## Install and configure SSL/TLS (Secure Sockets Layer/Transport Layer Security) support on Tomcat 10 and automate the configuration as much as possible using Java/Groovy.

SSL/TLS support is normally installed as part of a Tomcat installation, but a case could be made for having to re-configure it for each instance of Tomcat where a separate application resides.

1. Create a Keystore file

%JAVA\_HOME%\bin\keytool -genkey -alias tomcat -keyalg RSA -keystore sample.keystore -storepass mypassword

**Note (1)** This creates a keystore file named *sample.keystore*. You can specify a different name is desired.

**Note (2)** The command above uses a password of *mypassword* —you want to change this

**Note (3)** There are other parameters that are gathered by part of this process. These parameters can be added on the command line, see the script that follows.

2. Edit the Tomcat configuration file (server.xml). This is in the *conf* directory under the root of the Tomcat installation (eg C:\apache-tomcat-10.1.23\apache-tomcat-10.1.23\conf). However, when setting up the server.xml for a seperate instance of the Tomcat server, the one to modify will be in the directory $CATALINA\_BASE/conf/server.xml

Edit the <Connector> element. Eg.

<Connector port="8080" protocol="HTTP/1.1"

connectionTimeout="20000"

redirectPort="8443"

maxParameterCount="1000"

/>

Add the following attributes:

keystoreFile="path/to/sample.keystore

keystorePass="mypassword"

keystoreType="JKS"

(Note that the name of the *keystore* and its *password* are the same as in step 1.)

3. Install a certificate from a certificate authority. It is placed in the conf directory under the root of the Tomcat installation. To install the certificate, run the following command:

%JAVA\_HOME%\bin\keytool -import -alias tomcat -file certificate.pem -keystore sample.truststore -storepass mypassword

**Note (1)** that the *password* is the same as in step 1.

**Note (2)** that the *sample.truststore* is being created. It has a very similar name to the *sample.keystore* created previously. This is just something to be aware of.

4. Once this is done the changes are implemented to the server when the server is initialized (which would require a restart if the Tomcat server is running).

A Groovy script to automates the configuration of SSL/TLS support on Tomcat 10:

import groovy.io.FileType

import java.io.File

import groovy.xml.XmlSlurper

import groovy.xml.XmlParser

// TODO: Ultimately we would want to create a UI interface to gather the info set below/used to execute the keytool application.

// I am making the assumption tha this will be converted into a Java/Groovy application in some IDE

def tomcatHome = "C:\\apache-tomcat-10.1.23\\apache-tomcat-10.1.23"

def keystoreFile = new File(tomcatHome, "conf\\sample.keystore")

def truststoreFile = new File(tomcatHome, "conf\\sample.truststore")

def javaPath = System.getenv('JAVA\_HOME')

if (!keystoreFile.exists()) {

// Create a keystore file if it does not exist.

println 'Creating keystore...'

def firstCmd = """

$javaPath\\bin\\keytool -genkey -noprompt -keyalg RSA \

-alias tomcat -dname "CN=sas.com, OU=ID, OU=SAS, L=Cary, S=NC, C=US" \

-keystore $keystoreFile -storepass changeit -keypass changeit

"""

println "Executing ${firstCmd}"

def sout = new StringBuffer()

def serr = new StringBuffer()

// consumeProcessOutput() gets the output and error streams from a process

// and reads them to keep the process from blocking due to a full output

// buffer. The processed stream data is appended to the supplied

// OutputStream. For this, two Threads are started, so this method will

// return immediately. The threads will not be join()ed, even if waitFor()

// is called. To wait for the output to be fully consumed call

// waitForProcessOutput().

// TODO: At the current time, attempting the following produces an error of "Unknown password type:" Yet

// when I run it on the command line, it works. I am unsure why it doesn't work like this, but moving

// on so i am not bogged down here.

def process = firstCmd.execute()

process.consumeProcessOutput(sout, serr)

process.waitForProcessOutput()

println 'Result ---'

System.out << sout.toString()

System.out << serr.toString()

println 'end result ---'

} else {

println 'Keystore exists.'

}

// Edit the Tomcat configuration file to add the SSL/TLS settings.

// note that the port may be different based on the needs of the setup

println 'Edit server.xml file...'

def xmlFile = new File(tomcatHome, '\\conf\\server.xml')

def serverXml = new XmlParser().parse(xmlFile)

def thisConnector = serverXml.Service[0].Connector[0]

//can change the port in the following line

thisConnector.@port = 443

thisConnector.@keystoreFile = "${keystoreFile}"

// TODO: fillin the correct password from the user's inputs

thisConnector.@keystorePass = "mypassword"

thisConnector.@keystoreType = "JKS"

// Note: This does not preserve any connects in the xml file that was read in, modified, and then read out

// TODO: write the new server.xml file to a differently named xml file, perhaps one that the user of the app can specify

new XmlNodePrinter(new PrintWriter(new FileWriter(xmlFile))).print(serverXml)

//TODO: restart the Tomcat server

To set up different instances of the Tomcat server using different configurations of SSL/TLS, use the environment variable $CATALINA\_BASE. This is used as the starting point against which most relative paths are resolved.

If you have not configured Tomcat for multiple instances by setting a $CATALINA\_BASE directory, then $CATALINA\_BASE will be set to the value of $CATALINA\_HOME, the directory into which you have installed Tomcat.