

[f Share](#) [Tweet](#) [in Share](#) [g+ Share](#) [Pin](#)

Enable https in Spring Boot

February 8, 2019 / Spring Boot / By Java Development Journal / 4 COMMENTS



Updated on February 3rd, 2020

In this article of Spring Boot tutorial, we will see **how to enable HTTPS in Spring Boot application**. We will generate and configure the self-signed certificate for this example to **enable HTTPS in Spring Boot application**.

1. Create Self Signed SSL Certificate

In this article, we will use *Java keytool* to **generate the self-signed SSL certificate**. Other options is to get this certificate from a certification authority. For the production environment, always get your SSL certificate from these certifications authority (CA) also known as CA. To generate the SSL certificate, run the **keytool -genkey** command with the following parameters from the command prompt.

```
keytool -genkeypair -alias javadevjournal -keyalg RSA -keysize 2048 -storetype PKCS12 -keystore javadevjournal.p12 -v
```

On running above command, it will ask for certain information. This is how the command line will look like:

```
Last login: Thu Jan 24 23:02:57 on ttys000
System:~ umesh$ keytool -genkeypair -alias javadevjournal -keyalg RSA -keysize 2048 -storetype PKCS12 -keystore javadevjournal.p12 -validity 3650
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: javadevjournal
What is the name of your organizational unit?
[Unknown]:
What is the name of your organization?
[Unknown]: javadevjournal
What is the name of your City or Locality?
[Unknown]: san jose
What is the name of your State or Province?
[Unknown]: california
What is the two-letter country code for this unit?
[Unknown]: US
Is CN=javadevjournal, OU=javadevjournal, O=javadevjournal, L=san jose, ST=california, C=US correct?
[no]: y
System:~ umesh$
```

1.1 Keytool Command

Before moving ahead let's understand above command:

f Share

T Tweet

in Share

g+ Share

Pin

- `-keyalg RSA -keysize 2048 -validity 3650`—crypto algorithm, keysize and certificate validity.
- `-keystore javadevjournal.p12`—actual keystore where the certificate and public/private key stored.

We have the option to generate the following certificate while working on self signed SSL certificate:

1. **JKS** – Java KeyStore, limited to the Java environment only.
2. **PKCS12** – Public Key Cryptographic Standards is a password-protected format that can contain multiple certificates and keys; it's an industry-wide used format.

1.2 Certificate Details

To view the details of the keystore, use the `keytool -list` command:

```
keytool -list -keystore javadevjournal.p12
```

You will have similar output on the console:

```
System:~ umesh$ keytool -list -keystore javadevjournal.p12
Enter keystore password:
Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

javadevjournal, Jan 26, 2019, PrivateKeyEntry,
Certificate fingerprint (SHA1): 0C:C5:6C:0D:53:EC:CC:DB:D7:C4:4D:82:57:E7:88:E8:08:D6:89:8F
System:~ umesh$
```

2. Enabling HTTPS in Spring Boot

Create a new Spring Boot project to test our SSL certification. Spring Boot provides the option to *enable HTTPS using the application.properties file*. To configure SSL, use the `server.ssl.*` properties in `application.properties`. Read our article to

```
# The format used for the keystore. for JKS, set it as JKS
server.ssl.key-store-type=PKCS12
# The path to the keystore containing the certificate
server.ssl.key-store=classpath:keystore/javadevjournal.p12
# The password used to generate the certificate
server.ssl.key-store-password=use the same password which we added during certificate creation
# The alias mapped to the certificate
server.ssl.key-alias=javadevjournal
# Run Spring Boot on HTTPS only
server.port=8443
```

[f Share](#) [T Tweet](#) [in Share](#) [g+ Share](#) [Pin](#)

With above configuration, we have *enabled HTTPS in Spring Boot application*. Our application is ready to accept and server over HTTPS. Let's create a simple REST controller to test our application.

3. REST Controller

To test our application, let's create a simple REST controller.

```
package com.javadevjournal.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class SSLTestController {

    @GetMapping(value = "/ssl-test")
    public String greeting(){
        return "Self Signed SSL is Working!!";
    }
}
```

4. SSL in Action

To see our SSL configuration in action, build and deploy our Spring Boot application. You can also run your application through your IDE for testing purpose. Once your application is up and running Go to <https://localhost:8443/ssl-test> and you will get a browser warning since it is not issued by a trusted certificate authorities, add the exception to the browser and you will get a response from HTTPS server just created by you.



Your connection is not private

Attackers might be trying to steal your information from **localhost** (for example, passwords, messages, or credit cards). [Learn more](#)

NET::ERR_CERT_AUTHORITY_INVALID

☐ Help improve Safe Browsing by sending some [system information and page content](#) to Google.
[Privacy policy](#)

Advanced

Back to safety

f Share

T Tweet

in Share

g+ Share

Pin

“Configurations!! You are have **enabled HTTPS in Spring Boot application**

5. Redirect HTTP to HTTPS

This step becomes important if you qualify under following categories:

1. You want to make your application accessible over HTTP but want to redirect to HTTPS.
2. You are moving away from HTTP to HTTPS and want to make sure all HTTP requests redirects to HTTPS.

To handle this, add the below code to your configuration class:

```
@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;
}
```

Copy

X

If you are using Spring Boot < 2.0, above code will not work for you. For Spring Boot lower version, use the following code:

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

f Share

T Tweet

in Share

g+ Share

Pin

```
    }  
};  
  
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;  
}
```

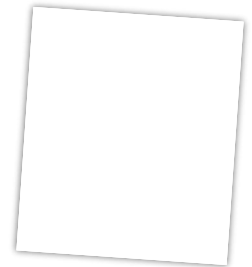
Summary

In this article, we saw how to *create a self-signed SSL certificate* and **enable HTTPS in Spring Boot application**. We also learned the steps and configuration to **redirect HTTP traffic to HTTPS**. Source code for this article is available over [GitHub](#)

Java Development Journal

Hello!! Welcome to the Java Development Journal. We love to share our knowledge with our readers and love to build a thriving community.

follow me on:  



Further Reading

18 Feb, 2020

[Spring Boot Configuration Properties](#)

04 Feb, 2020

[Multi-Module Project With Spring Boot](#)

29 Oct, 2019

[Introduction to Spring Boot Scheduler](#)

20 Aug, 2019

[Spring Boot CORS](#)

13 Aug, 2019

[Swagger 2 with Spring REST API](#)

f Share

T Tweet

in Share

g+ Share

P Pin

Custom Type Converter in Spring mvc

4 Leave a Reply



Join the discussion...

1 3 0

3

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)

Subscribe

newest **oldest** most voted

Guest

Raju Gurung

Using spring boot 2.1.4 and I used the http redirect code above and I get exception when sending a post request. It looks like it's converting my POST request to a get and failing { "status": 401, "message": "Authentication method not supported", "errorCode": 10, "timestamp": "2019-05-16T03:44:15.007+0000" } This is the log: Checking match of request : '/api/auth/login'; against '/logout' OrRequestMatcher : Trying to match using Ant [pattern='/logout', POST] AntPathRequestMatcher : Request 'GET /api/auth/login' doesn't match 'POST /logout' OrRequestMatcher : Trying to match using Ant [pattern='/logout', PUT] AntPathRequestMatcher : Request 'GET /api/auth/login' doesn't match 'PUT /logout' OrRequestMatcher : Trying to match... [Read more »](#)

0

Reply

10 months ago



Admin

Java Development Journal

The above code is not related to the SSL connection or redirect. It seems that you have Spring security configured for your application and that is causing the issue. I need a more context of the application or may be the code snippet to provide more input.

0

Reply

10 months ago



Guest

jitu

protected void postProcessContext(Context context) // here i cant understand about Context and where i can get Context Class or interface

0

Reply

8 months ago



Admin

Java Development Journal

That is the Tomcat Context. For more information read Context

0

Reply

7 months ago

f

Share

🐦

Tweet

in

Share

g+

Share

📌

Pin

SPRING BOOT TUTORIALS

SPRING MVC TUTORIAL

SPRING

NEWSLETTER

REST

SHOPIZER

SHOPIZER SETUP

SPRING

SPRING BOOT

SPRING MVC

DOCUMENTS


JAVA AND SPRING INTERVIEW QUESTIONS

PRIVACY POLICY

SPRING BOOT TUTORIALS

SPRING MVC TUTORIAL

WRITE AND EARN



Java Development Jour