**Technology choice #2 — WebSockets**

WebSockets is another protocol that you can use to build event-driven APIs. In a nutshell, WebSockets are a thin transport layer built on top of a device’s TCP/IP stack. The intent is to provide what is essentially an as-close-to-raw-as-possible TCP communication layer to web application developers while adding a few abstractions to eliminate certain friction that would otherwise exist concerning the way the web works.

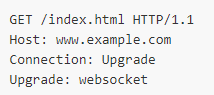
Unlike Webhooks, the WebSocket protocol allows for constant, bi-directional communication between the server and client, which means both parties can communicate and exchange data as and when needed.

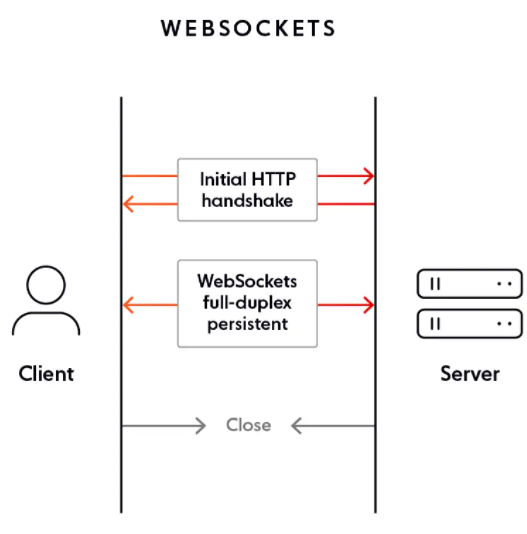
Similar to Webhooks, you’ll have to consider two things when building an event-driven API with WebSockets.

**1. Allow API consumers to subscribe to your APIs**

To keep getting event notifications, the consumer has to establish a WebSocket connection with your API in the first place.

A WebSocket connection is established by making an HTTP call to the API and then asking for an upgrade on that connection. After that, the communication takes place over a single TCP connection using the WebSocket protocol.





The Connection header tells the server that the client would like to negotiate a change in the way the socket is being used. The accompanying value, Upgrade, indicates that the transport protocol currently in use via TCP should change.

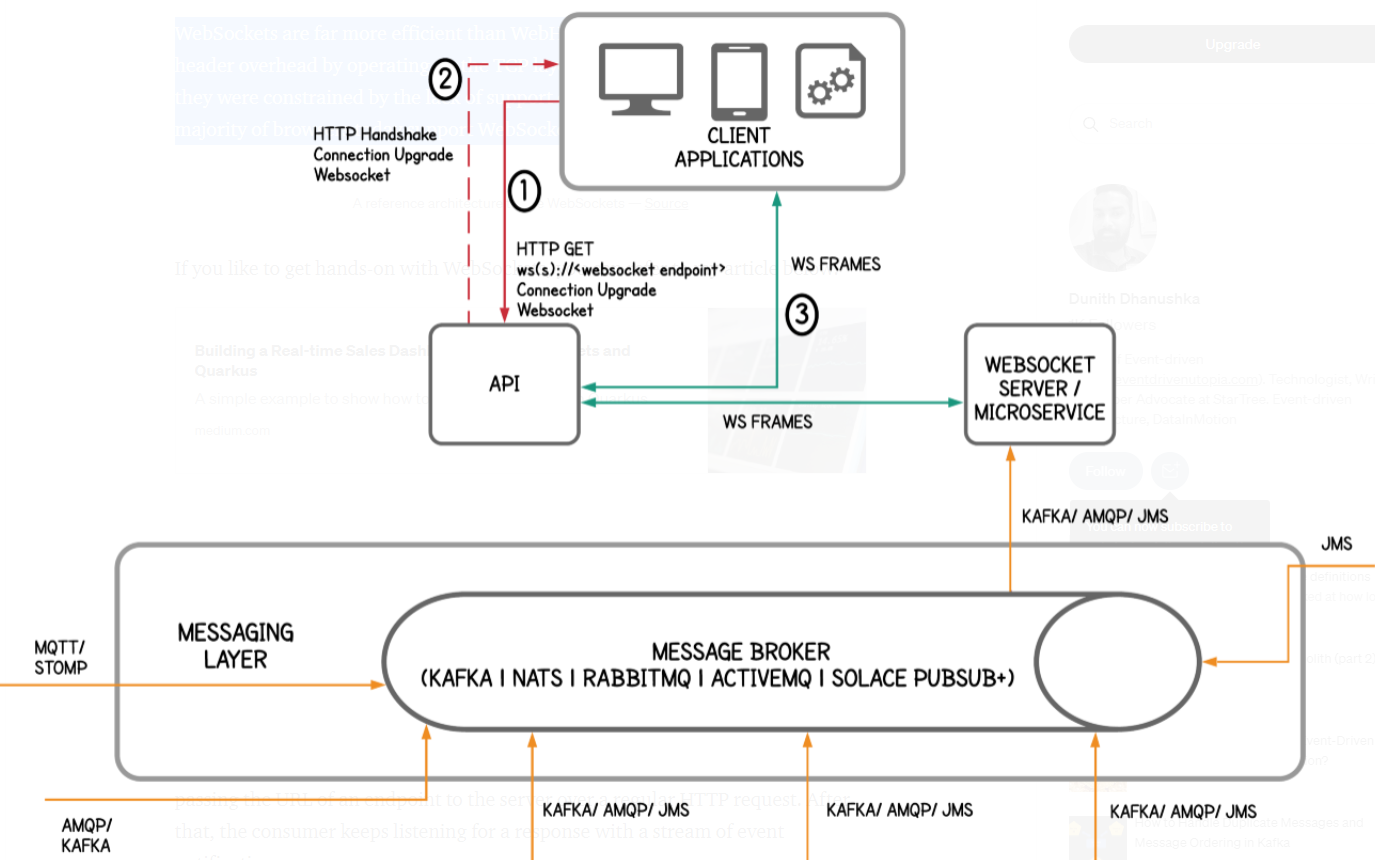
Now that the server knows that the client wants to upgrade the protocol currently in use over the active TCP socket, the server knows to look for the corresponding Upgrade header, which will tell it which transport protocol the client wants to use for the remaining lifetime of the connection. As soon as the server sees WebSocket as the value of the Upgrade header, it knows that a WebSocket handshake process has begun.

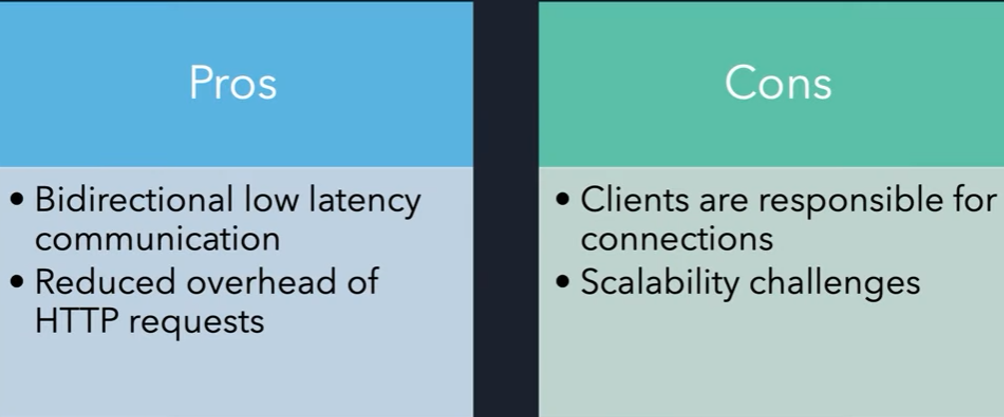
**2. Deliver events to API consumers asynchronously**

When the API has something interesting, it can notify the consumer by writing a data frame to its WebSockets connection. From a programming point of view, it is very similar to writing to a socket.

The consumer who had already established the connection can parse the event data from the wire and update its UI accordingly.

WebSockets are far more efficient than WebHooks as they remove the HTTP header overhead by operating on the TCP layer. In the early days of WebSockets, they were constrained by the lack of support offered by browsers. However, a majority of browsers today support WebSockets natively.

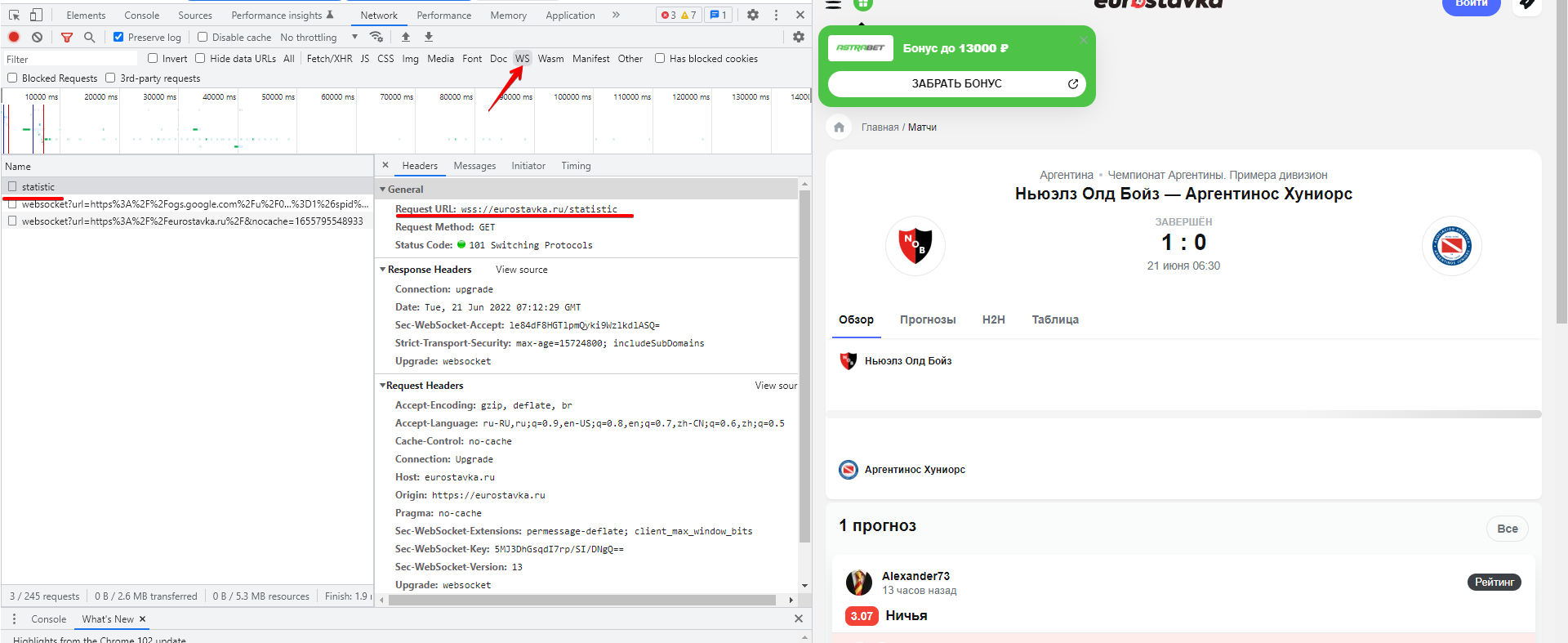




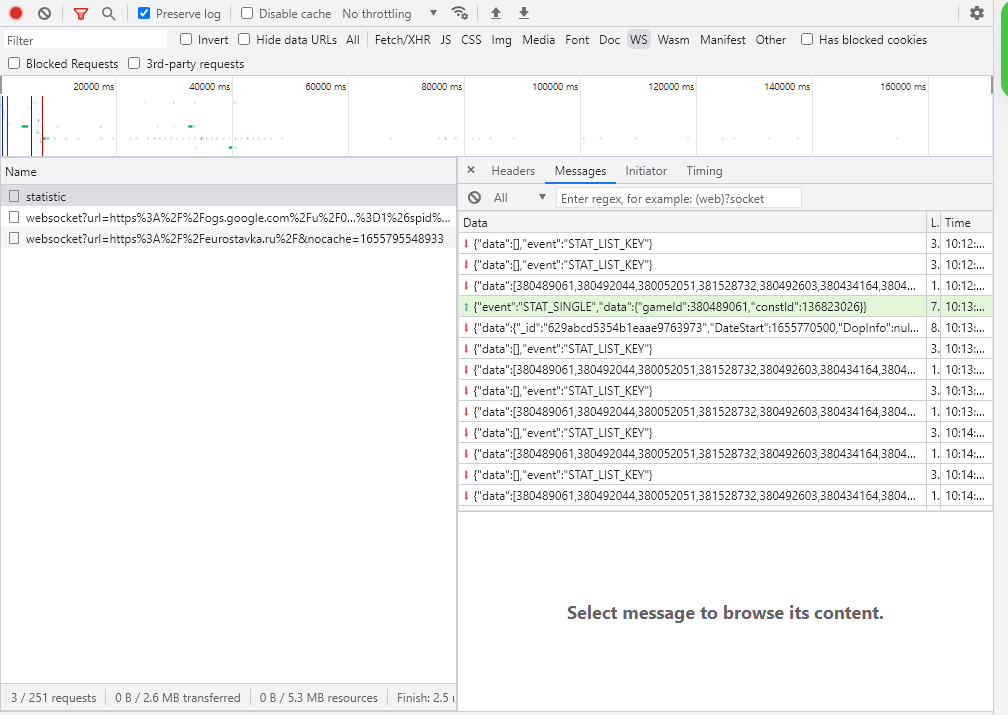
У вебсокетов URL начинается с wss://url.com

Например сайт евроставка использует вебсокеты для того, что бы динамически обновлять данные о матчах и счете

Основная страница использует HTTPs, но для модуля который реализован через вебсокеты идет подмена URL



Запросы отображаются зеленым – ответы красными стрелочками



Стоит использовать WebSockets когда нужно получать постоянно новые данные от сервера и важно что бы не было задержки (чаты, биржа, …). Если же мы можем себе позволить задержку в пару секунд, то лучше использовать HTTP Long Polling, AJAX. Так как они проще в разработке и поддержке

Что бы понять что websocket соединение оборвалось, то нужно посмотреть на “**Time**” в DevTools. WebSocket должен быть всегда в состоянии “**pending**” (так как он никогда не обрывается), если же там указано время, значит вебсокет оборвался и нам покаызваетс я время жихни вебсокет соединения.