Get Hands-On with HTTP

Estimated Time: **1 hour**

In this unit you'll be building dynamic web applications in Python using Flask and SQLAlchemy. Before looking at the details of using Flask, in this assignment you'll be taking a hands-on look at the protocol which forms the backbone of the web: HTTP.

Making HTTP requests using cURL

HTTP is the protocol which powers the web. Every time you visit a website you are making HTTP requests, and receiving responses from a server. To explore this you can use a little program called cURL. According to the manual, " `curl` is a tool to transfer data from or to a server" using a variety of protocols. One of those protocols is HTTP. And that's all it really is: a protocol for transferring data. Open up a terminal, and let's write your first curl command:

```
$ curl http://httpbin.org/ip
{
  "origin": "54.235.221.217"
}
```

This is a JSON-encoded response with the IP address of the origin of the request. So, what happened there?

This command sends a request, using the HTTP protocol, to the server at the location `httpbin.org` and requests the `/ip` endpoint. Specifically, the location of `httpbin.org` is 107.20.202.171; a Domain Name Server (DNS) points us to that numeric IP address when we send the request to `httpbin.org`. The DNS acts a bit like the white pages – it tells you the physical location of a server (the IP address) when you look up a website's URL.

On the httpbin server, there is a program setup to respond whenever a client (that's you!) sends a request. In the case of the `/ip` endpoint, the server simply reads the IP address of the client and sends it back in JSON format. `httpbin.org` is actually an open source Python program, and you can see the specific code for generating this response here. If you're confused by what that code means right now don't worry! You'll understand it by the end of the course when you learn to make Flask apps of your own.

This example illustrates a dynamic site: the response depends on information provided by the client; in this case your IP address. This is as opposed to a static site, which simply responds with pre-generated HTML files for viewing or downloading. It also shows the foundational architecture of the Internet. There is a **client**, in this case your computer*, sending a request to a **server**, in this case `httpbin.org`. Whenever you open a web page or do anything on the Internet that loads data to or from somewhere else, there is a client-server interaction going on somewhere.

* Technically, the client in this request is your Cloud9 workspace, but it's all the same to your in-browser terminal.

HTTP headers

HTTP as a protocol does more than just send IP addresses, however. Try this request:

```
$ curl http://httpbin.org
```

This requests the homepage of `httpbin.org` . Curl automatically outputs the body of the response to your terminal. You should see a bunch of HTML markup scroll on the screen. In this case, the server at `httpbin.org` was not configured to send a JSON-encoded response, but instead sent an HTML-encoded response. Your terminal doesn't know how to display HTML/CSS, but your browser does! Try opening http://httpbin.org and http://httpbin.org/ip in your browser. You'll see that the root endpoint displays a lovely web page, while the `/ip` endpoint just displays the same thing as your terminal.

But how does your browser know to display the HTML as a web page, and the JSON as just text? Well, HTTP also includes headers in every request. These headers contain fields and values which the client uses to interpret the body of the request. Try each of these commands separately:

```
$ curl -I http://httpbin.org
...
$ curl -I http://httpbin.org/ip
```

The `-I` switch tells curl to only fetch the HTTP-header set by the server. Take a look at the `Content-Type` field. The first command should have responded with `text/html` , the second with `application/json` . This field tells the client how to interpret the information, so the browser knows whether it should render the body of the request as an HTML web page, or simply display the text as JSON-encoded data.

Okay, so now you understand how to request and receive information from a server. But the Internet (and HTTP) does more than just look up info. You can send commands with HTTP to tell the server what to do. These are called the CRUD commands because they allow you to **c**reate, **r**ead, **u**pdate, and **d**elete information on the server. The specific HTTP commands are:

- **POST**: *create* a new record or piece of information on the server.
- **GET**: *retrieve* information from the server.
- **PUT**: *update* some piece of information.
- **DELETE**: *destroy* the specified information on the server.

HTTP has more commands, but these are the basics. Note that all of the commands are verbs; that is because these commands **do something**. Notice how HTTP uses the same CRUD design pattern as the SQL database you looked at in the previous unit. To learn more about these other types of requests, go to RequestBin in your browser and click "Create a RequestBin."

When you don't provide any arguments to curl it defaults to making a GET request. You can make your first POST request like this, replacing "UIDHERE" with the specific endpoint that RequestBin gives you:

```
$ curl -X POST -d "fizz=buzz" http://requestb.in/UIDHERE
ok
```

If all went well, the RequestBin server responded with `ok` . If you refresh the RequestBin URL in your browser, you'll see a webpage with all of the details of your POST request. Although it's not displayed in this example, the OK response from the server will also have a status code attached to it -- in this case 200.

It's useful to know the different HTTP codes, which are categorized by the hundreds:

- 1xx: Informational
- 2xx: Successful
- 3xx: Redirection
- 4xx: Client Error
- 5xx: Server Error

These general codes are all you need to know for basic debugging purposes. The specific codes can be looked up at the W3's HTTP Spec, or via Google.

Try It!

Use curl to explore:

- What headers your client sends to httpbin. If you make a GET request to http://httpbin.org/headers, httpbin will return JSON containing these headers.
- What status code is used when httpbin redirects your client -- view the headers returned when making a GET request to http://httpbin.org/redirect-to?url=http://google.com
- What headers some of your favourite websites set. Try looking up some of the more interesting sounding ones to see what they are used for.

Final words

This was a basic introduction to HTTP and client-server interaction. There is a great deal more which you could learn about HTTP; the specification runs to hundreds of pages of text. However, it's often a good idea to dive in and start exploring these concepts hands-on. Lucky for you, in the next assignment you get to build a tasty web application with Flask. Code responsibly.

What is HTTP?

> *HTTP (hypertext transfer protocol) is a protocol governing how messages sent between client and server should be formatted.*

What's the difference between an HTTP response header and body?

> *The header contains metadata about the request, such as content type, size, and response status code. The body (if any) contains the requested resource: for instance, an HTML page or image file.*

How can you use Google Chrome Developer tools to inspect HTTP request and response objects?

> *Open the Network tab in Chrome Developer tools.*

What is a DNS and how does it relate to URLs?

> *A DNS is a Domain Name Server. It is like a phone book where you look up a domain name and get back its numeric IP address, which is where your HTTP request actually is made to.*

What is the role of the server in a *static* vs. *dynamic site*?

> *For a static site, the server just supplies unchanging assets that remain the same no matter when the request is made. For a dynamic site, the server is responsible for running a program that dynamically generates the file(s) that get returned to the browser in the response object.*

☆ ☆ ☆ ☆ ☆ · Report a typo or other issue

✓ Mark as completed

Previous

Next