

# Running WebSocket applications on IBM i

## Using WebSocket for real-time web applications

Tian Gang  
Xu Meng  
Zheng Chang Qing

February 25, 2016  
(First published February 25, 2016)

WebSocket represents the next evolutionary step in web communication after Comet and Ajax. Both Java™ and Node.js provide support for server side WebSocket. `mod_proxy_wstunnel` is a new module of Apache 2.4 that provides support for the tunneling of web socket connections to a back-end WebSocket server. These features can be bundled together to run WebSocket solutions on IBM i. This article illustrates how to create different WebSocket server-side implementations and associate them with the Apache HTTP Server to run WebSocket applications on IBM i.

## Introduction

This article describes how to create a WebSocket application on IBM i using IBM Integrated Web Application Server for i and Node.js on IBM i, and how to associate a WebSocket application with the IBM HTTP Server for i. A simple web-based IBM i console example can help in introducing and guiding you through the process of creating a WebSocket application in Java and JavaScript. We can also explore the step-by-step process to set up the HTTP Server environment.

The WebSocket specification was developed as part of the HTML5 initiative and introduced the WebSocket JavaScript interface, which defines a full-duplex single socket connection over which messages can be sent between the client and server. The WebSocket protocol allows real-time interaction between a browser and a website, facilitating interactive live content. This can be used for the creation of real-time games, for example. This is made possible by providing a standardized way for the server to send content to the browser without being solicited by the client for every request, and allowing messages to be passed back and forth while keeping the connection open. The WebSocket represents the next evolutionary step in web communication compared to Comet and Ajax.

The WebSocket protocol is currently supported in most major browsers including Google Chrome, Microsoft® Internet Explorer, Mozilla Firefox, Safari, and Opera. This article focuses on using

WebSocket in the following server-side web technologies. It also explains how to configure and establish a WebSocket connection in a simple application.

- IBM Integrated Web Application Server for i is a lightweight Java-based application runtime on IBM i. The latest version has been updated based on WebSphere Application Server Liberty profile V8.5. The Liberty profile is a composable and dynamic application server and provides support for WebSocket 1.0 and 1.1.
- Node.js is an open source project based on the Google Chrome V8 engine. It provides a platform for server-side JavaScript applications running without the need for a browser. Node.js is now supported on the IBM i platform. It also provides support for WebSocket. To get started with Node.js on IBM i, it is recommended to read the article, [Native JavaScript applications on IBM i with Node.js](#).
- IBM HTTP Server (powered by Apache) for i is a complete web server product, which offers several components and features to assist in your website configuration and development. With the IBM HTTP Server (powered by Apache) for i, you quickly have everything you need and can easily establish a web presence and get your business running on the web. On IBM i 7.2, the HTTP Server has been upgraded to Apache 2.4.12 which introduces a new module, `mod_proxy_wstunnel`. This new module provides support for the tunneling of WebSocket connections to a back-end WebSocket server. On IBM i 7.1, IBM HTTP Server for i is based on Apache 2.2.11 which doesn't provide the WebSocket support.

## Software prerequisites

In order to get your WebSocket applications successfully associated with the HTTP Server and running on IBM i, the following licensed programs and program temporary fix (PTF) groups are required.

### Required license program:

- 5770SS1, option 30 – QSH
- 5770SS1, option 33 – Portable App Solutions Environment
- 5770DG1, \*BASE – IBM HTTP Server for i
- 5733OPS, option 1 – IBM i Open Source Solutions
- 5770JV1, option 14 – Java SE 7 32 bit
- 5733SC1, option 1 – OpenSSH, OpenSSL, zlib

### Required PTF group:

- IBM i 7.2 HTTP Server for i PTF Group SF99713 – level 12 (or later)
- IBM i 7.2 Java PTF Group SF99716 – latest level is recommended
- IBM i 7.1 HTTP Server for i PTF Group SF99368 – level 37 (or later)
- IBM i 7.1 Java PTF Group SF99572 – latest level is recommended

**Note:** 5733OPS is supported on IBM i 7.1 and later. If you need to configure the HTTP Server for i for WebSocket tunneling, you need to be on IBM i 7.2 or later.

In the following sections, we will be creating three separate examples:

- WebSocket server-side Java application and the client web-based JavaScript application
- WebSocket server-side application in JavaScript
- IBM HTTP Server for i to work together with the WebSocket applications

The following table provides the test environment for the examples.

**Table 1. Test environment scenarios**

| Scenario                             | IP            | Non-SSL port | SSL Port | WebSocket URI   |
|--------------------------------------|---------------|--------------|----------|---|
| WebSocket Server on Liberty          | 192.168.0.101 | 10000        | 10443    | ws://192.168.0.101:10000/WSConsole/CLRunner<br>wss://192.168.0.101:10443/WSConsole/CLRunner |
| WebSocket Server on Node.js          | 192.168.0.102 | 8888         | 8889     | ws://192.168.0.102:8888/CLRunner<br>wss://192.168.0.102:8889/CLRunner                       |
| HTTP Server using mod_proxy_wstunnel | 192.168.0.103 | 80           | 443      | ws://192.168.0.103/ws<br>wss://192.168.0.103/wss  |

In the following two sections, we demonstrate two examples with detailed steps to show you how to create and run a WebSocket web application using Java and Node.js respectively. In the last section, we demonstrate how to associate the WebSocket examples to IBM i HTTP Server by using the new `mod_proxy_wstunnel` module provided by Apache 2.4.

In the WebSocket examples we present, a client enters a Control language (CL) command in a web browser. The CL command is sent to the server to be executed and the results are returned back to browser. This can be implemented by using the traditional HTTP requests and responses, but here, we show you how to do that more easily by using the more advanced WebSocket technology.

## Create a WebSocket server application using Java

This section illustrates the steps needed to create a WebSocket server application using Java. It is assumed that you are familiar with creating Java-based web applications using a Java development environment.

### Step 1. Set up the required Java environment

WebSocket is only supported in Java EE 1.7 and later. So, install Java EE 1.7 on your development system and adopt it in your development tool or copy the required **javax.websocket-api-1.1.jar** file to your development environment and add it to the class path. The .jar file just provides the WebSocket related interface but not the implementation. The implementation is provided by the application server. After completing the setup, create a web application project in your Java development tool. Recommended development tools include Rational Developer for i, Rational Application Developer, or standard Eclipse-based tooling.

## Step 2. Create the required Java classes

Create the WSConsoleServerEndPoint.java file with the content, as shown in Listing 1.

### Listing 1. WebSocket server implementation in Java

```
package com.ibm.ws.demo.server;

import java.io.IOException;
import java.net.InetAddress;
import java.util.logging.Logger;

import javax.websocket.CloseReason;
import javax.websocket.OnClose;
import javax.websocket.OnMessage;
import javax.websocket.OnOpen;
import javax.websocket.OnError;
import javax.websocket.Session;
import javax.websocket.server.ServerEndpoint;

import com.ibm.ws.demo.CLRunner;

@ServerEndpoint(value = "/CLRunner")
public class WSConsoleServerEndPoint {

    private Logger logger = Logger.getLogger(this.getClass().getName());

    @OnOpen
    public void onOpen(Session session) {
        logger.info("[Server] Connected. " + session.getId());
        session.setMaxIdleTimeout(300000);

        try {
            InetAddress ia = InetAddress.getLocalHost();
            session.getBasicRemote().sendText("[From Server] Connected to " + ia.getHostName() +
                " [" + ia.getHostAddress() + "].");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @OnMessage
    public String onMessage(String message, Session session) {
        logger.info("[Server] Received:" + message);
        String result = CLRunner.runCL(message);
        return result;
    }

    @OnClose
    public void onClose(Session session, CloseReason closeReason) {
        logger.info(String.format("[Server] Session %s closed because of %s", session.getId(),
            closeReason));
    }

    @OnError
    public void onError(Session session, Throwable error) {
        error.printStackTrace();
    }
}
```

The class is an implementation of a WebSocket server. Java uses annotation for easy and simple WebSocket implementation. Except for the Java code, no configuration is required for a WebSocket server. The following line of code defines the URI to the WebSocket.

```
@ServerEndpoint(value = "/CLRunner")
```

The following four attributes are the WebSocket server skeleton. They are called when specific events happen.

**Table 2. WebSocket object attributes**

| Attribute | Description                          |
|-----------|--------------------------------------|
| OnOpen    | Invoked when a connection is set up  |
| OnMessage | Invoked when a message is received   |
| OnClose   | Invoked when a connection is dropped |
| OnError   | Invoked when an error occurs         |

The main functionality of most WebSocket servers takes place in the `onMessage` method. This method is invoked when the client sends a message, usually a *string*, to the server. The server code then uses the input message to create a response message that is then returned by the `onMessage` method.

The following line of code is used to set the timeout for the WebSocket connection. The connection closes on the server side automatically if the timeout is reached. The client side receives the time out event.

```
session.setMaxIdleTimeout(300000);
```

### Step 3. Create the HTML file for the WebSocket client

Now we create the WebSocket client in JavaScript to access the WebSocket server. You just need to create a WebSocket object and define the methods for the different WebSocket events as shown in the following listing.

**Listing 2. Create and initialize WebSocket object**

```
websocket = new WebSocket(wsUri);
websocket.onopen = function(evt) { onOpen(evt) };
websocket.onclose = function(evt) { onClose(evt) };
websocket.onmessage = function(evt) { onMessage(evt) };
websocket.onerror = function(evt) { onError(evt) };
```

The three methods shown in the following table are provided by the WebSocket object.

**Table 3. WebSocket object methods**

| Method                     | Description  |
|----------------------------|--|
| WebSocket WebSocket(WSUrl) | This method creates a WebSocket object. WSUrl is the URL to which you need to connect.                               |
| void close()               | This method closes the WebSocket connection or connection attempt, if any.   |
| void send(data)            | This method transmits data to the server over the WebSocket connection. Data is a text string to send to the server. |

On the sample web page, the client receives a CL command from the user. It then uses the `send()` method to send the CL command to the server as shown in the following example.

```
websocket.send(message);
```

The HTML file, `WSConsole.html`, in the attachment contains the code details. You can download and use it in the example.

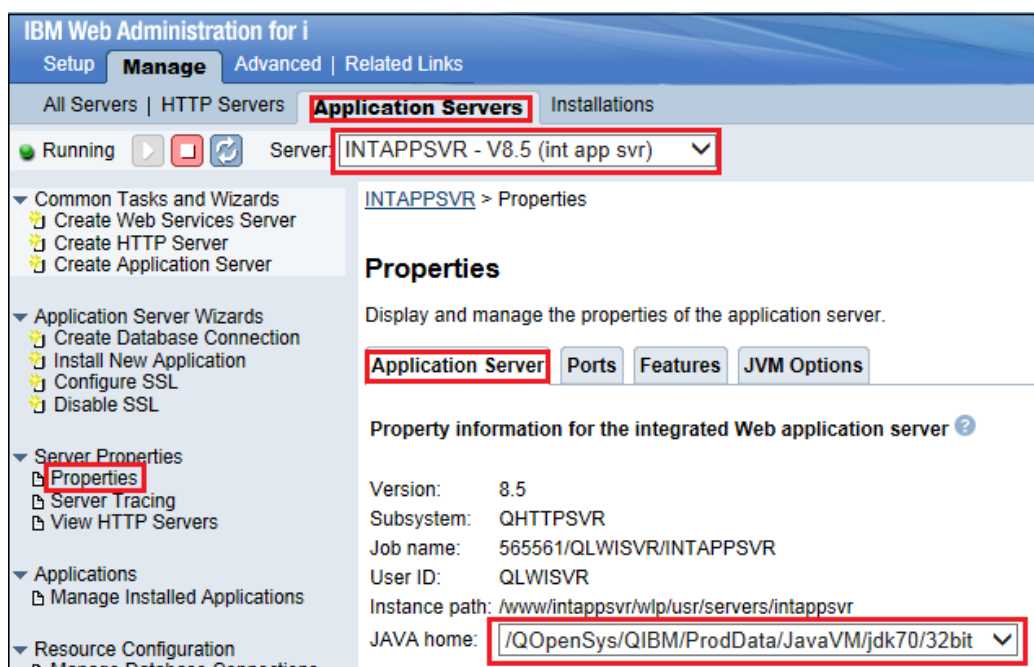
## Step 4. Create an Integrated Web Application Server instance and set up the environment for WebSocket

IBM Web Administration for i (<http://<your.server.name>:2001/HTTPAdmin>) can be used easily to create an instance of the Integrated Web Application Server for i. Refer to the [IBM i Knowledge Center](#) for details on how to create an instance of the Integrated Web Application Server for i V8.5. This article uses an Integrated Web Application Server for i V8.5 instance, named `INTAPPSVR`, as an example. It listens to port 10000 and the root directory is `/www/INTAPPSVR`.

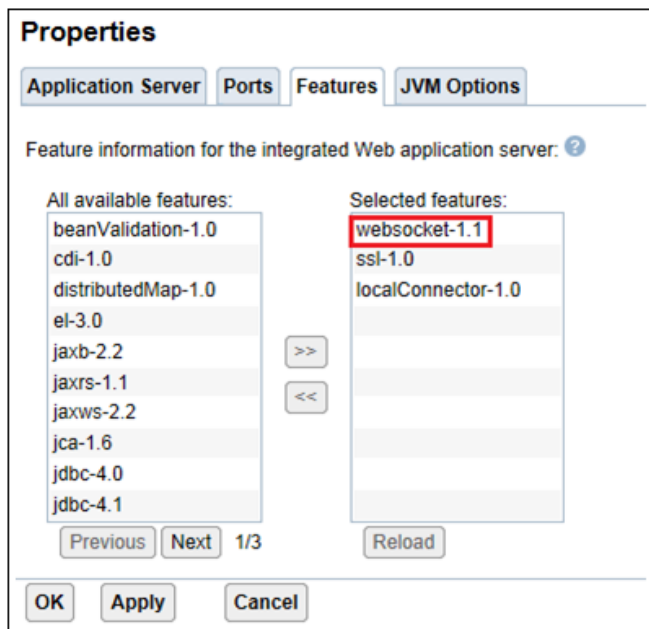
By default, the Integrated Web Application Server for i is initialized to use JDK 6 on IBM i 7.1 and 7.2. This needs to be updated to use JDK 7 to make WebSocket work.

Launch the IBM Web Administration for i, click the **Application Servers** tab, select the instance just created in the server list, and click **Properties** in the left pane. On the **Application Server** tab, all the JDKs installed on the system are listed in the **Java home** field. Select the JDK 7 32 bit from the list and click **Apply**.



## Figure 1. Change the JDK used by Integrated Web Application Server to Java version 7



Click the **Features** tab, select the **websocket-1.1** feature and add it to the **Selected features** list.

**Figure 2. Add the websocket-1.1 feature**

Then, click **Apply** for this to take effect. By default, there are only three features loaded in the instance. Feature ssl-1.0 is required for Secure Sockets Layer (SSL) configuration, feature localConnector-1.0 is required to manage the instance in the IBM Web Administration for i GUI. Other features can be added based on the needs of your applications.

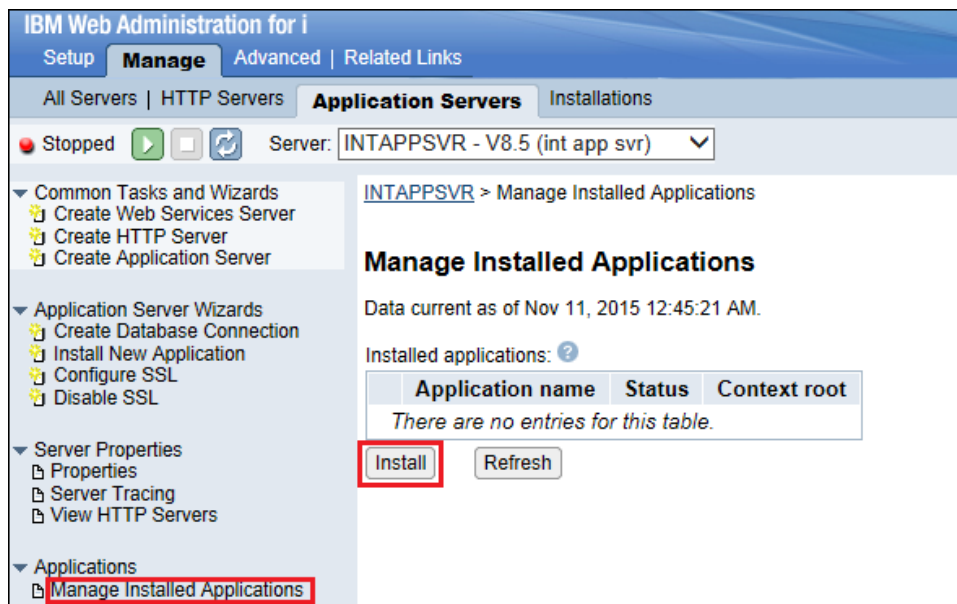
Now that the server has been changed, it needs to be restarted for these setting to go live. Click the stop server  icon to stop the server and click the start server  icon to start the server again.

## Step 5. Deploy the WebSocket application

Export the web application from your development tool as a war package file, WSConsole.war, and copy it to an existing IBM i integrated file system (IFS) directory, for example, /home/.

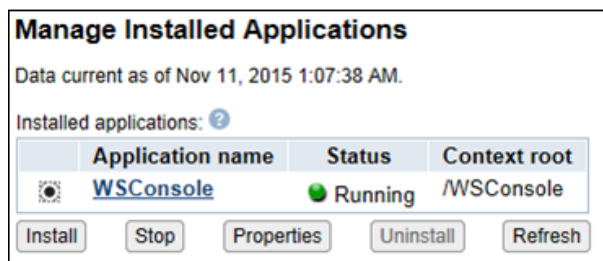
On the IBM Web Administration for i GUI, Click **Manage Installed Applications** on the left panel and click **Install** to launch the installation wizard.

**Figure 3. Launching the Install New Application wizard to install the WebSocket application**



Follow the wizard, accept all default values except for the installation location for the WAR file for the application. Specify the IFS directory and file for the actual application. Click **Finish** to complete the application installation. The application should be listed in the **Installed applications** table and started automatically. The WebSocket server is now ready to test.

**Figure 4. WebSocket applications listed on the Manage Installed Applications page**

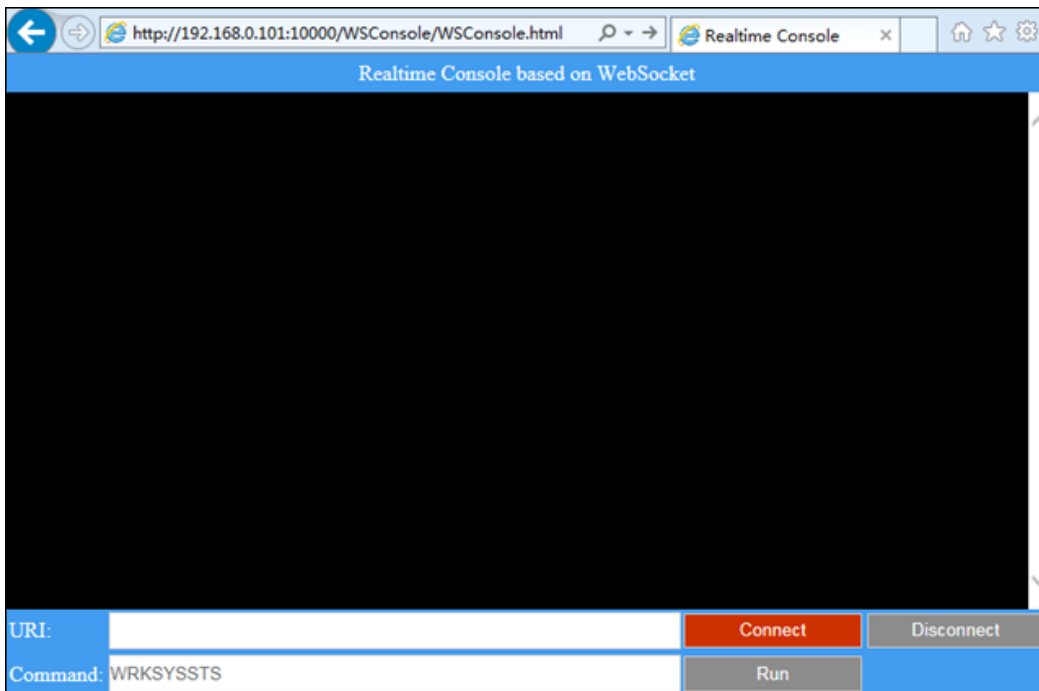


### Step 6. Visit the web page to verify the WebSocket application

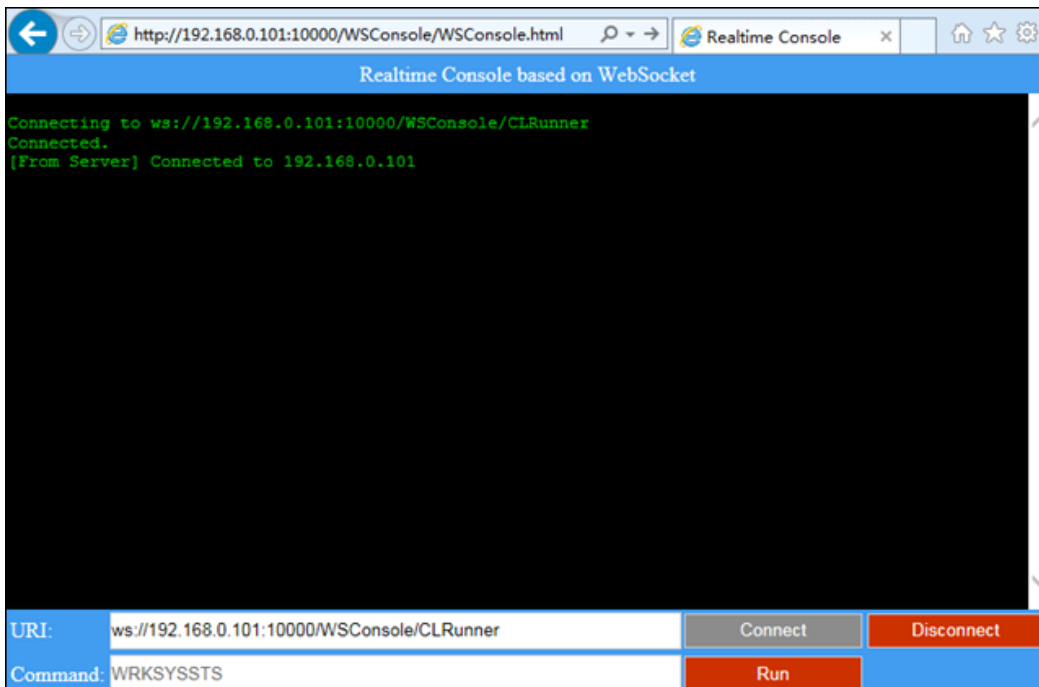
To use the HTML file you downloaded from the examples list, open a browser and enter the URI:  
<http://your.server.name:10000/WSConsole/WSConsole.html>

If everything is fine, the following web page will be displayed.



**Figure 5. Accessing the WebSocket client web page**

Enter the WebSocket server URI, *ws://your.server.name:10000/WSConsole/CLRunner* and click **Connect**. If there are no errors, the text as shown in Figure 6 is displayed.

**Figure 6. WebSocket server in Java**

To actually test the WebSocket connection we are going to input our favorite CL Command. Enter WRKSBS at the command prompt. The output, as shown in Figure 7 is displayed.

### Figure 7. Running the CL command on IBM i through WebSocket

Realtime Console based on WebSocket

Connecting to ws://192.168.0.101:10000/WSConsole/CLRunner

Connected.

[From Server] Connected to 192.168.0.101

WRKSBS

Work with Subsystems

12/14/15 21:12:36

Page 0001

Subsystem

Active

Total

-----

Subsystem Pools-----

Subsystem

Number

Jobs

Status

Storage (M)

1

2

QBATCH

011043

0

ACTIVE

.00

2

QCMN

011044

7

ACTIVE

.00

2

QCTL

011019

1

ACTIVE

.00

2

QHTTSPVR

120595

22

ACTIVE

.00

2

QINTER

011042

5

ACTIVE

.00

2

3

QSERVER

011036

25

ACTIVE

.00

2

QSPL

011045

0

ACTIVE

.00

2

4

QSYSWRK

011020

155

ACTIVE

.00

2

QUSWRK

011038

36

ACTIVE

.00

2

\*\*\*\*\* END OF LISTING \*\*\*\*\*

URI:ws://192.168.0.101:10000/WSConsole/CLRunner

Connect

Disconnect

Command: WRKSBS

Run

Enter other CL commands as required. Each command should return the results quickly without any additional connection.

An alternate way of implementing this application is to use traditional HTTP requests and responses. In this approach, the CL command is included in the HTTP request and the result is included in the HTTP response each time the user runs a CL command. It takes additional time to create a HTTP connection for each request and we can see a lot of duplicated data such as the common HTTP request and response headers being passed between the client and the server. When using WebSocket, only the necessary data, in this case the CL commands and the results, are passed over the connection. This saves both bandwidth and time, and this saving is very critical for a real-time web application.

The HTTP protocol only allows a client to send a request to server, and the server side can't initialize a request to the client. Now, by using the new WebSocket technique, we have the ability for the server to send a request to the client. Because the WebSocket connection is a full-duplex single-socket connection that is established between a client and a server, messages can be sent between the client and server in every direction. For example, the server side can actively terminate the connection and send the closed event to the client side if the timeout is reached and no more new connections are needed to be created in the entire session, thus saving time and system resources.

## Step 7. Configure the Integrated Web Application Server instance to use SSL

WebSocket also supports communication through SSL. In this step, we will configure the Integrated Web Application Server to use SSL and then test the WebSocket connection through SSL.

Click **Configure SSL** on the left panel to launch the Configure SSL wizard. In this article, we will use the default settings to create a SSL configuration. Click **Next** to specify the SSL port. In this example, port **10443** is chosen.

**Figure 8. Specify the port information for the SSL configuration**

IBM Web Administration for i  
Setup Manage Advanced Related Links  
All Servers HTTP Servers Application Servers Installations  
Running Server: INTAPPSVR - V8.5 (int app svr)  
INTAPPSVR > Configure SSL  
Configure SSL  
Specify SSL port and protocol - Step 2 of 9  
Specify the port number for secure communication and the SSL protocol for handshake. Most browsers make secure requests to port 443 by default.  
SSL port: 10443  
SSL protocol: TLS v1.0, v1.1, and v1.2  
Disable the non-SSL port?:  
☒ Yes, disable non-SSL port while configuring SSL port  
☐ No, leave non-SSL port enabled while still configuring SSL port  
Back Next Cancel

Click **Next** to specify the SSL keystore information. Retain the default value on the panel to create a new keystore.

**Figure 9. Specify the keystore information**

Configure SSL  
Specify keystore information - Step 3 of 9  
Keystore is a storage facility for cryptographic keys and certificates. The truststore contains signer certificates that are necessary for making trust decisions. For both keystore and truststore, either specify an existing one or one to be created.  
Specify keystore information:  
☒ Specify keystore path and type  
Keystore path: /www/intappsvr/wlp/usr/servers/intappsvr/resources/security/key.jks Browse eg./home/mykey.jks  
Keystore type: JKS  
☐ Use Digital Certificate Manager (DCM) SYSTEM store  
☐ Specify different path for truststore  
Back Next Cancel

Specify the password for the newly created keystore file and click **Next**.

**Figure 10. Specify the password for the keystore**

**Configure SSL**  
Specify keystore password - Step 4 of 9

Keystore is stored in a protected format to prevent unauthorized access. Specify the password to load and store the keystore. The password is also used to protect the key entry in the keystore. Keep this password in a safe place for future access to it.



Specify the password for the keystore ?

Password:

Confirm password:

Back Next Cancel

On the summary panel click **Finish** to complete the SSL configuration.

Click the stop  icon to stop the server and click the start  icon to start the server again.

## Step 8. Verify the WebSocket server through SSL

Access the <https://your.server.name:10443/WSConsole/WSConsole.html> web page to verify that the application is working. If everything is fine, the page as mentioned in the step 6 will be displayed again. Enter <wss://your.server.name:10000/WSConsole/CLRunner> in the URI field and connect to the WebSocket server through SSL. For normal WebSocket connection through HTTP, the URI starts with 'ws'. For WebSocket connection through SSL, the URI starts with 'wss'.

**Figure 11. Connecting to WebSocket using SSL**

Realtime Console based on WebSocket

```
Connecting to wss://192.168.0.101:10443/WSConsole/CLRunner
Connected.
[From Server] Connected to 192.168.0.101
WRKSBS
```

2/19/16 0:18:04 Page 0001 Subsystem Active Total

| Subsystem | Number | Jobs | Status | Storage (M) | 1 | 2 | 3 |
|-----------|--------|------|--------|-------------|---|---|---|
| QBASE     | 140275 | 29   | ACTIVE | .00         | 2 | 3 |   |
| QHTPSVR   | 161076 | 18   | ACTIVE | .00         | 2 |   |   |
| QSERVER   | 140303 | 20   | ACTIVE | .00         | 2 |   |   |
| QSPL      | 140314 | 0    | ACTIVE | .00         | 2 | 4 |   |
| QSYSWRK   | 140276 | 105  | ACTIVE | .00         | 2 |   |   |
| QOSRWRK   | 140304 | 24   | ACTIVE | .00         | 2 |   |   |
| QWAS7     | 144130 | 2    | ACTIVE | .00         | 2 |   |   |
| QWAS85    | 157610 | 0    | ACTIVE | .00         | 2 |   |   |
| ZENDSVR6  | 155312 | 6    | ACTIVE | .00         | 2 |   |   |

\*\*\*\*\* END OF LISTING \*\*\*\*\*

URI:  Connect Disconnect

Command:  Run

In the previous section, we created a WebSocket web application using Java, then configured SSL, and then ran the application on the IBM i platform. We also demonstrated some advantages of using WebSocket to run the example by comparing it with traditional HTTP request / response model. In the next section, we will show you how to create the same example by using Node.js on IBM i.

## Create a WebSocket server using Node.js

In this next section, we will explore creating and accessing a WebSocket server using Node.js running natively on IBM i.

### Step 1. Set up the Node.js environment

Node.js is included in option 1 of the 5733OPS product. For a basic Node.js environment setup and verification, refer to the article, [Native JavaScript applications on IBM i with Node.js](#). For this example, we will be using the exact same client-side JavaScript. One of the more powerful WebSocket add-ons for Node.js is Socket.IO. It is designed for real-time application interaction. You can find more details about the Socket.IO add-on at <http://socket.io/>.

In this section, we will create a project using a simple [WebSocket add-on](#). You can distribute and deploy this project to any other Node.js environment after completing these steps.

Initialize a Node.js project.

- a. Create the project directory.

In the PASE environment, create an empty directory named **wsexample** and step in to it.

```
mkdir wsexample
cd wsexample
```

- b. Initialize the project.

Issue the following command to initialize the Node.js project.

```
npm init
```

The above command guides you through the setup of the project and creation of a package.json file with the necessary information.

## Figure 12. Creating and configuring a new Node.js project

```
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See 'npm help json' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg> --save' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (wsexample)
version: (1.0.0)
description: my first websocket app
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /home/xumeng/wsexample/package.json:

{
  "name": "wsexample",
  "version": "1.0.0",
  "description": "my first websocket app",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}

Is this ok? (yes)
$
```

For this example, we will keep the default options and the created package.json file as in the following listing.

### Listing 3. Project configuration

```
{
  "name": "wsexample",
  "version": "1.0.0",
  "description": "my first websocket app",
  "main": "index.js",
  "author": ""
}
```

#### c. Install the dependencies.

Run the following command to install the websocket add-on as a dependency for your project.

```
npm install websocket --save
```

**Note:** The WebSocket add-on installation requires **gcc**. Also, refer to more information about [how to install gcc in PASE](#). The GCC support is now available through option 3 of 5733OPS.

If no error is reported, the package.json file will be automatically updated with the new dependency information.

```
"dependencies": {
  "websocket": "^1.0.22"
}
```

Now the Node.js WebSocket application, wsexample, is initialized and ready for the application to be added.

## Step 2. Create a WebSocket application for Node.js

This example uses the same HTML application that is in the **WSConsole.html** file that was used in the previous section as the WebSocket protocol is independent of the implementation. Copy the **WSConsole.html** file to the same directory of the **wsexample** project on your server.

The Node.js WebSocket server's function is exactly the same as the example we showed earlier in Java. The example supports both WebSocket connections through SSL and non-SSL. If you need to use a secure connection, you will need to prepare your private key and certificate file and put them into the project directory.

Create a JavaScript file named **index.js** in the **wsexample** project directory with following lines of code. This creates an HTTP instance listening to the 8888 port and a HTTPS instance listening to the 8889 port.

### Listing 4. WebSocket server implementation in JavaScript

```
var http = require('http');
var https = require('https'); // If need wss support
var fs = require('fs');
var url = require('url');
var WebSocketServer = require('websocket').server;
var xt = require('/QOpenSys/QIBM/ProdData/Node/os400/xstoolkit/lib/itoolkit');
var conn = new xt.iConn("*LOCAL");

// If need wss support
var options = {
  key: fs.readFileSync('privateKey.key'),
  cert: fs.readFileSync('certificate.crt')
};

function respond(req, res) {
  console.log((new Date()) + ' Received request for ' + req.url);
  var realPath = __dirname + url.parse(req.url).pathname;
  fs.exists(realPath, function(exists){
    if(exists){
      var file = fs.createReadStream(realPath);
      res.writeHead(200, {'Content-Type': 'text/html'});
      file.on('data', res.write.bind(res));
      file.on('close', res.end.bind(res));
    } else {
      res.writeHead(404, {'Content-Type': 'text/plain'});
      res.end("404 Not Found");
    }
  });
}

var server = http.createServer(respond);
var sserver = https.createServer(options, respond); // If need wss support

var wsServer = new WebSocketServer({
  httpServer: [server, sserver], // If need both ws and wss support, use this.
  // httpServer: server, // If only need ws support, use this.
  autoAcceptConnections: false
});

wsServer.on('request', function(request) {
  var path = url.parse(request.resource).pathname;
  console.log((new Date()) + ' Received request for ' + path + '.');
  var connection = request.accept(null, request.origin);
  var addr = connection.socket.localAddress;
  console.log((new Date()) + ' Peer ' + connection.remoteAddress + " connected.");
```

```
connection.sendUTF("[From Server] Connected to " + addr);
connection.on('message', function(message) {
  if (/^\/CLRunner\/?$/.test(path) && message.type === 'utf8') {
    console.log((new Date()) + ' Received Message: ' + message.utf8Data);
    conn.add(xt.iSh("system -i '" + message.utf8Data + "'"));
    conn.run(function(result) {
      connection.sendUTF("[From Server] " + addr + "\n" + result.slice(50, -18));
    });
  }
});
connection.on('close', function(reasonCode, description) {
  console.log((new Date()) + ' Peer ' + connection.remoteAddress + ' disconnected.');
```

In the code in Listing 4, we created a WebSocket server using the existing HTTP / HTTPS object. Actually the construct function of `WebSocketServer` allows binding to multiple HTTP objects represented as an array parameter. It is recommended to disable `autoAcceptConnections` to reject invalid requests for security reasons.

When the WebSocket server receives a connection request, the event handler `wsServer.on('request')` establishes the connection. Then the connection handler `connection.on('message')` reads the message and responds to the client.

Start the WebSocket server by calling **node index.js** in the project root directory.

### Figure 13. Launching the Node.js WebSocket application

```
node index.js
Tue Nov 17 2015 10:30:52 GMT+0000 (CST) HTTP Server is listening on port 8888
Tue Nov 17 2015 10:30:52 GMT+0000 (CST) HTTPS Server is listening on port 8889
```

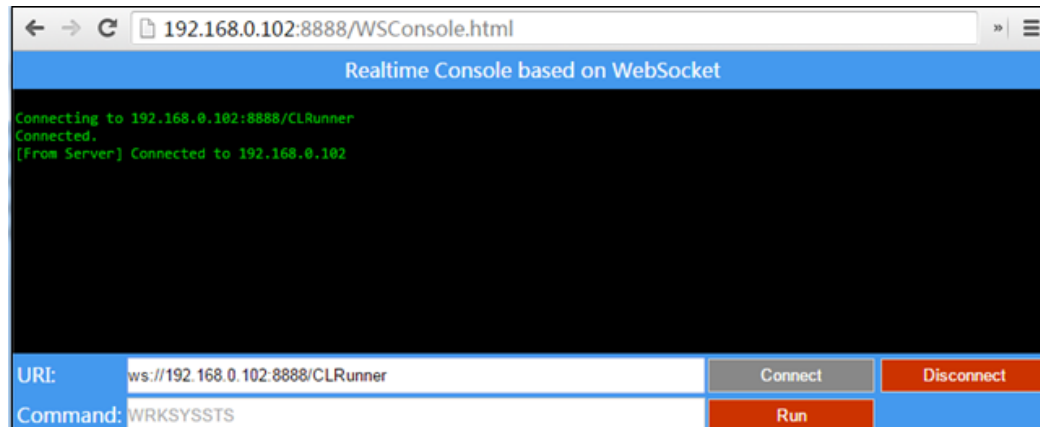
### Step 3. Verify the WebSocket server

Because we are using the same test client, ensure that the **WSConsole.html** file is copied into the project directory. Launch your favorite browser and enter *http://your.server.ip:8888/WSConsole.html* or *https://your.server.ip:8889/WSConsole.html* if you have SSL configured. If no problem occurs, the same page as shown in the Java example will be displayed.

In the first text box, fill in the WebSocket server address. It should be something like *ws://your.server.ip:8888/CLRunner* or *wss://your.server.ip:8889/CLRunner* if you need SSL / Transport Layer Security (TLS) connections. Then click **Connect**.

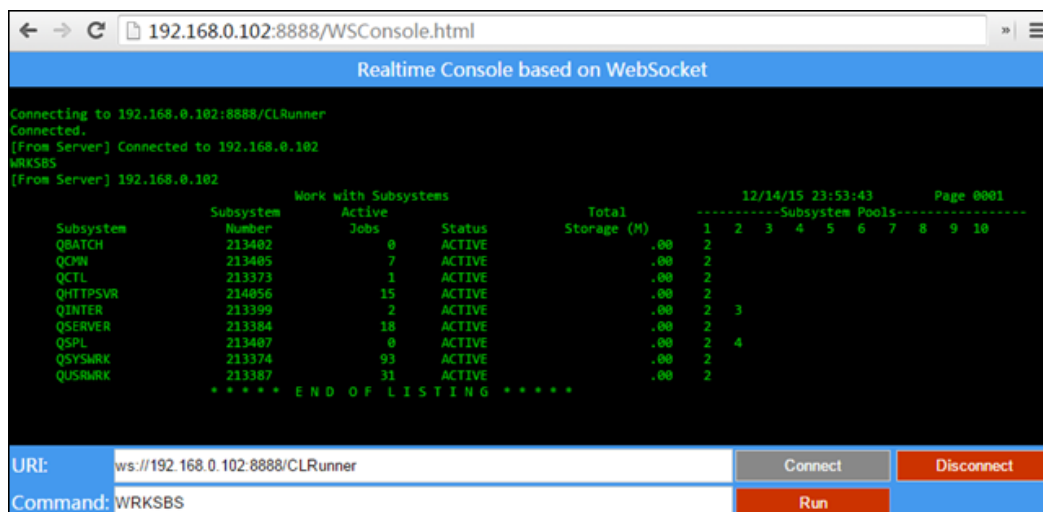


**Figure 14. Connecting to the WebSocket server in Node.js**



If there are no problems, the server responds with a message indicating that the connection has been established. Now, we can fill in the second text box with the CL command (for example, WRKSBS) you need to run. Click **Run** to send the request. You will see the response from the server, as shown in Figure 15.

**Figure 15. Run a CL command through WebSocket in Node.js**

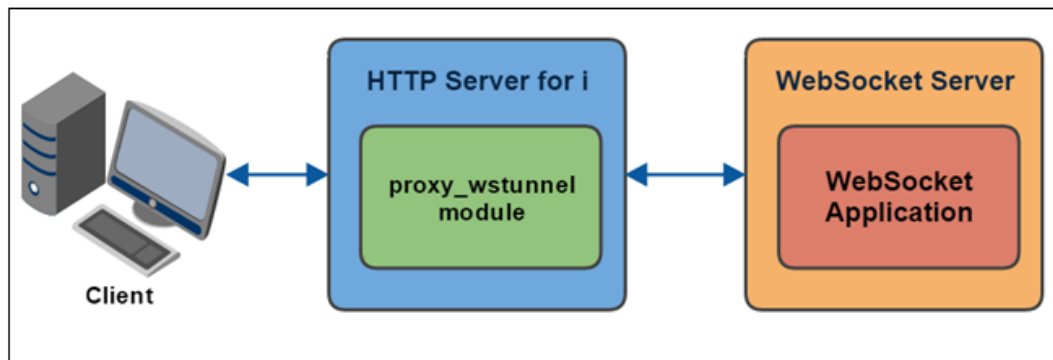


Now that you have a simple WebSocket application (**wsexample**) running, you can copy the **wsexample** directory and distribute it to any other system that has proper version of Node.js installed and configured.

## Associate HTTP Server for i with WebSocket

This section introduces how to associate the HTTP Server for i with the WebSocket applications we just created to make a full production web environment on IBM i.

**Figure 16. Tunnel WebSocket request to back-end WebSocket server through HTTP Server on i**



### Step 1. Create an instance of HTTP Server for i

An instance of HTTP Server for i must exist before you can associate the HTTP Server with the WebSocket applications. IBM Web Administration for i (<http://your.server.name:2001/HTTPAdmin>) can help to create an HTTP Server for i instance. For details on how to create an instance of the HTTP Server for i, refer to [IBM i Knowledge Center](#).

This article uses HTTP server WebSocket (root directory: /www/websocket, port: 80, SSL virtual host port: 443) running on server 192.168.0.103 as an example to demonstrate the configuration and association of HTTP Server for i with WebSocket on IBM i.

### Step 2. Configure HTTP Server for i

In order to associate HTTP Server for i with a WebSocket, the new Apache 2.4.x **mod\_proxy\_wstunnel** module is needed. This module requires the service of mod\_proxy and provides support for the tunneling of WebSocket connections to a back-end WebSocket server. The connection is automatically upgraded to a WebSocket connection through the following HTTP headers.

```
Upgrade: WebSocket
Connection: Upgrade
```

A simple way to create proxy requests to the WebSocket server is to use the ProxyPass directive, as shown in the following example.

```
ProxyPass /ws/ ws://echo.websocket.org/
ProxyPass /wss/ wss://echo.websocket.org/
```

In the following example, we associate the HTTP Server to the Java WebSocket server that we created in the previous section. Then, specify the following directives in the HTTP Server configuration (httpd.conf) file.

### Listing 5. HTTP Server configuration (httpd.conf) file snippet

```
LoadModule ibm_ssl_module /QSYS.LIB/QHTTPSVR.LIB/QZSRVSSL.SRVPGM
```

```

LoadModule proxy_module /QSYS.LIB/QHTTPSVR.LIB/QZSRCORE.SRVPGM
LoadModule proxy_http_module /QSYS.LIB/QHTTPSVR.LIB/QZSRCORE.SRVPGM
LoadModule proxy_connect_module /QSYS.LIB/QHTTPSVR.LIB/QZSRCORE.SRVPGM
LoadModule proxy_ftp_module /QSYS.LIB/QHTTPSVR.LIB/QZSRCORE.SRVPGM
LoadModule proxy_wstunnel_module /QSYS.LIB/QHTTPSVR.LIB/QZSRCORE.SRVPGM

ProxyPass /ws ws://192.168.0.101:10000/WSSConsole/CLRunner

<VirtualHost *:443>
    SSLEngine On
    SSLAppName IBM_SSL_TEST
    SSLProtocolDisable SSLv2 SSLv3
    SSLProxyEngine on
    SSLProxyAppName IBM_SSL_TEST
    SSLProxyProtocolDisable SSLv2 SSLv3
    SSLProxyVerify 1
    ProxyPass /wss wss://192.168.0.101:10443/WSSConsole/CLRunner
</VirtualHost>

```

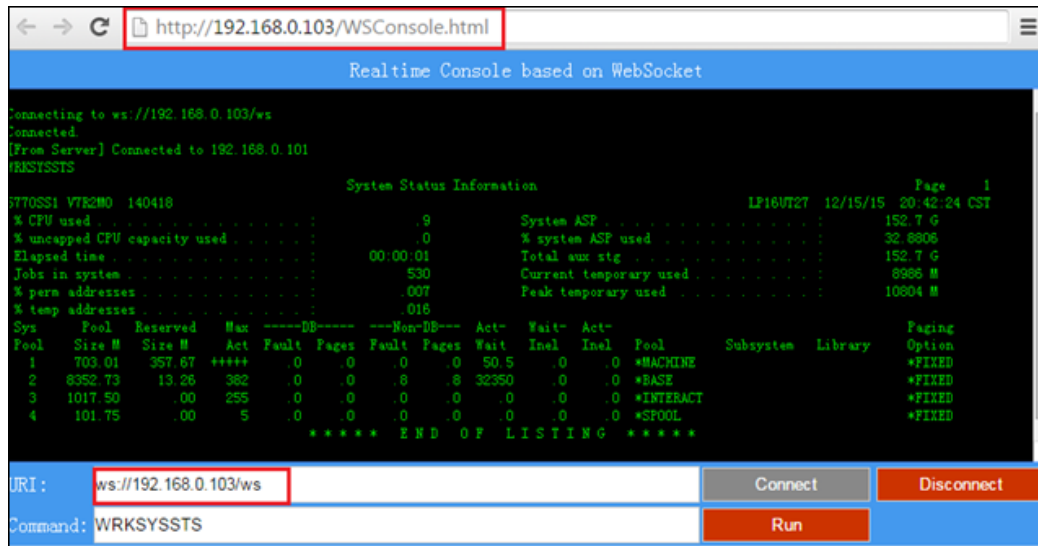
**Note:**

- For SSL support, create an application ID, IBM\_SSL\_TEST, and assign a valid certificate to it using the WebAdmin GUI HTTP SSL wizard or the Digital Certificate Manager (DCM) tool.
- To allow the testing client interface to work, place the same WSSConsole.html file in the HTTP Server document root and ensure that the default HTTP Server user profile QTMHHTTP has the \*RX authority to it.
- Replace 192.168.0.101 with your server IP address or host name on which your Java WebSocket is running.

**Step 3. Verify the association between the WebSocket server and HTTP Server for i**

Start the HTTP Server for i with the CL command **STRTCPSVR SERVER(\*HTTP) HTTPSVR (websocket)** or by using the Web Admin GUI. Open a web browser and enter the URL: `http://192.168.0.103/WSSConsole.html`.

The testing interface would be displayed. Enter `ws://192.168.0.103/ws` in the URI field and click **Connect** to connect to the WebSocket server. When it shows that the server has been successfully connected, enter the CL command, WRKSYSSTS in the Command field and click **Run**. The output of the CL command is displayed as shown in the following figure.

**Figure 17. Connecting to the WebSocket URI provided by HTTP Server**

To establish a WebSocket connection, the client sends a WebSocket handshake request, for which the server returns a WebSocket handshake response. We captured the followed HTTP request and response headers for this example.

## Listing 6. HTTP request

```

GET http://192.168.0.103/ws HTTP/1.1
Origin: http://192.168.0.103
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: GqQ0T2I5N+f9lp+daNccwQ==
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: 192.168.0.103

```

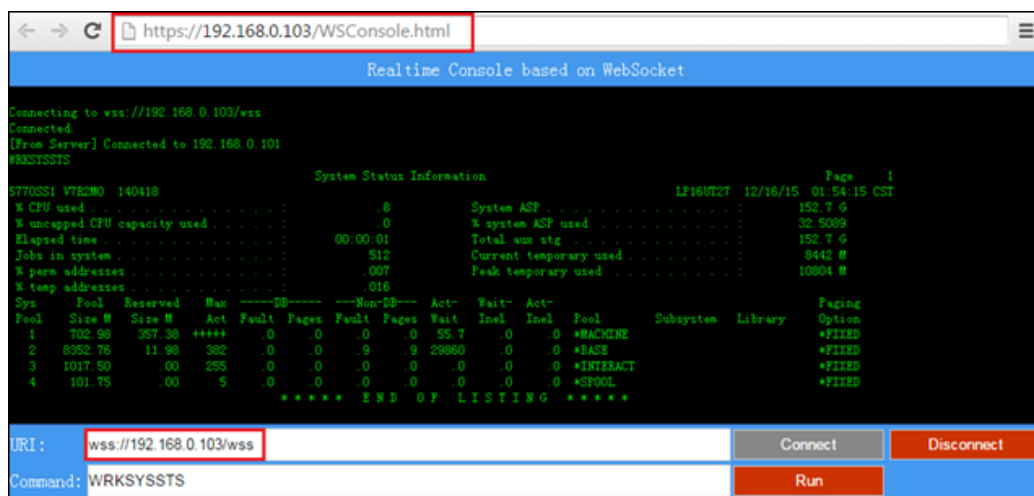
## Listing 7. HTTP response

```

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: 2M7J7SdJKynSSUCMPPVifgYKZ7E=
Origin: http://192.168.0.103

```

**Figure 18. Connect to the WebSocket SSL URI provided by HTTP Server**



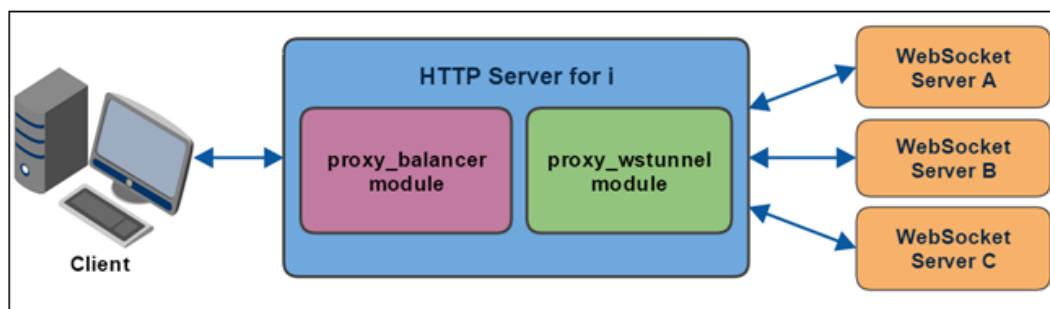
You can modify the `ProxyPass` directive to use the Node.js WebSocket server as the back-end server and do the same test.

In the previous section, we demonstrated the simplest way to associate the IBM i HTTP Server to the Java-based WebSocket application by using the new `proxy_wstunnel` module and access it in both non-SSL and SSL modes. The next section shows you a more complex scenario that uses WebSocket and HTTP Server for load balancing.

## Using WebSocket and HTTP Server for load balancing

The previous section demonstrates how to connect to a WebSocket server using the `ProxyPass` directive. In addition, it can also allow us to use the WebSocket in a HTTP Server as a load-balancing server. The following section shows you how to do that.

**Figure 19. Connecting to WebSocket server through HTTP Server as a load balancer**



Remove or comment out the previous `ProxyPass` directives, add the `proxy_balancer` and `byrequests` load-balancing method modules to the `httpd.conf` file (note the lines that are in **bold**).

### Listing 8. HTTP Server configuration (`httpd.conf`) file snippet

```
LoadModule ibm_ssl_module /QSYS.LIB/QHTTPSVR.LIB/QZSRVSSL.SRVPGM
```

```

LoadModule proxy_module /QSYS.LIB/QHTTPSVR.LIB/QZSRCORE.SRVPGM
LoadModule proxy_http_module /QSYS.LIB/QHTTPSVR.LIB/QZSRCORE.SRVPGM
LoadModule proxy_connect_module /QSYS.LIB/QHTTPSVR.LIB/QZSRCORE.SRVPGM
LoadModule proxy_ftp_module /QSYS.LIB/QHTTPSVR.LIB/QZSRCORE.SRVPGM
LoadModule proxy_balancer_module /QSYS.LIB/QHTTPSVR.LIB/QZSRCORE.SRVPGM
LoadModule proxy_wstunnel_module /QSYS.LIB/QHTTPSVR.LIB/QZSRCORE.SRVPGM
LoadModule lbmethod_byrequests_module /QSYS.LIB/QHTTPSVR.LIB/QZSRCORE.SRVPGM

BalancerInherit Off
ProxyPassInherit Off

ProxyPass /ws balancer://mycluster/CLRunner
ProxyPassReverse /ws balancer://mycluster/CLRunner

<Proxy balancer://mycluster/CLRunner>
BalancerMember ws://192.168.0.101:10000/WSConsole
    BalancerMember ws://192.168.0.102:8888
</Proxy>

<VirtualHost *:443>
    SSLEngine On
    SSLAppName IBM_SSL_TEST
    SSLProtocolDisable SSLv2 SSLv3
    SSLProxyEngine on
    SSLProxyAppName IBM_SSL_TEST
    SSLProxyProtocolDisable SSLv2 SSLv3
    SSLProxyVerify 1

    ProxyPass /wss balancer://mycluster/CLRunner
    ProxyPassReverse /wss balancer://mycluster/CLRunner

    <Proxy balancer://mycluster/CLRunner>
        BalancerMember wss://192.168.0.101:10443/WSConsole
        BalancerMember wss://192.168.0.102:8889
    </Proxy>
</VirtualHost>

```

We use the default `byrequests` load balancing method in our example. You can also choose `bybusyness` or `bytraffic` load-balancing methods.

**Note:** If you choose to use the `bybusyness` or the `bytraffic` load-balancing method, you also have to do the following changes to your `httpd.conf` file. In this example, we use `bytraffic`.

### 1. Load the `bytraffic` module.

```
LoadModule lbmethod_bytraffic_module /QSYS.LIB/QHTTPSVR.LIB/QZSRCORE.SRVPGM
```

### 2. Set the `lbmethod` balancer parameter.

```

<Proxy balancer://mycluster/CLRunner>
    BalancerMember wss://192.168.0.101:10443/WSConsole
    BalancerMember wss://192.168.0.102:8889
    ProxySet lbmethod=bytraffic
</Proxy>

```

Perform the same test as in the previous section. Then you will find the first time HTTP Server connects to the Node.js WebSocket server to run CL commands. Click **Disconnect** or open a new browser to connect again. The second time, the HTTP server connects to the Java WebSocket server to run the CL commands. The third time, it connects to the Node.js WebSocket server,

and for the fourth time, it connects to the Java Websocket server, and so on. The back-end Node.js and Java WebSocket server take turns to be connected by the HTTP Server when a new connection is created, and that is how the `byrequest` load-balancing method works.

**Figure 20. Connecting to the WebSocket server and verifying the load balance**

Connecting to wss://192.168.0.103/wss  
Connected  
[From Server] Connected to 192.168.0.103  
WRKSYSSTS

System Status Information

| Sys | Pool    | Reserved | Max  | DB | Non-DB | Act | Wait  | Incl | Incl | Pool | Subsystem | Library | Paging |
|-----|---------|----------|------|----|--------|-----|-------|------|------|------|-----------|---------|--------|
| 1   | 702.71  | 357.41   | ++++ | 0  | 0      | 0   | 50.1  | 0    | 0    | 0    | *MACHINE  |         | *FIXED |
| 2   | 8353.03 | 12.30    | 382  | 0  | 0      | 0   | 34665 | 0    | 0    | 0    | *BASE     |         | *FIXED |
| 3   | 1017.50 | .00      | 255  | 0  | 0      | 0   | 0     | 0    | 0    | 0    | *INTERACT |         | *FIXED |
| 4   | 101.75  | .00      | 5    | 0  | 0      | 0   | 0     | 0    | 0    | 0    | *SPOOL    |         | *FIXED |

\*\*\*\*\* END OF LISTING \*\*\*\*\*

Disconnected.  
Connecting to wss://192.168.0.103/wss  
Connected  
[From Server] Connected to 192.168.0.102  
WRKSYSSTS  
[From Server] 192.168.0.102

System Status Information

| Sys | Pool    | Reserved | Max  | DB  | Non-DB | Act | Wait | Incl  | Incl | Pool | Subsystem | Library | Paging |
|-----|---------|----------|------|-----|--------|-----|------|-------|------|------|-----------|---------|--------|
| 1   | 944.56  | 484.51   | ++++ | 0   | 0      | 0   | 53.0 | 0     | 0    | 0    | *MACHINE  |         | *FIXED |
| 2   | 9907.85 | 7.93     | 199  | 2.6 | 27.4   | 1.7 | 2.6  | 16657 | 0    | 0    | *BASE     |         | *FIXED |
| 3   | 25      | .00      | 5    | 0   | 0      | 0   | 0    | 0     | 0    | 0    | *SPOOL    |         | *FIXED |
| 4   | 5385.32 | .01      | 406  | 0   | 0      | 0   | 0    | 0     | 0    | 0    | *INTERACT |         | *FIXED |

\*\*\*\*\* END OF LISTING \*\*\*\*\*

URI: wss://192.168.0.103/wss [Connect] [Disconnect]  
Command: WRKSYSSTS [Run]

## Summary

WebSocket has become a very hot technology in the web communication area, and more and more customers are willing to use it to run their own web solutions on the IBM i platform. Using HTTP Server for i at the very front of your web environment is a sensible approach for your web solution. It is very useful to bundle them together to take full use of these advantages to provide a new web solution on IBM i.

## Resources

- [WebSocket.org](http://WebSocket.org)
- [About WebSocket](#)
- [IBM i 7.2 Knowledge Center](#)
- [About Apache HTTP Server Version 2.4 Documentation](#)
- [Native JavaScript applications on IBM i with Node.js](#)
- [Running Node.js with IBM HTTP Server for i](#)
- [Node.js project](#)

- [IBM SDK for Node.js](#)
- [IBM i Open Source Technologies](#)



## Downloadable resources

| Description    | Name   | Size    |
|----------------|--|---------|
| Code sample    | <a href="#">CLRunner.java</a>                | 1.41 KB |
| Code sample    | <a href="#">index.js</a>                     | 2.51 KB |
| Console sample | <a href="#">WSConsole.html</a>               | 4.16 KB |
| Code sample    | <a href="#">WSConsoleServerEndPoint.java</a> | 1.52 KB |

© Copyright IBM Corporation 2016

([www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml))

[Trademarks](#)

([www.ibm.com/developerworks/ibm/trademarks/](http://www.ibm.com/developerworks/ibm/trademarks/))