## Installeer Java9 early access build

Simpel. Die haal je van <https://jdk9.java.net/download/>

Tip: laat nog wel je bestaande Java7/8 installatie(s) staan! Bij de laatste dialoog kun je aangeven dat die nog *niet* verwijderd moet(en) worden.

## Installeer curl

Optioneel, maar wel superhandig. Tenzij je bang wordt van CLI’s ;-)

De nieuwste Win64 build haal je van <https://dl.uxnr.de/build/curl/curl_winssl_msys2_mingw64_stc/curl-7.53.1/curl-7.53.1.zip>.

Het enige dat je nodig hebt uit die ZIP is src/curl.exe

## Installeren van een IDE & build omgeving

Tsja, dat valt al tegen… ‘The big three’ ondersteunen Java9 nog slechts gedeeltelijk, of zijn simpelweg buggy.

Beste optie is voorlopig Maven en een gewone teksteditor.

* IntelliJ 2016.3 <https://blog.jetbrains.com/idea/2016/12/intellij-idea-2016-3-1-is-out> ondersteunt nog geen Java9. Misschien 2017.1 EAP <https://confluence.jetbrains.com/display/IDEADEV/IDEA+2017.1+EAP> wel. Ik heb 'm nog niet geprobeerd…
* Eclipse JEE Oxygen 4.7 dev build <http://download.eclipse.org/eclipse/downloads/drops4/S-4.7M5-201701261030>  met Java 9 Support Plugin <https://marketplace.eclipse.org/content/java-9-support-beta-neon> wel. Dit is de Eclipse plugin update site URL van die plugin: <http://download.eclipse.org/eclipse/updates/4.7-P-builds>.
  + Evt. toepassen: "add --add-modules=java.se.ee to the vmargs (in eclipse.ini), zie <https://www.eclipse.org/eclipse/news/4.6/platform.php#java-9>"
  + HTTP2 code wordt helaas niet gecompileerd - module jdk.incubator.http wordt niet gevonden.
* Maven (welke versie dan ook!) werkt out of the box:
  + wel even een <je homefolder>/.m2/toolchains.xml file aanmaken (zie onder - pas de folders aan je eigen systeem aan!)
* NetBeans:
  + installeer nieuwste JDK9 preview: <https://jdk9.java.net/download>
  + download latest NB9 daily dev build a.k.a. trunk nightly: <http://bits.netbeans.org/download/trunk/nightly/latest>
  + Registreer JDK 9: Tools | Java Platforms | Add Platform | browse naar JDK9 folder
    - tip: stel NetBeans in om *intern* op JDK8 te draaien: - zet dit in C:\Users\<username>\AppData\Roaming\NetBeans\9\netbeans.conf of <je install locatie>\etc\netbeans.conf - bijvoorbeeld C:\Program Files\NetBeans Dev 201702080002\etc\netbeans.conf:  
      netbeans\_jdkhome="C:\Program Files\Java\jdk1.8.x\_yy"
  + File | Open Project | browse naar folder waar pom.xml staat | Ok  
    rechtsklik project | Properties | Build > Compile | Java Platform: JDK 9
  + dit werkt, maar huidige NetBeans build geeft continu popups. Zelfs een newline toevoegen lukt niet. Momenteel dus niet bruikbaar als IDE!

Maven toolchains cfg:   
<toolchains>  
    <toolchain>  
        <type>jdk</type>  
        <provides>  
            <version>1.8</version>  
            <vendor>sun</vendor>  
        </provides>  
        <configuration>  
            <jdkHome>C:\Program Files\Java\jdk1.8.0\_25</jdkHome>  
        </configuration>  
    </toolchain>  
    <toolchain>  
        <type>jdk</type>  
        <provides>  
            <version>1.9</version>  
            <vendor>sun</vendor>  
        </provides>  
        <configuration>  
            <jdkHome>C:\Program Files\Java\jdk-9</jdkHome>  
        </configuration>  
    </toolchain>  
</toolchains>

Zorg dat je ook het pad voor Java8 goed instelt!

## Lab recipes

Hieronder staan enkele ‘lab recipes’ die je in willekeurige volgorde kunt uitvoeren. En ja, er is weinig Java9 en/of programmeren bij. Ik snijd mijn spreekwoordelijke vinger af…

Onderaan de recipes staat onder het kopje ‘impressie’ een screenshot met het verwachte resultaat.

## Caveat – let op!

Let bij copy-pasten op dat Word helaas de nummers meeneemt, en dat quotes (“ ipv ") en mintekens (– ipv -) wel eens omgezet kunnen zijn naar niet-ASCII varianten…

HTTP/2 gaat in principe uit van een TLS-beveiligde (“SSL” in de volksmond) verbinding, en dus zijn er altijd private keys en certificaten nodig. De diverse voorbeelden maken gebruik van self-signed certificaten. In de browser moet je die expliciet doordrukken; bij curl commando’s wordt -k (ook wel --insecure) gebruikt voor datzelfde doel.

In de Tomcat9 recipe staan optionele instructies hoe je met openssl zelf certificaten kunt aanmaken. Een manier om gratis aan CA-signed (en dus in de browser vertrouwde) certificaten te komen is <https://letsencrypt.org/>. Hiervoor heb je wel een ‘Certbot ACMEclient’ nodig.

## Challenges

Zijn onderstaande lab recipes nog niet genoeg, of niet interessant genoeg voor je, dan heb ik de volgende uitdagingen voor je – let wel, die heb ik zelf nog *niet* opgepakt!

1. configureer een IDE *succesvol* voor Java9
2. maak een NodeJs client&server o.b.v. <https://www.npmjs.com/package/http2>
3. ga aan de slag met curl --http2
4. vind uit waarom Java9 geen HTTP/2 wil spreken – zie het partiële recipe hier beneden
5. voer de bonusopdrachten van de recipes uit
6. regel een certificaat via <https://letsencrypt.org/>
7. maak een docker setup (of zoek bestaande) voor div. HTTP/2 servers, bijvoorbeeld: Apache, nginx, NodeJs, Jetty, Tomcat en spring-boot. Enkele startpunten:

* <https://github.com/kikov79/docker-tomcat9>
* <https://www.youtube.com/watch?v=oCFwgMvouis> (Tomcat9 HTTP/2 quickstart)
* <https://vanwilgenburg.wordpress.com/2016/04/01/spring-boot-http2/>

## HTTP/2 & Jetty (server-side)

**Case:** In deze lab start je een *ready-made* Jetty HTTP/2 server example, gemaakt door Fabian Stäber.

Let op: deze code werkt *alleen* met Java8! Via de maven toolchains configuratie gaat de compilatie goed, voor het runnen echter moet je expliciet verwijzen naar Java8! In het betreffende commando ga ik ervanuit dat je een environment variabele ‘JAVA8\_HOME’ hebt ingesteld.

**Taak:** download wat software en voer de commando’s uit.

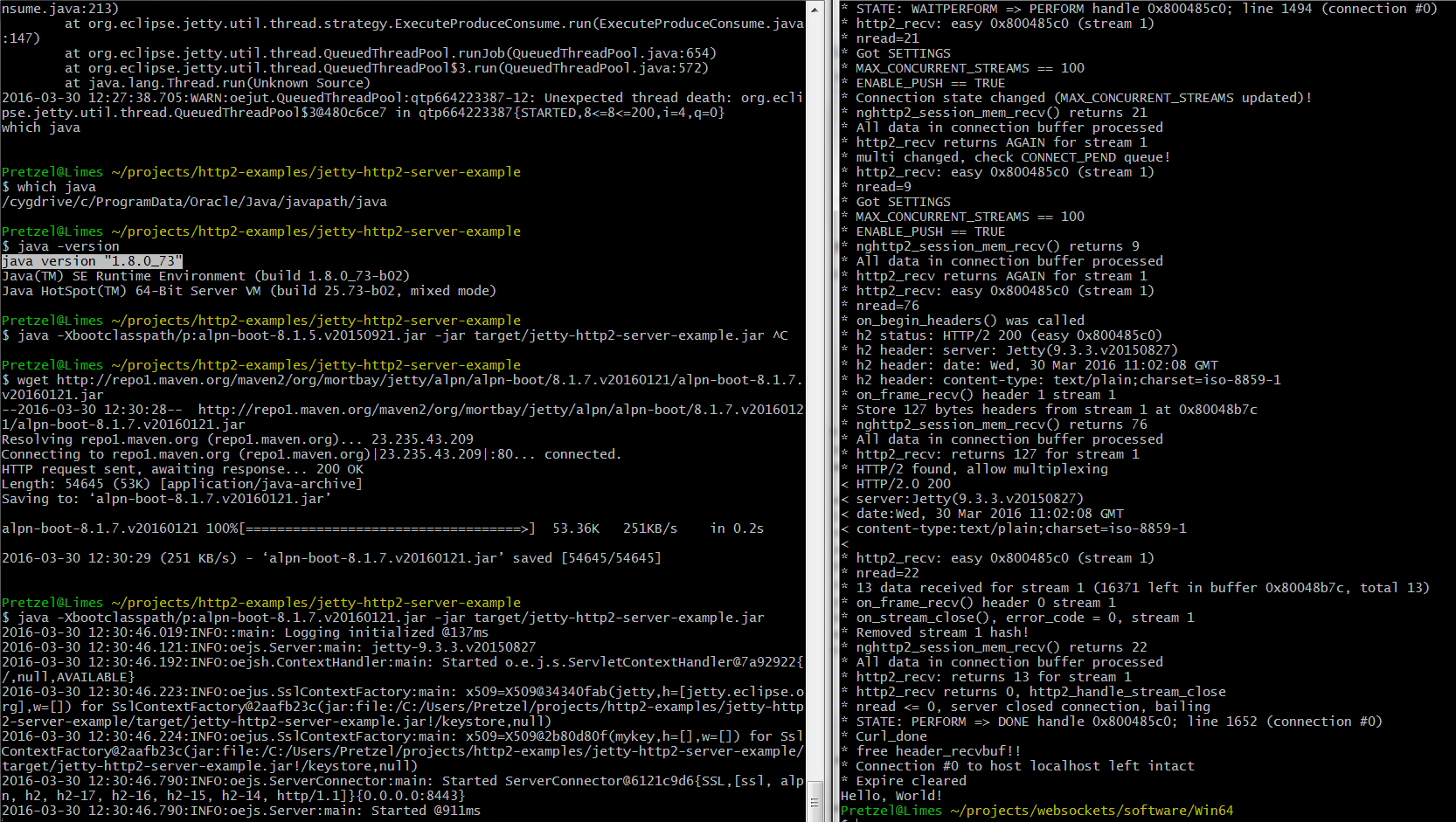
**Benodigd**:

* Maven
* JDK8, geconfigureerd via toolchain en met environment variabele
* Curl (kan ook met een browser)

**Stappen:**

1. start een console / shell / terminal / command prompt
2. git clone https://github.com/fstab/http2-examples
3. cd http2-examples/jetty-http2-server-example
4. mvn package
5. %JAVA8\_HOME%/bin/java.exe -version
   * Zoek op <https://www.eclipse.org/jetty/documentation/current/alpn-chapter.html#alpn-versions> welke ALPN package je nodig hebt.
   * Voor 1.8.0\_73 of \_72 is dat bijv. 8.1.7.v20160121
   * 1.8.0u111 is dat 8.1.9.v20160720
   * Etc.
6. NB Gebruik in de onderstaande 2 commando’s dus het *voor jou* juiste alpn-boot versienummer!
7. 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>
   * of via de browser deze JAR downloaden & op de huidige directory zetten
8. %JAVA8\_HOME%/bin/java.exe -Xbootclasspath/p:alpn-boot-8.1.7.v20160121.jar -jar target/jetty-http2-server-example.jar
9. start een andere console:
10. curl -v -k https://localhost:8443/
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.

Ook deze tool is van Fabian Stäber.

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

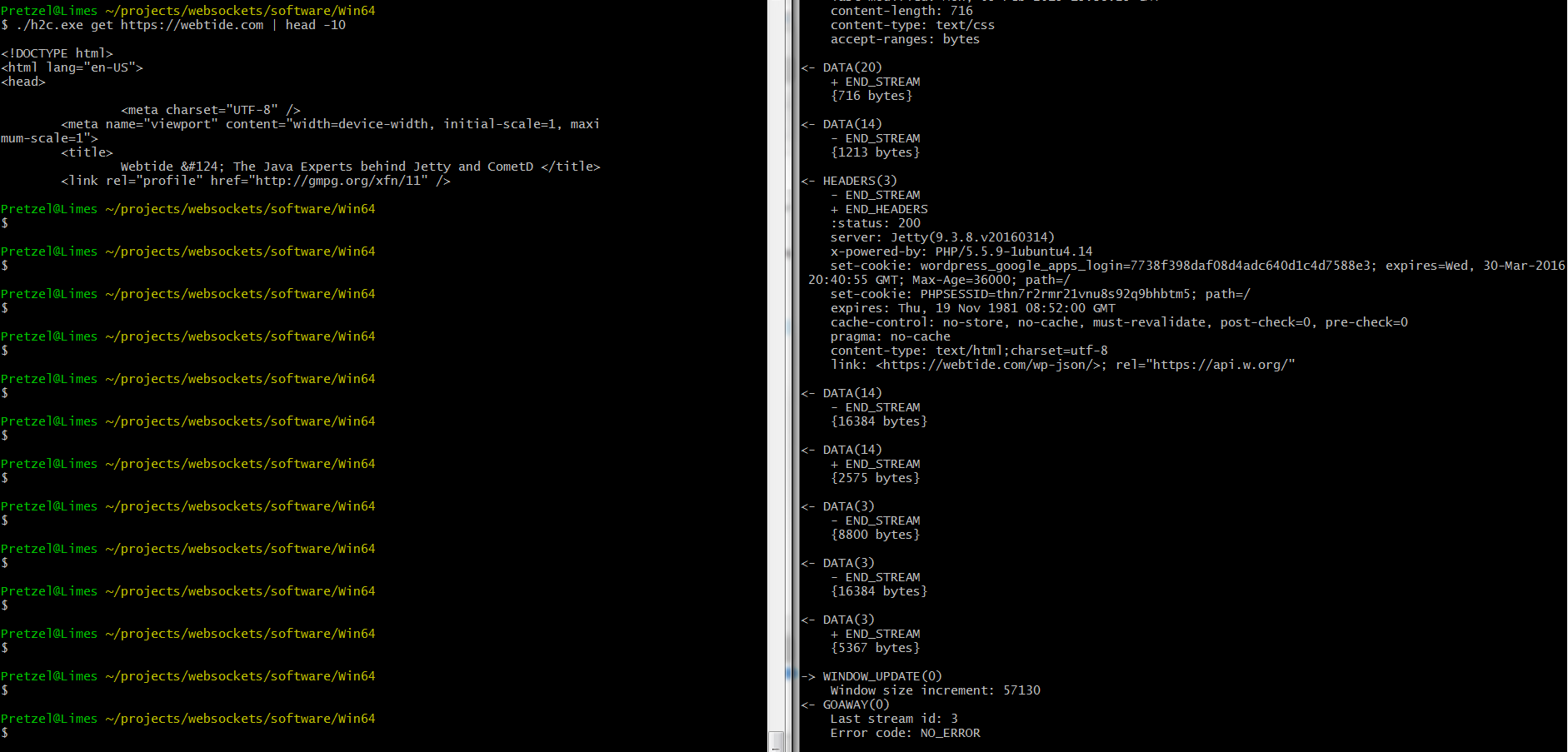
**Benodigd**:

* H2c – een lokale HTTP/2 proxy

**Stappen:**

1. Download van <https://github.com/fstab/h2c/releases/download/v0.0.12/h2c-v0.0.12.zip>
2. Haal hieruit *voor jouw platform* de juiste h2c(.exe)
3. Zet ‘m ergens op je PATH
4. start een console / shell
5. h2c start --dump
6. start een andere console
7. h2c get https://localhost:8443
   * of een andere lokaal draaiende HTTP/2 site – zie de gophertiles, Tomcat9 en Jetty recipes
8. h2c disconnect
9. h2c get https://webtide.com
10. Bekijk de frames – de webtide reponses bevatten PUSH\_PROMISE(1)’s!

**Impressie:**



Achtergrondinformatie:

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

## Gophertiles Apache/docker

**Case:** In deze lab start je een *ready made* docker image o.b.v. Apache met de HTTP/2 ‘gophertiles’ demo. Deze staat op <https://hub.docker.com/r/fstab/gophertiles> en is gemaakt door Fabian Stäber.

De applicatie heet Gophertiles aangezien deze in Go is geschreven en de snelheid van HTTP/2 demonstreert met een plaatje van twee gophers (grondeekhoorns) dat in tiles (tegels) is opgedeeld. Lache man ;-)

De applicatie staat ook live op <https://http2.golang.org/gophertiles> maar dat lijkt een oudere versie – zegt nl. altijd ‘Unfortunately, you're **not** using HTTP/2 right now.’

**Taak**: installeer docker, start de container en bekijk de demo in de browser.

**Benodigd**:

* Docker

**Stappen:**

1. Installeer docker: <https://docs.docker.com/engine/installation/>
2. Start docker machine; voor Windows met cygwin is dat:
   1. $> docker-machine start
   2. $> eval $(docker-machine.exe env default)
3. start de container: (duurt even; de image moet gedownload worden)
   1. docker run -d -p 4430:4430 -p 8080:8080 fstab/gophertiles
4. start je browser:
   1. URL: <https://192.168.99.100:4430/gophertiles>
5. Klik afwisselend op ‘HTTP/2, 200ms latency’ en ‘HTTP/1, 200ms latency’ – en zie hoe HTTP/1 last heeft van ‘Head of Line’ blocking en HTTP/2 niet.
   1. Tip: start eens de WebDeveloper (F12) en check hoe de network timing eruit ziet.

**Impressie:**



## Java9

Doel: Java 9 HTTP/2 client laten communiceren met diverse servers

Het stappenplan van dit recipe is het volgende:

* Installeer JDK9
* Stel toolchain.xml correct in
* Ga in de clone van de git repo naar dircteory ‘java9’
* mvn clean package exec:exec -P ResponseAsync  
  “This will fetch a list of target URIs asynchronously”
* Zie UriProvider: dit haalt momenteel 3 URLs op en zet ze in de target folder
* Momenteel valt dit helaas terug op HTTP/1.x
* Met de jul.properties file en java.logging.format property in de pom.xml zou extra logging getoond moeten worden – dat gebeurt echter niet

De uitdaging aan jullie: krijg dit werkend!

In de java9 directory vind je ook ZIPjes met de sources en javadoc van de http2client incubator module (van ea build 153), om je wat op weg te helpen.

## HTTP/2 & Tomcat9

**Case:** In deze lab ontwikkel je een server push WAR o.b.v. een in Tomcat9 meegeleverd HTTP/2 example.

**Taak**: installeer, configureer & start Tomcat, en programmeer dan het een en ander

**Benodigd**:

* Tomcat9
* JDK8   
  (of JDK9 - dan zie je wel in de Tomcat logs:  
  java.lang.reflect.InaccessibleObjectException: Unable to make public jdk.internal.ref.Cleaner java.nio.DirectByteBuffer.cleaner() accessible: module java.base does not "opens java.nio" to unnamed module. Het lijkt verder wel te werken... de Tomcat logs geven nog wel wat behulpzame hints:  
  *When running on Java 9 you need to add "--add-opens=java.base/java.lang=ALL-UNNAMED" to the JVM command line*, en

*When running on Java 9 you need to add "--add-opens=java.rmi/sun.rmi.transport=ALL-UNNAMED"*)

* Optioneel openssl

**Stappen:**

1. Download & unzip Tomcat 9.0.0.M17 (of hoger) van <https://tomcat.apache.org/download-90.cgi>
   * Download ook de [(Windows) APR binary](https://tomcat.apache.org/download-native.cgi) van <http://apache.hippo.nl/tomcat/tomcat-connectors/native/1.2.10/binaries/tomcat-native-1.2.10-win32-bin.zip> en pak deze uit in de bin folder van Tomcat.
   * Deze APR (‘Apache Portable Runtime’) is nodig omdat de NIO en NIO.2 implementaties van Java(8) SSL niet goed ondersteunen. Dit is een vergelijkbaar probleem als waarom voor Jetty de alpn-boot nodig is.
2. Kopieer naar apache-tomcat-9.0.0.M17\conf de volgende 2 files uit je clone van de github repo van deze workshop:
   * tomcat-cert.pem
   * tomcat-key.pem
3. Voeg aan apache-tomcat-9.0.0.M17\conf\server.xml (vanaf regel 114) het blok hieronder toe
4. Start apache-tomcat-9.0.0.M17\bin\startup.bat
   * Zoek in de logging naar: “INFO [main] org.apache.coyote.http11.AbstractHttp11Protocol.configureUpgradeProtocol The ["https-openssl-apr-8443"] connector has been configured to support negotiation to [h2] via ALPN” als bevestiging dat HTTP/2 werkt
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 *alleen* 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: een gePUSH’t plaatje!
9. Oke, dat is natuurlijk een beetje te eenvoudig – een meegeleverd & al gecompileerd example starten. Nu gaan we het zelf doen!
   * Maak een WAR projectje aan (maven, groovy: wat je zelf wil)
   * Voeg deze dependency toe: javax.servlet:javax.servlet-api:4.0.0-b02
   * Zet in src\main\webapp een te pushen static resource (bijv. een plaatje)
   * Maak een class aan die extend van HttpServlet
   * Override doGet() met:
     + PushBuilder pb = req.getPushBuilder().path("*je-static-file*");

pb.push();

* + - Zet iets in de response met o.a. <img src=”*je-static-file*"/>
  + Maak een src\main\webapp\WEB-INF\web.xml file aan die verwijst naar je Servletje
    - Hint: gebruik <servlet-class>

1. Bouw de WAR, copy-deploy in Tomcat & open in de browser
2. Oeps! Je krijgt (als het goed is ;-) een java.lang.NoSuchMethodError: javax.servlet.http.PushBuilder.push()V
   * Hmm, de Servlet 4.0 API is nog niet stabiel… De JAR die in Tomcat9-M17 wordt meegeleverd (4.0.EDR-b01) is anders dan die van onze dependency 4.0.0-b02!
   * Zie commando pb.push() – het return type is veranderd van boolean naar void
   * Je zou op zoek kunnen gaan in Maven Central naar de juiste dependency, maar de volgende Reflection hack werkt ook:  
     pb.getClass().getMethod("push").invoke(pb);
   * Solved!
3. Lees de PushBuilder javadoc door: <http://download.eclipse.org/jetty/stable-9/apidocs/org/eclipse/jetty/server/PushBuilder.html>
4. Probeer eens wat er gebeurt als je:
   * vele statics pusht
   * URLs pusht die een request naar je Servlet tot gevolg hebben
     + .. waarin je Thread.sleep toevoegt
   * De gepushte statics pas na een klik in de browser ophaalt
     + .. waarbij je Tomcat STOPT voordat je klikt!  
       (dat zou nl. geen probleem moeten zijn aangezien de statics al gePUSHt zijn en dus in de browser cache staan!!)
     + NB dit lijkt niet te werken voor .html’s – wellicht zul je met jQuery of Angular aan de slag moeten

Server.xml toevoeging:

<Connector port="8443" protocol="org.apache.coyote.http11.Http11AprProtocol"

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"

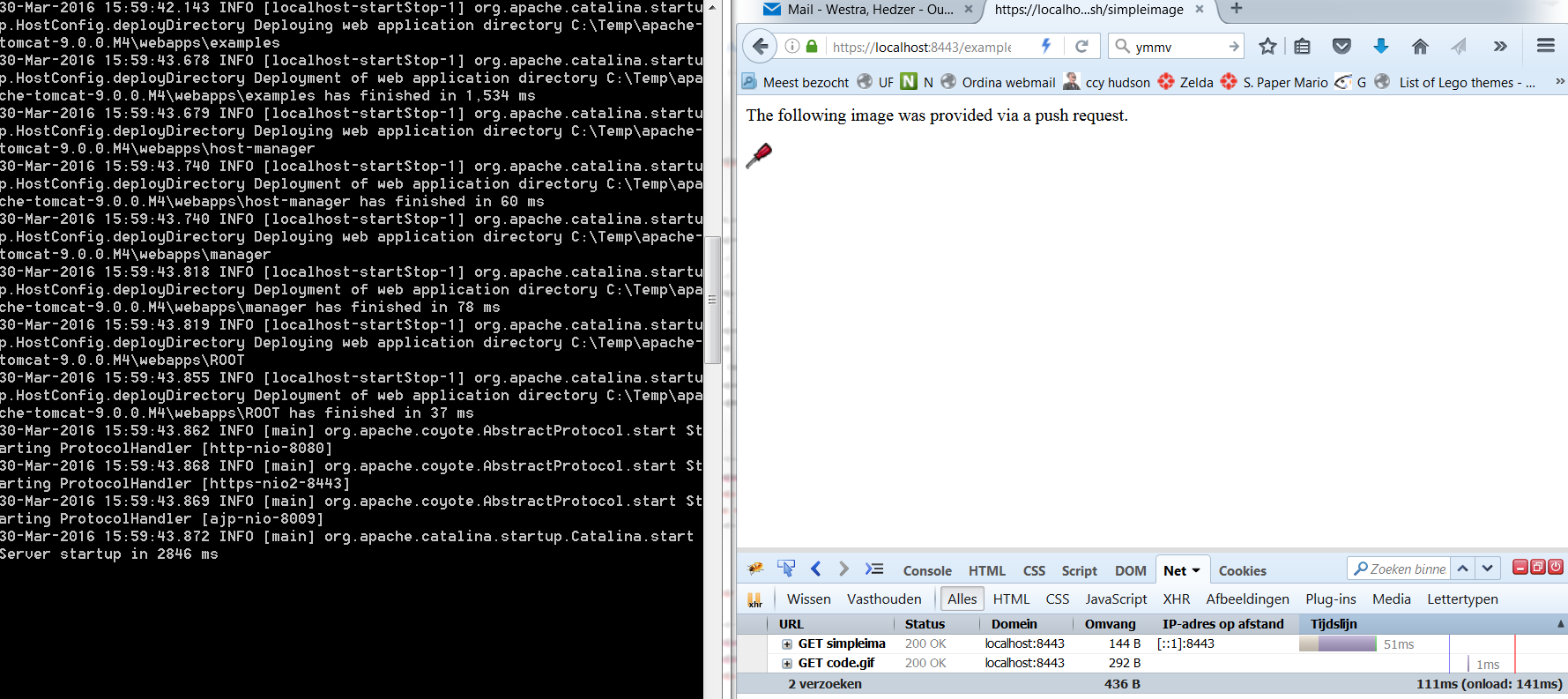
certificateKeyPassword="tomcat"

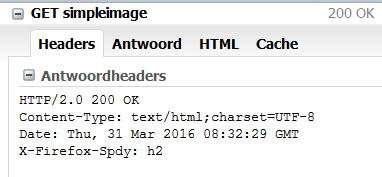
type="RSA" />

</SSLHostConfig>

</Connector>

**Impressie:**





**Bonus:**

* Benader de site via curl:
  + curl -v -k https://localhost:8443/examples/servlets/serverpush/simpleimage
  + Je ziet dan halverwege: “Got PUSH\_PROMISE, ignore it!” - curl kan/doet nl. (nog) niks met PUSH promises
* Uncomment het grijze stuk in server.xml en probeer unencrypted HTTP/2 aan de praat te krijgen.

**Achtergrondinformatie:**

Server push Java code

De Apache example code vind je in apache-tomcat-9.0.0.M17\webapps\examples\WEB-INF\classes\http2\SimpleImagePush.java

Dit gebruikt javax.servlet.http.PushBuilder, die (zij het niet exact hetzelfde) in Nexus / Maven central staat onder GAV javax.servlet:javax.servlet-api:4.0.0-b02. Dit is een *preview* van OpenJDK JSR-369 Servlet 4.0.

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 & Jetty (client-side)

**Case:** In deze lab maak je een Jetty HTTP/2 client o.b.v. de low-level API.

De high-level API programmeert lekkerder, maar daarmee kun je geen push berichten opvangen.

Let op: deze code werkt *alleen* met Java8! Via de maven toolchains configuratie gaat de compilatie goed, voor het runnen echter moet je expliciet verwijzen naar Java8! In het betreffende commando ga ik ervanuit dat je een environment variabele ‘JAVA8\_HOME’ hebt ingesteld.

**Benodigd**:

* Maven
* JDK8, geconfigureerd via toolchain en met environment variabele
* Een draaiende lokale HTTP/2 server (optioneel)

**Stappen:**

1. Maak een project aan (mag in maven, hoeft niet)
2. Voeg een dependency toe naar org.eclipse.jetty.http2:http2-client:9.4.0.v20161208
3. Maak een class aan met public static void main() methode
4. Voeg hierin toe:
5. HTTP2Client client = new HTTP2Client();

SslContextFactory sslContextFactory = new SslContextFactory(true);

client.addBean(sslContextFactory);

client.start();

1. Initialiseer ‘uri’, ‘host’ en ‘port’ variabelen
2. Tweede stuk code:

FuturePromise<Session> sessionPromise = new FuturePromise<>();

client.connect(sslContextFactory, new InetSocketAddress(host, port), new ServerSessionListener.Adapter(), sessionPromise);

Session session = sessionPromise.get(5, TimeUnit.SECONDS);

1. Derde stuk code:

HttpFields requestFields = new HttpFields();

MetaData.Request metaData = new MetaData.Request("GET", new HttpURI(uri), HttpVersion.HTTP\_2, requestFields);

HeadersFrame headersFrame = new HeadersFrame(metaData, null, true);

1. Vierde:

session.newStream(headersFrame, new Promise.Adapter<>(), new Stream.Listener.Adapter() {

@Override public void onHeaders(Stream stream, HeadersFrame frame) {

// print frame & frame.getMetaData()

}

@Override public void onData(Stream stream, DataFrame frame, Callback callback) {

// print frame & frame.getMetaData()

callback.succeeded();

}

@Override public Stream.Listener onPush(Stream stream, PushPromiseFrame frame) {

// print frame & frame.getMetaData()

return this;

}

1. Voeg een Thread.sleep() in
2. Afsluiten: client.stop();

Run deze code *met* het -Xbootclasspath/p:.. VM argument. En dan liefst tegen de hiervoor gebouwde Tomcat9 server, of de in docker gestarte gophertiles server. Je krijgt dan de PUSH\_PROMISES binnen zoals verwacht.