**HTML5, WebSockets & server push**

**Cookbook met recipes**

**voor**

**Hands-on code labs & demo’s**

Inhoud

[Software-installatie 4](#_Toc447631681)

[JDK 4](#_Toc447631682)

[Git 4](#_Toc447631683)

[Maven 5](#_Toc447631684)

[Eclipse 5](#_Toc447631685)

[NodeJs 5](#_Toc447631686)

[Glassfish 6](#_Toc447631687)

[Wireshark 6](#_Toc447631688)

[Curl 7](#_Toc447631689)

[Fiddler 7](#_Toc447631690)

[H2c 8](#_Toc447631691)

[Optionele pakketten 9](#_Toc447631692)

[Firefox nightly 9](#_Toc447631693)

[Chrome Canary 9](#_Toc447631694)

[OpenSSL 9](#_Toc447631695)

[Vagrant & Virtual Box 10](#_Toc447631696)

[Cygwin 10](#_Toc447631697)

[URLs van beschikbare software per OS / environment 11](#_Toc447631698)

[Java 11](#_Toc447631699)

[Windows 11](#_Toc447631700)

[Linux 11](#_Toc447631701)

[Mac OS X 12](#_Toc447631702)

[Handige command-line opdrachten 13](#_Toc447631703)

[Windows 13](#_Toc447631704)

[Linux / Mac 13](#_Toc447631705)

[Websockets code lab recipe 14](#_Toc447631706)

[Websocket client side 14](#_Toc447631707)

[Websocket server side 15](#_Toc447631708)

[Server Sent Events code lab recipe 18](#_Toc447631709)

[Jersey JAX-RS met SSE extensie 18](#_Toc447631710)

[Servlet 3.0 code lab recipes 21](#_Toc447631711)

[Servlet 3.0 21](#_Toc447631712)

[Jersey JAX-RS async 23](#_Toc447631713)

[WebSockets lab recipes 25](#_Toc447631714)

[Wireshark & websockets 25](#_Toc447631715)

[Fiddler & websockets 27](#_Toc447631716)

[Chrome & websockets 30](#_Toc447631717)

[SPDY lab recipe 31](#_Toc447631718)

[Demo SPDY 34](#_Toc447631719)

[Klaarzetten 34](#_Toc447631720)

[Uitvoeren demo SPDY 35](#_Toc447631721)

[HTTP/2 lab recipes 37](#_Toc447631722)

[HTTP/2 & browsers 37](#_Toc447631723)

[HTTP/2 & NodeJs 39](#_Toc447631724)

[Windows cmd.exe: 40](#_Toc447631725)

[Linux/Mac/cygwin: 40](#_Toc447631726)

[Variant via TLS-loze HTTP/2 op poort 80 (alleen Linux/Mac/cygwin syntax): 42](#_Toc447631727)

[HTTP/2(-loze) connectie met *externe sites* 43](#_Toc447631728)

[curl HTTP/2 43](#_Toc447631729)

[Vanuit de browser 44](#_Toc447631730)

[HTTP/2 & Jetty 45](#_Toc447631731)

[HTTP/2 & h2c 47](#_Toc447631732)

[HTTP/2 & Tomcat9 49](#_Toc447631733)

[HTTP/2 & Wireshark 52](#_Toc447631734)

# Software-installatie

**Binaries** vind je op de USB stick. De download URLs hieronder kun je evt. ook gebruiken. Er wordt uitgegaan van destijds (30-oct-2014) laatste 64bit Intel *stable* versies. Nieuwer mag natuurlijk ook.

Gebruik voor installeren de **package manager** die hoort bij je OS.

Genoemde **paden** zijn geldig onder Windows (7 / 8.1). Op Linux/Mac vanzelfsprekend deze paden aanpassen naar bijv. /usr/local/bin of ~.

Installatie op Windows bij voorkeur naar c:\Program Files (dus niet ‘ (x86)’) ofwel **c:\Progra~1** (zelfde pad maar dan zonder spaties – vaak betrouwbaarder). Bijv. Vagrant gaat default naar C:\Hashicorp\Vagrant – vermijd dit soort afwijkende paden zo mogelijk.  
Uitzondering is GlassFish. Zet die aub direct onder c:\ om permissieproblemen te voorkomen!

Waar extra configuratie nodig is, zijn **command line** shell (command prompt) **instructies** genoemd. Voel je vrij om ze desgewenst via bv. een file browser (explorer) uit te voeren…

Pas op: voorbereiding van dit *hands-on lab* is alleen op **Windows** (7 Ultimate en 8.1) uitgevoerd. Windows 10, **Mac en Linux** zou grotendeels hetzelfde moeten gaan & functioneren, maar: YMMV!

Ter “leeringh ende vermaeck” wordt bij elk **pakket** een aantal **alternatieven** genoemd. Gebruik die gerust bij de labs, maar: you’re on your own!

## JDK

Wat: Java compiler, bibliotheek en virtuele machine

Alternatieven: NodeJs, Ruby

Doel: div. code labs.

1. Installeer JDK
2. voeg toe aan PATH: C:\Progra~1\Java\jdk1.8.0\_20\bin
3. definieer environment variable JAVA\_HOME C:\Progra~1\Java\jdk1.8.0\_20

## Git

Wat: gedistribueerd versiebeheersysteem

Alternatieven: Subversion, Mercurial

Doel: code binnenhalen voor de labs.

1. Installeer git
2. voeg toe aan PATH: C:\Progra~1\git\bin

## Maven

Wat: bouwen van softwarecomponenten en beheren van afhankelijkheden ertussen

Alternatieven: Gradle, Grunt+npm

Doel: div. code labs, o.a. Jersey JAX-RS async

Stappen:

1. Installeer maven
2. voeg toe aan PATH: C:\Progra~1\apache-maven-3.3.9\bin

## Eclipse

Wat: geintegreerde ontwikkelomgeving

Alternatieven: NetBeans, IntelliJ

Doel: div. code labs, o.a. Jersey JAX-RS async

Stappen:

1. unzip eclipse\*.zip naar bijv. C:\Progra~1\eclipse

## NodeJs

Wat: javascript virtuele machine (node) en beheren van softwarecomponentafhankelijkheden (npm)

Alternatieven: Rhino, Nashorn

Doel: websockets en HTTP/2 labs.

Stappen:

1. Installeer NodeJs  
   Dit zet automatisch npm en node op het pad
2. Cmdline: mkdir C:\Users\<id>\AppData\Roaming\npm
   1. op Linux/Mac wordt dit waarschijnlijk ~/.npm ???

Voer je stap 2 *niet* uit, dan zie je bij npm install waarschijnlijk de volgende error:

Error: ENOENT, stat 'C:\Users\*JeGebruikersnaam*\AppData\Roaming\npm'

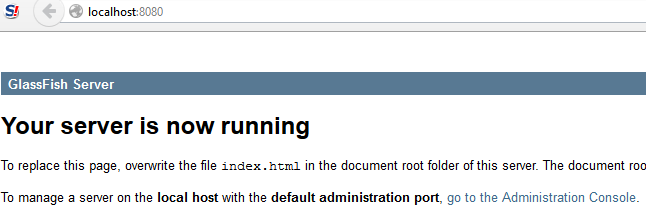
## Glassfish

Wat: applicatieserver voor JEE. Versie 4 is de referentieimplementatie voor JEE7.

Alternatieven: Tomcat 8 & Jetty 9.1.x (alleen voor Servlets), WildFly 8 (JEE7-compliant)

Doel: JEE7 websocket hands-on lab.

Stappen:

1. unzip glassfish-4.1\*zip naar c:\  
   NB heb je glassfish al via een installer geïnstalleerd, dan kan het hieronder gebruikte padnaam verschillen!
2. voeg toe aan PATH: C:\glassfish4\bin
3. cmdline: c:\glassfish4\bin\asadmin start-domain
4. admin console: <http://localhost:4848>   
   
5. site: <http://localhost:8080>   
   

## Wireshark

Wat: netwerksniffer

Alternatieven: Fiddler, tcpdump, tcptrace

Doel: websockets en HTTP/2 labs.

Stappen:

1. Installeer Wireshark
   * Linux: via je package manager
2. Zet RawCap.exe op een via cmdline te bereiken plek neer

## Curl

Wat: command line HTTP communicatie

Alternatieven: wget, postman Chrome plugin, RESTclient of Poster firefox plugins, SoapUI

Doelen:

* POST’en van data bij Jersey JAX-RS code lab
* checken lijndata bij SSE code lab
* HTTP/2 traffic

Stappen:

1. Windows: installeer / unzip meegeleverde curl
2. Cygwin: via setup.exe
3. Linux / Mac: via package manager

## Fiddler

Wat: web debugging proxy. Ondersteunt nog (steeds!) geen HTTP/2, en websockets maar deels.

Alternatieven: Wireshark, Charles, tcptrace

Doel: websockets lab

Stappen:

1. Installeer Fiddler
   1. Linux/Mac: installeer eerst Mono; zie URL hieronder
2. Voor Firefox:
   1. Extra > Monitor with Fiddler > Automatic
   2. Zo nodig proxy handmatig naar localhost:8888 verwijzen
3. Kopieer naar C:\Users\<User>\Documents\Fiddler2\Scripts  
   *(voor 2 verschillende recipes!)*
   1. CustomRules - Fiddler fakewebsocket.js  
      .. rename naar CustomRules.js & (her)start Fiddler
   2. CustomRules - Fiddler ws2log.js  
      .. en rename naar CustomRules.js & (her)start Fiddler

## H2c

Wat: locale HTTP/2 proxy

Alternatieven: Wireshark

Doel: HTTP/2 lab

Stappen:

1. Download van <https://github.com/fstab/h2c/releases/download/v0.0.11/h2c-v0.0.11.zip>
   1. of haal uit de Generic of Win64 folders in de repo.
2. Either way, haal hieruit *voor jouw platform* de juiste h2c(.exe)
3. Zet ‘m ergens op je PATH

# Optionele pakketten

## Firefox nightly

Wat: *bleeding edge* versie van de bekende web browser.

Alternatieven: Chrome Canary

Doel: in het verleden voor het HTTP/2 lab. Alle features lijkt nu in de normale release te zitten.

Stappen:

1. Download van <https://nightly.mozilla.org/>
2. Installeer

## Chrome Canary

Wat: *bleeding edge* versie van de bekende web browser.

Alternatieven: Firefox nightly

Doel: in het verleden voor het HTTP/2 lab. Alle features lijkt nu in de normale release te zitten.

Stappen:

1. Download van <http://www.google.com/intl/en/chrome/browser/canary.html>
2. Installeer

## OpenSSL

Wat: command line tool voor alles rondom crypto, SSL, certificaten & keystores.

Alternatieven: alleen *payware*

Doel: Optioneel! Gebruikt door uw docent bij inrichten van demo-omgevingen.

Stappen:

1. (Windows-only) Installeer de VC++ 2008 redistributable
2. Installeer OpenSSL

## Vagrant & Virtual Box

Wat: respectievelijk snel *opzetten* van een virtuele omgeving, en het *draaien* ervan

Alternatieven & toevoegingen:

* Puppet, boxgrinder ??, docker, ansible
* VMware Player/Fusion/Workstation/ vSphere Hypervisor, Parallels (Mac), Xen (Linux), KVM (Linux)

Doel: Optioneel! Gebruikt door uw docent bij inrichten van demo-omgevingen.

Stappen:

1. Installeer Vagrant
2. Installeer Virtual Box
3. Restart

## Cygwin

Wat: Voegt Unix-like omgeving toe aan Windows. Love it or hate it!

Alternatieven: MinGW, PowerShell, coLinux

Doel: Optioneel! Favoriete omgeving van uw cursusdocent…

Stappen:

1. Download setup\*.exe (zie URL beneden)
2. Start setup\*exe
3. Klik Next > Next > Next etc
4. Kies de packages die je wilt, bijv.
   1. openssl
   2. curl
   3. git
   4. patch
5. Installeer

# URLs van beschikbare software per OS / environment

## Java

* <http://apache.mirror1.spango.com/maven/maven-3/3.3.9/binaries/apache-maven-3.3.9-bin.zip>
* <http://mirror.nl.webzilla.com/apache/tomcat/tomcat-8/v8.0.12/bin/apache-tomcat-8.0.12.zip>
* <http://mirrors.supportex.net/apache/tomcat/tomcat-9/v9.0.0.M4/bin/apache-tomcat-9.0.0.M4-windows-x64.zip>
* <http://dlc.sun.com.edgesuite.net/glassfish/4.1/release/glassfish-4.1.zip>

## Windows

* <portable> <https://1.eu.dl.wireshark.org/win32/WiresharkPortable-1.12.1.paf.exe>
* <https://1.eu.dl.wireshark.org/win64/Wireshark-win64-1.12.1.exe>
* <http://eclipse.mirror.triple-it.nl/technology/epp/downloads/release/luna/R/eclipse-jee-luna-R-win32-x86_64.zip>
* <https://nodejs.org/dist/v5.10.0/node-v5.10.0-x64.msi>
* <http://www.telerik.com/docs/default-source/fiddler/fiddler2setup.exe?sfvrsn=2>
* <http://www.telerik.com/docs/default-source/fiddler/fiddler4setup.exe?sfvrsn=2>
* <Windows service> <http://apache.mirror1.spango.com/tomcat/tomcat-8/v8.0.12/bin/apache-tomcat-8.0.12.exe>
* <http://download.oracle.com/otn-pub/java/jdk/8u20-b26/jdk-8u20-windows-x64.exe?AuthParam=1411632876_6294683ae3b260561f5e53233a5a10a9>
* <http://winampplugins.co.uk/curl/curl_7_48_0_openssl_nghttp2_x64.7z>
* <http://slproweb.com/download/Win64OpenSSL-1_0_1i.exe>
* VC++ 2008 x64 redistributable: <http://www.microsoft.com/en-us/download/details.aspx?id=15336>
* <https://dl.bintray.com/mitchellh/vagrant/vagrant_1.6.5.msi>
* <http://download.virtualbox.org/virtualbox/4.3.16/VirtualBox-4.3.16-95972-Win.exe>
* Cygwin installer & package manager:
  + <https://cygwin.com/setup-x86.exe> (32b)
  + <https://cygwin.com/setup-x86_64.exe> (64b - af te raden; veel minder packages)

## Linux

* < WireShark: via pkg mgr>
* <http://eclipse.mirror.triple-it.nl/technology/epp/downloads/release/luna/R/eclipse-jee-luna-R-linux-gtk-x86_64.tar.gz>
* <https://nodejs.org/dist/v5.10.0/node-v5.10.0-linux-x64.tar.gz>
* <Fiddler: zie <http://fiddler.wikidot.com/mono>>
* <http://download.oracle.com/otn-pub/java/jdk/8u20-b26/jdk-8u20-linux-x64.rpm?AuthParam=1411632845_8f3db891c4c88929e5e992afc3ede757>
* <curl: installeer via je distro>
* <git: installeer via je distro>
* <openssl: installeer via je distro>
* <https://dl.bintray.com/mitchellh/vagrant/vagrant_1.6.5_x86_64.deb>  
  of  
  <https://dl.bintray.com/mitchellh/vagrant/vagrant_1.6.5_x86_64.rpm>
* Div. Virtual Box downloads, zie <https://www.virtualbox.org/wiki/Linux_Downloads>

## Mac OS X

* <https://1.eu.dl.wireshark.org/osx/Wireshark%201.12.1%20Intel%2064.dmg>
* <http://eclipse.mirror.triple-it.nl/technology/epp/downloads/release/luna/R/eclipse-jee-luna-R-macosx-cocoa-x86_64.tar.gz>
* <https://nodejs.org/dist/v5.10.0/node-v5.10.0.pkg>
* <Fiddler: zie <http://fiddler.wikidot.com/mono>>
* <http://download.oracle.com/otn-pub/java/jdk/8u20-b26/jdk-8u20-macosx-x64.dmg?AuthParam=1411632858_49eafbb2e3556e89c617f4883db906ee>
* <curl: zie <http://pdb.finkproject.org/pdb/package.php/curl>>
* <http://switch.dl.sourceforge.net/project/git-osx-installer/git-2.0.1-intel-universal-snow-leopard.dmg>
* <openssl: installeer via je distro>
* <https://dl.bintray.com/mitchellh/vagrant/vagrant_1.6.5.dmg>
* <http://download.virtualbox.org/virtualbox/4.3.16/VirtualBox-4.3.16-95972-OSX.dmg>

# Handige command-line opdrachten

## Windows

services.msc

<as admin> net stop tomcat8

netstat -ao / -ano

ipconfig

## Linux / Mac

sudo su

service apache2 restart

netstat

ifconfig

# Websockets code lab recipe

Doel: werken met websockets.

## Websocket client side

**Case:** fake “number cruncher” service die via websockets op een vast interval live ‘berekende’ data (bijv. responsetijden) toont als staafdiagram.

Client side maakt gebruik van Angular, d3 en jQuery. Statics worden geserveerd door NodeJs.

Server side maakt gebruik van een NodeJs websockets component. De data wordt ‘berekend’ door Math.random().

**Taak:** javascript client side code werkend maken; de server side is compleet.

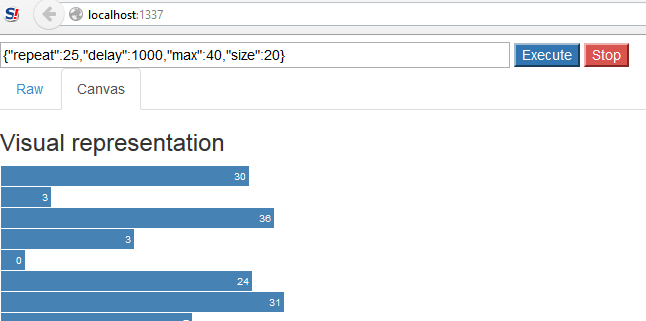
**Benodigd:**

* NodeJs
* Browser
* Internetverbinding (voor NPM)

**Stappen:**

1. Ga op de command line naar <workspace>\node-ng-d3-websocket
2. Haal dependencies op: npm install
3. vul de ontbrekende delen (TODO) in app\js\numbers.app.js in – zie sheets & commentaar
4. Start server side: node ws-server.js
5. open client side (numbers.app.js) in de browser: <http://localhost:1337>
6. druk op knoppen Execute / Stop om de ws verbinding te openen resp. sluiten
7. zie wat er gebeurt in de Raw & Canvas tabs
8. tip: de JSON parameters zijn instelbaar
9. sluit de NodeJs ws server (op de command line) met Ctrl-C

**Impressie:**



## Websocket server side

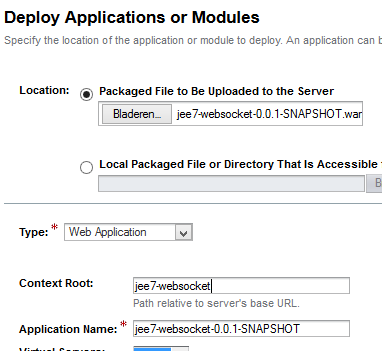
**Case:** “echo” websocket service die extra gegevens toevoegt. Server side bevat enkele “toeters en bellen”.

**Taak:** server code werkend maken; de client side functioneert.

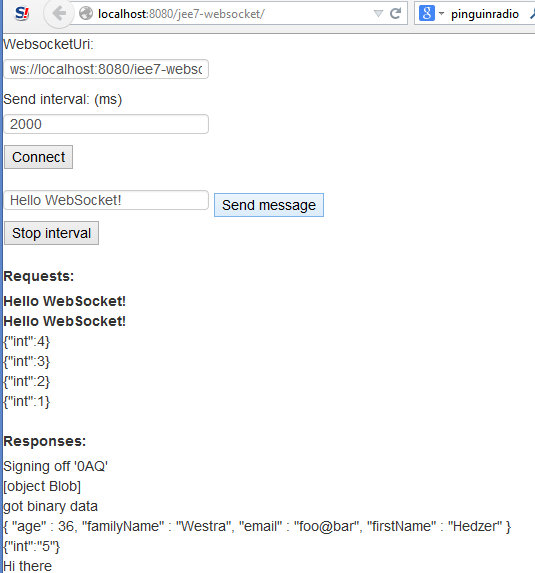
**Benodigd:**

* IDE
* Maven
* Glassfish
* Browser
* Internetverbinding (of een al complete maven local repo)

**Stappen:**

1. Start IDE
2. Open of importeer het jee7-websocket project
3. Vul de ‘TODO’ stukken in – gebruik de informatie in de sheets en/of javadoc!
4. Op de command line:
   1. Zet basedir: cd <workspace>\jee7-websocket
   2. Draai integratie tests: mvn test
   3. Bouw WAR: mvn package
   4. Start Glassfish: C:\glassfish4.1\bin\asadmin start-domain
5. Open in browser: <http://localhost:4848>
6. Applications > Deploy > Browse > selecteer <checkout locatie>\workspace\jee7-websocket\target\jee7-websocket\*.jar
7. Zet Context root naar jee7-websocket (dus zonder versienummer)  
   
8. Klik Ok > wacht
9. Open in browser: <http://localhost:8080/jee7-websocket>
10. Druk op Connect
11. De client stuurt nu elke 2s (da’s best vaak!) een bericht en krijgt een aantal antwoorden terug (binary, JSON en de input ge-echo’d). Requests en responses worden met laatste-bovenaan getoond.
    1. Druk op ‘Stop interval’ om de client te stoppen
    2. Druk op ‘Send message’ om een tekstbericht te sturen
12. Logs vind je in C:\glassfish4.1\glassfish\domains\domain1\logs\server.log
13. Stop Glassfish met C:\glassfish4.1\bin\asadmin stop-domain

**Impressie:**

****

**Bonus**:

1. Stop Glassfish: C:\glassfish4.1\bin\asadmin stop-domain
2. Start Jetty: mvn jetty:run
3. Open <http://localhost:8080>
   1. … maar functioneert helaas niet (Jetty logs: “Cannot append to finished buffer”). Een upgrade van Jetty zou kunnen helpen.  
      *Left as an exercise for the reader!*

**Achtergrondinformatie:**

De gebruikte websocket URL is   
 ws://localhost:8080/jee7-showcase/myWebSocketEndpoint/myCoolClientId  
(Jetty: ws://localhost:8080/myWebSocketEndpoint/myCoolClientId )  
Deze wordt afgeleid van de hostname waar de html staat.

De ‘myCoolClientId’-parameter – die eenmalig, bij opzetten van de ws connectie, door de client kan worden opgegeven - kan in elke @OnMessage worden geïnjecteerd.

# Server Sent Events code lab recipe

**Doel**: werken met frontend en backend SSE code, en inspecteren lijndata

## Jersey JAX-RS met SSE extensie

**Case**: simpele event source, die elke seconde een bericht met id stuurt, zowel in plaintext als JSON

**Taak**: vul de ontbrekende stukken Java code in m.b.v. de informatie in de sheets

NB de client (javascript/HTML) code functioneert al.

**Benodigd:**

* IDE
* Maven
* Curl
* Browser
* Internetverbinding (of een al complete maven local repo)
* Evt. Fiddler of tcptrace

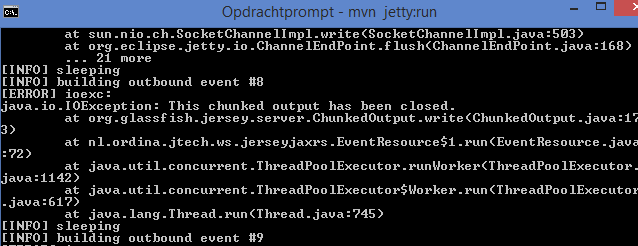
**Stappen:**

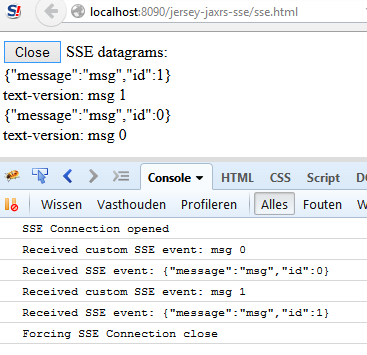
1. Start IDE
2. Open het jersey-jaxrs-sse project
3. Vul de ‘TODO’ stukken in (EventResource) – gebruik de informatie in de sheets en javadoc!
4. Zet basedir: cd <workspace>\jersey-jaxrs-async
5. Start Jetty: mvn jetty:run
6. Open in browser: <http://localhost:8090/jersey-jaxrs-sse/sse.html>
7. De berichten stromen meteen binnen – elke 1s één.
   1. Druk F12: Op de developer toolbar (Console) wordt extra informatie getoond.
8. De server verbreekt na 10x de verbinding – de browser reconnect automatisch
   1. Dit zie je ook terug in de console logs:  
      *‘SSE Connection closed from server side - browser will reopen it automagically’*
   2. … en in de Jetty INFO & ERROR logs op de command line  
      ‘done sending 10 items - connection will close from server side but browser reopens’
9. Druk op Close om de verbinding permanent te verbreken
   1. Jetty zal dan een probleem zien en loggen:  
      ‘This chunked output has been closed’
10. Bekijk de lijndata (2 verschillende events, interleaved) via een 2e command prompt waarin je direct het SSE endpoint uitvraagt:  
    curl <http://localhost:8090/jersey-jaxrs-sse/async-webapi/event>
11. Bekijk de (inline javascript) client side code in <workspace>\jersey-jaxrs-sse\src\main\webapp\sse.html
12. Stop Jetty m.b.v. Ctrl-C

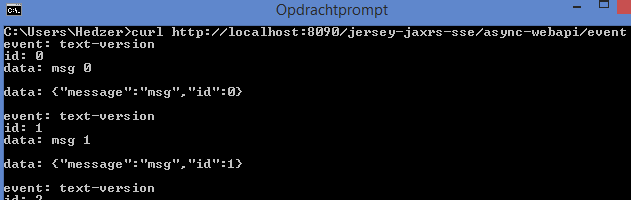
**Achtergrondinformatie:**

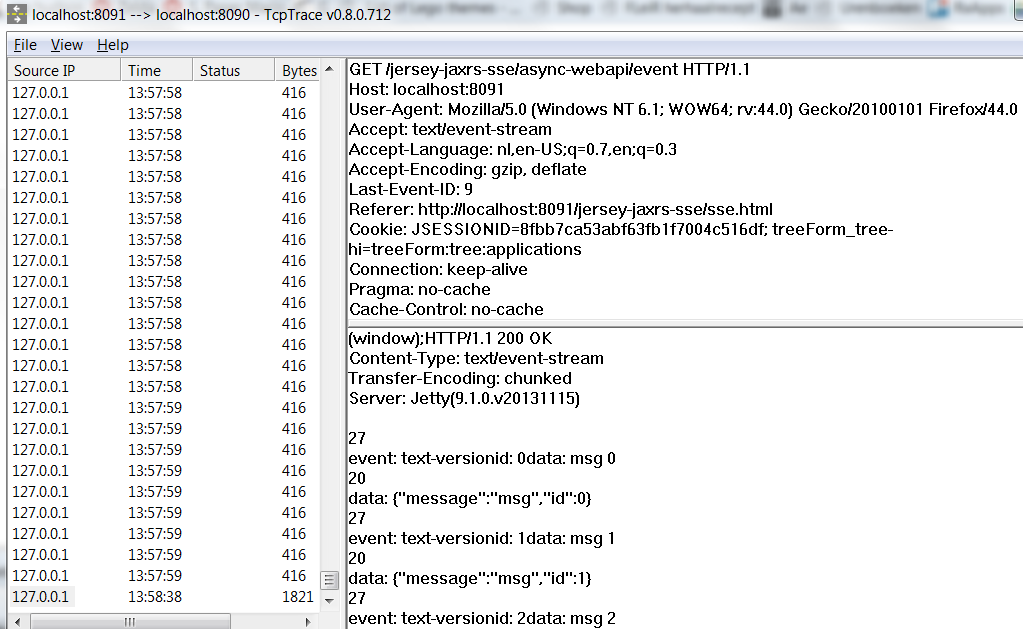
1. curl voert *geen* reconnect uit (implementeert nl. geen SSE .js API!)
2. Firefox (FireBug Net panel) toont helaas helemaal geen SSE data
3. Fiddler – zie een ander *recipe* uit dit *cookbook* – toont SSE events alleen *na* het verbreken van de verbinding
4. In Fiddler (en tcptrace) valt te zien dat elk bericht vooraf wordt gegaan door de lengte (NB: in hex!) op een losse regel. Curl en de browser decoden dat automatisch client side; Jersey encode het automatisch aan de server side.  
   Zie <https://en.wikipedia.org/wiki/Chunked_transfer_encoding> over Transfer-Encoding: chunked

**Impressie:**









**Bonus**:

1. Pas EventSourceClientIT zo aan dat de events te integration-testen zijn
2. Zie <https://jersey.java.net/documentation/latest/sse.html#d0e11897>
3. Ga met broadcasting aan de gang
   1. Zie <https://jersey.java.net/documentation/latest/sse.html#d0e11773>
4. Zet een SSE project op in GlassFish
   1. Zie <http://en.kodcu.com/2013/11/jaxrs-2-html-5-server-sent-events-on-glassfish-4/>

# Servlet 3.0 code lab recipes

**Doel**: werken met Servlet 3.x asynchrone code

## Servlet 3.0

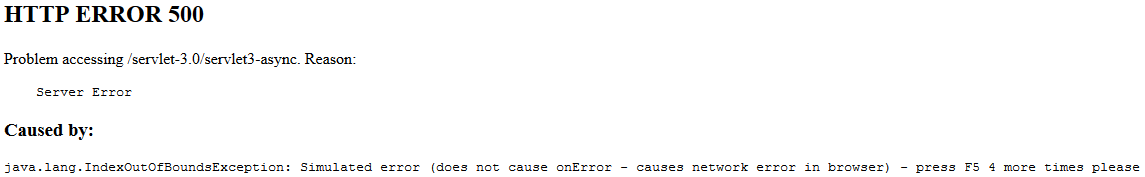
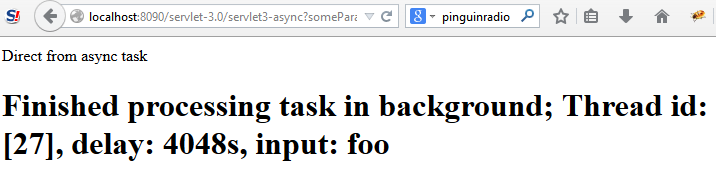
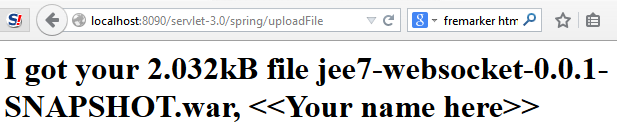
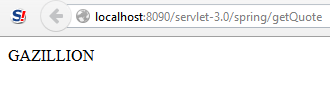
**Case**: enkele *gefingeerde* service requests die asynchroon afgehandeld worden. Naast standaard Servlet 3.0 (geen JEE7 overigens!) wordt ook Spring gebruikt. Sla dat stuk gerust over als je ge-Adam-Bien’ified bent!

**Taak**: vul de ontbrekende stukken Java code in m.b.v. de informatie in de sheets

**Benodigd**:

* IDE
* Maven
* Browser
* Optioneel: Tomcat
* Optioneel: Glassfish
* Internetverbinding (of een al gevulde maven local repo)

**Stappen**:

1. Start IDE
2. Open het servlet-3.0 project
3. Vul de voor de 3 requests hieronder de ‘TODO’ stukken in – gebruik de informatie in de sheets en javadoc!
4. Open basedir: cd <workspace>\servlet-3.0
5. Start jetty: mvn jetty:run
6. Bezoek test URLs:
   1. URL 1; Servlet 3.x standaard: <http://localhost:8090/servlet-3.0/servlet3-async?someParam=foo>   
      (kopieer deze URL in de browser; Ctrl-klik werkt niet vanwege error 500)  
      geeft 1e 5 keer **expres een error** **500** (lees de Exception message!)   
        
      en geeft daarna – na een random sleep:  
      
   2. URL 2; Spring variant – file upload en Callable: <http://localhost:8090/servlet-3.0/spring/uploadFile>   
      Dit accepteert een file via Spring MVC; afhandeling vindt plaats in de achtergrond d.m.v. teruggeven van een Callable<String>.
      1. NB: File wordt geplaatst in c:\temp (sorry, op Mac/Linux zelf even aanpassen!) en mag max. 100MiB groot zijn  
         
   3. URL 3, spring variant - DeferredResult: <http://localhost:8090/servlet-3.0/spring/getQuote>  
      fingeert het ophalen van een offertebedrag, waarbij de offerteservice batchgewijs elke 5 seconden de dan opgevraagde quotes opvraagt. De gebruikte Spring feature is DeferredResult<String>. Is de offerteservice te traag (>3s) dan volgt een timeout logregel, en in de browser een error 503, anders simpelweg:  
        
      Het kan best even duren voor je een ‘hit’ krijgt! Wees geduldig met F5…  
      Opvallend is dat de logs aantonen dat bij een 503 de methode getQuote() nogmaals aangeroepen wordt?! Onduidelijk is of dat vanuit de browser, de appserver of Spring MVC getriggerd wordt  
        
      Achtergrondinfo:
      1. de Spring async configuratie staat in servlet-context-async.xml  
         (o.a. <mvc:async-support/>)   
         Twee Spring @Service-componenten worden hierbij gebruikt:
      2. De quotes worden verzameld & beheerd in DeferredResultContainer
      3. Batchgewijze afhandling van offertes vindt plaats in QuoteUpdateScheduler
      4. Bij gebruik van Tomcat i.p.v. Jetty treedt een HTTP 500 Internal Server Error op bij timeout – bij Jetty is dat een 503 Service Unavailable! Dat laatste lijkt correcter.
7. Stop Jetty m.b.v. Ctrl-C

**Bonus**:

1. De 3s timeout voor DeferredResult in getQuote() wordt nergens expliciet gezet. Dit lijkt een Spring built-in timeout. Zoek dit uit & pas de timeout aan.
2. deploy & test het geheel op GlassFish, en vervang daarbij AsyncServlet’s zelf aangemaakte ExecutorService door een geinjecteerde @Resource ManagedExecutorService executorService.

## Jersey JAX-RS async

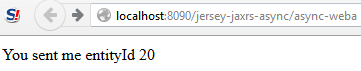
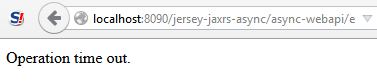
**Case**: een gefingeerde RESTful service die ‘zware’ objecten (naam: “entity”) aanmaakt & oplevert, dus pas na enige tijd antwoord geeft.

**Taak**: vul de ontbrekende stukken Java code in m.b.v. de informatie in de sheets

**Benodigd**:

* IDE
* Maven
* Curl
* Browser
* Internetverbinding (of een al gevulde maven local repo)

**Stappen**:

1. Start IDE
2. Open het jersey-jaxrs-async project
3. Vul de ‘TODO’ stukken in – gebruik de informatie in de sheets & javadoc!
4. Open basedir: cd <workspace>\jersey-jaxrs-async
5. start een embedded Jetty & voer integratietests uit: mvn verify
6. Start Jetty: mvn jetty:run
7. Test URL 1 – heavy background processing: <http://localhost:8090/jersey-jaxrs-async/async-webapi/entity/20>  
   dit doet 4s ‘heavy processing’ (gefaked door Thread.sleep()) en geeft dan:  
     
   Let ook op de Jetty logs op de console
8. Test URL 2 - timeouts: <http://localhost:8090/jersey-jaxrs-async/async-webapi/entity>   
   (copy-paste deze URL in de browser; Ctrl-klik werkt niet!)  
   geeft na 3s een timeout met HTTP 503 en “Operation time out.”  
   
9. Voer in 2e command prompt een RESTful POST uit:  
   curl <http://localhost:8090/jersey-jaxrs-async/async-webapi/entity> --data-urlencode "name=foo"  
   geeft na 4s de response ‘Created entity with name foo’:  
   NB als je liever POSTman o.i.d. gebruikt (als browserplugin) dan mag dat natuurlijk ook.
10. Stop Jetty m.b.v. Ctrl-C

**Bonus**:

1. breid AsyncEntityClient (gebruikt in de integratietest) uit met asynchroon gedrag, evt. zelfs non-blocking, dus met een callback om de response bij de aanroeper (AsyncEntityClientIT) terug te geven.

# WebSockets lab recipes

**Doel**: zien hoe WebSockets-verkeer herkenbaar is in de browser, en hoe de lijndata te volgen is & eruit ziet

## Wireshark & websockets

**Case**: werken met Wireshark om websocket data te debuggen

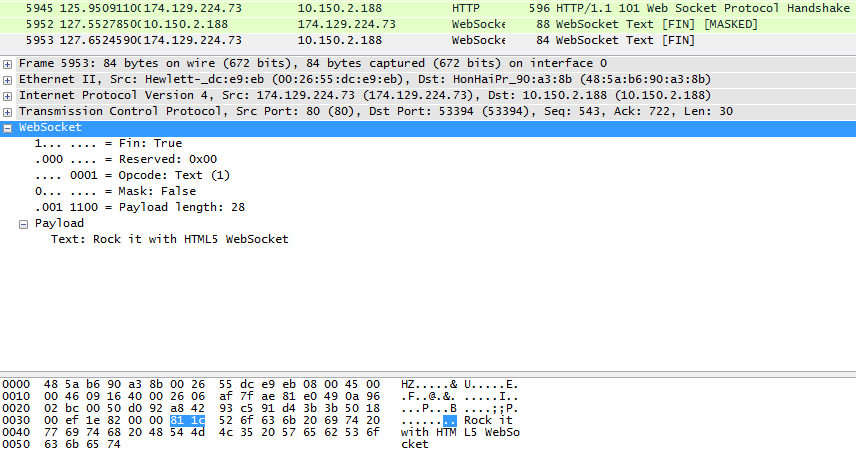
**Benodigd**:

* Browser
* Wireshark
* Internetverbinding

**Stappen**:

1. Start Wireshark
2. Start een trace op je WiFi en/of LAN adapter
3. Stel de filter in op
   1. Filter: (ip.dst == 174.0.0.0/8 && ip.src == 192.168.0.0/16) || (ip.dst == 192.168.0.0/16 && ip.src == 174.0.0.0/8) && (http2||websocket||spdy||http||ssl)
      1. NB Vervang ‘192.168.0.0/16’ door eigen ip-adres, op te vragen d.m.v. ipconfig in een command prompt!
      2. Andere filtervoorbeelden zijn:
         1. ip.src == 192.168.0.0/16 && tcp.dstport==80 && http
         2. ip.src == 192.168.0.0/16 && (tcp.dstport==80||tcp.dstport==443) && (ssl||http)
         3. ip.src == 192.168.0.0/16 && (http2||websocket||spdy)
         4. tcp.port==8080
4. Ga in chrome en/of firefox naar <http://www.websocket.org/echo.html>
5. Druk op Connect, Send (3x) en Disconnect
6. In Wireshark:
   1. Zo nodig: TCP stream > Decode as > Transport > HTTP > OK
   2. Selecteer de regel met HTTP/1.1 101 Web Socket Protocol Handshake
   3. Rechtsklik > Follow TCP stream
   4. Je ziet de complete stream; eerst HTTP plaintext en na upgrade de websocket binary stream.
7. Controleer dat de handshake (UPGRADE) in HTTP plaatsvindt
8. Controleer dat je de websocket datagrams kunt zien als je het “WebSocket packet” openklapt, en daarin de “(Unmasked) Payload”. Klik op een protocolnaam om de Bytes onderin gemarkeerd te krijgen.
   1. Masked: van browser
   2. Unmasked: echo response van server
9. Het valt op dat de Disconnect niet correct door wireshark geparsed wordt; “malformed packet”. Een lege *close reason* mag van het protocol, maar dit lijkt Wireshark niet te snappen. Voor een 2 Byte *close response* geldt hetzelfde.
10. Doe nogmaals hetzelfde, nadat je “Use secure WebSocket (TLS)” hebt aangevinkt op de websocket.org Echo Test. (je ziet nu 139 Bytes per bericht: “Rock it with HTML5 WebSocket”)
11. Kun je nu nog het verkeer tracen? Waarom wel/niet?

**Impressie:**



## Fiddler & websockets

**Case**: werken met Fiddler om websocket data te debuggen

**Taak**: 2 varianten doorlopen

**Benodigd**:

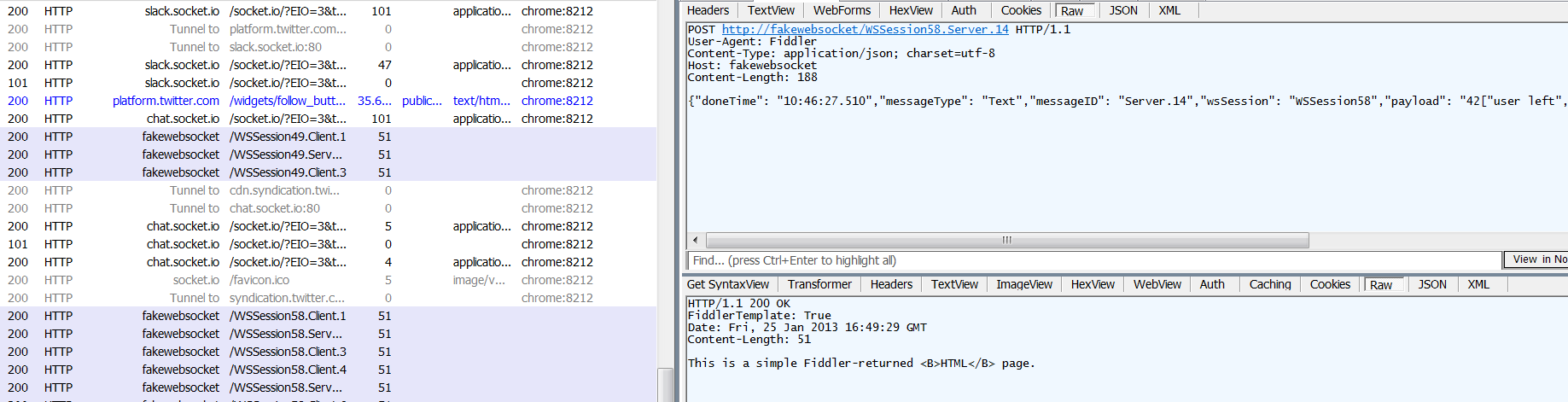
* Chrome en/of Firefox
* Fiddler
* Internetverbinding

**Nota bene**: Fiddler ondersteunt websockets nog (steeds!) niet optimaal. Momenteel zijn er 2 manieren om websocket datagrams te volgen. We zullen ze beide uitvoeren.

**Variant 1**:

1. Kopieer & rename de meegeleverde ‘<Workspace>\cfg\CustomRules - Fiddler fakewebsocket.js’ naar C:\Users\<user>\Documents\Fiddler2\Scripts\**CustomRules.js**
2. Start Fiddler. Ter informatie:
   1. Fiddler start een proxy op localhost:8888 en stelt die automatisch in als systeemproxy
      1. NB: IE en Chrome gebruiken die automatisch; Firefox niet
   2. Met Tools > Fiddler Options > HTTPS > Decrypt HTTPS traffic kun je zelfs HTTPS verkeer volgen. Je moet wel in de browser een certificaat van Fiddler toestaan; Fiddler voert feitelijk een “Man in the Middle” (MitM) aanval uit.
3. Start Firefox en/of Chrome
   1. Voor Firefox: Extra > Monitor with Fiddler > Use Fiddler automatically
4. Fiddler: activeer de tab rechtsboven: AutoResponder
   1. button Add Rule
   2. vervang ‘StringToMatch[1]’ (of ‘EXACT:http://213.19.196.8/’) onderaan door: ‘regex:http://fakewebsocket/.\*’ (zonder quotes)
   3. Kies de 2e keuze in de dropdown eronder (200\_SimpleHTML.dat)
   4. button Save
   5. check de checkbox van je rule onder ‘If request matches…’ (staat waarschijnlijk al aan)
   6. check de twee checkboxes boven de Add Rule button: ‘Enable rules‘ en ‘Unmatched requests passthrough’
5. In browser: <http://www.websocket.org/echo.html>
6. Druk op Connect, Send (3x) en Disconnect
7. In browser: <http://socket.io/demos/chat>
8. ‘log in’ en zend enkele berichten
9. Fiddler:
   1. selecteer links 1 v/d ‘fakewebsocket’ berichten
   2. selecteer rechts de Inspectors tab
   3. kies in het bovenste (Request) panel de view Raw of JSON, veld ‘payload’
   4. bekijk de berichten
      1. in de URL kolom zie je of het bericht van Client of Server komt
      2. zie dat het frame format van de chat demo het patroon 'n[{..}]' volgt; het getal n vervult de rol die in standaard HTTP door Content-Length wordt vervuld.
   5. merk op over het custom script dat:
      1. websocket berichten omgezet worden naar fake messages, die vervolgens op een vrij standaard manier te bekijken zijn
      2. er een vertraging van 2 seconden in zit, om te voorkomen dat Fiddler 100% CPU nodig heeft voor tracen van websocket berichten
      3. de JSON parser niet perfect werkt - Socket.IO berichten kunnen alleen in Raw mode bekeken worden

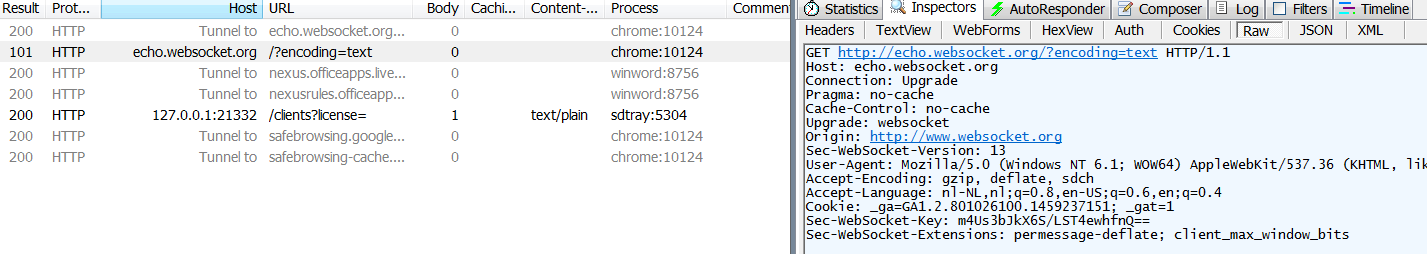
**Impressie:**

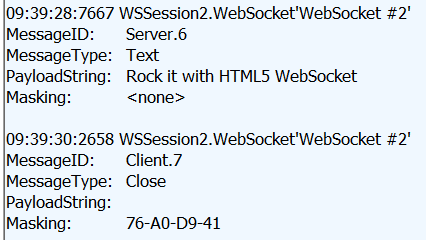


**Variant 2**:

1. Kopieer & rename de meegeleverde ‘<workspace>\cfg\CustomRules - Fiddler ws2log.js’ naar C:\Users\<user>\Documents\Fiddler2\Scripts\**CustomRules.js**
2. Herstart Fiddler
3. (her)open <http://www.websocket.org/echo.html>
4. Druk op Connect
5. In Fiddler:
   1. selecteer links de regel met Host: echo.websocket.org
   2. selecteer rechts de tab Inspectors
   3. selecteer de subtab Raw
   4. bekijk de HTTP headers. Wat zijn de websocket headers?
6. Nog steeds in Fiddler: activeer de Log tab rechtsboven
7. In de browser: druk op Send (3x) en Disconnect
8. Fiddler: check de berichten die in de log langskomen
9. Merk op dat de Mask van *client* berichten getoond wordt
   1. Die werden niet getoond in variant 1
   2. Waarom wordt geen mask van *server* berichten getoond?
10. Merk op dat deze log erg snel volloopt bij gebruik van websockets. Fijn debuggen/tracen is het niet.

**Impressie:**





**Achtergrondinformatie**:

* <http://blogs.telerik.com/fiddler/posts/13-06-04/what-s-new-in-fiddler-2-4-4-5>
* <http://www.codeproject.com/Articles/718660/Debug-Inspect-WebSocket-traffic-with-Fiddler>

## Chrome & websockets

**Case**: werken met Chrome om websocket data te debuggen

**Taak**: stappen doorlopen

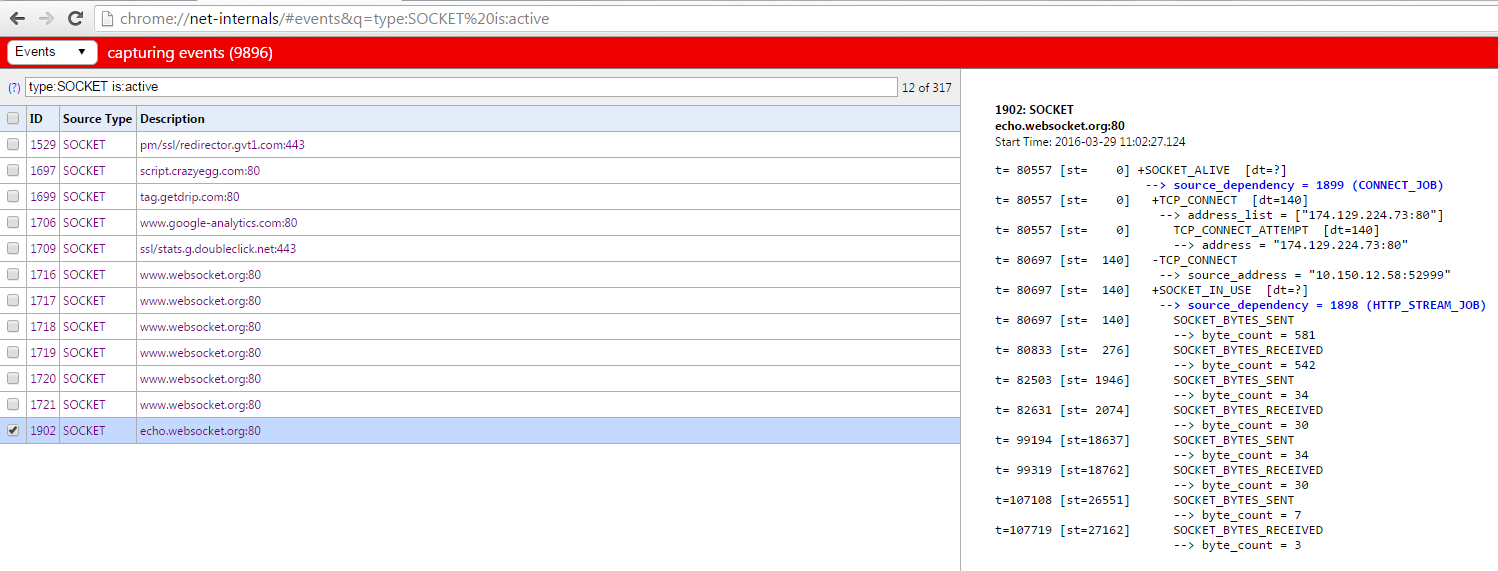
**Benodigd**:

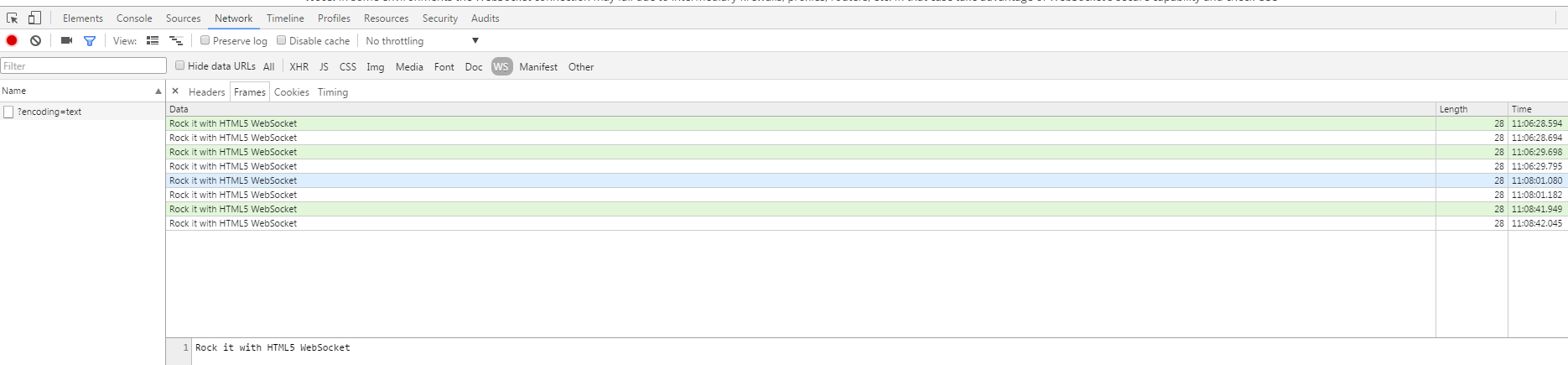
* Chrome
* Internetverbinding

**Stappen**:

1. Open <http://www.websocket.org/echo.html>
2. Druk op Connect en Send (3x) maar *geen* Disconnect
3. Open in een 2e tab: chrome://net-internals/#sockets
   1. Je ziet nu alle sockets die Chrome in de huidige sessie heeft geopend
4. Open in de 2e tab: chrome://net-internals/#events&q=type:SOCKET%20is:active
5. zoek naar een socket van type SOCKET\_STREAM of SOCKET
   1. je ziet rechts diverse header waarden en/of byte\_count’s maar geen data
6. ga terug naar de 1e tab
7. druk F12 (Developer Toolbar) – kies Network – rechts van Filter: selecteer ‘WS’ – klik op actieve websocket (status 101) – zie Header & Frames tabjes
   1. je ziet nu de individuele berichten. De meest recente staat bovenaan.

**Impressie**





# SPDY lab recipe

Obsolete! SPDY wordt snel vervangen door HTTP/2.

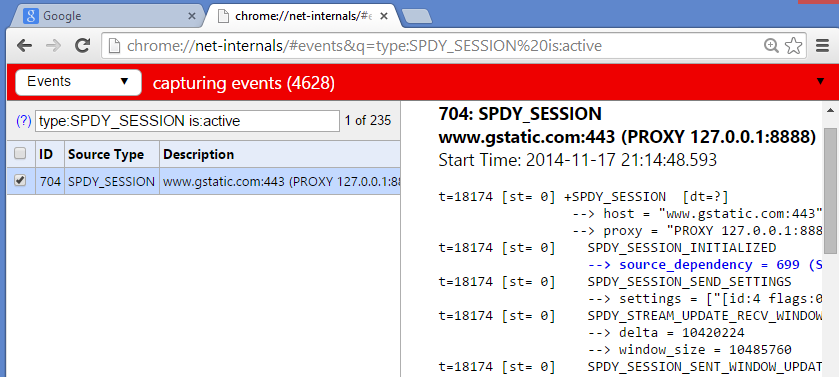
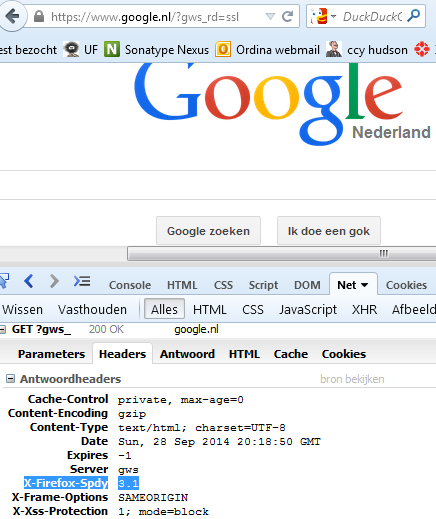
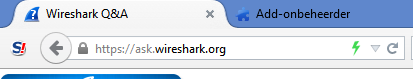
**Doel**: zien hoe SPDY-verkeer herkenbaar is in de browser, en hoe de lijndata te volgen is & eruit ziet

**Taak**: stappen doorlopen

**Benodigd:**

* Chrome (gewone versie is voldoende)
* Firefox incl. FireBug (gewone versie)
* ~~Wireshark~~
* Internetverbinding

**Stappen:**

1. Start chrome, evt. met command line opties
   1. --use-spdy=no-ssl (“Forces Chrome to always use SPDY, but without SSL”) of
   2. --use-spdy=npn (“By default Chrome disables SPDY for a small percentage of users; using the --use-spdy=npn flag simply ensures that Chrome does not do this.”)
2. Start Firefox
3. ~~Start Wireshark~~
4. ~~Start een trace op je WiFi en/of LAN adapter~~
5. Ga in chrome en firefox naar
   1. <https://www.google.nl> en/of
   2. <https://ask.wireshark.org>
6. Chrome:
   1. Open chrome://net-internals#spdy  
      
   2. Open chrome://net-internals/#events&q=type:SPDY\_SESSION%20is:active  
      
7. Firefox:
   1. druk op F12 (FireBug)
   2. ga naar Net panel
   3. zo nodig Inschakelen
   4. druk F5 (refresh)
   5. Open van de bovenste URL de Headers sectie
   6. Bij Antwoordheaders zie je staan   
      **“X-Firefox-Spdy 3.1”**Dit betekent dat spdy/3.1 actief is!  
      
   7. Optioneel: installeer “SPDY indicator” plugin – deze toont een groene bliksemschicht bij elke SPDY pagina  
      

**Achtergrondinformatie**:

* <http://www.chromium.org/spdy/spdy-tools-and-debugging>

# Demo SPDY

Obsolete! SPDY wordt snel vervangen door HTTP/2.

Deze demo zal door de docent gegeven worden. Om evt. thuis na te spelen vind je hier de instructies.

## Klaarzetten

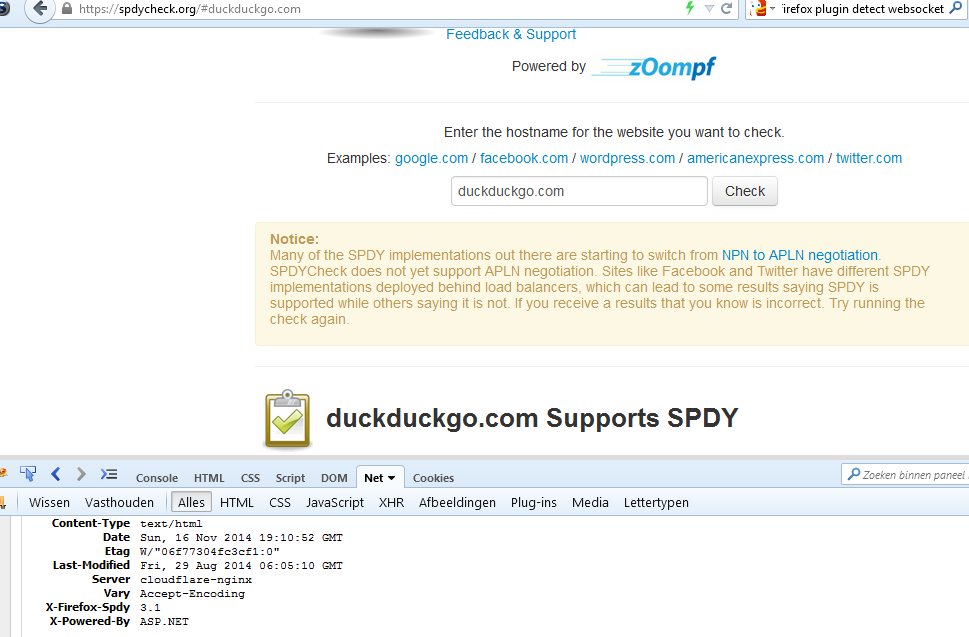
1. Installeer Vagrant & Virtual Box
2. vagrant init
3. vagrant box add hashicorp/precise64
4. vi Vagrantfile
   1. box: hashicorp/precise64
   2. port forwarding:  
      80 =>10080  
      443 => 10443
5. vagrant up
6. vagrant ssh
7. sudo su
8. apt-get update
9. apt-get install apache2
10. download (op host OS) mod\_spdy\*.deb (mod-spdy-beta\_current\_amd64.deb) van <https://developers.google.com/speed/spdy/mod_spdy/>
    1. Ter info: heeft Apache 2.2.4+ met mod\_ssl enabled nodig.
    2. Zorg dat .deb op /vagrant benaderbaar is
11. touch /etc/default/mod-spdy
12. dpkg -i /vagrant/mod-spdy-\*.deb
    1. of apt-get -f install mod-spdy-\*.deb
13. Cfgfile vind je nu in /etc/apache2/mods-available/spdy.conf met o.a. content:  
    <IfModule spdy\_module>  
     SpdyEnabled on  
    </IfModule>
14. cd /etc/apache2
15. vi apache2.conf
16. voeg ‘ServerName localhost’ toe
17. mkdir ssl
18. cd ssl
19. openssl genrsa -out key.pem 1024
20. openssl req -new -key key.pem -out request.pem
21. openssl x509 -req -days 1095 -in request.pem -signkey key.pem -out cert.pem
    1. beter: openssl req -x509 **-nodes** -days 1095 **-newkey rsa:2048** -out /etc/apache2/ssl/server.crt **-keyout /etc/apache2/ssl/server.key**
    2. Common Name (CN) must match FQDN (==localhost)!
22. a2enmod ssl
23. ln -s /etc/apache2/sites-available/default-ssl /etc/apache2/sites-enabled/000-default-ssl
24. vi /etc/apache2/sites-enabled/000-default-ssl
    1. voeg toe:  
       SSLCertificateFile /etc/apache2/ssl/server.crt  
       SSLCertificateKeyFile /etc/apache2/ssl/server.key  
       SSLStrictSNIVHostCheck Off
    2. inside block <VirtualHost \_default\_:443>, line below Server Admin email:  
        ServerName localhost:443
25. service apache2 restart

## Uitvoeren demo SPDY

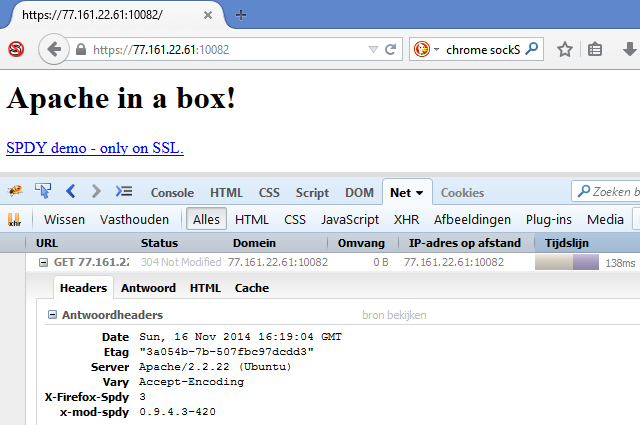
**Benodigd**:

* Draaiende vagrant-spdy VM
* Firefox
* Chrome

**Externe sites:**

1. open <https://spdycheck.org>
2. Toon enkele sites  
   

**Apache met mod\_spy:**

1. vagrant up (of start via VirtualBox, evt. met shift voor headless)
2. vagrant ssh
3. tail -F /var/log/apache2/\*.log
4. Open de volgende URLs:
   1. <https://77.161.22.61:10082/> (extern via experia router)
   2. <http://77.161.22.61:10081/> (extern via experia router – plain SPDY!)
   3. <https://192.168.2.5:10443> (Flow home WiFi)
   4. <https://localhost:10443>
5. Firefox: check **X-Firefox-Spdy** header
6. Beide: check Apache response header: x-mod-spdy  
   
7. Chrome: check chrome://net-internals#spdy  
   

# HTTP/2 lab recipes

**Doel**: op drie manieren kijken naar HTTP/2

## HTTP/2 & browsers

**Case:** In deze lab ontdek je hoe je een HTTP/2 sessie in de browser kunt herkennen.

**Taak**: stappen doorlopen

**Benodigd**:

* Firefox
* Chrome
* Internetverbinding

**Stappen:**

**Firefox**

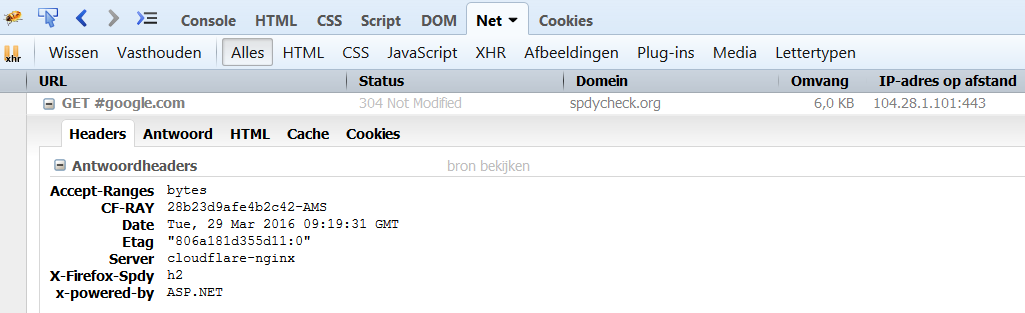
1. Start Firefox
2. Installeer zo nodig de extensie FireBug
3. Druk F12
4. Activeer Net panel
5. Open <https://spdycheck.org/#google.com>
6. FireBug: klap de GET uit
7. Check de Response header:
   1. De X-Firefox-Spdy header heeft de waarde h2 (dit betekent: HTTP/2, final version)
      1. W.b. de naam ‘Spdy’: in het verleden is deze meta-tag toegevoegd toen het SPDY transport protocol, voorloper van HTTP/2, werd toegevoegd.
      2. NB deze header wordt door Firefox als meta-informatie toegevoegd! De header gaat niet daadwerkelijk over de lijn.

**Chrome**

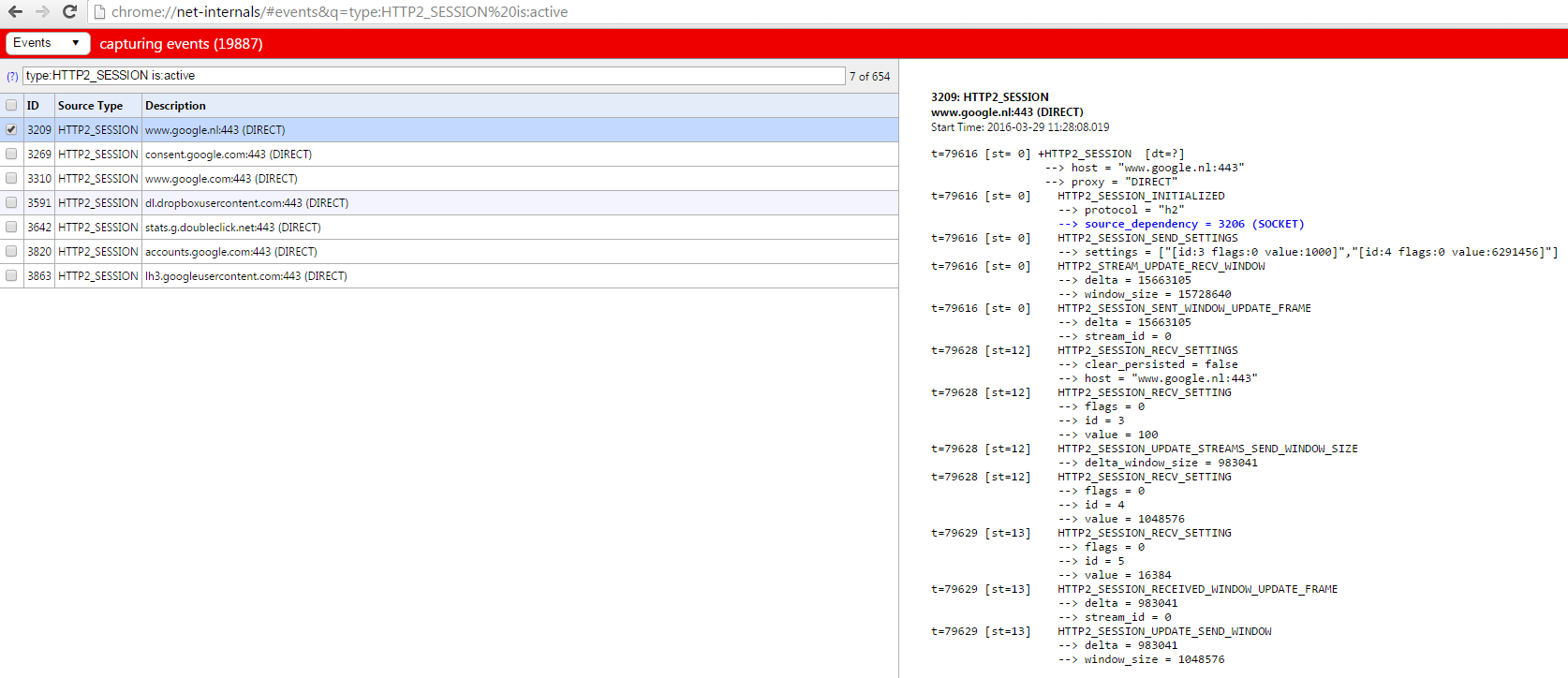
1. Start Chrome
2. Druk F12 om de Developer toolbar te openen
3. Tab 1: Open <https://spdycheck.org/#google.com>
4. Tab 2: Open <chrome://net-internals/#http2>
5. Tab 3: Open <chrome://net-internals/#events&q=type:HTTP2_SESSION%20is:active>
6. Selecteer in de 3e tab een HTTP2\_SESSION regel
   1. NB Soms moet je wat heen een weer klikken, refreshen e.d. om hier wat te zien!
   2. Rechts zie je nu de verschillende packets

**Impressie:**

Firefox:



Chrome:



## HTTP/2 & NodeJs

**Case:** In deze lab start je via de command line zowel een HTTP/2 client als een server. Code komt van hithub en is geschreven in NodeJs javascript.

**Taak:** na opzetten voer je 5 varianten van client requests uit: lokaal over TLS, lokaal unencrypted, extern, lokaal vanuit curl, lokaal vanuit je browser

**Benodigd**:

* NodeJs 5+ (voor HTTP/2 TLS o.b.v. ALPN)
* Windows: liefst ook cygwin
* Firefox en/of Chrome
* Curl met libnghttp2 (bijv. versie 7.48.0)

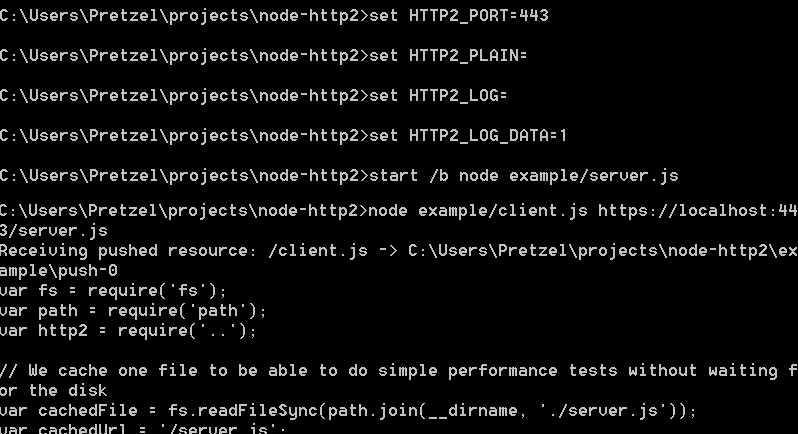
**Stappen:**

1. Open een command prompt
   1. bij voorkeur niet cmd.exe, maar als je echt niks anders hebt.. Vooruit dan maar!  
      Je krijgt dan echter helaas geen logs te zien.
2. git clone https://github.com/molnarg/node-http2.git
3. cd node-http2
4. npm install bunyan
5. kopieer index.html van <workspace>\cfg naar node-http2\example
6. Zie de stappen hieronder voor jouw shell naar keuze

### Windows cmd.exe:

1. set HTTP2\_PORT=443
2. set HTTP2\_PLAIN=
3. set HTTP2\_LOG=
   1. helaas werkt bunyan logger niet op Windows console!?
4. set HTTP2\_LOG\_DATA=1
5. start /b node example/server.js
   1. dit start een HTTP/2 server in de achtergrond
6. node example/client.js https://localhost:443/localhost.crt
   1. toont localhost.crt op cmdline
7. node example/client.js https://localhost:443/server.js
   1. toont server.js op cmline
   2. **en krijgt client.js gepusht; wordt op disk gezet als ‘push-0’ in de example map**
8. sluiten van NodeJs server.js op de achtergrond: tsja.. dat moet via de taskmanager… Kill alle processen van node.

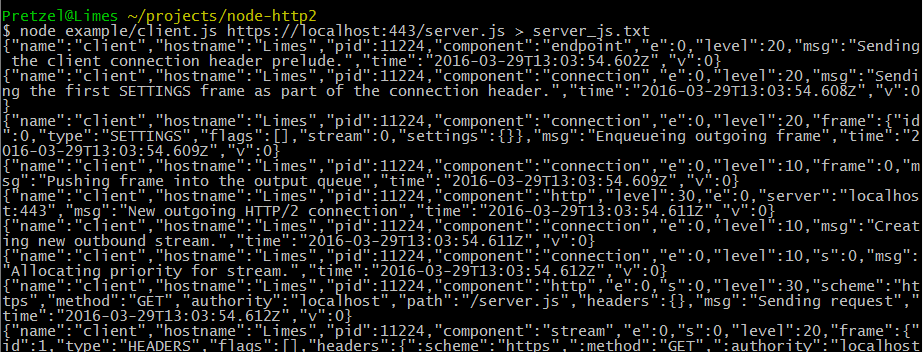
**Impressie:**

****

### Linux/Mac/cygwin:

1. export set HTTP2\_PORT=443
2. export set HTTP2\_PLAIN=
3. export set HTTP2\_LOG=trace
4. export set HTTP2\_LOG\_DATA=1
5. node example/server.js &
6. node example/client.js https://localhost:443/localhost.crt > localhost\_crt.txt
   1. toont logs op cmdline en schrijft data naar localhost\_crt.txt
7. node example/client.js https://localhost:443/server.js > server\_js.txt
   1. toont logs op cmline en schrijft data naar server\_js.txt
   2. **en krijgt client.js gepusht; wordt op disk gezet als ‘push-0’ in de example map**
8. jobs (om het NodeJs server.js achtergrondproces te zien)
9. kill %1 (als jobs slechts item [1] oplevert)

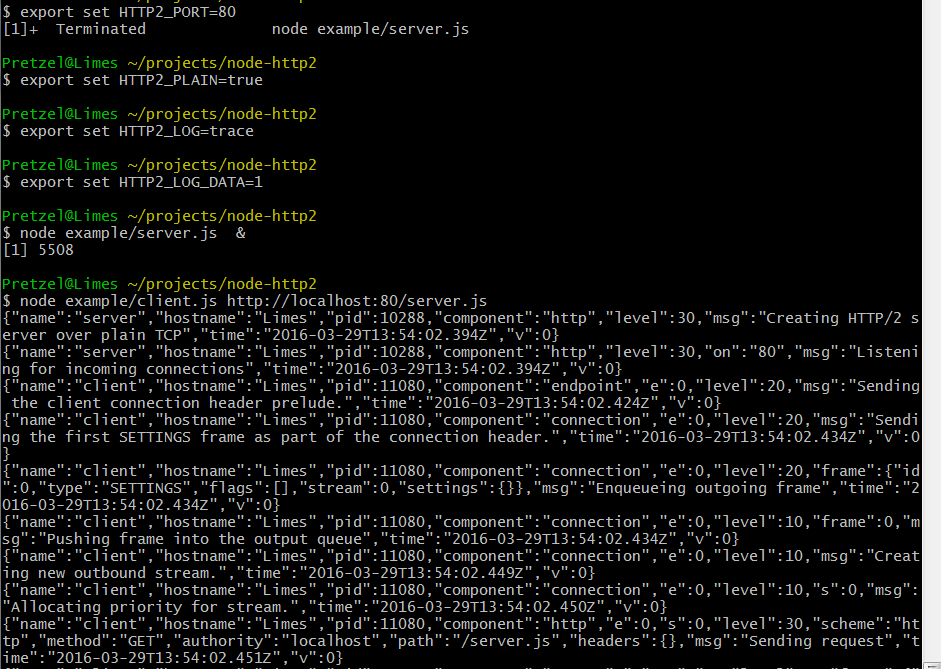
**Impressie:**



### Variant via TLS-loze HTTP/2 op poort 80 (alleen Linux/Mac/cygwin syntax):

1. export set HTTP2\_PORT=**80**
2. export set HTTP2\_PLAIN=**true**
3. export set HTTP2\_LOG=trace
4. export set HTTP2\_LOG\_DATA=1
5. node example/server.js &
6. node example/client.js http://localhost:80/server.js
7. node example/client.js http://localhost:80/localhost.crt

**Impressie:**



HTTP/2(-loze) connectie met *externe sites* (deze keer alleen Windows cmd.exe syntax):

1. set HTTP2\_PLAIN=
2. set HTTP2\_LOG=trace
3. set HTTP2\_LOG\_DATA=1
4. node example/client.js https://google.com
   1. wel HTTP/2, getuige de grote hoeveelheid logs
5. node example/client.js https://www.ordina.nl
   1. *geen* HTTP/2, getuige de totale afwezigheid van logs

### curl HTTP/2

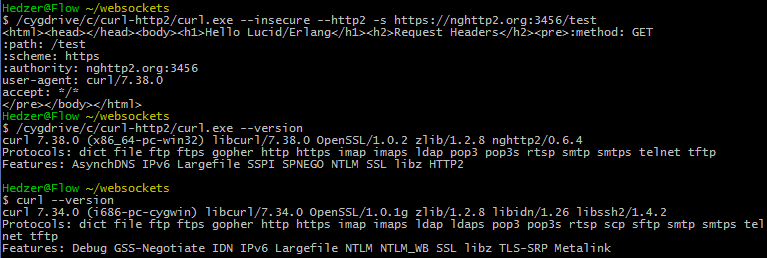
1. curl.exe –version geeft:  
   curl 7.48.0 (x86\_64-pc-win32) libcurl/7.48.0 OpenSSL/1.0.2g **nghttp2/1.9.1**

Protocols: dict file ftp ftps gopher http https imap imaps ldap pop3 pop3s rtsp smb smbs smtp smtps telnet tftp

Features: AsynchDNS Largefile NTLM SSL **HTTP2**

1. curl –-insecure --http1.1 –v https://localhost:443/server.js
   1. serveert de data op HTTP/1.1, dus geen server push  
      logs: “Falling back to simple HTTPS”
   2. --insecure is nodig omdat het (self signed) server certificaat niet in de curl truststore staat
2. curl.exe --insecure --http2 -s -o server\_js.txt -v https://localhost:443/server.js
   1. werkt, maar de server.js crasht met: “Error: Sending illegal frame (DATA) in CLOSED state.” Het lijkt erop dat curl geen push file accepteert & de stream sluit, en NodeJs implementatie hier niet goed mee omgaat.
3. Opvallend is dat curl -v http://localhost:80/localhost.crt eindigt in "Client connection header prelude does not match" .. "PROTOCOL\_ERROR" .. "Fatal error, closing connection" .. "GOAWAY" – over TLS (dus encrypted) werkt *good old* HTTP/1.1 wel, getuige: "Falling back to simple HTTPS" in stap 2
   1. NB wel even de server starten in plain modus voordat je dit naspeelt
   2. Blijkbaar support de NodeJs server implementatie geen unencrypted HTTP/2 van andere clients dan zijn eigen client.js.

**Impressie:**

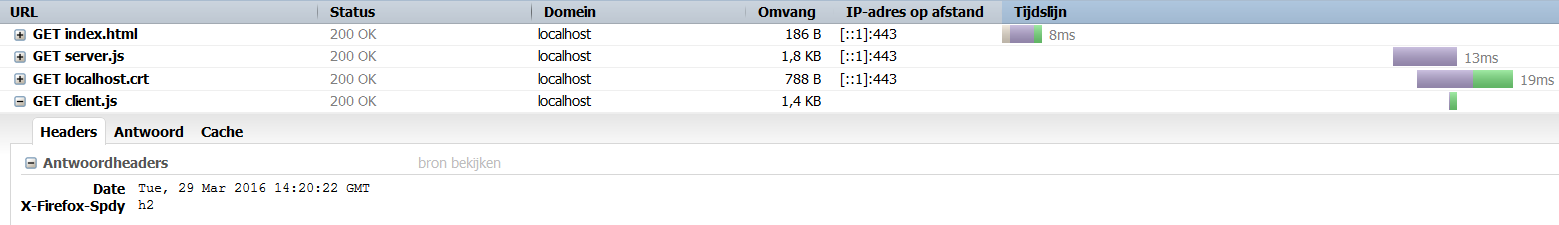


### Vanuit de browser

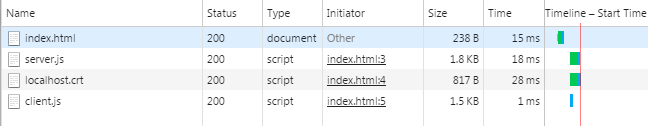
1. Zorg dat de NodeJs server nog draait in TLS mode
2. Open in de browser <https://localhost/index.html>
3. Druk de certificaatfout door
4. Check de Net panel
   1. Zie dat de client.js *snel* geserveerd kan worden omdat deze al gepusht wordt bij ophalen van server.js

**Impressie:**

**Firefox**



**Chrome**



**Achtergrondinformatie**:

* <https://github.com/molnarg/node-http2/wiki/Public-API>
* <https://github.com/molnarg/node-http2>

## HTTP/2 & Jetty

**Case:** In deze lab start je een ready-made Jetty HTTP/2 example.

**Taak:** voer de stappen uit

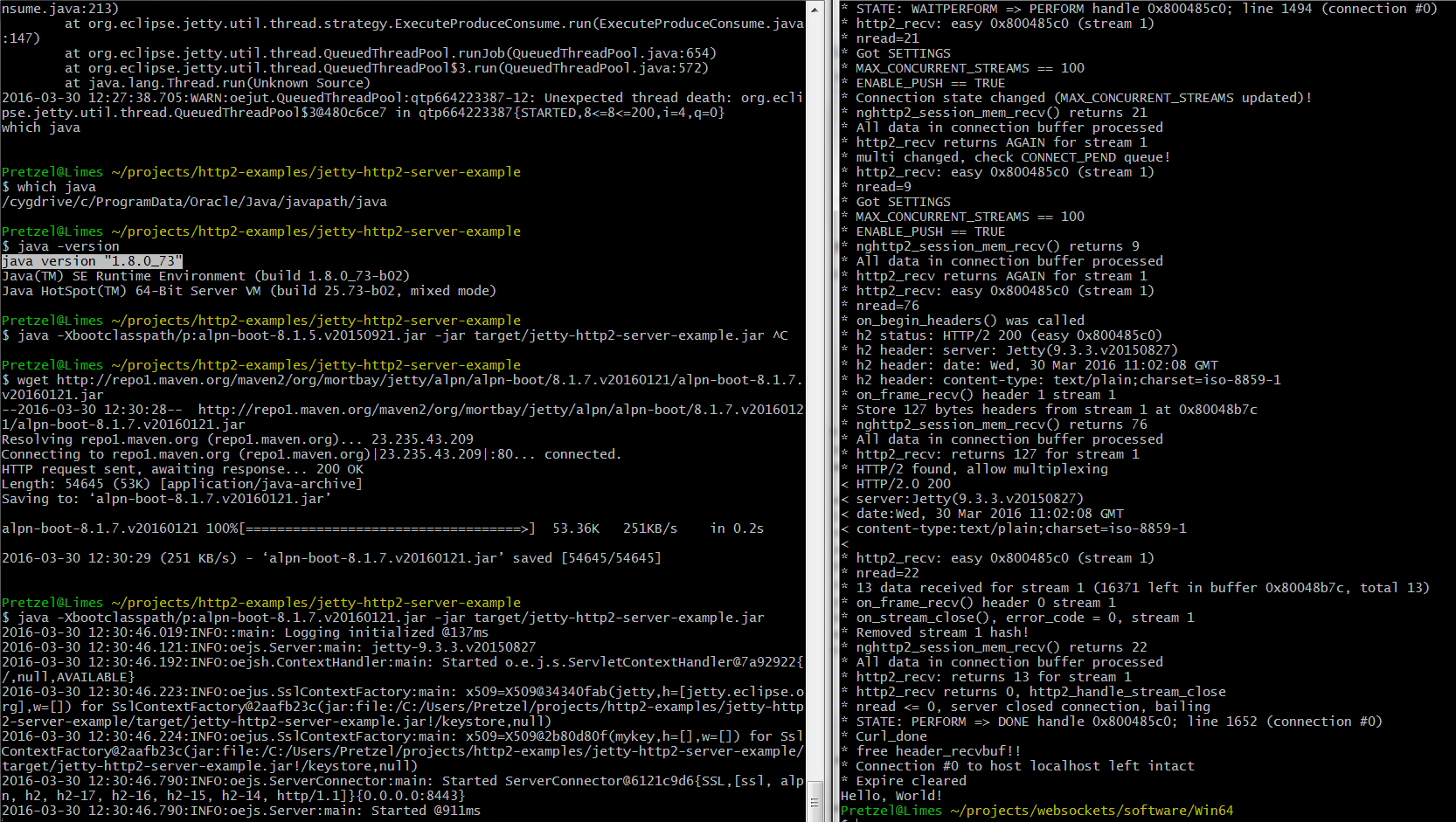
**Benodigd**:

* Maven
* JDK8
* curl

**Stappen:**

1. start een console
2. git clone https://github.com/fstab/http2-examples
3. cd http2-examples/jetty-http2-server-example
4. mvn package
5. curl -s -o alpn-boot-8.1.7.v20160121.jar http://repo1.maven.org/maven2/org/mortbay/jetty/alpn/alpn-boot/8.1.7.v20160121/alpn-boot-8.1.7.v20160121.jar
6. java -version
   * moet 1.8.0\_73 of \_72 zijn! Anders krijg je ws. ALPN TLS problemen
7. java -Xbootclasspath/p:alpn-boot-8.1.7.v20160121.jar -jar target/jetty-http2-server-example.jar
8. start een andere console
9. curl -v -k https://localhost:8443/
10. node example/client.js https://localhost:8443/
    * faalt met reden: cert expired at 18 augustus 2015
11. check de Jetty embedded server start code in http2-examples\jetty-http2-server-example\src\main\java\de\consol\labs\h2c\examples\server\**Http2Server.java**
    * Servlet.java in dezelfde package is een *2-regel plain-old Servlet*

**Impressie:**



**Bonus:**

* Probeer ook de andere projecten in deze repo (Undertow, Netty, REST + HTTP/2 push, etc. Jummy!)

## HTTP/2 & h2c

**Case:** In deze lab start je via de command line een HTTP/2 proxy en zie je HTTP/2 frames zoals ze over het netwerk gaan.

**Taak:** ‘installeer’ h2c en voer de stappen uit

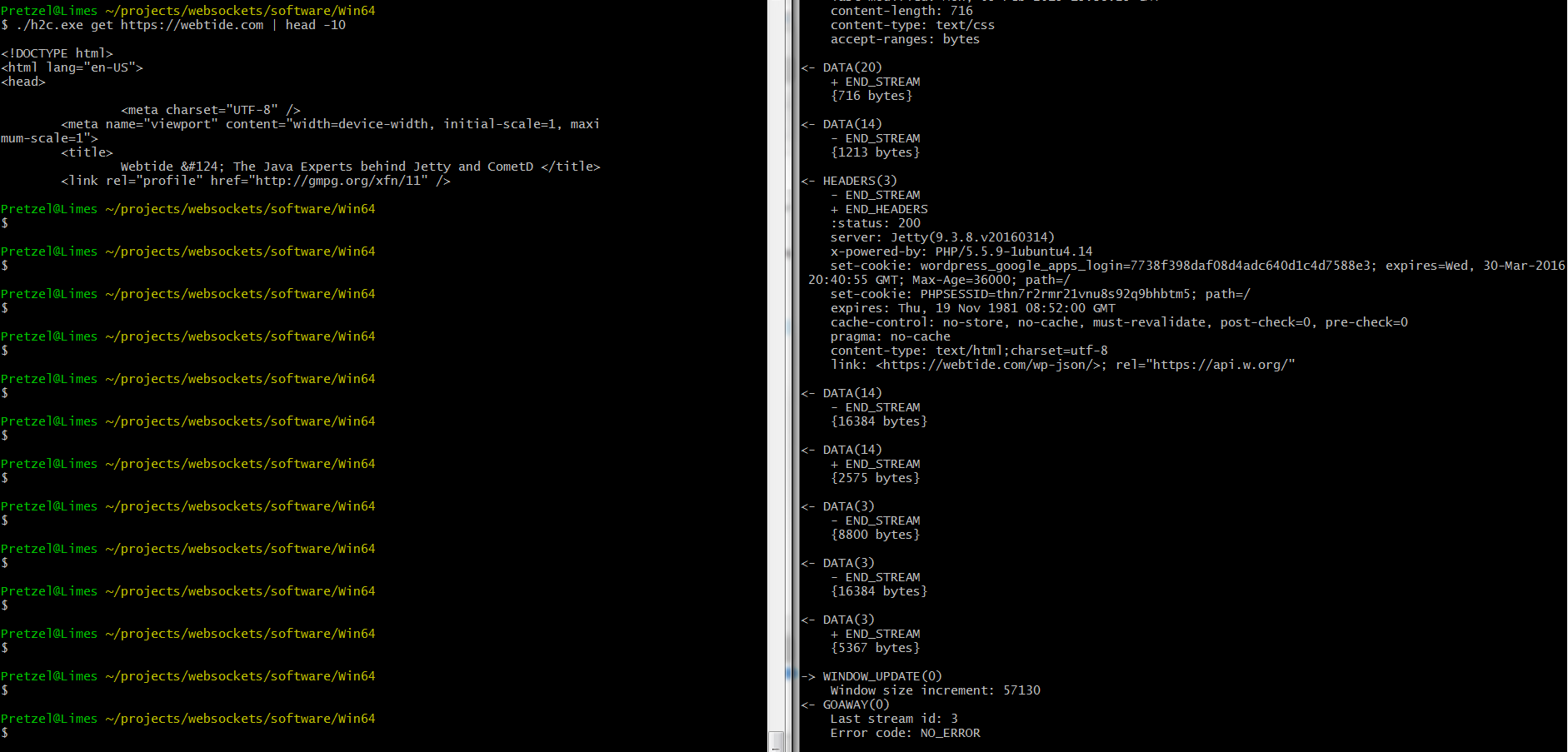
**Benodigd**:

* H2c

**Stappen:**

1. start een console
2. h2c start --dump
3. start een andere console
4. h2c get https://localhost:8443
   * of een andere lokaal draaiende HTTP/2 site
5. h2c disconnect
6. h2c get https://webtide.com
7. Bekijk de frames

**Impressie:**



Achtergrondinformatie:

* <https://github.com/fstab/h2c>

## HTTP/2 & Tomcat9

**Case:** In deze lab start je een in Tomcat9 meegeleverd HTTP/2 server push example.

**Taak**: stappen doorlopen

**Benodigd**:

* Tomcat9
* JDK8
* Optioneel openssl

**Stappen:**

1. Unzip Tomcat 9.0.0.M4
2. Kopieer naar apache-tomcat-9.0.0.M4\conf de volgende 2 files:
   * <repo>\workspace\cfg\tomcat-cert.pem
   * <repo>\workspace\cfg\tomcat-key.pem
3. Voeg aan apache-tomcat-9.0.0.M4\conf\server.xml (vanaf regel 109) het blok hieronder toe
4. Start apache-tomcat-9.0.0.M4\bin\startup.bat
   * Zoek in de logging naar: “INFO [main] org.apache.coyote.http11.AbstractHttp11Protocol.configureUpgradeProtocol The ["https-nio2-8443"] connector has been configured to support negotiation to [h2] via ALPN”
5. Open in de browser <http://localhost:8080/examples/servlets/serverpush/simpleimage>
   * Zie error: “Server push requests are not supported by the HTTP/1.1 protocol”
   * Reden: op poort 8080 draait een HTTP/1.1 listener; zie server.xml
6. Open in de browser <https://localhost:8443/examples/servlets/serverpush/simpleimage>
7. Druk de certificate error door
8. Zie het resultaat
9. Patch node-http2\example\client.js (zie onder)
10. Draai op cmdline (op locatie van node-http2):   
    node example/client.js <https://localhost:8443/examples/servlets/serverpush/simpleimage>
    * Observeer: “Receiving pushed resource: /examples/servlets/images/code.gif -> <checkout>\node-http2\example\push-0” !

Server.xml toevoeging:

<Connector port="8443" protocol="org.apache.coyote.http11.Http11Nio2Protocol"

maxThreads="150" SSLEnabled="true">

<UpgradeProtocol className="org.apache.coyote.http2.Http2Protocol" />

<SSLHostConfig honorCipherOrder="false" >

<Certificate certificateKeyFile="conf\tomcat-key.pem"

certificateFile="conf\tomcat-cert.pem"

certificateKeyPassword="tomcat"

type="RSA" />

</SSLHostConfig>

</Connector>

<!--<Connector port="8080" protocol="org.apache.coyote.http11.Http11AprProtocol"

connectionTimeout="20000"

redirectPort="8443" >

<UpgradeProtocol className="org.apache.coyote.http2.Http2Protocol" />

</Connector>-->

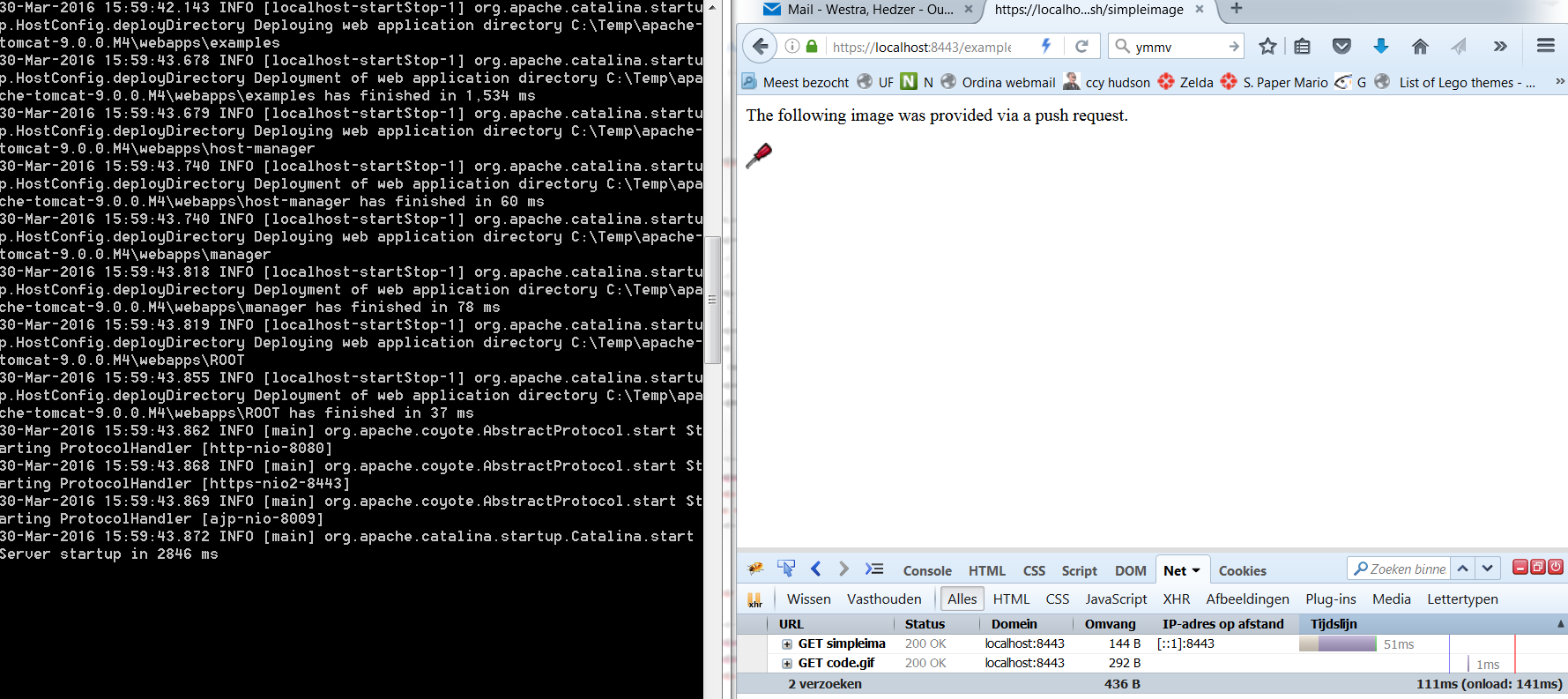
node-http2\example\client.js patch (vanaf regel 18):

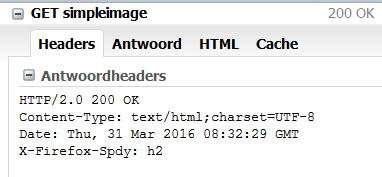
options.key = fs.readFileSync(path.join(\_\_dirname, '/tomcat-key.pem'));

options.passphrase = 'tomcat';

options.ca = fs.readFileSync(path.join(\_\_dirname, '/tomcat-cert.pem'));

**Impressie:**





**Bonus:**

* Uncomment het grijze stuk in server.xml en probeer unencrypted HTTP/2 aan de prata te krijgen.

**Achtergrondinformatie:**

De ‘magic’ zit in 2 regels code in class http2.SimpleImagePush:

javax.servlet.http.PushBuilder pb =

req.getPushBuilder().path("servlets/images/code.gif");

pb.push();

Aanmaken certificaten:

openssl req -x509 -newkey rsa:2048 -keyout tomcat-key.pem -out tomcat-cert.pem -days 365

# passphrase: **tomcat**

# Common Name (e.g. server FQDN or YOUR name) []:**localhost**

# Email Address []: **<vul hier evt. je emailadres in>**

Extra logging:

Zie conf/logging.properties

# To see debug messages for HTTP/2 handling, uncomment the following line:

#org.apache.coyote.http2.level = FINE

## HTTP/2 & Wireshark

**Case:** In deze lab kijk je op lijndataniveau naar HTTP/2 verkeer. Je ervaart de huidige (on)mogelijkheden van HTTP/2 sniffing & inspectie.

**Taak**: stappen doorlopen

**Benodigd**:

* Wireshark

**Stappen:**

1. Open (van de USB-stick data) Sample of HTTP2 (draft-14).trc
   * Rechtsklik 1 van de packets > decode as … > kies http2
   * Je ziet dat WireShark ze herkent als HTTP2 draft-**13** packets
2. Open node-http2-local-trace-unencrypted.pcap
   * Is een opgeslagen trace van de node2-http lab (unencrypted)
   * Deels te decoden als HTTP/2 door WireShark; deels geeft “malformed packet” of wordt gewoon niet herkend
   * Op packet van 927 Bytes: rechtsklik > Follow TCP Stream: je ziet beide documenten
3. Open node-http2-local-trace-encrypted.pcap
   * rechtsklik op een packet met poort 443 > decode as … > kies ssl > Ok
4. Edit > Preferences > Protocols > SSL > RSA key list > edit
5. Invullen: 0.0.0.0 / 0 / http / <USB-stick>\ node-http2-example.key
6. Activeer evt. filter ‘tcp.stream eq 2’
7. Rechtsklik packet van 871 Bytes > follow SSL stream
8. zie data, incl. (als eerste) het gepush’te private key document
9. Als follow SSL stream niet werkt, dan kun je ook per frame onderaan de tab ‘Decrypted SSL data’ openen.

**Achtergrondinformatie**:

* Om op Windows de lokale traffic op te vangen:
  + RawCap.exe -f 127.0.0.1 dumpfile.pcap
  + Wel als admin draaien!
* <http://wiki.wireshark.org/HTTP2>
* <http://wiki.wireshark.org/SSL>
* <http://wiki.wireshark.org/CaptureSetup/Loopback>
* <http://www.netresec.com/?page=Blog&month=2011-04&post=RawCap-sniffer-for-Windows-released>
* <http://www.youtube.com/watch?v=QEQy1dAXnTQ> “Decrypting and reading HTTP and SPDY traffic in Wireshark over TLS/SSL” (zonder audio helaas)

**EINDE!**