# Table of Contents

1. Websockets ................................................................................................................. 1
2. Websocket in Java ........................................................................................................... 1
3. Websocket Example Code for Websocket clients (FOSS) ................................................ 1
4. BCP: Websocket Configuration ....................................................................................... 1
5. Demo with VPN Policy Model .......................................................................................... 2
   5.1. A Websocket Configuration for the VPN Domain ....................................................... 2
   5.2. Start APEX Engine .................................................................................................... 3
   5.3. Run the Websocket Echo Client .................................................................................. 4
   5.4. Run the Websocket Console Client ........................................................................... 5
   5.5. Send Events .............................................................................................................. 6
1. Websockets

Websocket is a protocol to run sockets of HTTP. Since it in essence a socket, the connection is realized between a server (waiting for connections) and a client (connecting to a server). Server/client separation is only important for connection establishment, once connected, everyone can send/receive on the same socket (as any standard socket would allow).

Standard Websocket implementations are simple, no publish/subscribe and no special event handling. Most servers simply send all incoming messages to all connections. There is a PubSub definition on top of Websocket called WAMP. APEX does not support WAMP at the moment.

2. Websocket in Java

In Java, JSR 356 defines the standard Websocket API. This JSR is part of Java EE 7 standard. For Java SE, several implementations exist in open source. Since Websockets are a stable standard and simple, most implementations are stable and ready to use. A lot of products support Websockets, like Spring, JBoss, Netty, ... there are also Kafka extensions for Websockets.

3. Websocket Example Code for Websocket clients (FOSS)

There are a lot of implementations and examples available on Github for Websocket clients. If one is using Java EE 7, then one can also use the native Websocket implementation. Good examples for clients using simply Java SE are here:

- Websocket implementation
- Websocket sending client example, using AWT
- Websocket receiving client example (simple echo client)

For Java EE, the native Websocket API is explained here:

- Oracle docs
- link: An example

4. BCP: Websocket Configuration

The probably best is to configure APEX for Websocket servers for input (ingress, consume) and output (egress, produce) interfaces. This means that APEX will start Websocket servers on named ports and wait for clients to connect. Advantage: once APEX is running all connectivity infrastructure is running as well. Consequence: if APEX is not running, everyone else is in the dark, too.

The best protocol to be used is JSON string. Each event on any interface is then a string with a JSON encoding. JSON string is a little bit slower than byte code, but we doubt that this will be noticeable. A further advantage of JSON strings over Websockets with APEX starting the servers: it is very easy to connect web browsers to such a system.
Simple connect the web browser to the APEX sockets and send/read JSON strings.

Once APEX is started you simply connect Websocket clients to it, and send/receive event. When APEX is terminated, the Websocket servers go down, and the clients will be disconnected. APEX does not (yet) support auto-client reconnect nor WAMP, so clients might need to be restarted or reconnected manually after an APEX boot.

5. Demo with VPN Policy Model

We assume that you have an APEX installation using the full package, i.e. APEX with all examples, of version 8.5.6 or higher. We will use the VPN policy from the APEX examples here.

Now, have the following ready to start the demo:

- 3 terminals on the host where APEX is running (we need 1 for APEX and 1 for each client)
- the events in the file `$APEX_HOME/examples/events/VPN/SetupEvents.json` open in an editor (we need to send those events to APEX)
- the events in the file `$APEX_HOME/examples/events/VPN/Link09Events.json` open in an editor (we need to send those events to APEX)

5.1. A Websocket Configuration for the VPN Domain

Create a new APEX configuration using the VPN policy model and configuring APEX as discussed above for Websockets. Copy the following configuration into `$APEX_HOME/examples/config/VPN/Ws2WsServerAvroContextJsonEvent.json` (for Windows use `%APEX_HOME%\examples\config\VPN\Ws2WsServerAvroContextJsonEvent.json`):
5.2. Start APEX Engine

In a new terminal, start APEX with the new configuration for Websocket-Server ingress/egress:

```bash
#: $APEX_HOME/bin/apexEngine.sh -c $APEX_HOME/examples/config/VPN/Ws2WsServerAvroContextJsonEvent.json
#: %APEX_HOME%\bin\apexEngine.bat -c %APEX_HOME%\examples\config\VPN\Ws2WsServerAvroContextJsonEvent.json
```
Wait for APEX to start, it takes a while to create all Websocket servers (about 8 seconds on a standard laptop without cached binaries). depending on your log messages, you will see no (some, a lot) log messages. If APEX starts correctly, the last few messages you should see are:

```plaintext
VPNPolicyModelAvro:0.0.1 added to the engine-AxArtifactKey:(name=VPNApexEngine-0,version=0.0.1)
2017-07-28 13:17:21,057 Apex [Apex-apex-engine-service-0:0] INFO c.e.a.s.engine.runtime.EngineService -
Engine AxArtifactKey:(name=VPNApexEngine-0,version=0.0.1) processing ...
2017-07-28 13:17:21,296 Apex [main] INFO c.e.a.s.e.r.impl.EngineServiceImpl - Added the action listener to
the engine
```

APEX is running in the new terminal and will produce output when the policy is triggered/executed.

## 5.3. Run the Websocket Echo Client

The echo client is included in an APEX full installation. To run the client, open a new shell (Unix, Cygwin) or command prompt (cmd on Windows). Then use the APEX application launcher to start the client.

### APEX engine needs to run first

The example assumes that an APEX engine configured for *produce* carrier technology Websocket and *JSON* event protocol is executed first.

<table>
<thead>
<tr>
<th>Unix, Cygwin</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td><code># $APEX_HOME/bin/apexApps.sh ws-echo [args]</code></td>
<td><code>&gt; %APEX_HOME%\bin\apexApps.bat ws-echo [args]</code></td>
</tr>
</tbody>
</table>

Use the following command line arguments for server and port of the Websocket server. The port should be the same as configured in the APEX engine. The server host should be the host on which the APEX engine is running

- `-p` defines the Websocket port to connect to (defaults to 8887)
- `-s` defines the host on which a Websocket server is running (defaults to `localhost`)

Let's assume that there is an APEX engine running, configured for *produce* Websocket carrier technology, as server, for port 42452, with *produce* event protocol *JSON*. If we start the console client on the same host, we can omit the `-s` options. We start the console client as:

```plaintext
# $APEX_HOME/bin/apexApps.sh ws-echo -p 42452 ①
> %APEX_HOME%\bin\apexApps.bat ws-echo -p 42452 ②
```

① Start client on Unix or Cygwin
② Start client on Windows

Once started successfully, the client will produce the following messages (assuming we used `-p 42452` and an APEX engine is running on `localhost` with the same port:
ws-simple-echo: starting simple event echo
--> server: localhost
--> port: 42452

Once started, the application will simply print out all received events to standard out.
Each received event will be prefixed by '---' and suffixed by '===='

ws-simple-echo: opened connection to APEX (Web Socket Protocol Handshake)

### 5.4. Run the Websocket Console Client

The console client is included in an APEX full installation. To run the client, open a new shell (Unix, Cygwin) or command prompt (cmd on Windows). Then use the APEX application launcher to start the client.

<table>
<thead>
<tr>
<th>Unix, Cygwin</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td># $APEX_HOME/bin/apexApps.sh ws-console [args]</td>
<td>&gt; %APEX_HOME%/bin/apexApps.bat ws-console [args]</td>
</tr>
</tbody>
</table>

Use the following command line arguments for server and port of the Websocket server. The port should be the same as configured in the APEX engine. The server host should be the host on which the APEX engine is running

- `-p` defines the Websocket port to connect to (defaults to 8887)
- `-s` defines the host on which a Websocket server is running (defaults to localhost)

Let's assume that there is an APEX engine running, configured for consume carrier technology Websocket and JSON event protocol is executed first. If we start the console client on the same host, we can omit the `-s` options. We start the console client as:

```
# $APEX_HOME/bin/apexApps.sh ws-console -p 42450 ①
> %APEX_HOME%/bin/apexApps.bat ws-console -p 42450 ②
```

① Start client on Unix or Cygwin
② Start client on Windows

Once started successfully, the client will produce the following messages (assuming we used `-p 42450` and an APEX engine is running on localhost with the same port:

ws-simple-console: starting simple event console
--> server: localhost
--> port: 42450

- terminate the application typing 'exit<enter>' or using 'CTRL+C'
- events are created by a non-blank starting line and terminated by a blank line

ws-simple-console: opened connection to APEX (Web Socket Protocol Handshake)
5.5. Send Events

Now you have the full system up and running:

- Terminal 1: APEX ready and loaded
- Terminal 2: an echo client, printing received messages produced by the VPN policy
- Terminal 2: a console client, waiting for input on the console (standard in) and sending text to APEX

We started the engine with the VPN policy example. So all the events we are using now are located in files in the following example directory:

```plaintext
#: $APEX_HOME/examples/events/VPN
> $APEX_HOME\examples\events\VPN
```

To sends events, simply copy the content of the event files into Terminal 3 (the console client). It will read multi-line JSON text and send the events. So copy the content of `SetupEvents.json` into the client. APEX will trigger a policy and produce some output, the echo client will also print some events created in the policy. In Terminal 1 (APEX) you’ll see some status messages from the policy as:

```json
{Link=L09, LinkUp=true}
L09     true
outFields: {Link=L09, LinkUp=true}
{Link=L10, LinkUp=true}
L09     true
L10     true
outFields: {Link=L10, LinkUp=true}
{CustomerName=C, LinkList=L09 L10, SlaDT=300, YtdDT=300}
*** Customers ***
C 300 300 [L09, L10]
outFields: {CustomerName=C, LinkList=L09 L10, SlaDT=300, YtdDT=300}
{CustomerName=A, LinkList=L09 L10, SlaDT=300, YtdDT=50}
*** Customers ***
A 300 50 [L09, L10]
C 300 300 [L09, L10]
```

In Terminal 2 (echo-client) you see the received events, the last two should look like:

```json
{CustomerName=C, LinkList=L09 L10, SlaDT=300, YtdDT=300}
*** Customers ***
C 300 300 [L09, L10]
```

In Terminal 2 (echo-client) you see the received events, the last two should look like:
Congratulations, you have triggered a policy in APEX using Websockets, the policy did run through, created events, picked up by the echo-client.

Now you can send the Link 09 and Link 10 events, they will trigger the actual VPN policy and some calculations are made. Let's take the Link 09 events from Link09Events.json, copy them all into Terminal 3 (the console). APEX will run the policy (with some status output), and the echo client will receive and print events.

To terminate the applications, simply press **CTRL+C** in Terminal 1 (APEX). This will also terminate the echo-client in Terminal 2. Then type **exit<enter>** in Terminal 3 (or **CTRL+C**) to terminate the console-client.