Spring Boot 2 Tutorial

Spring Boot – Introduction
Spring Boot – Starter parent
Spring Boot – Starter templates
Spring Boot – Multi-module project
Spring Boot – Annotations
Spring Boot – Auto configuration
Spring Boot – AOP
Spring Boot – Logging
Spring Boot – DevTools
Spring Boot – WAR Packaging
Spring Boot – REST API
Spring Boot – CRUD
Spring Boot – OAuth2
Spring Boot – Testing
Spring Boot – RestTemplate
Spring Boot – Thymeleaf
Spring Boot – Hibernate
Spring Boot – DataSource
Spring Boot – Error Handling

Spring Boot SSL [https] Example

By Sajal Chakraborty | Filed Under: Spring Boot

In this spring boot example, learn to configure web application to run on SSL (HTTPS) with self-signed certificate. Also learn to **create SSL cert**, as well.

SSL Configuration for Impatients

Spring boot HTTPS Config

```
server.port=8443
server.ssl.key-alias=selfsigned_localhost_sslserver
server.ssl.key-password=changeit
server.ssl.key-store=classpath:ssl-server.jks
server.ssl.key-store-provider=SUN
server.ssl.key-store-type=JKS
```

Redirect from HTTP to HTTPS

Search Tutorials

Type and Press ENTER...

Spring Boot – Caching
Spring Boot – Retry
Spring Boot – BasicAuth
Spring Boot – H2 Database
Spring Boot – Ehcache 3.x
Spring Boot – Gson
Spring Boot – RMI
Spring Boot – Send Email
Spring Boot – Interview Questions

Spring Boot Tutorial

Spring Boot – CommandLineRunner
Spring Boot – Configure Jetty
Spring Boot – Tomcat Default Port
Spring Boot – Context Root
Spring Boot – SSL [https]
Spring Boot – Get all loaded beans
Spring Boot – PropertyEditor
Spring Boot – @EnableScheduling
Spring Boot – Jersey
Spring Boot – SOAP Webservice
Spring Boot – SOAP Client
Spring Boot – JMSTemplate
Spring Boot – REST APIs

```java
private Connector redirectConnector() {
  Connector connector = new Connector("org.apache.coyote.http11.Http11
  connector.setScheme("http");
  connector.setPort(8080);
  connector.setSecure(false);
  connector.setRedirectPort(8443);
  return connector;
}
```

For detailed tutorial on how to setup whole thing, continue reading.

Table of Contents

Terminology
Create your own self signed SSL certificate
Create Spring-boot application and configure SSL
Redirect to HTTPS from HTTP

Spring Boot – JSP View
Spring Boot – Actuator endpoints
Spring Boot – Role Based Security
Spring Boot – RSS / ATOM Feed
Spring Boot – Ehcache 2.x

Popular Tutorials

Java 14 Tutorial
Java 8 Tutorial
Core Java Tutorial
Collections in Java
Java Concurrency
Java Date and Time
Java Examples
Spring Boot Tutorial
Spring AOP Tutorial
Spring MVC Tutorial
Spring Security Tutorial
Hibernate Tutorial
Python Tutorial
Jersey Tutorial
Maven Tutorial
Log4j Tutorial
Regex Tutorial

Terminology

Before moving further, let’s understand what specific terms such as SSL or TLS means.

SSL – stands for *Secure Sockets Layer*. It is the industry standard protocol for keeping an internet connection secure by safeguarding all sensitive data that is being sent between two systems, preventing hackers from reading and modifying any information transferred.

TLS – (Transport Layer Security) is an updated, more secure, version of SSL. It adds more features. Today, certificates provided by certificate authorities are based on TLS only. But regarding secured communication over network, the term SSL is still common as it is the old and just become popular among community.

HTTPS – (Hyper Text Transfer Protocol Secure) appears in the URL when a website is secured by an SSL certificate. It is the secured version of HTTP protocol.

Truststore and Keystore – Those are used to store SSL certificates in Java but there is little difference between them. `truststore` is used to store public certificates while `keystore` is used to store private certificates of client or server.

Create your own self signed SSL certificate

To get SSL digital certificate for our application we have two options –

1. to create a self-signed certificate
2. to obtain SSL certificate from certification authority(CA) we call it CA certificate.

For today’s demo purpose we will create self-signed certificate generated by java `keytool` command. We need to run the `keytool -g` enkey command from command prompt.

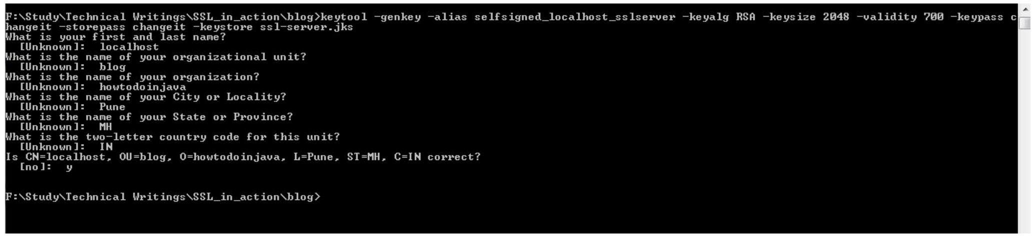
Here is the exact command we will use –

```
keytool -genkey -alias selfsigned_localhost_sslserver -keyalg RSA -key:
```

Let's understand above command –

- `-genkey` – is the keytool command to generate the certificate, actually keytool is a multipurpose and robust tool which has several options
- `-alias selfsigned_localhost_sslserver` – indicates the alias of the certificate, which is used by SSL/TLS layer
- `-keyalg RSA -keysize 2048 -validity 700` – are self descriptive parameters indicating the crypto algorithm, keysize and certificate validity.
- `-keypass changeit -storepass changeit` – are the passwords of our truststore and keystore
- `-keystore ssl-server.jks` – is the actual keystore where the certificate and public/private key will be stored. Here we are using JKS fromat – *Java Key Store*, there are other formats as well for keystore.

Once we execute above command, it will ask for certain information and finally this will look like this.



```
F:\Study\Technical Writings\SSL_in_action\blog>keytool -genkey -alias selfsigned_localhost_sslserver -keyalg RSA -keysize 2048 -validity 700 -keypass c
hangeit -storepass changeit -keystore ssl-server.jks
What is your first and last name?
  [Unknown]:  localhost
What is the name of your organizational unit?
  [Unknown]:  blog
What is the name of your organization?
  [Unknown]:  howtodoinjava
What is the name of your City or Locality?
  [Unknown]:  Pune
What is the name of your State or Province?
  [Unknown]:  MH
What is the two-letter country code for this unit?
  [Unknown]:  IN
Is CN=localhost, OU=blog, O=howtodoinjava, L=Pune, ST=MH, C=IN correct?
  [no]:  y


F:\Study\Technical Writings\SSL_in_action\blog>
```

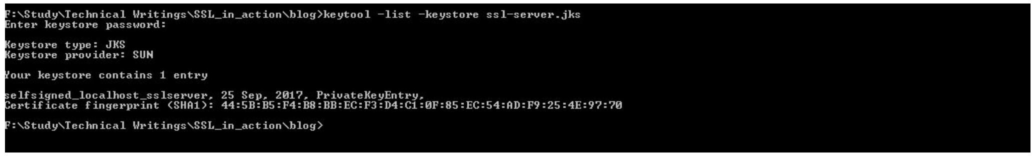
keytool to generate the certificate

That’s all we need at this point regarding certification generation. This will generate the `ssl-server.jks` keystore file containing our self signed certificates in the directory from where keytool command has been executed.

To view what is inside this keystore we can again use the `keytool -list` command as bellow.

```
keytool -list -keystore ssl-server.jks
```

Output will be something like –



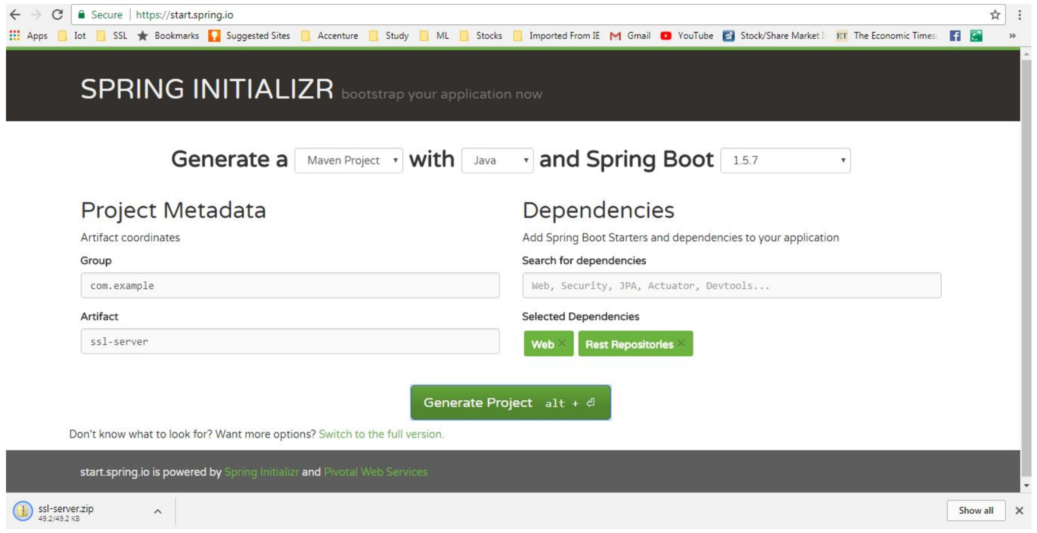
```
F:\Study\Technical Writings\SSL_in_action\blog>keytool -list -keystore ssl-server.jks
Enter keystore password:

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

selfsigned_localhost_sslserver, 25 Sep, 2017, PrivateKeyEntry,
Certificate fingerprint (SHA1): 44:5B:B5:F4:B8:BB:EC:F3:D4:C1:0F:85:EC:54:AD:F9:25:4E:97:70

F:\Study\Technical Writings\SSL_in_action\blog>
```

keytool -list option

Create Spring-boot project and configure SSL

Generate spring boot project

Create one spring boot project from SPRING INITIALIZR site with dependencies `Web` and `Rest Repositories`. After selecting the dependencies and giving the proper maven GAV coordinates, we will get download option in zipped format. Download the skeleton project, unzip and then import that in eclipse as maven project.



SPRING INITIALIZR bootstrap your application now

Generate a Maven Project with Java and Spring Boot 1.5.7

Project Metadata
Artifact coordinates
Group
com.example
Artifact
ssl-server

Dependencies
Add Spring Boot Starters and dependencies to your application
Search for dependencies
Web, Security, JPA, Actuator, Devtools...
Selected Dependencies
Web ✕ Rest Repositories ✕

Generate Project alt + ⏎

Don't know what to look for? Want more options? Switch to the full version.

start.spring.io is powered by Spring Initializr and Pivotal Web Services

Spring boot project generation

Add REST endpoint

For testing purpose we will use one simple REST endpoint. To do that open the already generated spring boot application class annotated with `@SpringBootApplication` and add this code. This will expose one rest endpoint with relative URL `/secured` in the server.

```
@RestController
class SecuredServerController{

    @RequestMapping("/secured")
    public String secured(){
        System.out.println("Inside secured()");
        return "Hello user !!! : " + new Date();
    }
}
```

That's all we need to add web contents in our application. You can add more like adding pages, images to create a fully functional web application.

Spring boot SSL Configuration

First we need to copy the generated keystore file (`ssl-server.jks`) into the `resources` folder and then open the `application.properties` and add the below entries.

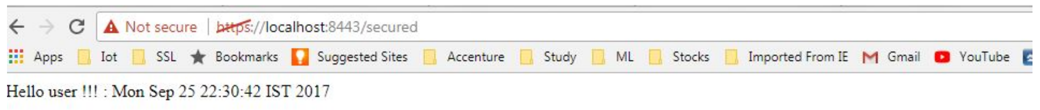
```
server.port=8443
server.ssl.key-alias=selfsigned_localhost_sslserver
server.ssl.key-password=changeit
server.ssl.key-store=classpath:ssl-server.jks
server.ssl.key-store-provider=SUN
server.ssl.key-store-type=JKS
```

That's all we need to enable https. It's pretty easy, right? Thanks to spring boot for making everything possible very easily.

Demo

Now it is time to do a final maven build by command `mvn clean install` and start the application by `java -jar target\ssl-server-0.0.1-SNAPSHOT.jar` command. This will start our secured application in `localhost 8443` port and our end point url will be https://localhost:8443/secured.

Since our REST endpoint is exposed over GET, we can test it through browser only. Go to https://localhost:8443/secured and you will get some browser warning like certificate is not issued from trusted certificate authorities, add exception to that in browser and you will get response from HTTPS server just created by you.



```
Hello user !!! : Mon Sep 25 22:30:42 IST 2017
```

Browser output

Redirect HTTP requests to HTTPS

This is an optional step in case you want to redirect your HTTP traffic to HTTPS, so that the full site becomes secured. To do that in spring boot, we need to add HTTP connector at `8080` port and then we need to set redirect port `8443`. So that any request in `8080` through http, it would be automatically redirected to `8443` and https.

To do that you just need to add below configuration.

```
@Bean
public EmbeddedServletContainerFactory servletContainer() {
  TomcatEmbeddedServletContainerFactory tomcat = new TomcatEmbeddedSer
      @Override
      protected void postProcessContext(Context context) {
        SecurityConstraint securityConstraint = new SecurityConstraint
        securityConstraint.setUserConstraint("CONFIDENTIAL");
        SecurityCollection collection = new SecurityCollection();
        collection.addPattern("/*");
        securityConstraint.addCollection(collection);
        context.addConstraint(securityConstraint);
      }
    };

  tomcat.addAdditionalTomcatConnectors(redirectConnector());
  return tomcat;
}

private Connector redirectConnector() {
  Connector connector = new Connector("org.apache.coyote.http11.Http11
  connector.setScheme("http");
```

```
    connector.setPort(8080);
    connector.setSecure(false);
    connector.setRedirectPort(8443);

    return connector;
  }
```

Do a final maven build by command `mvn clean install` and start the application. Test http://localhost:8080/secured. It would be automatically redirected to HTTPS secured URL.

Summary

So today we learned, how we can **enable HTTPS in spring boot application** and also we have seen how we can **redirect HTTP traffic to HTTPS**. We also learned to **create self signed SSL certificate**.

Drop me your questions in comments section.

Download Source code

Happy Learning !!

Feedback, Discussion and Comments

Nipun Gupta

February 14, 2020

I am not able to make a POST/PUT call , getting error as Forbidden. However Get works as expected.

Reply

Pablo

October 2, 2019

Literally thank you so much! Wasted hours on this one.. Keep it up!

Reply

Anil Kumar

July 25, 2019

“Go to https://localhost:8443/secured and you will get some browser warning like certificate is not issued from trusted certificate authorities, add exception to that in browser and you will get response from HTTPS server just created by you.”

How to overcome the browser warning? What next I need to perform to overcome browser warning.

Thanks

Reply

Dineth Senevirathne

June 23, 2019

https://github.com/hdineth/spring-boot-ssl-https-example

Reply

Abner Hernandez

May 28, 2019

Hello, I got this error:
IllegalArgumentException: Private key must be accompanied by certificate chain
Any help?
Using: Spring Tool Suite 4
Version: 4.2.1.RELEASE
Build Id: 201904170946
Java 1.8
Thanks

Reply

Raj

August 8, 2019

replace server.ssl.key-password with server.ssl.key-store-password in application.properties

Reply

Constantijn Blondel

January 8, 2020

I can confirm this is correct.

Reply

saif farooqui

March 28, 2019

How to run this same project on jdk 1.7?

Reply

dew

January 15, 2019

Hi,
Thank you for sharing your knowledge.
I wonder where I should put the configuration code for "Redirect HTTP requests to HTTPS"?
Is it a separate configuration file ? Does it require a class for these code?

Thank you so much for your sharing.
Dew

Reply

Raj Modi

August 8, 2019

Just paste the code in main class. no need to create separate configuration file

Reply

vishnu dixit

November 4, 2018

i want clarity on SSL and TLS what was the ssl and what is the TLS so, i want to make two project configuring with SSL certificate protocol and another one with TLS protocol certificate so, i will be able to see clearly , what are features in the ssl and what are in TLS

Reply

Ramya

July 25, 2018

How can I avoid getting not secure exception? Is there a way to sign certificate and make it trusted?

Reply

Qi Shan

July 19, 2018

when trying added POST on http://localhost:8080/… , it handles the POST (sent through postman), as GET — as if a GET was sent. it returns the same response as GET request.

Qi Shen

Reply

Rojalin

July 19, 2018

This example redirect me to a login page with URL :
https://localhost:8098/login
I have not added the "Redirect HTTP requests to HTTPS" part yet.

Reply

venkat

November 17, 2019

Remove any spring security dependencies if any…

Reply

Savani

June 5, 2018

Many Thanks for such a great tutorials. Could you please create some tutorials for the Spring Boot 2.0?

Reply

Lokesh Gupta

June 5, 2018

You can find some at https://howtodoinjava.com/category/spring/spring-boot2/

Reply

Mohammed Thaqi

May 29, 2018

This page isn’t working
localhost didn’t send any data.

I am getting this error, when hitting “http://localhost:8443/secured”. Please suggest.

Reply

Lokesh Gupta

May 29, 2018

Plrease use “https” in place of “http” in URL.

Reply

Adit Choudhary

May 26, 2018

The class has been removed and replaced by org.springframework.boot.web.embedded.tomcat.TomcatServletWebServerFactory

Use the below code for Spring boot 2.0.0 release.

@Bean
public ServletWebServerFactory servletContainer() {
TomcatServletWebServerFactory tomcat = new TomcatServletWebServerFactory() {
@Override
protected void postProcessContext(Context context) {
SecurityConstraint securityConstraint = new SecurityConstraint();
securityConstraint.setUserConstraint(“CONFIDENTIAL”);

SecurityCollection collection = new SecurityCollection();
collection.addPattern("/*");
securityConstraint.addCollection(collection);
context.addConstraint(securityConstraint);
}
};
tomcat.addAdditionalTomcatConnectors(redirectConnector());
return tomcat;
}

private Connector redirectConnector() {
Connector connector = new
Connector("org.apache.coyote.http11.Http11NioProtocol");
connector.setScheme("http");
connector.setPort(8080);
connector.setSecure(false);
connector.setRedirectPort(8443);
return connector;
}

Reply

Joaquin Ponte

May 4, 2018

Hi, did you find any solution for this? I have the same problem

Reply

Lokesh Gupta

May 4, 2018

What problem?

Reply

Irko

April 16, 2018

This code works on version 1.5.7.RELEASE, however it doesn't compile on the current 2.0.1.RELEASE of SpringBoot.

Reply

Kumar

March 23, 2018

I need HTTPS using TLS 1.2 in Spring boot. How can I achieve that

Reply

Akash

February 2, 2018

Hi can you tell how about integrating a Apache ssl certified that is issued from gadaddy? They give 3 different files. Any help on it. Thanks

Reply

Tushar

February 15, 2020

In My case it asks for some username/password , what would be the default username/password for TLS ?

Reply

Ask Questions & Share Feedback

Please do not submit a comment only to say "Thank you".

Comment

*Want to Post Code Snippets or XML content? Please use [java] ... [/java] tags otherwise code may not appear partially or even fully. e.g.

```
[java]
public static void main (String[] args) {
...
}
[/java]
```

Name *

Email *

Website

POST COMMENT

Meta Links

Advertise
Contact Us
Privacy policy
About Me

Recommended Reading

10 Life Lessons
Secure Hash Algorithms
How Web Servers work?
How Java I/O Works Internally?
Best Way to Learn Java
Java Best Practices Guide
Microservices Tutorial
REST API Tutorial
How to Start New Blog