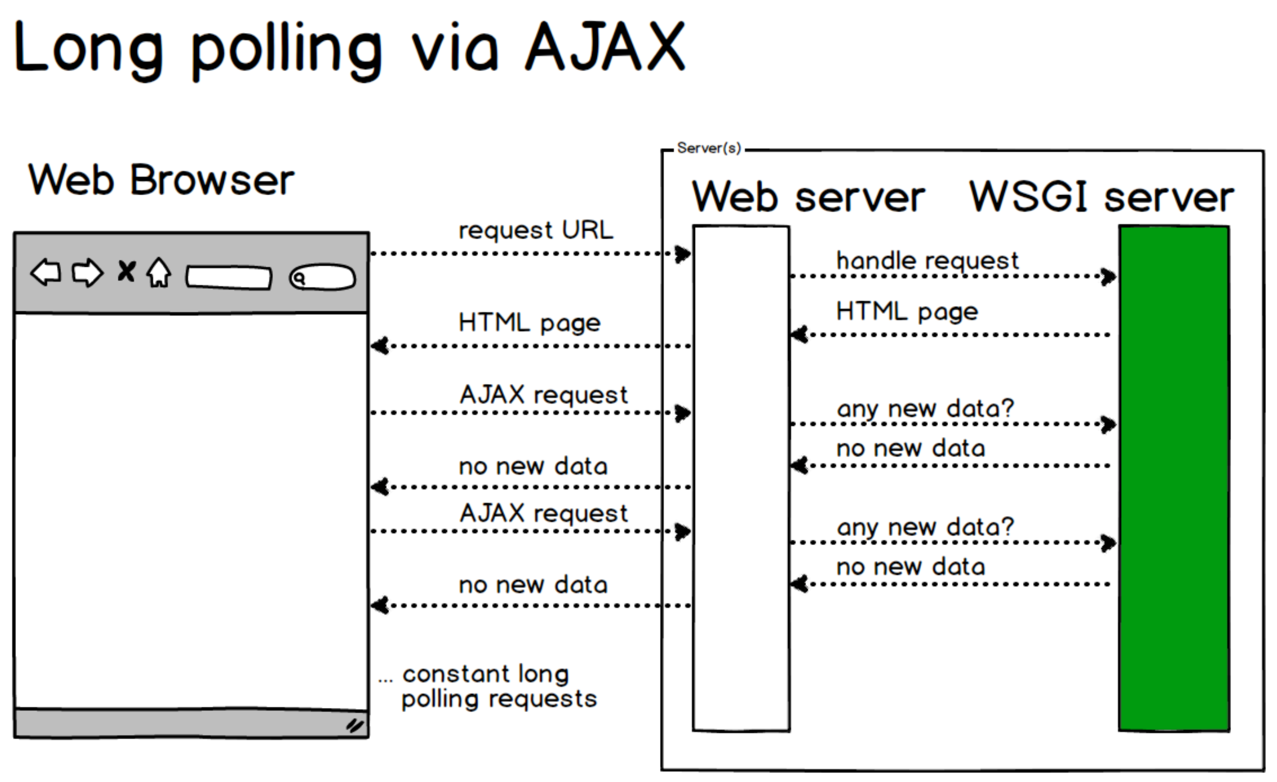
WebSockets

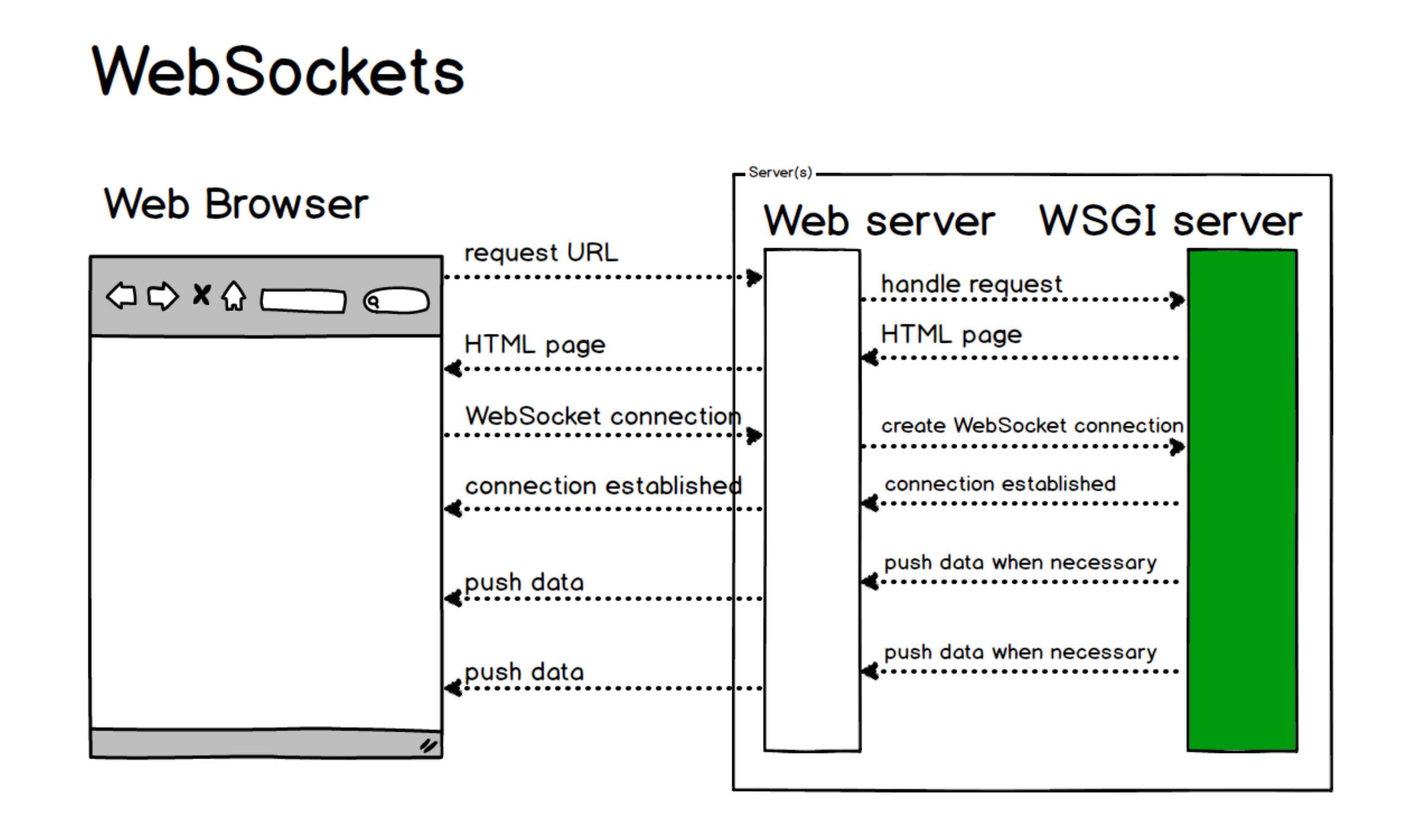
A WebSocket is a standard protocol for two-way data transfer between a client and server. The WebSockets protocol does not run over HTTP, instead it is a separate implementation on top of TCP.

Why use WebSockets?

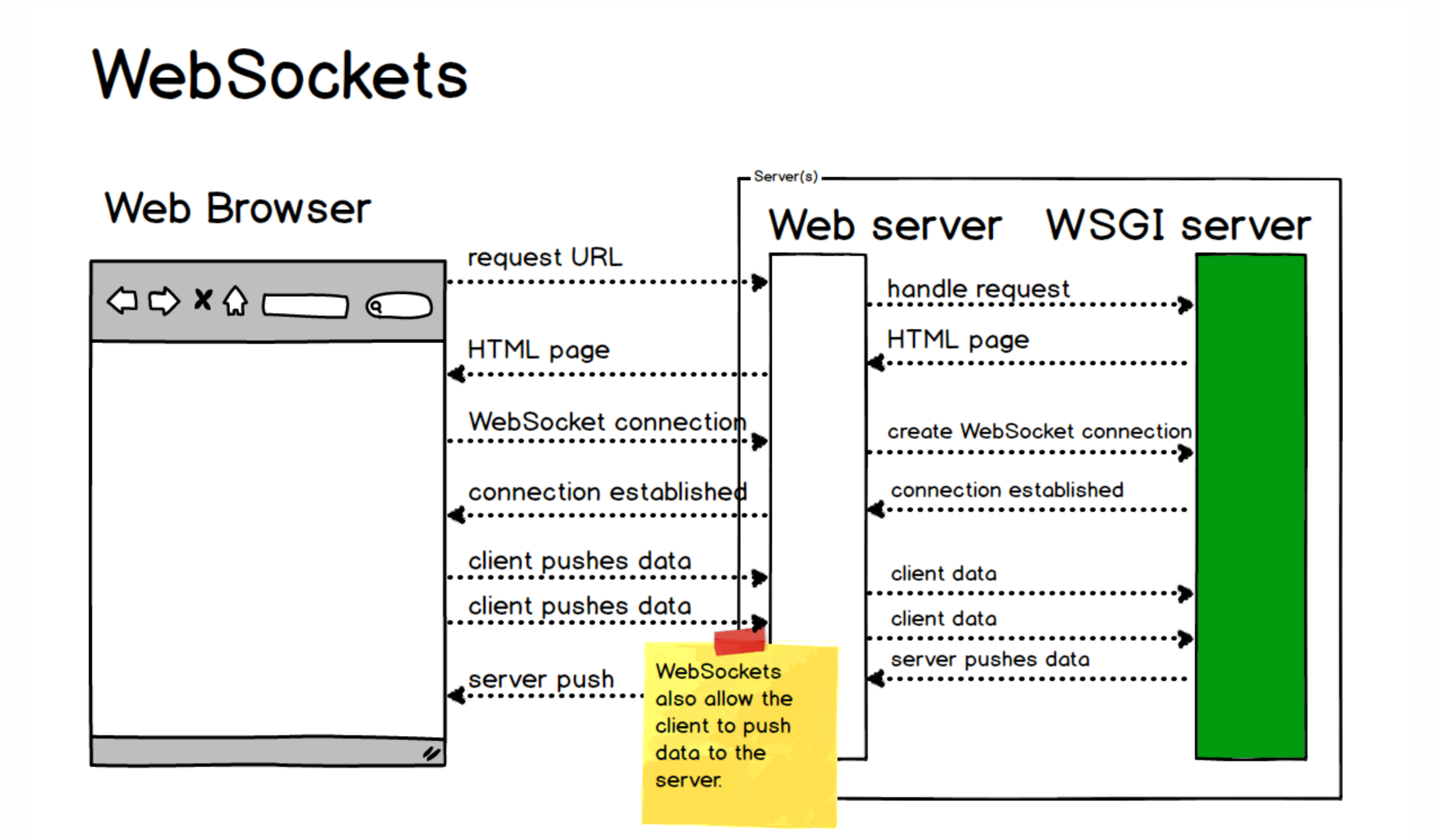
A WebSocket connection allows full-duplex communication between a client and server so that either side can push data to the other through an established connection. The reason why WebSockets, along with the related technologies of Server-sent Events (SSE) and WebRTC data channels, are important is that HTTP is not meant for keeping open a connection for the server to frequently push data to a web browser. Previously, most web applications would implement long polling via frequent Asynchronous JavaScript and XML (AJAX) requests as shown in the below diagram.



Server push is more efficient and scalable than long polling because the web browser does not have to constantly ask for updates through a stream of AJAX requests.



While the above diagram shows a server pushing data to the client, WebSockets is a full-duplex connection so the client can also push data to the server as shown in the diagram below.



The WebSockets approach for server- and client-pushed updates works well for certain categories of web applications such as chat room, which is why that's often an example application for a WebSocket library.

Implementing WebSockets

Both the web browser and the server must implement the WebSockets protocol to establish and maintain the connection. There are important implications for servers since WebSockets connections are long lived, unlike typical HTTP connections.

A multi-threaded or multi-process based server cannot scale appropriately for WebSockets because it is designed to open a connection, handle a request as quickly as possible and then close the connection. An asynchronous server such as Tornado or Green Unicorn monkey patched with gevent is necessary for any practical WebSockets server-side implementation.

On the client side, it is not necessary to use a JavaScript library for WebSockets. Web browsers that implement WebSockets will expose all necessary client-side functionality through the WebSockets object.

However, a JavaScript wrapper library can make a developer's life easier by implementing graceful degradation (often falling back to long-polling when WebSockets are not supported) and by providing a wrapper around browser-specific WebSocket quirks. Examples of JavaScript client libraries and Python implementations are shown in a section below.

Nginx WebSocket proxying

Nginx officially supports WebSocket proxying as of version 1.3. However, you have to configure the Upgrade and Connection headers to ensure requests are passed through Nginx to your WSGI server. It can be tricky to set this up the first time.

Here are the configuration settings I use in my Nginx file as part of my WebSockets proxy.

```
# this is where my WSGI server sits answering only on localhost
# usually this is Gunicorn monkey patched with gevent
upstream app_server_wsgiapp {
  server localhost:5000 fail_timeout=0;
}

server {

  # typical web server configuration goes here

  # this section is specific to the WebSockets proxying
  location /socket.io {
    proxy_pass http://app_server_wsgiapp/socket.io;
    proxy_redirect off;

    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_read_timeout 600;
    }
}
```

Note if you run into any issues with the above example configuration you'll want to scope out the official HTTP proxy module documentation.

The following resources are also helpful for setting up the configuration properly.

- Nginx has an official page for WebSocket proxying.
- Proxying WebSockets with Nginx shows how to proxy with Socket.io.

Python WebSockets implementations

The following projects either implement WebSockets in Python or provide example code you can follow to use WebSockets in your own projects.

- Autobahn uses Twisted and asyncio to create the server-side WebSockets component while AutobahnJS assists on the client web browser side.
- Django Channels is built on top of WebSockets and is easy to integrate with existing or new Django projects.
- Flask-SocketIO is a Flask extension that relies upon eventlet or gevent to create server-side WebSockets connections.
- websocket-client provides low level APIs for WebSockets and works with both Python 2 and 3.
- Crossbar.io builds upon Autobahn and includes a separate server for handling the WebSockets connections if desired by the web app developer.

JavaScript client libraries

JavaScript is used to create the client side of the WebSocket connection because the client is typically a web browser. While you do not need one of these client-side libraries to use WebSockets, they are useful for minimizing the amount of boilerplate code to establish and handle your connections.

- Socket.io's client side JavaScript library can be used to connect to a server side WebSockets implementation.
- web-socket-js is a Flash-based client-side WebSockets implementation.
- AutobahnJS assists on the client web browser side.

Example code for WebSockets in Python

There are numerous Python-based implementations for WebSockets so sometimes it's just easiest to examine an example so you can build something for your own project.

- The Flask-SocketIO project has a chat web application that demos sending server generated events as well as input from users via a text box input on a form.
- The python-websockets-example contains code to create a simple web application that provides WebSockets using Flask, Flask-SocketIO and gevent.

Python-specific WebSockets resources

- The "Async Python Web Apps with WebSockets & gevent" talk I gave at San Francisco Python in January 2015 is a live-coded example Flask web app implementation that allows the audience to interact with WebSockets as I built out the application.
- Real-time in Python provides Python-specific context for how the server push updates were implemented in the past and how Python's tools have evolved to perform server side updates.
- websockets is a WebSockets implementation for Python 3.3+ written with the asyncio module.
- Speeding up Websockets 60X is a cool experiment in coding loops different ways to eek out more performance from WebSockets connections. It is unclear how generalizable the results in the blog post are to other programs but it is a good example of how tweaking and tuning can produce outsized returns in some applications.
- Adding Real Time to Django Applications shows how to use Django and Crossbar.io to implement a publish/subscribe feature in the application.
- Async with Bottle shows how to use greenlets to support WebSockets with the Bottle web framework.
- If you're deploying to Heroku, there is a specific WebSockets guide for getting your Python application up and running.
- The Reddit thread for this page has some interesting comments on what's missing from the above content that I'm working to address.
- Creating Websockets Chat with Python shows code for a Twisted server that handles WebSockets connections on the server side along with the JavaScript code for the client side.
- Synchronize clients of a Flask application with WebSockets is a quick tutorial showing how to use Flask, the Flask-SocketIO extension and Socket.IO to update

values between web browser clients when changes occur.

General WebSockets resources

WebSockets have wide browser support and therefore many web frameworks across all major programming languages have libraries to make creating WebSockets connections easier. The following resources are general guides and tutorials that provide context for the protocol without getting into the weeds of how to use WebSockets in Python.

- The official W3C candidate draft for WebSockets API and the working draft for WebSockets are good reference material but can be tough for those new to the WebSockets concepts. I recommend reading the working draft after looking through some of the more beginner-friendly resources list below.
- WebSockets 101 by Armin Ronacher provides a detailed assessment of the subpar state of HTTP proxying in regards to WebSockets. He also discusses the complexities of the WebSockets protocol including the packet implementation.
- The "Can I Use?" website has a handy WebSockets reference chart for which web browsers and specific versions support WebSockets.
- WebSockets for fun and profit has a nice concise overview of WebSockets alternatives like long polling and Server-Sent Events (SSE) before it goes into a WebSockets example that includes JavaScript code for the client-side implementation.
- Mozilla's Developer Resources for WebSockets is a good place to find documentation and tools for developing with WebSockets.
- WebSockets from Scratch gives a nice overview of the protocol then shows how the lower-level pieces work with WebSockets, which are often a black box to developers who only use libraries like Socket.IO.
- websocketd is a WebSockets server aiming to be the "CGI of WebSockets". Worth a look.
- How WebSockets Work – With Socket.io Demo walks through the HTTP-to-WebSocket upgrade handshake and explains a bit about how WebSockets work. The provided code is NodeJS on the backend but the SocketIO client side JavaScript is the same as you would implement in a Python-backed web application.
- Can WebSockets and HTTP/2 Co-exist? compares and contrasts the two protocols and shows how they have differences which will likely lead to WebSockets sticking around for awhile longer.

- A Brief Introduction to WebSockets and Socket.io by Saleh Hamadeh is a video on WebSockets basics and using the Socket.io JavaScript library to wrap WebSockets functionality in web browsers.
- Benchmarking and Scaling WebSockets: Handling 60000 concurrent connections is a detailed examination of how WebSockets connections can scale to tens of thousands of users.
- Writing WebSocket servers gets into the nitty-gritty of how WebSockets work. Well worth reading to get a deep understanding of WebSockets connections.

What's next for your web application?

What runs a Python application execute on the server?

How should Python libraries be installed on a server?

What editor should I use to code my Python app?

Chapters