

#### **Spring blog**

All Posts 🔊

Engineering

Releases 🔊

News and Events

# Spring Security without the WebSecurityConfigurerAdapter

ENGINEERING | ELEFTHERIA STEIN-KOUSATHANA | FEBRUARY 21, 2022 | 101 COMMENTS

In Spring Security 5.7.0-M2 we deprecated the WebSecurityConfigurerAdapter, as we encourage users to move towards a component-based security configuration.

To assist with the transition to this new style of configuration, we have compiled a list of common use-cases and the suggested alternatives going forward.

In the examples below we follow best practice by using the Spring Security lambda DSL and the method <a href="httpSecurity#authorizeHttpRequests">httpSecurity#authorizeHttpRequests</a> to define our authorization rules. If you are new to the lambda DSL you can read about it in this blog post. If you would like to learn more about why we choose to use <a href="httpSecurity#authorizeHttpRequests">httpSecurity#authorizeHttpRequests</a> you can check out the reference documentation.

# **Configuring HttpSecurity**

In Spring Security 5.4 we introduced the ability to configure HttpSecurity by creating a SecurityFilterChain bean.

Below is an example configuration using the WebSecurityConfigurerAdapter that secures all endpoints with HTTP Basic:

```
@Configuration

public class SecurityConfiguration extends WebSecurityConfigurerAdapter

@Override

protected void configure(HttpSecurity http) throws Exception {
   http

   .authorizeHttpRequests((authz) -> authz

   .anyRequest().authenticated()

)
```

### Get the Spring newsletter

**Email Address** 

Yes, I would like to be contacted by The Sp and VMware for new promotions and ever

**SUBSCRIBE** 



```
}
```

Going forward, the recommended way of doing this is registering a

SecurityFilterChain bean:

# **Configuring WebSecurity**

In Spring Security 5.4 we also introduced the WebSecurityCustomizer.

The WebSecurityCustomizer is a callback interface that can be used to customize WebSecurity .

Below is an example configuration using the WebSecurityConfigurerAdapter that ignores requests that match /ignore1 or /ignore2:

```
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter

@Override
public void configure(WebSecurity web) {
    web.ignoring().antMatchers("/ignore1", "/ignore2");
}
```



Going forward, the recommended way of doing this is registering a WebSecurityCustomizer bean:

```
@Configuration
public class SecurityConfiguration {

    @Bean
    public WebSecurityCustomizer webSecurityCustomizer() {
        return (web) -> web.ignoring().antMatchers("/ignore1", "/ignore2")
    }
}
```

**WARNING**: If you are configuring WebSecurity to ignore requests, consider using permitAll via HttpSecurity#authorizeHttpRequests instead. See the configure Javadoc for additional details.

#### **LDAP Authentication**

In Spring Security 5.7 we introduced the

EmbeddedLdapServerContextSourceFactoryBean ,

LdapBindAuthenticationManagerFactory and

LdapPasswordComparisonAuthenticationManagerFactory which can be used to

create an embedded LDAP Server and an AuthenticationManager that performs LDAP authentication.

Below is an example configuration using WebSecurityConfigurerAdapter the that creates an embedded LDAP server and an AuthenticationManager that performs LDAP authentication using bind authentication:



Going forward, the recommended way of doing this is using the new LDAP classes:

```
COPY
@Configuration
public class SecurityConfiguration {
               @Bean
                public EmbeddedLdapServerContextSourceFactoryBean contextSourceFactory
                                {\tt EmbeddedLdapServerContextSourceFactoryBean\ contextSourceFactoryBean\ contextSourceFactoryB
                                                EmbeddedLdapServerContextSourceFactoryBean.fromEmbeddedLdapS
                                contextSourceFactoryBean.setPort(0);
                                return contextSourceFactoryBean;
                }
                @Bean
                AuthenticationManager ldapAuthenticationManager(
                                                BaseLdapPathContextSource contextSource) {
                                LdapBindAuthenticationManagerFactory factory =
                                                new LdapBindAuthenticationManagerFactory(contextSource);
                                factory.setUserDnPatterns("uid={0},ou=people");
                                factory.setUserDetailsContextMapper(new PersonContextMapper());
                                return factory.createAuthenticationManager();
                }
```

#### **JDBC Authentication**

Below is an example configuration using the WebSecurityConfigurerAdapter with an embedded DataSource that is initialized with the default schema and has a single user:



The recommended way of doing this is registering a <code>JdbcUserDetailsManager</code> bean:

```
COPY
@Configuration
public class SecurityConfiguration {
    @Bean
    public DataSource dataSource() {
        return new EmbeddedDatabaseBuilder()
            .setType(EmbeddedDatabaseType.H2)
            .addScript(JdbcDaoImpl.DEFAULT_USER_SCHEMA_DDL_LOCATION)
            .build();
    }
    @Bean
    public UserDetailsManager users(DataSource dataSource) {
        UserDetails user = User.withDefaultPasswordEncoder()
            .username("user")
            .password("password")
            .roles("USER")
            .build();
        JdbcUserDetailsManager users = new JdbcUserDetailsManager(dataSc
        users.createUser(user);
        return users;
    }
}
```

Note: In these examples, we use the method

User.withDefaultPasswordEncoder() for readability. It is not intended for production and instead we recommend hashing your passwords externally. One way to do that is to use the Spring Boot CLI as described in the reference documentation.

## **In-Memory Authentication**



```
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter
  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws {
    UserDetails user = User.withDefaultPasswordEncoder()
        .username("user")
        .password("password")
        .roles("USER")
        .build();
    auth.inMemoryAuthentication()
        .withUser(user);
  }
}
```

The recommended way of doing this is registering an

InMemoryUserDetailsManager bean:

Note: In these examples, we use the method

User.withDefaultPasswordEncoder() for readability. It is not intended for production and instead we recommend hashing your passwords externally. One way to do that is to use the Spring Boot CLI as described in the reference documentation.

# **Global AuthenticationManager**

To create an AuthenticationManager that is available to the entire application you can simply register the AuthenticationManager as a @Bean.



# **Local Authentication Manager**

In Spring Security 5.6 we introduced the method

HttpSecurity#authenticationManager that overrides the default

AuthenticationManager for a specific SecurityFilterChain. Below is an example configuration that sets a custom AuthenticationManager as the default:

### **Accessing the local AuthenticationManager**

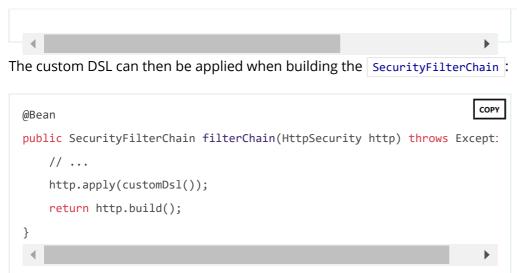
The local AuthenticationManager can be accessed in a custom DSL. This is actually how Spring Security internally implements methods like HttpSecurity.authorizeRequests().

```
public class MyCustomDsl extends AbstractHttpConfigurer<MyCustomDsl,
@Override

public void configure(HttpSecurity http) throws Exception {
    AuthenticationManager authenticationManager = http.getSharedObje
    http.addFilter(new CustomFilter(authenticationManager));
}

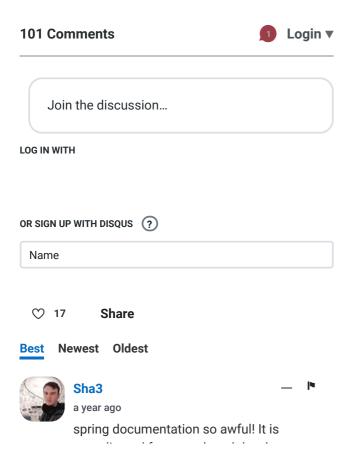
public static MyCustomDsl customDsl() {
    return new MyCustomDsl();</pre>
```





#### **Getting Involved**

We are excited to share these updates with you and we look forward to further enhancing Spring Security with your feedback! If you are interested in contributing, you can find us on GitHub.





#### **Get ahead**

VMware offers training and certification to turbo-charge your progress.

Learn more

Batch

#### **Get support**

Spring Runtime offers support and binaries for OpenJDK™, Spring, and Apache Tomcat® in one simple subscription.

Learn more

# Upcoming events

Check out all the upcoming events in the Spring community.

View all

Why Spring	Learn
Microservices	Quickstart
Reactive	Guides
Event Driven	Blog
Cloud	
Web Applications	
Serverless	

Community Projects

Events Training

Team Thank You

Support

Security Advisories



© 2023 VMware, Inc. or its affiliates. Terms of Use • Privacy• Trademark Guidelines • Your California Privacy Rights • Cookie Settings

Apache®, Apache Tomcat®, Apache Kafka®, Apache Cassandra™, and Apache Geode™ are trademarks or registered trademarks of the Apache Software Foundation in the United States and/or other countries. Java™, Java™ SE, Java™ EE, and OpenJDK™ are trademarks of Oracle and/or its affiliates. Kubernetes® is a registered trademark of the Linux Foundation in the United States and other countries. Linux® is the registered trademark of Linus Torvalds in the United States and other countries. Windows® and Microsoft® Azure are registered trademarks of Microsoft Corporation. "AWS" and "Amazon Web Services" are trademarks or registered trademarks of Amazon.com Inc. or its affiliates. All other trademarks and copyrights are property of their respective owners and are only mentioned for informative purposes. Other names may be trademarks of their respective owners.