Marcin Warczygłowa
Nov 10 2015

Real-time Web Application with Websockets and Vert.x

At Allegro, you can sell items at a fixed price (buy now) or at auction. Auctions are still a popular sales format, especially in categories such as antiques and art or clothing. So far, buyers fighting for an item had to refresh the web page in the last seconds of the auction to verify that the offer had not been overbid. This made bidding difficult and less fun. Last year real time bidding process for all mobile users was introduced. In this article I want to show how to create a simple application that provides real-time bidding, based on Allegro auctions. We will use WebSockets, SockJS and the latest, third version of Vert.x. We will create a frontend for fast bidding that communicates with a microservice written in Java 8 and based on Vert.x.

What are Websockets?

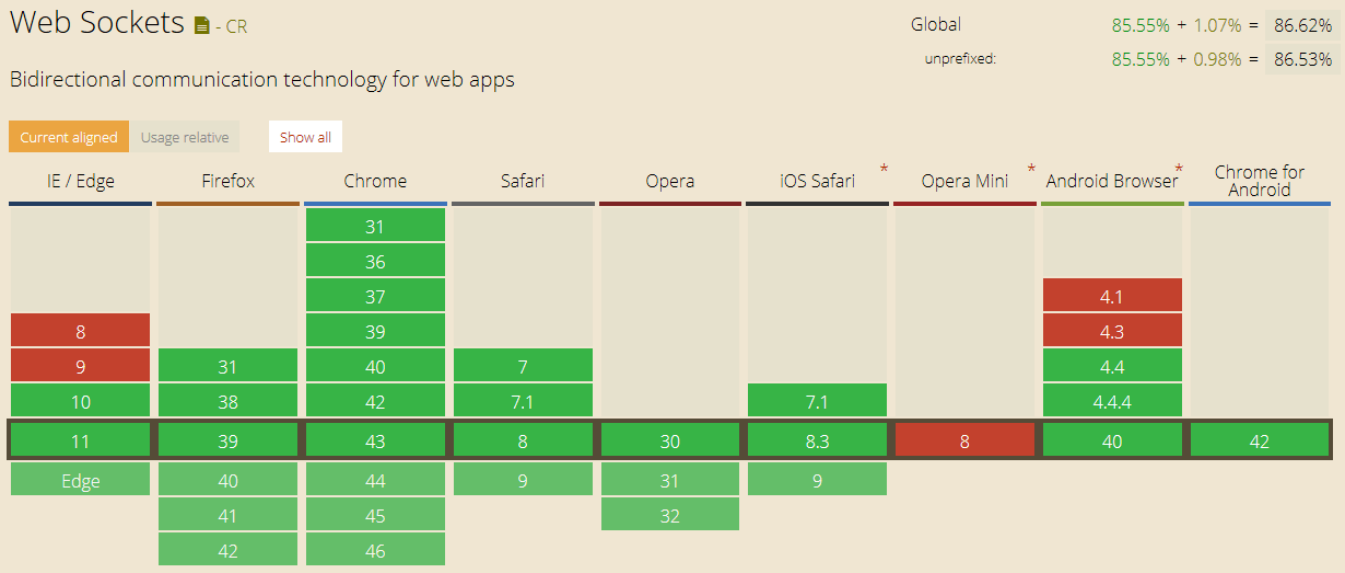
WebSocket is asynchronous, bidirectional, full-duplex protocol that provides a communication channel over a single TCP connection. With the WebSocket API it provides bidirectional communication between the website and a remote server. Originally, WebSocket was supposed to be a part of the HTML 5 specification, but a later revision of the protocol is described in a separate document RFC 6455.

WebSockets solve many problems which prevented the HTTP protocol from being suitable for use in modern, real-time applications. Workarounds like polling are no longer needed, which simplifies application architecture. WebSockets do not need to open multiple HTTP connections, they provide a reduction of unnecessary network traffic and reduce latency.

Each WebSockets connection begins as an HTTP request. In addition, an updated HTTP header indicates that the client wants to change the connection to WebSocket protocol. The initial

HTTP connection is replaced by a WebSocket connection using the same underlying TCP/IP connection. At this point, each side can start sending data.

WebSockets are supported by most web browsers (source):



Some examples of good use cases for WebSockets include:

- chat applications
- multiplayer games
- social feeds
- collaborative editing or coding
- sports updates

Websocket API vs SockJS

Unfortunately, WebSockets are not supported by all web browsers. However, there are libraries that provide a fallback when WebSockets are not available. One such library is SockJS. SockJS starts from trying to use the WebSocket protocol. However, if this is not possible, it uses a variety of browser-specific transport protocols. SockJS is a library designed to work in all modern browsers and in environments that do not support WebSocket protocol, for instance behind restrictive corporate proxy. SockJS provides an API similar to the standard WebSocket API. A simple example of using the SockJS library might look like the one below. First, load

```
<script src="//cdn.jsdelivr.net/sockjs/0.3.4/sockjs.min.js"></script>
```

Then we establish the connection to the SockJS server:

```
var sock = new SockJS('http://mydomain.com/my_prefix');

sock.onopen = function() {
    console.log('open');
};

sock.onmessage = function(e) {
    console.log('message', e.data);
};

sock.onclose = function() {
    console.log('close');
};

sock.send('test');
sock.close();
```

Vert.x

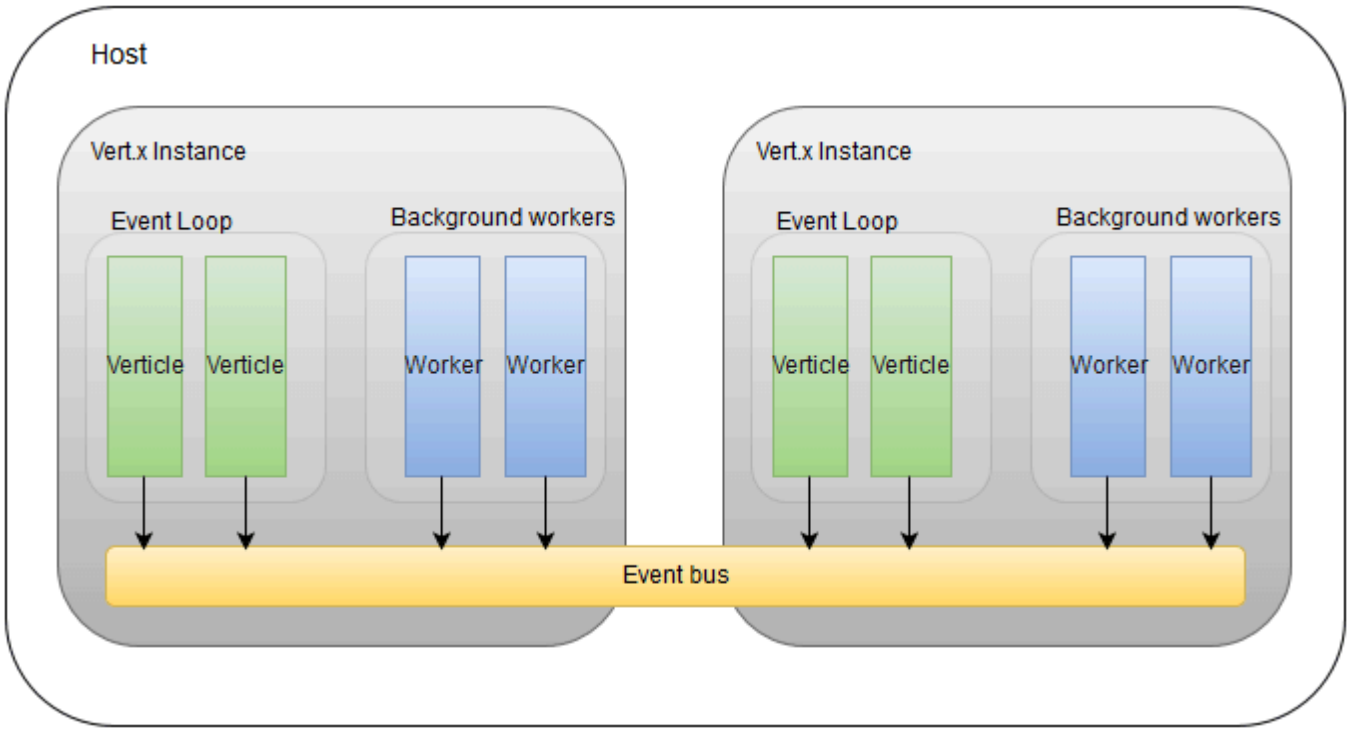
SockJS client requires the server-side part. For the Java language we can use, among other things, Spring Framework Java client & server, Atmosphere Framework or Vert.x. We are going to use the latter.

Vert.x is a polyglot, non-blocking, event-driven tool-kit for building applications on the JVM. Vert.x is pretty fast, which you can see on TechEmpower Benchmarks. The packages of code that Vert.x executes are called verticles. Verticles can be written in Java, Groovy, Ruby, JavaScript as well as in several programming languages mixed and matched in a single application. Many verticles can be executed concurrently in the same Vert.x instance. A single Vert.x instance runs inside its own JVM instance. Vert.x guarantees that a particular verticle instance is never executed by multiple threads concurrently. Verticles communicate by passing messages using an event bus.

Vert.x applications are mostly written by defining event handlers. Vert.x calls handlers using a thread called an event loop. The event loop delivers events to different handlers in succession as they arrive. None of the Vert.x APIs block threads, so you also need to remember not to block the event loop in handlers. Because nothing blocks, an event loop can potentially deliver a lot of events in a short time. We make guarantees that any specific handler will always be invoked by the same event loop. This means you can write your code as single threaded. Vert.x instance maintains several event loops. The default number of event loops is determined by the number of available cores on the machine.

There are two main types of verticles: standard and worker verticles. Standard verticles are always executed using an event loop thread. Workers are designed for executing blocking code. Workers are like standard verticles but use threads from a special worker thread pool. An alternative way to run blocking code is to use `executeBlocking` method directly from an event loop.

Typical application will consist of multiple verticles running on Vert.x instance:



There can be many Vert.x instances running on the same host or on different hosts on the network. Instances can be configured to cluster with each other forming a distributed event bus over which verticles can communicate. We can create a distributed bus encompassing

Frontend to fast bidding

Auction web page contains the bidding form and some simple JavaScript which loads current price from the service, opens an event bus connection to the SockJS server and offers bidding. HTML source code of sample web page on which we bid might look like this:

```
<h3>Auction 1</h3>
<div id="error_message"></div>
<form>
    Current price:
    <span id="current_price"></span>
    <div>
        <label for="my_bid_value">Your offer:</label>
        <input id="my_bid_value" type="text">
        <input type="button" onclick="bid();" value="Bid">
    </div>
    <div>
        Feed:
        <textarea id="feed" rows="4" cols="50" readonly></textarea>
    </div>
</form>
```

We use the **vertxbus.js** library to create a connection to the event bus. **Vertxbus.js** library is a part of the Vert.x distribution. **Vertxbus.js** internally uses SockJS library to send the data to the SockJS server. In the code snippet below we create an instance of the event bus. The parameter to the constructor is the URI where to connect to the event bus. Then we register the handler listening on address **auction.<auction_id>** . Each client has a possibility of registering at multiple addresses e.g. when bidding in the auction 1234, they register on the address **auction.1234** etc. When data arrives in the handler, we change the current price and the bidding feed on the auction's web page.

```
function registerHandlerForUpdateCurrentPriceAndFeed() {
    var eventBus = new vertx.EventBus('http://localhost:8080/eventbus');
    eventBus.onopen = function () {
        eventBus.registerHandler('auction.' + auction_id, function (message) {
            document.getElementById('current_price').innerHTML = JSON.parse(message).price;
```

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.