main process and the thread. Since we will register this STP with numbertestbeds=1, we should only have at most one item in our Queue.

**from** flask **import** Flask, request

**import** threading

**import** queue

**import** time

**import** subprocess

**import** os

**import** logging

LOGGER **=** logging**.**getLogger(\_\_name\_\_)

logging**.**basicConfig(

*# filename = 'tmp.log',*

level **=** logging**.**DEBUG,

format **=** '%(levelname)-8s | %(threadName)-10s | %(asctime)s | %(message)s',

)

app **=** Flask(\_\_name\_\_)

api\_ver **=** 2

required\_fields **=** ['version', 'caller\_name', 'caller\_context', 'stp',

'branch', 'target', 'refpoint', 'reason', 'image\_paths']

MAX\_QUEUE\_DEPTH **=** 1

**def** **imagesize\_test**(data):

''' Returns the image size in bytes of the bin file in

data['image\_paths'] '''

LOGGER**.**debug(data)

**try**:

hostpath **=** data['image\_paths']['bin']

**except** **KeyError**:

LOGGER**.**error('Error: Unable to find bin in image\_paths')

**return** None

host, path **=** hostpath**.**split(':')

*# Assumes passwordless ssh to host*

cmd **=** 'ssh %s ls -l %s' **%** (host, path)

LOGGER**.**info(cmd)

FNULL **=** open(os**.**devnull, 'w')

**try**:

popen **=** subprocess**.**Popen(cmd**.**split(' '), stdout**=**subprocess**.**PIPE,

stderr**=**FNULL)

output **=** popen**.**communicate()[0]

bytes **=** output**.**decode('utf-8')**.**split()[4]

LOGGER**.**debug('output: %s, bytes: %s' **%** (output, bytes))

**except**:

LOGGER**.**error('Error: Unable to get file size')

**return** None

**return** bytes

**def** **imagesize**():

''' Imagesize thread. Monitors requests queue and executes

imagesize\_test if item found. For now we will print the

output '''

**global** test\_running

**while** True:

**while** **not** requests\_q**.**empty():

test\_running **=** True

data **=** requests\_q**.**get()

size **=** imagesize\_test(data)

LOGGER**.**info('Image is %s bytes' **%** size)

test\_running **=** False

time**.**sleep(5)

@app.route("/runTest", methods**=**['GET', 'POST'])

**def** **runTest**():

**if** request**.**method **==** 'GET':

**return** "Hello World!"

**try**:

data **=** request**.**get\_json()

**except**:

**return** 'Unable to extract JSON data from message. Please check JSON format', 400

*# Validate all required fields are present*

missing\_fields **=** set(required\_fields) **-** set(data**.**keys())

**if** len(missing\_fields) **!=** 0:

**return** 'Missing fields: %s' **%** missing\_fields, 400

*# Add to Queue*

**if** requests\_q**.**qsize() **<** MAX\_QUEUE\_DEPTH **and** **not** test\_running:

requests\_q**.**put(data)

**else**:

**return** 'STP Queue is full', 500

**return** 'OK'

**if** \_\_name\_\_ **==** "\_\_main\_\_":

test\_running **=** False

requests\_q **=** queue**.**Queue()

t **=** threading**.**Thread(target**=**imagesize)

t**.**start()

app**.**run(host**=**'0.0.0.0', port**=**8005)

**Tip**

To kill the running application, you need to press Ctrl-C twice now that we have a running thread and a main application

Give it a try by sending the curl command in the previous step. You should see that the image size is 617687867.

We’re almost there, now we need to publish the result back to Lighthouse.

**STEP 4: Publish test results to Lighthouse**

Test results published using the putTestResults API. We will use the Python library requests to construct the HTTP PUT message.

Here we add the publish\_results function that tries to write to Lighthouse.

**from** flask **import** Flask, request

**import** threading

**import** queue

**import** time

**import** subprocess

**import** os

**import** requests

**import** logging

LOGGER **=** logging**.**getLogger(\_\_name\_\_)

logging**.**basicConfig(

*# filename = 'tmp.log',*

level **=** logging**.**DEBUG,

format **=** '%(levelname)-8s | %(threadName)-10s | %(asctime)s | %(message)s',

)

app **=** Flask(\_\_name\_\_)

api\_ver **=** 2

required\_fields **=** ['version', 'caller\_name', 'caller\_context', 'stp',

'branch', 'target', 'refpoint', 'reason', 'image\_paths']

MAX\_QUEUE\_DEPTH **=** 1

LH\_URL **=** '<http://ott-lhv3-lnx:8005/stp/putTestResults>'

**def** **publish\_results**(data, size):

''' Sends HTTP PUT message to Lighthouse with test results '''

**try**:

test\_result **=** {

'testcase\_name': 'imagesize',

'type': 'int',

'value': int(size)

}

**except** **ValueError**:

**print**('Error: could not convert size to integer')

**return**

payload **=** {

'version': api\_ver,

'caller\_name': data['caller\_name'],

'caller\_context': data['caller\_context'],

'stp': data['stp'],

'branch': data['branch'],

'target': data['target'],

'refpoint': data['refpoint'],

'reason': data['reason'],

'test\_results': [test\_result],

}

LOGGER**.**info('Publishing Result: %s', payload)

r **=** requests**.**put(LH\_URL, json**=**payload)

**if** **not** r**.**ok:

LOGGER**.**error('Error: Unable to post results to Lighthouse')

LOGGER**.**error('%s: %s', r**.**status\_code, r**.**text)

**def** **imagesize\_test**(data):

''' Returns the image size in bytes of the bin file in

data['image\_paths'] '''

LOGGER**.**debug(data)

**try**:

hostpath **=** data['image\_paths']['bin']

**except** **KeyError**:

LOGGER**.**error('Error: Unable to find bin in image\_paths')

**return** None

host, path **=** hostpath**.**split(':')

*# Assumes passwordless ssh to host*

cmd **=** 'ssh %s ls -l %s' **%** (host, path)

LOGGER**.**info(cmd)

FNULL **=** open(os**.**devnull, 'w')

**try**:

popen **=** subprocess**.**Popen(cmd**.**split(' '), stdout**=**subprocess**.**PIPE,

stderr**=**FNULL)

output **=** popen**.**communicate()[0]

bytes **=** output**.**decode('utf-8')**.**split()[4]

LOGGER**.**debug('output: %s, bytes: %s' **%** (output, bytes))

**except**:

LOGGER**.**error('Error: Unable to get file size')

**return** None

**return** bytes

**def** **imagesize**():

''' Imagesize thread. Monitors requests queue and executes

imagesize\_test if item found. For now we will print the

output '''

**global** test\_running

**while** True:

**while** **not** requests\_q**.**empty():

test\_running **=** True

data **=** requests\_q**.**get()

size **=** imagesize\_test(data)

publish\_results(data, size)

test\_running **=** False

time**.**sleep(5)

@app.route("/runTest", methods**=**['GET', 'POST'])

**def** **runTest**():

**if** request**.**method **==** 'GET':

**return** "Hello World!"

**try**:

data **=** request**.**get\_json()

**except**:

**return** 'Unable to extract JSON data from message. Please check JSON format', 400

*# Validate all required fields are present*

missing\_fields **=** set(required\_fields) **-** set(data**.**keys())

**if** len(missing\_fields) **!=** 0:

**return** 'Missing fields: %s' **%** missing\_fields, 400

*# Add to Queue*

**if** requests\_q**.**qsize() **<** MAX\_QUEUE\_DEPTH **and** **not** test\_running:

requests\_q**.**put(data)

**else**:

**return** 'STP Queue is full', 500

**return** 'OK'

**if** \_\_name\_\_ **==** "\_\_main\_\_":

test\_running **=** False

requests\_q **=** queue**.**Queue()

t **=** threading**.**Thread(target**=**imagesize)

t**.**start()

app**.**run(host**=**'0.0.0.0', port**=**8005)

Give it a try. You should see the following failure because we haven’t registered the STP with the Analytics Engine. The response you should now see:

$ python run\_api\_step4.py

\* Running on <http://0.0.0.0:8005/> **(**Press CTRL+C to quit**)**

127.0.0.1 - - **[**03/Nov/2016 21:29:11**]** "POST /runTest HTTP/1.1" 200 -

**{**'caller\_name': 'demo', 'version': 2, 'branch': 'polaris\_dev', 'target': 'edison\_super ', 'stp': 'tutorialSTP', 'caller\_context': 1234, 'refpoint': 1234, 'image\_paths': **{**'bin': 'ott-mcp-nfs1:/local\_nfs/polaris\_lite/saqib/prod/images/edison\_super/polaris\_dev/375/polaris\_dev-37506-cat3k\_caa-universalk9.bin'**}}**

ssh ott-mcp-nfs1 ls -l /local\_nfs/polaris\_lite/saqib/prod/images/edison\_super/polaris\_dev/375/polaris\_dev-37506-cat3k\_caa-universalk9.bin

output: b'-rw-r--r-- 1 cgathuru eng 617687867 Nov 2 17:45 /local\_nfs/polaris\_lite/saqib/prod/images/edison\_super/polaris\_dev/375/polaris\_dev-37506-cat3k\_caa-universalk9.bin\n', bytes: 437266078

Error: Unable to post results to Lighthouse

404: STP tutorialSTP does not exist, please register STP

**Tip**

If you see a proxy error, be sure to disable any proxy you might have set in the window running run\_api.py. In bash, you can use the command to disable the proxy.

$ unset http\_proxy

$ unset HTTP\_PROXY

**STEP 5: Register Your STP with Lighthouse**

Open a browser and navigate to [http://lighthouse](http://lighthouse/) and under the Admin tab click ‘Register STP’. Fill out the fields similar to the following with your user ID.

Wait a few minutes and soon you will start to see test requests from the Analytics Engine. If all goes well, you should see your results under the ‘View Data’ tab of [http://lighthouse](http://lighthouse/).

**STEP 6: View Data**

Click the View Data tab on [http://lighthouse](http://lighthouse/) and navigate to your STP to view your data.

[Next](https://lighthouse.cisco.com/docs/ssr_recipe.html)[Previous](https://lighthouse.cisco.com/docs/register_stp.html)