Spring Boot Reference Documentation

Phillip Webb, Dave Syer, Josh Long, Stéphane Nicoll, Rob Winch, Andy Wilkinson, Marcel Overdijk, Christian Dupuis, Sébastien Deleuze, Michael Simons, Vedran Pavić, Jay Bryant, Madhura Bhave, Eddú Meléndez, Scott

Frederick

Spring Boot参考文档

菲利普·韦伯（Phillip Webb），戴夫·赛尔（Dave Syer），乔什·朗（Josh Long），斯特凡·尼科尔（StéphaneNic​​oll），罗伯·温奇（Rob Winch），安迪·威尔金森（Andy Wilkinson），马塞尔·奥弗迪杰克（Marcel Overdijk），克里斯蒂安·杜普依斯（Christian Dupuis），塞巴斯蒂安·德勒兹（SébastienDeleuze），迈克尔·西蒙斯（Michael Simons），韦德兰·帕维奇（VedranPavić），杰伊·布莱恩特（Madhura Bhave），埃德杜·梅隆德斯（EddMeléndez），斯科特（Scott）

弗雷德里克

**Table of Contents** 目录

[Legal 1](#_Toc54423655)

[法律声明 1](#_Toc54423656)

[1.1. About the Documentation 有关文献 2](#_Toc54423657)

[1.2. Getting Help 获得帮助 2](#_Toc54423658)

[1.3. First Steps 第一步 3](#_Toc54423659)

[1.4. Working with Spring Boot 使用 Spring Boot 3](#_Toc54423660)

[1.5. Learning about Spring Boot Features 学习 Spring Boot 的特性 3](#_Toc54423661)

[1.6. Moving to Production 进入生产阶段 4](#_Toc54423662)

[1.7. Advanced Topics 进阶主题 4](#_Toc54423663)

[2.1. Introducing Spring Boot Spring Boot 简介 5](#_Toc54423664)

[2.2. System Requirements 系统需求 5](#_Toc54423665)

[2.2.1. Servlet Containers Servlet容器 6](#_Toc54423666)

[2.3. Installing Spring Boot Spring Boot安装 6](#_Toc54423667)

[2.3.1. Installation Instructions for the Java Developer 6](#_Toc54423668)

[2.3.2. Installing the Spring Boot CLI 安装Spring Boot CLI 8](#_Toc54423669)

[2.3.3. Upgrading from an Earlier Version of Spring Boot 从Spring Boot的早期版本升级 11](#_Toc54423670)

[2.4. Developing Your First Spring Boot Application 开发您的第一个Spring Boot应用程序 12](#_Toc54423671)

[2.4.1. Creating the POM 创建POM 13](#_Toc54423672)

[2.4.2. Adding Classpath Dependencies  添加环境变量依赖 14](#_Toc54423673)

[2.4.3. Writing the Code 编写代码 15](#_Toc54423674)

[2.4.4. Running the Example 17](#_Toc54423675)

[2.4.5. Creating an Executable Jar 17](#_Toc54423676)

[2.5. What to Read Next 20](#_Toc54423677)

[3.1. Build Systems 22](#_Toc54423678)

[3.1.1. Dependency Management 22](#_Toc54423679)

[3.1.2. Maven 22](#_Toc54423680)

[3.1.3. Gradle 22](#_Toc54423681)

[3.1.4. Ant 23](#_Toc54423682)

[3.1.5. Starters 24](#_Toc54423683)

[3.2. Structuring Your Code 28](#_Toc54423684)

[3.2.1. Using the “default” Package 28](#_Toc54423685)

[3.2.2. Locating the Main Application Class 28](#_Toc54423686)

[3.3. Configuration Classes 29](#_Toc54423687)

[3.3.1. Importing Additional Configuration Classes 30](#_Toc54423688)

[3.3.2. Importing XML Configuration 30](#_Toc54423689)

[3.4. Auto-configuration 30](#_Toc54423690)

[3.4.1. Gradually Replacing Auto-configuration 30](#_Toc54423691)

[3.4.2. Disabling Specific Auto-configuration Classes 30](#_Toc54423692)

[3.5. Spring Beans and Dependency Injection 31](#_Toc54423693)

[3.6. Using the @SpringBootApplication Annotation 32](#_Toc54423694)

[3.7. Running Your Application 34](#_Toc54423695)

[3.7.1. Running from an IDE 34](#_Toc54423696)

[3.7.2. Running as a Packaged Application 34](#_Toc54423697)

[3.7.3. Using the Maven Plugin 34](#_Toc54423698)

[3.7.4. Using the Gradle Plugin 35](#_Toc54423699)

[3.7.5. Hot Swapping 35](#_Toc54423700)

[3.8. Developer Tools 35](#_Toc54423701)

[3.8.1. Property Defaults 36](#_Toc54423702)

[3.8.2. Automatic Restart 37](#_Toc54423703)

[3.8.3. LiveReload 40](#_Toc54423704)

[3.8.4. Global Settings 41](#_Toc54423705)

[3.8.5. Remote Applications 42](#_Toc54423706)

[3.9. Packaging Your Application for Production 44](#_Toc54423707)

[3.10. What to Read Next 44](#_Toc54423708)

[4.1. SpringApplication 45](#_Toc54423709)

[4.1.1. Startup Failure 46](#_Toc54423710)

[4.1.2. Lazy Initialization 46](#_Toc54423711)

[4.1.3. Customizing the Banner 47](#_Toc54423712)

[4.1.4. Customizing SpringApplication 48](#_Toc54423713)

[4.1.5. Fluent Builder API 48](#_Toc54423714)

[4.1.6. Application Availability 49](#_Toc54423715)

[4.1.7. Application Events and Listeners 51](#_Toc54423716)

[4.1.8. Web Environment 53](#_Toc54423717)

[4.1.9. Accessing Application Arguments 53](#_Toc54423718)

[4.1.10. Using the ApplicationRunner or CommandLineRunner 54](#_Toc54423719)

[4.1.11. Application Exit 54](#_Toc54423720)

[4.1.12. Admin Features 55](#_Toc54423721)

[4.2. Externalized Configuration 55](#_Toc54423722)

[4.2.1. Configuring Random Values 58](#_Toc54423723)

[4.2.2. Accessing Command Line Properties 58](#_Toc54423724)

[4.2.3. Application Property Files 58](#_Toc54423725)

[4.2.4. Profile-specific Properties 60](#_Toc54423726)

[4.2.5. Placeholders in Properties 60](#_Toc54423727)

[4.2.6. Encrypting Properties 61](#_Toc54423728)

[4.2.7. Using YAML Instead of Properties 61](#_Toc54423729)

[4.2.8. Type-safe Configuration Properties 64](#_Toc54423730)

[4.3. Profiles 81](#_Toc54423731)

[4.3.1. Adding Active Profiles 82](#_Toc54423732)

[4.3.2. Programmatically Setting Profiles 82](#_Toc54423733)

[4.3.3. Profile-specific Configuration Files 82](#_Toc54423734)

[4.4. Logging 82](#_Toc54423735)

[4.4.1. Log Format 83](#_Toc54423736)

[4.4.2. Console Output 84](#_Toc54423737)

[4.4.3. File Output 85](#_Toc54423738)

[4.4.4. Log Levels 85](#_Toc54423739)

[4.4.5. Log Groups 86](#_Toc54423740)

[4.4.6. Custom Log Configuration 86](#_Toc54423741)

[4.4.7. Logback Extensions 89](#_Toc54423742)

[4.5. Internationalization 90](#_Toc54423743)

[4.6. JSON 91](#_Toc54423744)

[4.6.1. Jackson 91](#_Toc54423745)

[4.6.2. Gson 91](#_Toc54423746)

[4.6.3. JSON-B 91](#_Toc54423747)

[4.7. Developing Web Applications 91](#_Toc54423748)

[4.7.1. The “Spring Web MVC Framework” 92](#_Toc54423749)

[4.7.2. The “Spring WebFlux Framework” 102](#_Toc54423750)

[4.7.3. JAX-RS and Jersey 108](#_Toc54423751)

[4.7.4. Embedded Servlet Container Support 110](#_Toc54423752)

[4.7.5. Embedded Reactive Server Support 113](#_Toc54423753)

[4.7.6. Reactive Server Resources Configuration 113](#_Toc54423754)

[4.8. Graceful shutdown 113](#_Toc54423755)

[4.9. RSocket 114](#_Toc54423756)

[4.9.1. RSocket Strategies Auto-configuration 114](#_Toc54423757)

[4.9.2. RSocket server Auto-configuration 114](#_Toc54423758)

[4.9.3. Spring Messaging RSocket support 115](#_Toc54423759)

[4.9.4. Calling RSocket Services with RSocketRequester 115](#_Toc54423760)

[4.10. Security 116](#_Toc54423761)

[4.10.1. MVC Security 117](#_Toc54423762)

[4.10.2. WebFlux Security 117](#_Toc54423763)

[4.10.3. OAuth2 118](#_Toc54423764)

[4.10.4. SAML 2.0 121](#_Toc54423765)

[4.10.5. Actuator Security 122](#_Toc54423766)

[4.11. Working with SQL Databases 123](#_Toc54423767)

[4.11.1. Configure a DataSource 123](#_Toc54423768)

[4.11.2. Using JdbcTemplate 125](#_Toc54423769)

[4.11.3. JPA and Spring Data JPA 126](#_Toc54423770)

[4.11.4. Spring Data JDBC 129](#_Toc54423771)

[4.11.5. Using H2’s Web Console 129](#_Toc54423772)

[4.11.6. Using jOOQ 130](#_Toc54423773)

[4.11.7. Using R2DBC 132](#_Toc54423774)

[4.12. Working with NoSQL Technologies 135](#_Toc54423775)

[4.12.1. Redis 135](#_Toc54423776)

[4.12.2. MongoDB 136](#_Toc54423777)

[4.12.3. Neo4j 139](#_Toc54423778)

[4.12.4. Solr 142](#_Toc54423779)

[4.12.5. Elasticsearch 142](#_Toc54423780)

[4.12.6. Cassandra 145](#_Toc54423781)

[4.12.7. Couchbase 146](#_Toc54423782)

[4.12.8. LDAP 148](#_Toc54423783)

[4.12.9. InfluxDB 150](#_Toc54423784)

[4.13. Caching 150](#_Toc54423785)

[4.13.1. Supported Cache Providers 151](#_Toc54423786)

[4.14. Messaging 156](#_Toc54423787)

[4.14.1. JMS 156](#_Toc54423788)

[4.14.2. AMQP 161](#_Toc54423789)

[4.14.3. Apache Kafka Support 164](#_Toc54423790)

[4.15. Calling REST Services with RestTemplate 168](#_Toc54423791)

[4.15.1. RestTemplate Customization 169](#_Toc54423792)

[4.16. Calling REST Services with WebClient 169](#_Toc54423793)

[4.16.1. WebClient Runtime 170](#_Toc54423794)

[4.16.2. WebClient Customization 171](#_Toc54423795)

[4.17. Validation 171](#_Toc54423796)

[4.18. Sending Email 171](#_Toc54423797)

[4.19. Distributed Transactions with JTA 172](#_Toc54423798)

[4.19.1. Using an Atomikos Transaction Manager 172](#_Toc54423799)

[4.19.2. Using a Bitronix Transaction Manager 173](#_Toc54423800)

[4.19.3. Using a Java EE Managed Transaction Manager 173](#_Toc54423801)

[4.19.4. Mixing XA and Non-XA JMS Connections 173](#_Toc54423802)

[4.19.5. Supporting an Alternative Embedded Transaction Manager 174](#_Toc54423803)

[4.20. Hazelcast 174](#_Toc54423804)

[4.21. Quartz Scheduler 175](#_Toc54423805)

[4.22. Task Execution and Scheduling 176](#_Toc54423806)

[4.23. Spring Integration 177](#_Toc54423807)

[4.24. Spring Session 178](#_Toc54423808)

[4.25. Monitoring and Management over JMX 179](#_Toc54423809)

[4.26. Testing 179](#_Toc54423810)

[4.26.1. Test Scope Dependencies 179](#_Toc54423811)

[4.26.2. Testing Spring Applications 180](#_Toc54423812)

[4.26.3. Testing Spring Boot Applications 180](#_Toc54423813)

[4.26.4. Test Utilities 209](#_Toc54423814)

[4.27. WebSockets 211](#_Toc54423815)

[4.28. Web Services 212](#_Toc54423816)

[4.28.1. Calling Web Services with WebServiceTemplate 212](#_Toc54423817)

[4.29. Creating Your Own Auto-configuration 213](#_Toc54423818)

[4.29.1. Understanding Auto-configured Beans 213](#_Toc54423819)

[4.29.2. Locating Auto-configuration Candidates 213](#_Toc54423820)

[4.29.3. Condition Annotations 214](#_Toc54423821)

[4.29.4. Testing your Auto-configuration 216](#_Toc54423822)

[4.29.5. Creating Your Own Starter 218](#_Toc54423823)

[4.30. Kotlin support 222](#_Toc54423824)

[4.30.1. Requirements 222](#_Toc54423825)

[4.30.2. Null-safety 222](#_Toc54423826)

[4.30.3. Kotlin API 223](#_Toc54423827)

[4.30.4. Dependency management 223](#_Toc54423828)

[4.30.5. @ConfigurationProperties 224](#_Toc54423829)

[4.30.6. Testing 224](#_Toc54423830)

[4.30.7. Resources 225](#_Toc54423831)

[4.31. Building Container Images 225](#_Toc54423832)

[4.31.1. Building Docker images 225](#_Toc54423833)

[4.31.2. Buildpacks 228](#_Toc54423834)

[4.32. What to Read Next 228](#_Toc54423835)

[5.1. Enabling Production-ready Features 229](#_Toc54423836)

[5.2. Endpoints 229](#_Toc54423837)

[5.2.1. Enabling Endpoints 231](#_Toc54423838)

[5.2.2. Exposing Endpoints 231](#_Toc54423839)

[5.2.3. Securing HTTP Endpoints 233](#_Toc54423840)

[5.2.4. Configuring Endpoints 234](#_Toc54423841)

[5.2.5. Hypermedia for Actuator Web Endpoints 234](#_Toc54423842)

[5.2.6. CORS Support 235](#_Toc54423843)

[5.2.7. Implementing Custom Endpoints 235](#_Toc54423844)

[5.2.8. Health Information 238](#_Toc54423845)

[5.2.9. Kubernetes Probes 244](#_Toc54423846)

[5.2.10. Application Information 246](#_Toc54423847)

[5.3. Monitoring and Management over HTTP 248](#_Toc54423848)

[5.3.1. Customizing the Management Endpoint Paths 248](#_Toc54423849)

[5.3.2. Customizing the Management Server Port 249](#_Toc54423850)

[5.3.3. Configuring Management-specific SSL 249](#_Toc54423851)

[5.3.4. Customizing the Management Server Address 250](#_Toc54423852)

[5.3.5. Disabling HTTP Endpoints 250](#_Toc54423853)

[5.4. Monitoring and Management over JMX 251](#_Toc54423854)

[5.4.1. Customizing MBean Names 251](#_Toc54423855)

[5.4.2. Disabling JMX Endpoints 251](#_Toc54423856)

[5.4.3. Using Jolokia for JMX over HTTP 251](#_Toc54423857)

[5.5. Loggers 252](#_Toc54423858)

[5.5.1. Configure a Logger 252](#_Toc54423859)

[5.6. Metrics 253](#_Toc54423860)

[5.6.1. Getting started 253](#_Toc54423861)

[5.6.2. Supported monitoring systems 254](#_Toc54423862)

[5.6.3. Supported Metrics 260](#_Toc54423863)

[5.6.4. Registering custom metrics 265](#_Toc54423864)

[5.6.5. Customizing individual metrics 266](#_Toc54423865)

[5.6.6. Metrics endpoint 267](#_Toc54423866)

[5.7. Auditing 268](#_Toc54423867)

[5.7.1. Custom Auditing 268](#_Toc54423868)

[5.8. HTTP Tracing 268](#_Toc54423869)

[5.8.1. Custom HTTP tracing 268](#_Toc54423870)

[5.9. Process Monitoring 269](#_Toc54423871)

[5.9.1. Extending Configuration 269](#_Toc54423872)

[5.9.2. Programmatically 269](#_Toc54423873)

[5.10. Cloud Foundry Support 269](#_Toc54423874)

[5.10.1. Disabling Extended Cloud Foundry Actuator Support 269](#_Toc54423875)

[5.10.2. Cloud Foundry Self-signed Certificates 269](#_Toc54423876)

[5.10.3. Custom context path 269](#_Toc54423877)

[5.11. What to Read Next 269](#_Toc54423878)

[6.1. Deploying to Containers 269](#_Toc54423879)

[6.2. Deploying to the Cloud 269](#_Toc54423880)

[6.2.1. Cloud Foundry 269](#_Toc54423881)

[6.2.2. Kubernetes 269](#_Toc54423882)

[6.2.3. Heroku 269](#_Toc54423883)

[6.2.4. OpenShift 269](#_Toc54423884)

[6.2.5. Amazon Web Services (AWS) 269](#_Toc54423885)

[6.2.6. Boxfuse and Amazon Web Services 269](#_Toc54423886)

[6.2.7. Google Cloud 269](#_Toc54423887)

[6.3. Installing Spring Boot Applications 269](#_Toc54423888)

[6.3.1. Supported Operating Systems 269](#_Toc54423889)

[6.3.2. Unix/Linux Services 269](#_Toc54423890)

[6.3.3. Microsoft Windows Services 269](#_Toc54423891)

[6.4. What to Read Next 269](#_Toc54423892)

[7.1. Installing the CLI 269](#_Toc54423893)

[7.2. Using the CLI 269](#_Toc54423894)

[7.2.1. Running Applications with the CLI 269](#_Toc54423895)

[7.2.2. Applications with Multiple Source Files 269](#_Toc54423896)

[7.2.3. Packaging Your Application 269](#_Toc54423897)

[7.2.4. Initialize a New Project 269](#_Toc54423898)

[7.2.5. Using the Embedded Shell 269](#_Toc54423899)

[7.2.6. Adding Extensions to the CLI 269](#_Toc54423900)

[7.3. Developing Applications with the Groovy Beans DSL 269](#_Toc54423901)

[7.4. Configuring the CLI with settings.xml 269](#_Toc54423902)

[7.5. What to Read Next 269](#_Toc54423903)

[8.1. Spring Boot Maven Plugin 269](#_Toc54423904)

[8.2. Spring Boot Gradle Plugin 269](#_Toc54423905)

[8.3. Spring Boot AntLib Module 269](#_Toc54423906)

[8.3.1. Spring Boot Ant Tasks 269](#_Toc54423907)

[8.3.2. Using the “findmainclass” Task 269](#_Toc54423908)

[8.4. Supporting Other Build Systems 269](#_Toc54423909)

[8.4.1. Repackaging Archives 269](#_Toc54423910)

[8.4.2. Nested Libraries 269](#_Toc54423911)

[8.4.3. Finding a Main Class 269](#_Toc54423912)

[8.4.4. Example Repackage Implementation 269](#_Toc54423913)

[8.5. What to Read Next 269](#_Toc54423914)

[9.1. Spring Boot Application 269](#_Toc54423915)

[9.1.1. Create Your Own FailureAnalyzer 269](#_Toc54423916)

[9.1.2. Troubleshoot Auto-configuration 269](#_Toc54423917)

[9.1.3. Customize the Environment or ApplicationContext Before It Starts 269](#_Toc54423918)

[9.1.4. Build an ApplicationContext Hierarchy (Adding a Parent or Root Context) 269](#_Toc54423919)

[9.1.5. Create a Non-web Application 269](#_Toc54423920)

[9.2. Properties and Configuration 269](#_Toc54423921)

[9.2.1. Automatically Expand Properties at Build Time 269](#_Toc54423922)

[9.2.2. Externalize the Configuration of SpringApplication 269](#_Toc54423923)

[9.2.3. Change the Location of External Properties of an Application 269](#_Toc54423924)

[9.2.4. Use ‘Short’ Command Line Arguments 269](#_Toc54423925)

[9.2.5. Use YAML for External Properties 269](#_Toc54423926)

[9.2.6. Set the Active Spring Profiles 269](#_Toc54423927)

[9.2.7. Change Configuration Depending on the Environment 269](#_Toc54423928)

[9.2.8. Discover Built-in Options for External Properties 269](#_Toc54423929)

[9.3. Embedded Web Servers 269](#_Toc54423930)

[9.3.1. Use Another Web Server 269](#_Toc54423931)

[9.3.2. Disabling the Web Server 269](#_Toc54423932)

[9.3.3. Change the HTTP Port 269](#_Toc54423933)

[9.3.4. Use a Random Unassigned HTTP Port 269](#_Toc54423934)

[9.3.5. Discover the HTTP Port at Runtime 269](#_Toc54423935)

[9.3.6. Enable HTTP Response Compression 269](#_Toc54423936)

[9.3.7. Configure SSL 269](#_Toc54423937)

[9.3.8. Configure HTTP/2 269](#_Toc54423938)

[9.3.9. Configure the Web Server 269](#_Toc54423939)

[9.3.10. Add a Servlet, Filter, or Listener to an Application 269](#_Toc54423940)

[9.3.11. Configure Access Logging 269](#_Toc54423941)

[9.3.12. Running Behind a Front-end Proxy Server 269](#_Toc54423942)

[9.3.13. Enable Multiple Connectors with Tomcat 269](#_Toc54423943)

[9.3.14. Use Tomcat’s LegacyCookieProcessor 269](#_Toc54423944)

[9.3.15. Enable Tomcat’s MBean Registry 269](#_Toc54423945)

[9.3.16. Enable Multiple Listeners with Undertow 269](#_Toc54423946)

[9.3.17. Create WebSocket Endpoints Using @ServerEndpoint 269](#_Toc54423947)

[9.4. Spring MVC 269](#_Toc54423948)

[9.4.1. Write a JSON REST Service 269](#_Toc54423949)

[9.4.2. Write an XML REST Service 269](#_Toc54423950)

[9.4.3. Customize the Jackson ObjectMapper 269](#_Toc54423951)

[9.4.4. Customize the @ResponseBody Rendering 269](#_Toc54423952)

[9.4.5. Handling Multipart File Uploads 269](#_Toc54423953)

[9.4.6. Switch Off the Spring MVC DispatcherServlet 269](#_Toc54423954)

[9.4.7. Switch off the Default MVC Configuration 269](#_Toc54423955)

[9.4.8. Customize ViewResolvers 269](#_Toc54423956)

[9.5. Testing With Spring Security 269](#_Toc54423957)

[9.6. Jersey 269](#_Toc54423958)

[9.6.1. Secure Jersey endpoints with Spring Security 269](#_Toc54423959)

[9.6.2. Use Jersey Alongside Another Web Framework 269](#_Toc54423960)

[9.7. HTTP Clients 269](#_Toc54423961)

[9.7.1. Configure RestTemplate to Use a Proxy 269](#_Toc54423962)

[9.7.2. Configure the TcpClient used by a Reactor Netty-based WebClient 269](#_Toc54423963)

[9.8. Logging 269](#_Toc54423964)

[9.8.1. Configure Logback for Logging 269](#_Toc54423965)

[9.8.2. Configure Log4j for Logging 269](#_Toc54423966)

[9.9. Data Access 269](#_Toc54423967)

[9.9.1. Configure a Custom DataSource 269](#_Toc54423968)

[9.9.2. Configure Two DataSources 269](#_Toc54423969)

[9.9.3. Use Spring Data Repositories 269](#_Toc54423970)

[9.9.4. Separate @Entity Definitions from Spring Configuration 269](#_Toc54423971)

[9.9.5. Configure JPA Properties 269](#_Toc54423972)

[9.9.6. Configure Hibernate Naming Strategy 269](#_Toc54423973)

[9.9.7. Configure Hibernate Second-Level Caching 269](#_Toc54423974)

[9.9.8. Use Dependency Injection in Hibernate Components 269](#_Toc54423975)

[9.9.9. Use a Custom EntityManagerFactory 269](#_Toc54423976)

[9.9.10. Use Two EntityManagers 269](#_Toc54423977)

[9.9.11. Use a Traditional persistence.xml File 269](#_Toc54423978)

[9.9.12. Use Spring Data JPA and Mongo Repositories 269](#_Toc54423979)

[9.9.13. Customize Spring Data’s Web Support 269](#_Toc54423980)

[9.9.14. Expose Spring Data Repositories as REST Endpoint 269](#_Toc54423981)

[9.9.15. Configure a Component that is Used by JPA 269](#_Toc54423982)

[9.9.16. Configure jOOQ with Two DataSources 269](#_Toc54423983)

[9.10. Database Initialization 269](#_Toc54423984)

[9.10.1. Initialize a Database Using JPA 269](#_Toc54423985)

[9.10.2. Initialize a Database Using Hibernate 269](#_Toc54423986)

[9.10.3. Initialize a Database using basic SQL scripts 269](#_Toc54423987)

[9.10.4. Initialize a Database Using R2DBC 269](#_Toc54423988)

[9.10.5. Initialize a Spring Batch Database 269](#_Toc54423989)

[9.10.6. Use a Higher-level Database Migration Tool 269](#_Toc54423990)

[9.11. Messaging 269](#_Toc54423991)

[9.11.1. Disable Transacted JMS Session 269](#_Toc54423992)

[9.12. Batch Applications 269](#_Toc54423993)

[9.12.1. Specifying a Batch Data Source 269](#_Toc54423994)

[9.12.2. Running Spring Batch Jobs on Startup 269](#_Toc54423995)

[9.12.3. Running from the Command Line 269](#_Toc54423996)

[9.12.4. Storing the Job Repository 269](#_Toc54423997)

[9.13. Actuator 269](#_Toc54423998)

[9.13.1. Change the HTTP Port or Address of the Actuator Endpoints 269](#_Toc54423999)

[9.13.2. Customize the ‘whitelabel’ Error Page 269](#_Toc54424000)

[9.13.3. Sanitize Sensitive Values 269](#_Toc54424001)

[9.13.4. Map Health Indicators to Micrometer Metrics 269](#_Toc54424002)

[9.14. Security 269](#_Toc54424003)

[9.14.1. Switch off the Spring Boot Security Configuration 269](#_Toc54424004)

[9.14.2. Change the UserDetailsService and Add User Accounts 269](#_Toc54424005)

[9.14.3. Enable HTTPS When Running behind a Proxy Server 269](#_Toc54424006)

[9.15. Hot Swapping 269](#_Toc54424007)

[9.15.1. Reload Static Content 269](#_Toc54424008)

[9.15.2. Reload Templates without Restarting the Container 269](#_Toc54424009)

[9.15.3. Fast Application Restarts 269](#_Toc54424010)

[9.15.4. Reload Java Classes without Restarting the Container 269](#_Toc54424011)

[9.16. Build 269](#_Toc54424012)

[9.16.1. Generate Build Information 269](#_Toc54424013)

[9.16.2. Generate Git Information 269](#_Toc54424014)

[9.16.3. Customize Dependency Versions 269](#_Toc54424015)

[9.16.4. Create an Executable JAR with Maven 269](#_Toc54424016)

[9.16.5. Use a Spring Boot Application as a Dependency 269](#_Toc54424017)

[9.16.6. Extract Specific Libraries When an Executable Jar Runs 269](#_Toc54424018)

[9.16.7. Create a Non-executable JAR with Exclusions 269](#_Toc54424019)

[9.16.8. Remote Debug a Spring Boot Application Started with Maven 269](#_Toc54424020)

[9.16.9. Build an Executable Archive from Ant without Using spring-boot- antlib 269](#_Toc54424021)

[9.17. Traditional Deployment 269](#_Toc54424022)

[9.17.1. Create a Deployable War File 269](#_Toc54424023)

[9.17.2. Convert an Existing Application to Spring Boot 269](#_Toc54424024)

[9.17.3. Deploying a WAR to WebLogic 269](#_Toc54424025)

[9.17.4. Use Jedis Instead of Lettuce 269](#_Toc54424026)

[9.17.5. Use Testcontainers for integration testing 269](#_Toc54424027)

[Appendix A: Common Application properties 269](#_Toc54424028)

[10.A.1. Core properties 269](#_Toc54424029)

[10.A.2. Cache properties 269](#_Toc54424030)

[10.A.3. Mail properties 269](#_Toc54424031)

[10.A.4. JSON properties 269](#_Toc54424032)

[10.A.5. Data properties 269](#_Toc54424033)

[10.A.6. Transaction properties 269](#_Toc54424034)

[10.A.7. Data migration properties 269](#_Toc54424035)

[10.A.8. Integration properties 269](#_Toc54424036)

[10.A.9. Web properties 269](#_Toc54424037)

[10.A.10. Templating properties 269](#_Toc54424038)

[10.A.11. Server properties 269](#_Toc54424039)

[10.A.12. Security properties 269](#_Toc54424040)

[10.A.13. RSocket properties 269](#_Toc54424041)

[10.A.14. Actuator properties 269](#_Toc54424042)

[10.A.15. Devtools properties 269](#_Toc54424043)

[10.A.16. Testing properties 269](#_Toc54424044)

[Appendix B: Configuration Metadata 269](#_Toc54424045)

[10.B.1. Metadata Format 269](#_Toc54424046)

[10.B.2. Providing Manual Hints 269](#_Toc54424047)

[10.B.3. Generating Your Own Metadata by Using the Annotation Processor 269](#_Toc54424048)

[Appendix C: Auto-configuration Classes 269](#_Toc54424049)

[10.C.1. spring-boot-autoconfigure 269](#_Toc54424050)

[10.C.2. spring-boot-actuator-autoconfigure 269](#_Toc54424051)

[Appendix D: Test Auto-configuration Annotations 269](#_Toc54424052)

[10.D.1. Test Slices 269](#_Toc54424053)

[Appendix E: The Executable Jar Format 269](#_Toc54424054)

[10.E.1. Nested JARs 269](#_Toc54424055)

[10.E.2. Spring Boot’s “JarFile” Class 269](#_Toc54424056)

[10.E.3. Launching Executable Jars 269](#_Toc54424057)

[10.E.4. PropertiesLauncher Features 269](#_Toc54424058)

[10.E.5. Executable Jar Restrictions 269](#_Toc54424059)

[10.E.6. Alternative Single Jar Solutions 269](#_Toc54424060)

[Appendix F: Dependency versions 269](#_Toc54424061)

[10.F.1. Managed Dependency Coordinates 269](#_Toc54424062)

[10.F.2. Version Properties 269](#_Toc54424063)

# Legal

2.3.4.RELEASE

Copyright © 2012-2020

Copies of this document may be made for your own use and for distribution to others, provided that you do not charge any fee for such copies and further provided that each copy contains this Copyright Notice, whether distributed in print or electronically.

# 法律声明

2.3.4. 发布版本

版权所有©2012-2020

本文档的副本可以供您自己使用，也可以分发给他人，但前提是您不对此类副本收取任何费用，并且还应确保每份副本均包含本版权声明（无论是印刷版本还是电子版本）。

1. **Spring Boot Documentation Spring Boot 文献**

This section provides a brief overview of Spring Boot reference documentation. It serves as a map for the rest of the document.

本节简要概述了springboot参考文档。它作为文档其余部分的映射。

## About the Documentation 有关文献

The Spring Boot reference guide is available as:

Spring Boot参考指南可通过以下方式获得：

* + - [Multi-page HTML](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/reference/html/) 多页HTML
    - [Single page HTML](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/reference/htmlsingle/) 单页HTML
    - [PDF](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/reference/pdf/spring-boot-reference.pdf)

The latest copy is available at [docs.spring.io/spring-boot/docs/current/reference/](https://docs.spring.io/spring-boot/docs/current/reference/).

最新的版本可以在 docs.spring.io/spring-boot/docs/current/reference/网站上找到。

Copies of this document may be made for your own use and for distribution to others, provided that you do not charge any fee for such copies and further provided that each copy contains this Copyright Notice, whether distributed in print or electronically.

本文档的副本可以供您自己使用，也可以分发给他人，但前提是您不对此类副本收取任何费用，并且还应确保每份副本均包含本版权声明（无论是印刷版本还是电子版本）。

## Getting Help 获得帮助

If you have trouble with Spring Boot, we would like to help.

如果您在使用Spring Boot时遇到问题，我们将为您提供帮助。

* + - Try the [How-to documents](#_bookmark525). They provide solutions to the most common questions.
    - 尝试使用操作文档。它们为最常见的问题提供了解决方案。
    - Learn the Spring basics. Spring Boot builds on many other Spring projects. Check the [spring.io](https://spring.io/) web-site for a wealth of reference documentation. If you are starting out with Spring, try one of the [guides](https://spring.io/guides).
    - 学习 Spring 的基础知识。Spring Boot 基于许多其他 Spring 项目。在 spring.io 网站上查找大量的参考文档。如果你是从 Spring 开始的，试试其中一个指南。
    - Ask a question. We monitor [stackoverflow.com](https://stackoverflow.com/) for questions tagged with [spring-boot](https://stackoverflow.com/tags/spring-boot).
    - 问一个问题。 我们会监视stackoverflow.com，以查看带有spring-boot标签的问题。
    - Report bugs with Spring Boot at [github.com/spring-projects/spring-boot/issues](https://github.com/spring-projects/spring-boot/issues).
    - 通过github.com/spring-projects/spring-boot/issues报告Spring Boot的bug。

All of Spring Boot is open source, including the documentation. If you find problems with the docs or if you want to improve them, please [get involved](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE).

Spring Boot 的所有内容都是开源的，包括文档。如果你发现文档有问题或者你想改进它们，请参与进来。

## First Steps 第一步

If you are getting started with Spring Boot or 'Spring' in general, start with [the following topics](#_bookmark9):

如果你刚开始使用 Spring Boot 或者‘ Spring’ ，可以从以下主题开始:

* + - **From scratch:** [Overview](#_bookmark10) | [Requirements](#_bookmark11) | [Installation](#_bookmark13)
    - 从头开始: 概述 | 需求 | 安装
    - **Tutorial:** [Part 1](#_bookmark26) | [Part 2](#_bookmark29)
    - 教程: 第一部分 | 第二部分
    - **Running your example:** [Part 1](#_bookmark33) | [Part 2](#_bookmark34)
    - 运行示例: 第一部分 | 第二部分

## Working with Spring Boot 使用 Spring Boot

Ready to actually start using Spring Boot? [We have you covered](#_bookmark36):

准备好真正开始使用 Spring Boot 了吗?我们为您提供：

* + - **Build systems:** [Maven](#_bookmark39) | [Gradle](#_bookmark40) | [Ant](#_bookmark41) | [Starters](#_bookmark42)
    - 构建系统：[Maven](#_bookmark39) | [Gradle](#_bookmark40) | [Ant](#_bookmark41) | [Starters](#_bookmark42)
    - **Best practices:** [Code Structure](#_bookmark48) | [@Configuration](#_bookmark51) | [@EnableAutoConfiguration](#_bookmark54) | [Beans and](#_bookmark57) [Dependency Injection](#_bookmark57)
    - 最佳实践：[Code Structure](#_bookmark48) | [@Configuration](#_bookmark51) | [@EnableAutoConfiguration](#_bookmark54) | [Beans and](#_bookmark57) [Dependency Injection](#_bookmark57)
    - **Running your code:** [IDE](#_bookmark60) | [Packaged](#_bookmark61) | [Maven](#_bookmark62) | [Gradle](#_bookmark63)
    - 代码运行：[IDE](#_bookmark60) | [Packaged](#_bookmark61) | [Maven](#_bookmark62) | [Gradle](#_bookmark63)
    - **Packaging your app:** [Production jars](#_bookmark82)
    - 应用程序打包：[Production jars](#_bookmark82)
    - **Spring Boot CLI:** [Using the CLI](#_bookmark493)
    - Spring Boot CLI: [Using the CLI](#_bookmark493)

## Learning about Spring Boot Features 学习 Spring Boot 的特性

Need more details about Spring Boot’s core features? [The following content is for you](#_bookmark84):

需要更多关于Spring Boot核心特性的详细信息吗？以下内容为您提供:

* + - **Core Features:** [SpringApplication](#_bookmark85) | [External Configuration](#_bookmark101) | [Profiles](#_bookmark126) | [Logging](#_bookmark130)
    - 核心功能：[SpringApplication](#_bookmark85)  |外部配置|配置文件|日志记录
    - **Web Applications:** [MVC](#_bookmark147) | [Embedded Containers](#_bookmark172)
    - Web应用程序：MVC |嵌入式容器
    - **Working with data:** [SQL](#_bookmark198) | [NO-SQL](#_bookmark221)
    - 处理数据：SQL | NO-SQL
    - **Messaging:** [Overview](#_bookmark267) | [JMS](#_bookmark268)
    - 消息传递: 概述 | JMS
    - **Testing:** [Overview](#_bookmark303) | [Boot Applications](#_bookmark306) | [Utils](#_bookmark334)
    - 测试：概述| 引导应用程序| 实用程序
    - **Extending:** [Auto-configuration](#_bookmark342) | [@Conditions](#_bookmark345)
    - 扩展:自动配置| @Conditions

## Moving to Production 进入生产阶段

When you are ready to push your Spring Boot application to production, we have [some tricks](#_bookmark378) that you might like:

当您准备将您的 Spring Boot 应用程序投入生产时，我们有一些您可能会喜欢的技巧:

* + - **Management endpoints:** [Overview](#_bookmark380)
    - 管理端点：概述
    - **Connection options:** [HTTP](#_bookmark407) | [JMX](#_bookmark413)
    - 连接选项: HTTP | JMX
    - **Monitoring:** [Metrics](#_bookmark421) | [Auditing](#_bookmark457) | [HTTP Tracing](#_bookmark459) | [Process](#_bookmark461)
    - 监控: 度量 | 审计 | HTTP 跟踪 | 流程

## Advanced Topics 进阶主题

Finally, we have a few topics for more advanced users:

最后，我们为进阶用户准备了一些主题:

* + - **Spring Boot Applications Deployment:** [Cloud Deployment](#_bookmark471) | [OS Service](#_bookmark485)
    - Spring Boot应用程序部署：云部署| 操作系统服务
    - **Build tool plugins:** [Maven](#_bookmark511) | [Gradle](#_bookmark512)
    - 构建工具插件：Maven|Gradle。
    - **Appendix:** [Application Properties](#_bookmark656) | [Configuration Metadata](#_bookmark673) | [Auto-configuration Classes](#_bookmark686) | [Test](#_bookmark689) [Auto-configuration Annotations](#_bookmark689) | [Executable Jars](#_bookmark691) | [Dependency Versions](#_bookmark705)
    - 附录: 应用程序属性 | 配置元数据 | 自动配置类 | 测试自动配置注释 | 可执行 jar | 依赖性版本

1. **Getting Started 入门**

If you are getting started with Spring Boot, or “Spring” in general, start by reading this section. It answers the basic “what?”, “how?” and “why?” questions. It includes an introduction to Spring Boot, along with installation instructions. We then walk you through building your first Spring Boot application, discussing some core principles as we go.

如果您是从Spring Boot或“ Spring”开始的，请先阅读本节。 它针对基本的“什么？”，“如何？” 和“为什么？” 问题提出答惑。 包括对Spring Boot的介绍以及安装说明。 然后，我们将引导您构建第一个Spring Boot应用程序，并讨论一些核心原理。

## Introducing Spring Boot Spring Boot 简介

Spring Boot helps you to create stand-alone, production-grade Spring-based Applications that you can run. We take an opinionated view of the Spring platform and third-party libraries, so that you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration.

Spring Boot帮助您创建可以独立的运行的、生产级的基于Spring的应用程序。我们对Spring平台和第三方库有自己的看法，这样您就可以从零开始。大多数Spring引导应用程序只需要很少的Spring配置。

You can use Spring Boot to create Java applications that can be started by using java -jar or more traditional war deployments. We also provide a command line tool that runs “spring scripts”.

您可以使用Spring Boot创建可以通过使用Java -jar或更传统的war部署启动的Java应用程序。我们还提供了一个运行“spring脚本”的命令行工具。

Our primary goals are:

* + - Provide a radically faster and widely accessible getting-started experience for all Spring development.
    - Be opinionated out of the box but get out of the way quickly as requirements start to diverge from the defaults.
    - Provide a range of non-functional features that are common to large classes of projects (such as embedded servers, security, metrics, health checks, and externalized configuration).
    - Absolutely no code generation and no requirement for XML configuration.

我们的主要目标是：

•为所有Spring开发提供彻底更快、可广泛访问的入门体验。

•跳出框框固执己见，但当需求开始偏离默认值时，要迅速闪开。

•提供一系列对大型项目常见的非功能特性(如嵌入式服务器、安全性、度量、运行状况检查和外部化配置)。

•绝对不需要代码生成，也不需要XML配置。

## System Requirements 系统需求

Spring Boot 2.3.4.RELEASE requires [Java 8](https://www.java.com/) and is compatible up to Java 14 (included). [Spring](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/) [Framework 5.2.9.RELEASE](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/) or above is also required.

Spring Boot 2.3.4.RELEASE需要Java 8支持，并且与Java 14（包括）兼容。 还需要Spring Framework 5.2.9.RELEASE或更高版本。

Explicit build support is provided for the following build tools:

为以下构建工具提供了明确的构建支持：

|  |  |
| --- | --- |
| **Build Tool** | **Version** |
| Maven | 3.3+ |
| Gradle | 6 (6.3 or later). 5.6.x is also supported but in a deprecated form  6(6.3或更高版本) . 5.6. x 也支持，但不推荐使用这个版本 |

### 2.2.1. Servlet Containers Servlet容器

Spring Boot supports the following embedded servlet containers:

Spring Boot支持以下嵌入式servlet容器：

|  |  |
| --- | --- |
| **Name** | **Servlet Version** |
| Tomcat 9.0 | 4.0 |
| Jetty 9.4 | 3.1 |
| Undertow 2.0 | 4.0 |

You can also deploy Spring Boot applications to any Servlet 3.1+ compatible container.

您还可以将Spring Boot应用程序部署到任何Servlet 3.1+兼容容器中。

## Installing Spring Boot Spring Boot安装

Spring Boot can be used with “classic” Java development tools or installed as a command line tool. Either way, you need [Java SDK v1.8](https://www.java.com/) or higher. Before you begin, you should check your current Java installation by using the following command:

Springboot 可以与“经典” Java 开发工具一起使用，也可以使用命令行工具安装。无论哪种方式，您都需要 Java SDK v1.8或更高版本。在开始之前，你应该使用以下命令检查你当前的 Java 安装:

$ java -version

If you are new to Java development or if you want to experiment with Spring Boot, you might want to try the [Spring Boot CLI](#_bookmark17) (Command Line Interface) first. Otherwise, read on for “classic” installation instructions.

如果您不熟悉Java开发，或者想尝试使用Spring Boot，则可能要先尝试使用Spring Boot CLI（命令行界面）。 否则，请继续阅读“经典”安装说明。

### Installation Instructions for the Java Developer 面向Java开发人员的安装说明

You can use Spring Boot in the same way as any standard Java library. To do so, include the appropriate spring-boot-\*.jar files on your classpath. Spring Boot does not require any special tools integration, so you can use any IDE or text editor. Also, there is nothing special about a Spring Boot application, so you can run and debug a Spring Boot application as you would any other Java program.

您可以像使用任何标准 Java 库一样使用 Spring Boot。为此，请在您的classpath里引入相应的 spring-boot-\*.jar 文件。Spring Boot 不需要任何特殊的工具集成，因此您可以使用任何 IDE 或文本编辑器。此外，Spring Boot 应用程序没有什么特别之处，因此您可以像运行其他 Java 程序一样运行和调试 Spring Boot 应用程序。

Although you *could* copy Spring Boot jars, we generally recommend that you use a build tool that supports dependency management (such as Maven or Gradle).

尽管您可以复制Spring Boot jar包，但是我们通常建议您使用支持依赖关系管理的构建工具（例如Maven或Gradle）。就是尽管可以使用jar包来引入，但是通常还是建议用Maven或Gradle。

#### Maven Installation Maven 安装

Spring Boot is compatible with Apache Maven 3.3 or above. If you do not already have Maven installed, you can follow the instructions at [maven.apache.org](https://maven.apache.org/).

Spring Boot兼容Apache Maven3.3或更高版本。 如果您尚未安装Maven，可以按照maven.apache.org上的说明进行操作。

On many operating systems, Maven can be installed with a package manager. If you use OSX Homebrew, try brew install maven. Ubuntu users can run sudo apt- get install maven. Windows users with [Chocolatey](https://chocolatey.org/) can run choco install maven from an elevated (administrator) prompt.

在许多操作系统中，Maven可以与包管理器一起安装。 如果您使用OSX Homebrew，请尝试BREW Install Maven。 Ubuntu用户可以运行sudo apt-get install maven。 使用chocolatey的Windows用户可以从有管理员权限的 chocolatey 来运行以下命令 install maven。

Spring Boot dependencies use the org.springframework.boot groupId. Typically, your Maven POM file inherits from the spring-boot-starter-parent project and declares dependencies to one or more [“Starters”](#_bookmark42). Spring Boot also provides an optional [Maven plugin](#_bookmark511) to create executable jars.

Spring Boot依赖项使用org.springframework.boot groupId。 通常，您的Maven POM文件继承自spring-boot-starter-parent项目，并声明对一个或多个“Starters”的依赖关系。 Spring Boot还提供了一个可选的Maven插件来创建可执行JAR。

More details on getting started with Spring Boot and Maven can be found in the [Getting Started](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/maven-plugin/reference/html/#getting-started) [section](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/maven-plugin/reference/html/#getting-started) of the Maven plugin’s reference guide.

有关使用Spring Boot和Maven入门的更多详细信息，请参见Maven插件参考指南的“入门”部分。

#### Gradle Installation Gradle 安装

Spring Boot is compatible with Gradle 6 (6.3 or later). Gradle 5.6.x is also supported but this support is deprecated and will be removed in a future release. If you do not already have Gradle installed, you can follow the instructions at [gradle.org](https://gradle.org/).

Spring Boot与Gradle 6（6.3或更高版本）兼容。 还支持Gradle 5.6.x，但不建议使用这个版本，Spring Boot在将来的版本中将删除该版本支持。 如果尚未安装Gradle，则可以按照gradle.org上的说明进行操作。

Spring Boot dependencies can be declared by using the org.springframework.boot group. Typically, your project declares dependencies to one or more [“Starters”](#_bookmark42). Spring Boot provides a useful [Gradle](#_bookmark512) [plugin](#_bookmark512) that can be used to simplify dependency declarations and to create executable jars.

可以使用org.springframework.boot group声明Spring Boot依赖项。 通常，您的项目声明对一个或多个“Starters”的依赖。 Spring Boot提供了一个有用的Gradle插件，可用于简化依赖项声明和创建可执行的jar。

**Gradle Wrapper**

The Gradle Wrapper provides a nice way of “obtaining” Gradle when you need to build a project. It is a small script and library that you commit alongside your code to bootstrap the build process. See [docs.gradle.org/current/userguide/gradle\_wrapper.html](https://docs.gradle.org/current/userguide/gradle_wrapper.html) for details.

**Gradle Wrapper**

Gradle Wrapper 在您需要构建项目时提供了一种很好的“获取” Gradle 的方法。它是一个很小的脚本和库，您可以在代码旁边提交它来引导构建过程。详情请参阅 docs.gradle.org/current/userguide/gradle\_wrapper.html。

Gradle Wrapper

Gradle Wrapper 在您需要构建项目时提供了一种很好的“获取” Gradle 的方法。它是一个很小的脚本和库，您可以在代码旁边提交它来引导构建过程。详情请参阅 docs.gradle.org/current/userguide/gradle\_wrapper.html。

More details on getting started with Spring Boot and Gradle can be found in the [Getting Started](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/gradle-plugin/reference/html/#getting-started) [section](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/gradle-plugin/reference/html/#getting-started) of the Gradle plugin’s reference guide.

更多关于Spring Boot和Gradle入门的细节可以在Gradle插件的入门指南中找到。

### Installing the Spring Boot CLI 安装Spring Boot CLI

The Spring Boot CLI (Command Line Interface) is a command line tool that you can use to quickly prototype with Spring. It lets you run [Groovy](https://groovy-lang.org/) scripts, which means that you have a familiar Java- like syntax without so much boilerplate code.

Spring Boot CLI（命令行界面）是一个命令行工具，可用于快速使用Spring进行原型设计。 它使您可以运行Groovy脚本，这意味着您具有类似Java的熟悉语法，而没有太多样板代码。

You do not need to use the CLI to work with Spring Boot, but it is definitely the quickest way to get a Spring application off the ground.

您无需使用CLI即可与Spring Boot一起使用，但这绝对是使Spring应用程序启动的最快方法。

#### Manual Installation 手动安装

You can download the Spring CLI distribution from the Spring software repository:

你可以从 Spring 软件资源库下载 Spring CLI 发行版:

* + - * [spring-boot-cli-2.3.4.RELEASE-bin.zip](https://repo.spring.io/release/org/springframework/boot/spring-boot-cli/2.3.4.RELEASE/spring-boot-cli-2.3.4.RELEASE-bin.zip)
      * [spring-boot-cli-2.3.4.RELEASE-bin.tar.gz](https://repo.spring.io/release/org/springframework/boot/spring-boot-cli/2.3.4.RELEASE/spring-boot-cli-2.3.4.RELEASE-bin.tar.gz)

Cutting edge [snapshot distributions](https://repo.spring.io/snapshot/org/springframework/boot/spring-boot-cli/) are also available.

也可以使用最先进的快照发行版。

Once downloaded, follow the [INSTALL.txt](https://raw.githubusercontent.com/spring-projects/spring-boot/v2.3.4.RELEASE/spring-boot-project/spring-boot-cli/src/main/content/INSTALL.txt) instructions from the unpacked archive. In summary, there is a spring script (spring.bat for Windows) in a bin/ directory in the .zip file. Alternatively, you can use java -jar with the .jar file (the script helps you to be sure that the classpath is set correctly).

下载完成后，请按照解压缩后的存档中的INSTALL.txt说明进行操作。 总而言之，.zip文件的bin /目录中有一个spring脚本（对于Windows是spring.bat）。 或者，您可以将Java -jar与.jar文件一起使用（脚本可帮助您确保正确设置了类路径）。

#### Installation with SDKMAN! 使用SDKMAN安装！

SDKMAN! (The Software Development Kit Manager) can be used for managing multiple versions of various binary SDKs, including Groovy and the Spring Boot CLI. Get SDKMAN! from [sdkman.io](https://sdkman.io/) and install Spring Boot by using the following commands:

SDKMAN！ （软件开发工具包管理器）可用于管理各种二进制SDK的多个版本，包括Groovy和Spring Boot CLI。 从sdkman.io获取SDKMAN！并使用以下命令安装Spring Boot：

$ sdk install springboot

$ spring --version

Spring Boot v2.3.4.RELEASE

If you develop features for the CLI and want access to the version you built, use the following commands:

如果您为CLI开发功能并希望访问所构建的版本，请使用以下命令：

$ sdk install springboot dev /path/to/spring-boot/spring-boot-cli/target/spring-boot- cli-2.3.4.RELEASE-bin/spring-2.3.4.RELEASE/

$ sdk default springboot dev

$ spring --version

Spring CLI v2.3.4.RELEASE

The preceding instructions install a local instance of spring called the dev instance. It points at your target build location, so every time you rebuild Spring Boot, spring is up-to-date.

前面的说明安装了一个名为dev实例的Spring本地实例。 它指向您的目标构建位置，因此每次您重新构建Spring Boot时，Spring都是最新的。

You can see it by running the following command:

你可以通过运行以下命令来查看它:

$ sdk ls springboot

================================================================================

Available Springboot Versions

================================================================================

> + dev

* 2.3.4.RELEASE

================================================================================

+ - local version

* - installed
* - currently in use

================================================================================

#### OSX Homebrew Installation OSX Homebrew安装

If you are on a Mac and use [Homebrew](https://brew.sh/), you can install the Spring Boot CLI by using the following commands:

如果您在Mac上并且使用Homebrew，则可以使用以下命令来安装Spring Boot CLI：

$ brew tap pivotal/tap

$ brew install springboot

Homebrew installs spring to /usr/local/bin.

Homebrew将Spring安装到/usr/local/bin。

If you do not see the formula, your installation of brew might be out-of-date. In that case, run brew update and try again.

如果看不到该公式，则说明brew的安装可能已过期。 在这种情况下，请运行brew update，然后重试。

#### MacPorts Installation MacPorts安装

If you are on a Mac and use [MacPorts](https://www.macports.org/), you can install the Spring Boot CLI by using the following command:

如果您在Mac上并且使用MacPorts，则可以使用以下命令来安装Spring Boot CLI：

$ sudo port install spring-boot-cli

#### Command-line Completion 命令行完成

The Spring Boot CLI includes scripts that provide command completion for the [BASH](https://en.wikipedia.org/wiki/Bash_%28Unix_shell%29) and [zsh](https://en.wikipedia.org/wiki/Z_shell) shells. You can source the script (also named spring) in any shell or put it in your personal or system-wide bash completion initialization. On a Debian system, the system-wide scripts are in /shell- completion/bash and all scripts in that directory are executed when a new shell starts. For example, to run the script manually if you have installed by using SDKMAN!, use the following commands:

Spring Boot CLI 包括为 BASH 和 zsh shell 提供命令完成的脚本。可以在任何 shell 中源代码脚本(也称为 spring) ，或者将其放入个人或系统范围的 bash 完成初始化中。在 Debian 系统上，系统范围的脚本在/shell-completion/bash 中，当新的 shell 启动时，该目录中的所有脚本都将执行。例如，如果您已经使用 SDKMAN 安装了脚本，那么可以手动运行该脚本！，使用以下命令:

$ . ~/.sdkman/candidates/springboot/current/shell-completion/bash/spring

$ spring <HIT TAB HERE>

grab help jar run test version

If you install the Spring Boot CLI by using Homebrew or MacPorts, the command- line completion scripts are automatically registered with your shell.

如果您使用 Homebrew 或 MacPorts 安装 Spring Boot CLI，命令行完成脚本将自动注册到您的 shell 中。

#### Windows Scoop Installation Windows Scoop安装

If you are on a Windows and use [Scoop](https://scoop.sh/), you can install the Spring Boot CLI by using the following commands:

如果你使用的是 Windows 系统并且使用了 Scoop，你可以使用以下命令安装 Spring Boot CLI:

* scoop bucket add extras
* scoop install springboot

Scoop installs spring to ~/scoop/apps/springboot/current/bin.

Scoop将spring安装到〜/ scoop / apps / springboot / current / bin中。

If you do not see the app manifest, your installation of scoop might be out-of-date. In that case, run scoop update and try again.

如果您没有看到应用清单，则可能是因为您安装的Scoop已过期。 在这种情况下，请运行scoop update，然后重试。

#### Quick-start Spring CLI Example 快速入门Spring CLI示例

You can use the following web application to test your installation. To start, create a file called

app.groovy, as follows:

您可以使用以下Web应用程序来测试安装。 首先，创建一个名为app.groovy的文件，如下所示：

@RestController

class ThisWillActuallyRun {

@RequestMapping("/") String home() {

"Hello World!"

}

}

Then run it from a shell, as follows:

然后在一个 shell 里运行它，如下所示:

$ spring run app.groovy

The first run of your application is slow, as dependencies are downloaded. Subsequent runs are much quicker.

应用程序的第一次运行速度很慢，因为要下载依赖项，后续的运行速度要快得多。

Open localhost:8080 in your favorite web browser. You should see the following output:

在您最喜欢的Web浏览器中打开localhost：8080。 您应该会看到以下输出：

Hello World!

### Upgrading from an Earlier Version of Spring Boot 从Spring Boot的早期版本升级

If you are upgrading from the 1.x release of Spring Boot, check the [“migration guide” on the project](https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.0-Migration-Guide) [wiki](https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.0-Migration-Guide) that provides detailed upgrade instructions. Check also the [“release notes”](https://github.com/spring-projects/spring-boot/wiki) for a list of “new and noteworthy” features for each release.

如果您正在从 Spring Boot 的1.x 版本升级，请查看项目 wiki 上的“迁移指南” ，该指南提供了详细的升级说明。还要检查“发行说明” ，以获得每个发行版的“新的和值得注意的”特性列表。

When upgrading to a new feature release, some properties may have been renamed or removed. Spring Boot provides a way to analyze your application’s environment and print diagnostics at startup, but also temporarily migrate properties at runtime for you. To enable that feature, add the following dependency to your project:

当升级到新的特性版本时，一些属性可能已被重命名或删除。Spring Boot 提供了一种在启动时分析应用程序环境和打印诊断信息的方法，而且还可以在运行时为您临时迁移属性。要启用该特性，请在项目中添加以下依赖项:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-properties-migrator</artifactId>

<scope>runtime</scope>

</dependency>

Properties that are added late to the environment, such as when using

@PropertySource, will not be taken into account.

后期添加到环境中的属性，例如在使用@PropertySource，则不会考虑。

Once you’re done with the migration, please make sure to remove this module from your project’s dependencies.

迁移完成后，请确保从项目的依赖项中删除此模块。

To upgrade an existing CLI installation, use the appropriate package manager command (for example, brew upgrade). If you manually installed the CLI, follow the [standard instructions](#_bookmark18), remembering to update your PATH environment variable to remove any older references.

要升级现有CLI安装，请使用相应的软件包管理器命令(例如，BREW升级)。 如果手动安装了CLI，请按照标准说明操作，记住要更新PATH环境变量以删除任何较旧的引用。

## Developing Your First Spring Boot Application 开发您的第一个Spring Boot应用程序

This section describes how to develop a small “Hello World!” web application that highlights some of Spring Boot’s key features. We use Maven to build this project, since most IDEs support it.

本节描述如何开发一个小型“Hello World!”这个web应用程序突出了Spring Boot的一些关键特性。我们使用Maven来构建这个项目，因为大多数ide都支持它。

The [spring.io](https://spring.io/) web site contains many “Getting Started” [guides](https://spring.io/guides) that use Spring Boot. If you need to solve a specific problem, check there first.

spring.io网站包含许多使用Spring Boot的“入门”指南。 如果您需要解决特定问题，请首先检查。

You can shortcut the steps below by going to [start.spring.io](https://start.spring.io/) and choosing the "Web" starter from the dependencies searcher. Doing so generates a new project structure so that you can [start coding right away](#_bookmark29). Check the [Spring Initializr](https://docs.spring.io/initializr/docs/current/reference/html/#user-guide) [documentation](https://docs.spring.io/initializr/docs/current/reference/html/#user-guide) for more details.

您可以通过访问 start.spring.io 并从依赖项/searcher 中选择“ Web” starter 来快捷完成下面的步骤。这样做会生成一个新的项目结构，以便您可以立即开始编码。有关更多详细信息，请查看Spring Initializr文档。

Before we begin, open a terminal and run the following commands to ensure that you have valid versions of Java and Maven installed:

在我们开始之前，打开一个终端并运行以下命令，以确保你已经安装了 Java 和 Maven 的有效版本:

$ java -version

java version "1.8.0\_102"

Java(TM) SE Runtime Environment (build 1.8.0\_102-b14)

Java HotSpot(TM) 64-Bit Server VM (build 25.102-b14, mixed mode)

$ mvn -v

Apache Maven 3.5.4 (1edded0938998edf8bf061f1ceb3cfdeccf443fe; 2018-06-17T14:33:14- 04:00)

Maven home: /usr/local/Cellar/maven/3.3.9/libexec Java version: 1.8.0\_102, vendor: Oracle Corporation

This sample needs to be created in its own directory. Subsequent instructions assume that you have created a suitable directory and that it is your current directory.

他的示例需要在自己的目录中创建。接下来的说明假设您已经创建了一个合适的目录，并且这个目录就是你的工作目录/目录。

### Creating the POM 创建POM

We need to start by creating a Maven pom.xml file. The pom.xml is the recipe that is used to build your project. Open your favorite text editor and add the following:

我们需要先创建一个Maven pom.xml文件。 pom.xml是用于构建项目的配方。 打开您喜欢的文本编辑器并添加以下内容：

<?xml version="1.0" encoding="UTF-8"?>

<project [xmlns="http://maven.apache.org/POM/4.0.0"](http://maven.apache.org/POM/4.0.0) [xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance)

[xsi:schemaLocation="http://maven.apache.org/POM/4.0.0](http://maven.apache.org/POM/4.0.0) https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>myproject</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.3.4.RELEASE</version>

</parent>

<description/>

<developers>

<developer/>

</developers>

<licenses>

<license/>

</licenses>

<scm>

<url/>

</scm>

<url/>

<!-- Additional lines to be added here... -->

</project>

The preceding listing should give you a working build. You can test it by running mvn package (for now, you can ignore the “jar will be empty - no content was marked for inclusion!” warning).

前面的清单应该会为您提供一个有效的构建。 您可以通过运行MVN包来测试它(目前，您可以忽略“JAR将为空-没有内容被标记为包含！” 警告)。

At this point, you could import the project into an IDE (most modern Java IDEs include built-in support for Maven). For simplicity, we continue to use a plain text editor for this example.

至此，您可以将项目导入 IDE (大多数现代 javaide 都包含对 Maven 的内置支持)。为了简单起见，我们继续对本例使用纯文本编辑器。

### Adding Classpath Dependencies  添加环境变量依赖

Spring Boot provides a number of “Starters” that let you add jars to your classpath. Our applications for smoke tests use the spring-boot-starter-parent in the parent section of the POM. The spring-boot-starter-parent is a special starter that provides useful Maven defaults. It also provides a [dependency-management](#_bookmark38) section so that you can omit version tags for “blessed” dependencies.

Springboot 提供了许多“Starters” ，可以将 jar 添加到classpath中。我们的冒烟测试应用程序使用 POM 的父节中的 spring-boot-starter-parent。Spring-boot-starter-parent 是一个特殊的 starter，它提供了有用的 Maven 默认值。它还提供了一个依赖关系管理部分，以便您可以省略版本标记。

Other “Starters” provide dependencies that you are likely to need when developing a specific type of application. Since we are developing a web application, we add a spring-boot-starter-web dependency. Before that, we can look at what we currently have by running the following command:

其他“Starters”提供了在开发特定类型的应用程序时可能需要的依赖项。因为我们正在开发一个 web 应用程序，所以我们添加了一个 spring-boot-starter-web 依赖项。在此之前，我们可以通过运行以下命令来查看我们当前拥有的内容:

$ mvn dependency:tree

[INFO] com.example:myproject:jar:0.0.1-SNAPSHOT

The mvn dependency:tree command prints a tree representation of your project dependencies. You can see that spring-boot-starter-parent provides no dependencies by itself. To add the necessary dependencies, edit your pom.xml and add the spring-boot-starter-web dependency immediately below the parent section:

MVN Dependency：tree命令打印项目依赖关系的树表示形式。 您可以看到Spring-boot-starter-parent本身没有提供依赖项。 要添加必要的依赖项，请编辑pom.xml并在父节的正下方添加Spring-boot-starter-web依赖项：

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

If you run mvn dependency:tree again, you see that there are now a number of additional dependencies, including the Tomcat web server and Spring Boot itself.

如果再次运行MVN Dependency：Tree，您会看到现在有许多额外的依赖项，包括Tomcat web服务器和Spring Boot本身。

### Writing the Code 编写代码

To finish our application, we need to create a single Java file. By default, Maven compiles sources from src/main/java, so you need to create that directory structure and then add a file named src/main/java/Example.java to contain the following code:

为了完成我们的应用程序，我们需要创建一个 Java 文件。默认情况下，Maven 从 src/main/java 编译源代码，因此您需要创建该目录结构，然后添加一个名为 src/main/java/Example.java 的文件，以包含以下代码:

import org.springframework.boot.\*;

import org.springframework.boot.autoconfigure.\*; import org.springframework.web.bind.annotation.\*;

@RestController @EnableAutoConfiguration public class Example {

@RequestMapping("/") String home() {

return "Hello World!";

}

public static void main(String[] args) { SpringApplication.run(Example.class, args);

}

}

Although there is not much code here, quite a lot is going on. We step through the important parts in the next few sections.

虽然这里没有太多代码，但有相当多的代码在运行。 在接下来的几个部分中，我们将逐步介绍重要的部分。

#### The @RestController and @RequestMapping Annotations @RestController和@RequestMapping注解

The first annotation on our Example class is @RestController. This is known as a *stereotype* annotation. It provides hints for people reading the code and for Spring that the class plays a specific role. In this case, our class is a web @Controller, so Spring considers it when handling incoming web requests.

我们的 Example 类的第一个注释是@restcontroller。这就是所谓的构造型注释。它为阅读代码的人提供了提示，并为 Spring 提供了类扮演特定角色的提示。在这种情况下，我们的类是一个 web@Controller，因此 Spring 在处理传入的 web 请求时会考虑它。

The @RequestMapping annotation provides “routing” information. It tells Spring that any HTTP request with the / path should be mapped to the home method. The @RestController annotation tells Spring to render the resulting string directly back to the caller.

@ requestmapping 注解提供“路由”信息。它告诉 Spring，任何带有/path 的 HTTP 请求都应该映射到 home 方法。@ restcontroller 注解告诉 Spring 将生成的字符串直接呈现回调用方。

The @RestController and @RequestMapping annotations are Spring MVC annotations (they are not specific to Spring Boot). See the [MVC section](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web.html#mvc) in the Spring Reference Documentation for more details.

@ restcontroller 和@requestmapping 注释是 Spring MVC 注释(它们不是特定于 Spring Boot 的)。有关更多细节，请参见 Spring 参考文档中的 MVC 部分。

#### The @EnableAutoConfiguration Annotation

The second class-level annotation is @EnableAutoConfiguration. This annotation tells Spring Boot to “guess” how you want to configure Spring, based on the jar dependencies that you have added. Since spring-boot-starter-web added Tomcat and Spring MVC, the auto-configuration assumes that you are developing a web application and sets up Spring accordingly.

**Starters and Auto-configuration**

Auto-configuration is designed to work well with “Starters”, but the two concepts are not directly tied. You are free to pick and choose jar dependencies outside of the starters. Spring Boot still does its best to auto-configure your application.

#### The “main” Method

The final part of our application is the main method. This is a standard method that follows the Java convention for an application entry point. Our main method delegates to Spring Boot’s SpringApplication class by calling run. SpringApplication bootstraps our application, starting Spring, which, in turn, starts the auto-configured Tomcat web server. We need to pass Example.class as an argument to the run method to tell SpringApplication which is the primary Spring component. The args array is also passed through to expose any command-line arguments.

### Running the Example

At this point, your application should work. Since you used the spring-boot-starter-parent POM, you have a useful run goal that you can use to start the application. Type mvn spring-boot:run from the root project directory to start the application. You should see output similar to the following:

$ mvn spring-boot:run

.

\_

\_ \_

/\\ / '\_ \_ \_(\_)\_ \_ \ \ \ \ ( ( )\ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ )| |\_)| | | | | || (\_| | ) ) ) ) ' | | . |\_| |\_|\_| |\_\ , | / / / /

=========|\_|==============| /=/\_/\_/\_/

:: Spring Boot :: (v2.3.4.RELEASE)

....... . . .

....... . . . (log output here)

....... . . .

........ Started Example in 2.222 seconds (JVM running for 6.514)

If you open a web browser to localhost:8080, you should see the following output:

Hello World!

To gracefully exit the application, press ctrl-c.

### Creating an Executable Jar

We finish our example by creating a completely self-contained executable jar file that we could run in production. Executable jars (sometimes called “fat jars”) are archives containing your compiled classes along with all of the jar dependencies that your code needs to run.

**Executable jars and Java**

Java does not provide a standard way to load nested jar files (jar files that are themselves contained within a jar). This can be problematic if you are looking to distribute a self- contained application.

To solve this problem, many developers use “uber” jars. An uber jar packages all the classes from all the application’s dependencies into a single archive. The problem with this approach is that it becomes hard to see which libraries are in your application. It can also be problematic if the same filename is used (but with different content) in multiple jars.

Spring Boot takes a [different approach](#_bookmark691) and lets you actually nest jars directly.

To create an executable jar, we need to add the spring-boot-maven-plugin to our pom.xml. To do so, insert the following lines just below the dependencies section:

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

The spring-boot-starter-parent POM includes <executions> configuration to bind the repackage goal. If you do not use the parent POM, you need to declare this configuration yourself. See the [plugin documentation](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/maven-plugin/reference/html/#getting-started) for details.

Save your pom.xml and run mvn package from the command line, as follows:

$ mvn package

[INFO] Scanning for projects... [INFO]

[INFO]

[INFO] Building myproject 0.0.1-SNAPSHOT [INFO]

[INFO] .... ..

[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ myproject ---

[INFO] Building jar: /Users/developer/example/spring-boot-example/target/myproject- 0.0.1-SNAPSHOT.jar

[INFO]

[INFO] --- spring-boot-maven-plugin:2.3.4.RELEASE:repackage (default) @ myproject ---

[INFO]

[INFO] BUILD SUCCESS [INFO]

If you look in the target directory, you should see myproject-0.0.1-SNAPSHOT.jar. The file should be around 10 MB in size. If you want to peek inside, you can use jar tvf, as follows:

$ jar tvf target/myproject-0.0.1-SNAPSHOT.jar

You should also see a much smaller file named myproject-0.0.1-SNAPSHOT.jar.original in the target

directory. This is the original jar file that Maven created before it was repackaged by Spring Boot.

To run that application, use the java -jar command, as follows:

$ java -jar target/myproject-0.0.1-SNAPSHOT.jar

.

\_

\_ \_

/\\ / '\_ \_ \_(\_)\_ \_ \ \ \ \ ( ( )\ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ )| |\_)| | | | | || (\_| | ) ) ) ) ' | | . |\_| |\_|\_| |\_\ , | / / / /

=========|\_|==============| /=/\_/\_/\_/

:: Spring Boot :: (v2.3.4.RELEASE)

....... . . .

....... . . . (log output here)

....... . . .

........ Started Example in 2.536 seconds (JVM running for 2.864)

As before, to exit the application, press ctrl-c.

## What to Read Next

Hopefully, this section provided some of the Spring Boot basics and got you on your way to writing

your own applications. If you are a task-oriented type of developer, you might want to jump over to [spring.io](https://spring.io/) and check out some of the [getting started](https://spring.io/guides/) guides that solve specific “How do I do that with Spring?” problems. We also have Spring Boot-specific “[How-to](#_bookmark525)” reference documentation.

Otherwise, the next logical step is to read [*Using Spring Boot*](#_bookmark36). If you are really impatient, you could also jump ahead and read about [*Spring Boot features*](#_bookmark84).

**Chapter 3. Using Spring Boot**

This section goes into more detail about how you should use Spring Boot. It covers topics such as build systems, auto-configuration, and how to run your applications. We also cover some Spring Boot best practices. Although there is nothing particularly special about Spring Boot (it is just another library that you can consume), there are a few recommendations that, when followed, make your development process a little easier.

If you are starting out with Spring Boot, you should probably read the [*Getting Started*](#_bookmark9) guide before diving into this section.

## Build Systems

It is strongly recommended that you choose a build system that supports [*dependency management*](#_bookmark38)and that can consume artifacts published to the “Maven Central” repository. We would recommend that you choose Maven or Gradle. It is possible to get Spring Boot to work with other build systems (Ant, for example), but they are not particularly well supported.

### Dependency Management

Each release of Spring Boot provides a curated list of dependencies that it supports. In practice, you do not need to provide a version for any of these dependencies in your build configuration, as Spring Boot manages that for you. When you upgrade Spring Boot itself, these dependencies are upgraded as well in a consistent way.

You can still specify a version and override Spring Boot’s recommendations if you need to do so.

The curated list contains all the Spring modules that you can use with Spring Boot as well as a refined list of third party libraries. The list is available as a standard Bills of Materials (spring-boot- dependencies) that can be used with both [Maven](#_bookmark39) and [Gradle](#_bookmark40).

Each release of Spring Boot is associated with a base version of the Spring Framework. We **highly** recommend that you not specify its version.

### Maven

To learn about using Spring Boot with Maven, please refer to the documentation for Spring Boot’s Maven plugin:

* + - * Reference ([HTML](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/maven-plugin/reference/html/) and [PDF](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/maven-plugin/reference/pdf/spring-boot-maven-plugin-reference.pdf))
      * [API](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/maven-plugin/api/)

### Gradle

To learn about using Spring Boot with Gradle, please refer to the documentation for Spring Boot’s Gradle plugin:

* + - * Reference ([HTML](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/gradle-plugin/reference/html/) and [PDF](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/gradle-plugin/reference/pdf/spring-boot-gradle-plugin-reference.pdf))
      * [API](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/gradle-plugin/api/)

### Ant

It is possible to build a Spring Boot project using Apache Ant+Ivy. The spring-boot-antlib “AntLib” module is also available to help Ant create executable jars.

To declare dependencies, a typical ivy.xml file looks something like the following example:

<ivy-module version="2.0">

<info organisation="org.springframework.boot" module="spring-boot-sample-ant" />

<configurations>

<conf name="compile" description="everything needed to compile this module" />

<conf name="runtime" extends="compile" description="everything needed to run this module" />

</configurations>

<dependencies>

<dependency org="org.springframework.boot" name="spring-boot-starter" rev="${spring-boot.version}" conf="compile" />

</dependencies>

</ivy-module>

A typical build.xml looks like the following example:

<project

xmlns:ivy="antlib:org.apache.ivy.ant"

xmlns:spring-boot="antlib:org.springframework.boot.ant" name="myapp" default="build">

<property name="spring-boot.version" value="2.3.4.RELEASE" />

<target name="resolve" description="--> retrieve dependencies with ivy">

<ivy:retrieve pattern="lib/[conf]/[artifact]-[type]-[revision].[ext]" />

</target>

<target name="classpaths" depends="resolve">

<path id="compile.classpath">

<fileset dir="lib/compile" includes="\*.jar" />

</path>

</target>

<target name="init" depends="classpaths">

<mkdir dir="build/classes" />

</target>

<target name="compile" depends="init" description="compile">

<javac srcdir="src/main/java" destdir="build/classes" classpathref="compile.classpath" />

</target>

<target name="build" depends="compile">

<spring-boot:exejar destfile="build/myapp.jar" classes="build/classes">

<spring-boot:lib>

<fileset dir="lib/runtime" />

</spring-boot:lib>

</spring-boot:exejar>

</target>

</project>

If you do not want to use the spring-boot-antlib module, see the [*Build an*](#_bookmark648)[*Executable Archive from Ant without Using spring-boot-antlib*](#_bookmark648) “How-to” .

### Starters

Starters are a set of convenient dependency descriptors that you can include in your application. You get a one-stop shop for all the Spring and related technologies that you need without having to hunt through sample code and copy-paste loads of dependency descriptors. For example, if you want to get started using Spring and JPA for database access, include the spring-boot-starter-data- jpa dependency in your project.

The starters contain a lot of the dependencies that you need to get a project up and running quickly and with a consistent, supported set of managed transitive dependencies.

**What’s in a name**

All **official** starters follow a similar naming pattern; spring-boot-starter-\*, where \* is a particular type of application. This naming structure is intended to help when you need to find a starter. The Maven integration in many IDEs lets you search dependencies by name. For example, with the appropriate Eclipse or STS plugin installed, you can press ctrl-space in the POM editor and type “spring-boot-starter” for a complete list.

As explained in the “[Creating Your Own Starter](#_bookmark355)” section, third party starters should not start with spring-boot, as it is reserved for official Spring Boot artifacts. Rather, a third-party starter typically starts with the name of the project. For example, a third-party starter project called thirdpartyproject would typically be named thirdpartyproject-spring-boot-starter.

The following application starters are provided by Spring Boot under the org.springframework.boot

group:

*Table 1. Spring Boot application starters*

|  |  |
| --- | --- |
| **Name** | **Description** |
| spring-boot-starter | Core starter, including auto-configuration support, logging and YAML |
| spring-boot-starter-activemq | Starter for JMS messaging using Apache ActiveMQ |
| spring-boot-starter-amqp | Starter for using Spring AMQP and Rabbit MQ |
| spring-boot-starter-aop | Starter for aspect-oriented programming with Spring AOP and AspectJ |
| spring-boot-starter-artemis | Starter for JMS messaging using Apache Artemis |
| spring-boot-starter-batch | Starter for using Spring Batch |
| spring-boot-starter-cache | Starter for using Spring Framework’s caching support |
| spring-boot-starter-data-cassandra | Starter for using Cassandra distributed database and Spring Data Cassandra |
| spring-boot-starter-data-cassandra-reactive | Starter for using Cassandra distributed database and Spring Data Cassandra Reactive |
| spring-boot-starter-data-couchbase | Starter for using Couchbase document-oriented database and Spring Data Couchbase |
| spring-boot-starter-data-couchbase-reactive | Starter for using Couchbase document-oriented database and Spring Data Couchbase Reactive |
| spring-boot-starter-data-elasticsearch | Starter for using Elasticsearch search and analytics engine and Spring Data Elasticsearch |
| spring-boot-starter-data-jdbc | Starter for using Spring Data JDBC |
| spring-boot-starter-data-jpa | Starter for using Spring Data JPA with Hibernate |

|  |  |
| --- | --- |
| **Name** | **Description** |
| spring-boot-starter-data-ldap | Starter for using Spring Data LDAP |
| spring-boot-starter-data-mongodb | Starter for using MongoDB document-oriented database and Spring Data MongoDB |
| spring-boot-starter-data-mongodb-reactive | Starter for using MongoDB document-oriented database and Spring Data MongoDB Reactive |
| spring-boot-starter-data-neo4j | Starter for using Neo4j graph database and Spring Data Neo4j |
| spring-boot-starter-data-r2dbc | Starter for using Spring Data R2DBC |
| spring-boot-starter-data-redis | Starter for using Redis key-value data store with Spring Data Redis and the Lettuce client |
| spring-boot-starter-data-redis-reactive | Starter for using Redis key-value data store with Spring Data Redis reactive and the Lettuce client |
| spring-boot-starter-data-rest | Starter for exposing Spring Data repositories over REST using Spring Data REST |
| spring-boot-starter-data-solr | Starter for using the Apache Solr search platform with Spring Data Solr |
| spring-boot-starter-freemarker | Starter for building MVC web applications using FreeMarker views |
| spring-boot-starter-groovy-templates | Starter for building MVC web applications using Groovy Templates views |
| spring-boot-starter-hateoas | Starter for building hypermedia-based RESTful web application with Spring MVC and Spring HATEOAS |
| spring-boot-starter-integration | Starter for using Spring Integration |
| spring-boot-starter-jdbc | Starter for using JDBC with the HikariCP connection pool |
| spring-boot-starter-jersey | Starter for building RESTful web applications using JAX-RS and Jersey. An alternative to  [spring-boot-starter-web](#_bookmark45) |
| spring-boot-starter-jooq | Starter for using jOOQ to access SQL databases. An alternative to [spring-boot-starter-data-jpa](#_bookmark43) or [spring-boot-starter-jdbc](#_bookmark44) |
| spring-boot-starter-json | Starter for reading and writing json |
| spring-boot-starter-jta-atomikos | Starter for JTA transactions using Atomikos |
| spring-boot-starter-jta-bitronix | Starter for JTA transactions using Bitronix. Deprecated since 2.3.0 |
| spring-boot-starter-mail | Starter for using Java Mail and Spring Framework’s email sending support |

|  |  |
| --- | --- |
| **Name** | **Description** |
| spring-boot-starter-mustache | Starter for building web applications using Mustache views |
| spring-boot-starter-oauth2-client | Starter for using Spring Security’s OAuth2/OpenID Connect client features |
| spring-boot-starter-oauth2-resource-server | Starter for using Spring Security’s OAuth2 resource server features |
| spring-boot-starter-quartz | Starter for using the Quartz scheduler |
| spring-boot-starter-rsocket | Starter for building RSocket clients and servers |
| spring-boot-starter-security | Starter for using Spring Security |
| spring-boot-starter-test | Starter for testing Spring Boot applications with libraries including JUnit, Hamcrest and Mockito |
| spring-boot-starter-thymeleaf | Starter for building MVC web applications using Thymeleaf views |
| spring-boot-starter-validation | Starter for using Java Bean Validation with Hibernate Validator |
| spring-boot-starter-web | Starter for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container |
| spring-boot-starter-web-services | Starter for using Spring Web Services |
| spring-boot-starter-webflux | Starter for building WebFlux applications using Spring Framework’s Reactive Web support |
| spring-boot-starter-websocket | Starter for building WebSocket applications using Spring Framework’s WebSocket support |

In addition to the application starters, the following starters can be used to add [*production ready*](#_bookmark378)

features:

*Table 2. Spring Boot production starters*

|  |  |
| --- | --- |
| **Name** | **Description** |
| spring-boot-starter-actuator | Starter for using Spring Boot’s Actuator which provides production ready features to help you monitor and manage your application |

Finally, Spring Boot also includes the following starters that can be used if you want to exclude or swap specific technical facets:

*Table 3. Spring Boot technical starters*

|  |  |
| --- | --- |
| **Name** | **Description** |
| spring-boot-starter-jetty | Starter for using Jetty as the embedded servlet  container. An alternative to [spring-boot-](#_bookmark47) [starter-tomcat](#_bookmark47) |
| spring-boot-starter-log4j2 | Starter for using Log4j2 for logging. An alternative to [spring-boot-starter-logging](#_bookmark46) |
| spring-boot-starter-logging | Starter for logging using Logback. Default logging starter |
| spring-boot-starter-reactor-netty | Starter for using Reactor Netty as the embedded reactive HTTP server. |
| spring-boot-starter-tomcat | Starter for using Tomcat as the embedded servlet container. Default servlet container starter used by [spring-boot-starter-web](#_bookmark45) |
| spring-boot-starter-undertow | Starter for using Undertow as the embedded  servlet container. An alternative to [spring-boot-](#_bookmark47) [starter-tomcat](#_bookmark47) |

To learn how to swap technical facets, please see the how-to documentation for [swapping web](#_bookmark544) [server](#_bookmark544) and [logging system](#_bookmark587).

For a list of additional community contributed starters, see the [README file](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-project/spring-boot-starters/README.adoc) in the

spring-boot-starters module on GitHub.

## Structuring Your Code

Spring Boot does not require any specific code layout to work. However, there are some best practices that help.

### Using the “default” Package

When a class does not include a package declaration, it is considered to be in the “default package”. The use of the “default package” is generally discouraged and should be avoided. It can cause particular problems for Spring Boot applications that use the @ComponentScan, @ConfigurationPropertiesScan, @EntityScan, or @SpringBootApplication annotations, since every class from every jar is read.

We recommend that you follow Java’s recommended package naming conventions and use a reversed domain name (for example, com.example.project).

### Locating the Main Application Class

We generally recommend that you locate your main application class in a root package above other classes. The [@SpringBootApplication annotation](#_bookmark58) is often placed on your main class, and it implicitly defines a base “search package” for certain items. For example, if you are writing a JPA application, the package of the @SpringBootApplication annotated class is used to search for @Entity items. Using

a root package also allows component scan to apply only on your project.

If you don’t want to use @SpringBootApplication, the @EnableAutoConfiguration and @ComponentScan annotations that it imports defines that behaviour so you can also use those instead.

The following listing shows a typical layout:

com

+- example

+- myapplication

+- Application.java

|

+- customer

| +- Customer.java

| +- CustomerController.java

| +- CustomerService.java

| +- CustomerRepository.java

|

+- order

+- Order.java

+- OrderController.java

+- OrderService.java

+- OrderRepository.java

The Application.java file would declare the main method, along with the basic

@SpringBootApplication, as follows:

package com.example.myapplication;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication public class Application {

public static void main(String[] args) { SpringApplication.run(Application.class, args);

}

}

## Configuration Classes

Spring Boot favors Java-based configuration. Although it is possible to use SpringApplication with XML sources, we generally recommend that your primary source be a single @Configuration class. Usually the class that defines the main method is a good candidate as the primary @Configuration.

Many Spring configuration examples have been published on the Internet that use XML configuration. If possible, always try to use the equivalent Java-based configuration. Searching for Enable\* annotations can be a good starting point.

### Importing Additional Configuration Classes

You need not put all your @Configuration into a single class. The @Import annotation can be used to import additional configuration classes. Alternatively, you can use @ComponentScan to automatically pick up all Spring components, including @Configuration classes.

### Importing XML Configuration

If you absolutely must use XML based configuration, we recommend that you still start with a @Configuration class. You can then use an @ImportResource annotation to load XML configuration files.

## Auto-configuration

Spring Boot auto-configuration attempts to automatically configure your Spring application based on the jar dependencies that you have added. For example, if HSQLDB is on your classpath, and you have not manually configured any database connection beans, then Spring Boot auto-configures an in-memory database.

You need to opt-in to auto-configuration by adding the @EnableAutoConfiguration or

@SpringBootApplication annotations to one of your @Configuration classes.

You should only ever add one @SpringBootApplication or @EnableAutoConfiguration annotation. We generally recommend that you add one or the other to your primary @Configuration class only.

### Gradually Replacing Auto-configuration

Auto-configuration is non-invasive. At any point, you can start to define your own configuration to replace specific parts of the auto-configuration. For example, if you add your own DataSource bean, the default embedded database support backs away.

If you need to find out what auto-configuration is currently being applied, and why, start your application with the --debug switch. Doing so enables debug logs for a selection of core loggers and logs a conditions report to the console.

### Disabling Specific Auto-configuration Classes

If you find that specific auto-configuration classes that you do not want are being applied, you can use the exclude attribute of @SpringBootApplication to disable them, as shown in the following example:

import org.springframework.boot.autoconfigure.\*; import org.springframework.boot.autoconfigure.jdbc.\*;

@SpringBootApplication(exclude={DataSourceAutoConfiguration.class}) public class MyApplication {

}

If the class is not on the classpath, you can use the excludeName attribute of the annotation and specify the fully qualified name instead. If you prefer to use @EnableAutoConfiguration rather than @SpringBootApplication, exclude and excludeName are also available. Finally, you can also control the list of auto-configuration classes to exclude by using the spring.autoconfigure.exclude property.

 You can define exclusions both at the annotation level and by using the property.

Even though auto-configuration classes are public, the only aspect of the class that is considered public API is the name of the class which can be used for disabling

 the auto-configuration. The actual contents of those classes, such as nested

configuration classes or bean methods are for internal use only and we do not

recommend using those directly.

## Spring Beans and Dependency Injection

You are free to use any of the standard Spring Framework techniques to define your beans and their injected dependencies. We often find that using @ComponentScan (to find your beans) and using @Autowired (to do constructor injection) works well.

If you structure your code as suggested above (locating your application class in a root package), you can add @ComponentScan without any arguments. All of your application components ( @Component, @Service, @Repository, @Controller etc.) are automatically registered as Spring Beans.

The following example shows a @Service Bean that uses constructor injection to obtain a required

RiskAssessor bean:

package com.example.service;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Service;

@Service

public class DatabaseAccountService implements AccountService { private final RiskAssessor riskAssessor;

@Autowired

public DatabaseAccountService(RiskAssessor riskAssessor) { this.riskAssessor = riskAssessor;

}

// ...

}

If a bean has one constructor, you can omit the @Autowired, as shown in the following example:

@Service

public class DatabaseAccountService implements AccountService { private final RiskAssessor riskAssessor;

public DatabaseAccountService(RiskAssessor riskAssessor) { this.riskAssessor = riskAssessor;

}

// ...

}

Notice how using constructor injection lets the riskAssessor field be marked as

final, indicating that it cannot be subsequently changed.

## Using the @SpringBootApplication Annotation

Many Spring Boot developers like their apps to use auto-configuration, component scan and be able to define extra configuration on their "application class". A single @SpringBootApplication annotation can be used to enable those three features, that is:

* @EnableAutoConfiguration: enable [Spring Boot’s auto-configuration mechanism](#_bookmark54)
* @ComponentScan: enable @Component scan on the package where the application is located (see [the](#_bookmark48) [best practices](#_bookmark48))
* @Configuration: allow to register extra beans in the context or import additional configuration classes

package com.example.myapplication;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication // same as @Configuration @EnableAutoConfiguration @ComponentScan

public class Application {

public static void main(String[] args) { SpringApplication.run(Application.class, args);

}

}

@SpringBootApplication also provides aliases to customize the attributes of

@EnableAutoConfiguration and @ComponentScan.

None of these features are mandatory and you may choose to replace this single annotation by any of the features that it enables. For instance, you may not want to use component scan or configuration properties scan in your application:



package com.example.myapplication;

import org.springframework.boot.SpringApplication;

import org.springframework.context.annotation.ComponentScan import org.springframework.context.annotation.Configuration; import org.springframework.context.annotation.Import;

@Configuration(proxyBeanMethods = false) @EnableAutoConfiguration

@Import({ MyConfig.class, MyAnotherConfig.class }) public class Application {

public static void main(String[] args) { SpringApplication.run(Application.class, args);

}

}

In this example, Application is just like any other Spring Boot application except that @Component-annotated classes and @ConfigurationProperties-annotated classes are not detected automatically and the user-defined beans are imported explicitly (see @Import).

## Running Your Application

One of the biggest advantages of packaging your application as a jar and using an embedded HTTP server is that you can run your application as you would any other. The sample applies to debugging Spring Boot applications. You do not need any special IDE plugins or extensions.

This section only covers jar based packaging. If you choose to package your application as a war file, you should refer to your server and IDE documentation.

### Running from an IDE

You can run a Spring Boot application from your IDE as a Java application. However, you first need to import your project. Import steps vary depending on your IDE and build system. Most IDEs can import Maven projects directly. For example, Eclipse users can select Import… → Existing Maven Projects from the File menu.

If you cannot directly import your project into your IDE, you may be able to generate IDE metadata by using a build plugin. Maven includes plugins for [Eclipse](https://maven.apache.org/plugins/maven-eclipse-plugin/) and [IDEA](https://maven.apache.org/plugins/maven-idea-plugin/). Gradle offers plugins for [various IDEs](https://docs.gradle.org/current/userguide/userguide.html).

If you accidentally run a web application twice, you see a “Port already in use” error. STS users can use the Relaunch button rather than the Run button to ensure that any existing instance is closed.

### Running as a Packaged Application

If you use the Spring Boot Maven or Gradle plugins to create an executable jar, you can run your application using java -jar, as shown in the following example:

$ java -jar target/myapplication-0.0.1-SNAPSHOT.jar

It is also possible to run a packaged application with remote debugging support enabled. Doing so lets you attach a debugger to your packaged application, as shown in the following example:

$ java -Xdebug -Xrunjdwp:server=y,transport=dt\_socket,address=8000,suspend=n \

-jar target/myapplication-0.0.1-SNAPSHOT.jar

### Using the Maven Plugin

The Spring Boot Maven plugin includes a run goal that can be used to quickly compile and run your application. Applications run in an exploded form, as they do in your IDE. The following example shows a typical Maven command to run a Spring Boot application:

$ mvn spring-boot:run

You might also want to use the MAVEN\_OPTS operating system environment variable, as shown in the following example:

$ export MAVEN\_OPTS=-Xmx1024m

### Using the Gradle Plugin

The Spring Boot Gradle plugin also includes a bootRun task that can be used to run your application in an exploded form. The bootRun task is added whenever you apply the org.springframework.boot and java plugins and is shown in the following example:

$ gradle bootRun

You might also want to use the JAVA\_OPTS operating system environment variable, as shown in the following example:

$ export JAVA\_OPTS=-Xmx1024m

### Hot Swapping

Since Spring Boot applications are plain Java applications, JVM hot-swapping should work out of the box. JVM hot swapping is somewhat limited with the bytecode that it can replace. For a more complete solution, [JRebel](https://www.jrebel.com/products/jrebel) can be used.

The spring-boot-devtools module also includes support for quick application restarts. See the [Developer Tools](#_bookmark65) section later in this chapter and the [Hot swapping “How-to”](#_bookmark631) for details.

## Developer Tools

Spring Boot includes an additional set of tools that can make the application development experience a little more pleasant. The spring-boot-devtools module can be included in any project to provide additional development-time features. To include devtools support, add the module dependency to your build, as shown in the following listings for Maven and Gradle:

*Maven*

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

</dependencies>

*Gradle*

dependencies {

developmentOnly("org.springframework.boot:spring-boot-devtools")

}

Developer tools are automatically disabled when running a fully packaged application. If your application is launched from java -jar or if it is started from a special classloader, then it is considered a “production application”. If that does not apply to you (i.e. if you run your application from a container), consider excluding devtools or set the -Dspring.devtools.restart.enabled=false system property.

Flagging the dependency as optional in Maven or using the developmentOnly configuration in Gradle (as shown above) prevents devtools from being transitively applied to other modules that use your project.

Repackaged archives do not contain devtools by default. If you want to use a [certain remote devtools feature](#_bookmark79), you need to include it. When using the Maven plugin, set the excludeDevtools property to false. When using the Gradle plugin, [configure the task’s classpath to include the developmentOnly configuration](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/gradle-plugin/reference/html/#packaging-executable-configuring-including-development-only-dependencies).

### Property Defaults

Several of the libraries supported by Spring Boot use caches to improve performance. For example, [template engines](#_bookmark157) cache compiled templates to avoid repeatedly parsing template files. Also, Spring MVC can add HTTP caching headers to responses when serving static resources.

While caching is very beneficial in production, it can be counter-productive during development, preventing you from seeing the changes you just made in your application. For this reason, spring- boot-devtools disables the caching options by default.

Cache options are usually configured by settings in your application.properties file. For example, Thymeleaf offers the spring.thymeleaf.cache property. Rather than needing to set these properties manually, the spring-boot-devtools module automatically applies sensible development-time configuration.

Because you need more information about web requests while developing Spring MVC and Spring WebFlux applications, developer tools will enable DEBUG logging for the web logging group. This will give you information about the incoming request, which handler is processing it, the response outcome, etc. If you wish to log all request details (including potentially sensitive information), you can turn on the spring.mvc.log-request-details or spring.codec.log-request-details configuration properties.

If you don’t want property defaults to be applied you can set spring.devtools.add- properties to false in your application.properties.

For a complete list of the properties that are applied by the devtools, see [DevToolsPropertyDefaultsPostProcessor](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-devtools/src/main/java/org/springframework/boot/devtools/env/DevToolsPropertyDefaultsPostProcessor.java).

### Automatic Restart

Applications that use spring-boot-devtools automatically restart whenever files on the classpath change. This can be a useful feature when working in an IDE, as it gives a very fast feedback loop for code changes. By default, any entry on the classpath that points to a directory is monitored for changes. Note that certain resources, such as static assets and view templates, [do not need to restart](#_bookmark70) [the application](#_bookmark70).

**Triggering a restart**

As DevTools monitors classpath resources, the only way to trigger a restart is to update the classpath. The way in which you cause the classpath to be updated depends on the IDE that you are using. In Eclipse, saving a modified file causes the classpath to be updated and triggers a restart. In IntelliJ IDEA, building the project (Build +→+ Build Project) has the same effect.

As long as forking is enabled, you can also start your application by using the supported build plugins (Maven and Gradle), since DevTools needs an isolated application classloader to operate properly. By default, the Gradle and Maven plugins fork the application process.

Automatic restart works very well when used with LiveReload. [See the LiveReload](#_bookmark76) [section](#_bookmark76) for details. If you use JRebel, automatic restarts are disabled in favor of dynamic class reloading. Other devtools features (such as LiveReload and property overrides) can still be used.

DevTools relies on the application context’s shutdown hook to close it during a restart. It does not work correctly if you have disabled the shutdown hook (SpringApplication.setRegisterShutdownHook(false)).

When deciding if an entry on the classpath should trigger a restart when it changes, DevTools automatically ignores projects named spring-boot, spring-boot- devtools, spring-boot-autoconfigure, spring-boot-actuator, and spring-boot- starter.

DevTools needs to customize the ResourceLoader used by the ApplicationContext. If your application provides one already, it is going to be wrapped. Direct override of the getResource method on the ApplicationContext is not supported.

**Restart vs Reload**

The restart technology provided by Spring Boot works by using two classloaders. Classes that do not change (for example, those from third-party jars) are loaded into a *base* classloader. Classes that you are actively developing are loaded into a *restart* classloader. When the application is restarted, the *restart* classloader is thrown away and a new one is created. This approach means that application restarts are typically much faster than “cold starts”, since the *base* classloader is already available and populated.

If you find that restarts are not quick enough for your applications or you encounter classloading issues, you could consider reloading technologies such as [JRebel](https://jrebel.com/software/jrebel/) from ZeroTurnaround. These work by rewriting classes as they are loaded to make them more amenable to reloading.

#### Logging changes in condition evaluation

By default, each time your application restarts, a report showing the condition evaluation delta is logged. The report shows the changes to your application’s auto-configuration as you make changes such as adding or removing beans and setting configuration properties.

To disable the logging of the report, set the following property:

spring.devtools.restart.log-condition-evaluation-delta=false

#### Excluding Resources

Certain resources do not necessarily need to trigger a restart when they are changed. For example, Thymeleaf templates can be edited in-place. By default, changing resources in /META-INF/maven,

/META-INF/resources, /resources, /static, /public, or /templates does not trigger a restart but does

trigger a [live reload](#_bookmark76). If you want to customize these exclusions, you can use the spring.devtools.restart.exclude property. For example, to exclude only /static and /public you would set the following property:

spring.devtools.restart.exclude=static/\*\*,public/\*\*

If you want to keep those defaults and *add* additional exclusions, use the

spring.devtools.restart.additional-exclude property instead.

#### Watching Additional Paths

You may want your application to be restarted or reloaded when you make changes to files that are not on the classpath. To do so, use the spring.devtools.restart.additional-paths property to configure additional paths to watch for changes. You can use the spring.devtools.restart.exclude property [described earlier](#_bookmark70) to control whether changes beneath the additional paths trigger a full restart or a [live reload](#_bookmark76).

#### Disabling Restart

If you do not want to use the restart feature, you can disable it by using the spring.devtools.restart.enabled property. In most cases, you can set this property in your application.properties (doing so still initializes the restart classloader, but it does not watch for file changes).

If you need to *completely* disable restart support (for example, because it does not work with a specific library), you need to set the spring.devtools.restart.enabled System property to false before calling SpringApplication.run(…), as shown in the following example:

public static void main(String[] args) { System.setProperty("spring.devtools.restart.enabled", "false"); SpringApplication.run(MyApp.class, args);

}

#### Using a Trigger File

If you work with an IDE that continuously compiles changed files, you might prefer to trigger restarts only at specific times. To do so, you can use a “trigger file”, which is a special file that must be modified when you want to actually trigger a restart check.

Any update to the file will trigger a check, but restart only actually occurs if Devtools has detected it has something to do.

To use a trigger file, set the spring.devtools.restart.trigger-file property to the name (excluding any path) of your trigger file. The trigger file must appear somewhere on your classpath.

For example, if you have a project with the following structure:

src

+- main

+- resources

+- .reloadtrigger

Then your trigger-file property would be:

spring.devtools.restart.trigger-file=.reloadtrigger

Restarts will now only happen when the src/main/resources/.reloadtrigger is updated.

You might want to set spring.devtools.restart.trigger-file as a [global setting](#_bookmark77), so that all your projects behave in the same way.

Some IDEs have features that save you from needing to update your trigger file manually. [Spring](https://spring.io/tools) [Tools for Eclipse](https://spring.io/tools) and [IntelliJ IDEA (Ultimate Edition)](https://www.jetbrains.com/idea/) both have such support. With Spring Tools, you

can use the “reload” button from the console view (as long as your trigger-file is named

.reloadtrigger). For IntelliJ IDEA, you can follow the [instructions in their documentation](https://www.jetbrains.com/help/idea/spring-boot.html#application-update-policies).

#### Customizing the Restart Classloader

As described earlier in the [Restart vs Reload](#_bookmark68) section, restart functionality is implemented by using two classloaders. For most applications, this approach works well. However, it can sometimes cause classloading issues.

By default, any open project in your IDE is loaded with the “restart” classloader, and any regular

.jar file is loaded with the “base” classloader. If you work on a multi-module project, and not every module is imported into your IDE, you may need to customize things. To do so, you can create a META-INF/spring-devtools.properties file.

The spring-devtools.properties file can contain properties prefixed with restart.exclude and restart.include. The include elements are items that should be pulled up into the “restart” classloader, and the exclude elements are items that should be pushed down into the “base” classloader. The value of the property is a regex pattern that is applied to the classpath, as shown in the following example:

restart.exclude.companycommonlibs=/mycorp-common-[\\w\\d-\.]+\.jar restart.include.projectcommon=/mycorp-myproj-[\\w\\d-\.]+\.jar

All property keys must be unique. As long as a property starts with

restart.include. or restart.exclude. it is considered.

All META-INF/spring-devtools.properties from the classpath are loaded. You can package files inside your project, or in the libraries that the project consumes.

#### Known Limitations

Restart functionality does not work well with objects that are deserialized by using a standard ObjectInputStream. If you need to deserialize data, you may need to use Spring’s ConfigurableObjectInputStream in combination with Thread.currentThread().getContextClassLoader().

Unfortunately, several third-party libraries deserialize without considering the context classloader. If you find such a problem, you need to request a fix with the original authors.

### LiveReload

The spring-boot-devtools module includes an embedded LiveReload server that can be used to trigger a browser refresh when a resource is changed. LiveReload browser extensions are freely available for Chrome, Firefox and Safari from [livereload.com](http://livereload.com/extensions/).

If you do not want to start the LiveReload server when your application runs, you can set the

spring.devtools.livereload.enabled property to false.

You can only run one LiveReload server at a time. Before starting your application, ensure that no other LiveReload servers are running. If you start multiple applications from your IDE, only the first has LiveReload support.

### Global Settings

You can configure global devtools settings by adding any of the following files to the

$HOME/.config/spring-boot directory:

1. spring-boot-devtools.properties
2. spring-boot-devtools.yaml
3. spring-boot-devtools.yml

Any properties added to these file apply to *all* Spring Boot applications on your machine that use devtools. For example, to configure restart to always use a [trigger file](#_bookmark73), you would add the following property:

*~/.config/spring-boot/spring-boot-devtools.properties*

spring.devtools.restart.trigger-file=.reloadtrigger

If devtools configuration files are not found in $HOME/.config/spring-boot, the root of the $HOME directory is searched for the presence of a .spring-boot- devtools.properties file. This allows you to share the devtools global configuration with applications that are on an older version of Spring Boot that does not support the $HOME/.config/spring-boot location.

Profiles are not supported in devtools properties/yaml files.

Any profiles activated in .spring-boot-devtools.properties will not affect the loading of [profile-specific configuration files](#_bookmark106). Profile specific filenames (of the form spring-boot-devtools-<profile>.properties) and spring.profile sub- documents in YAML files are not supported.

#### Configuring File System Watcher

[FileSystemWatcher](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-devtools/src/main/java/org/springframework/boot/devtools/filewatch/FileSystemWatcher.java) works by polling the class changes with a certain time interval, and then waiting for a predefined quiet period to make sure there are no more changes. Since Spring Boot relies entirely on the IDE to compile and copy files into the location from where Spring Boot can read them, you might find that there are times when certain changes are not reflected when devtools restarts the application. If you observe such problems constantly, try increasing the spring.devtools.restart.poll-interval and spring.devtools.restart.quiet-period parameters to the values that fit your development environment:

spring.devtools.restart.poll-interval=2s spring.devtools.restart.quiet-period=1s

The monitored classpath directories are now polled every 2 seconds for changes, and a 1 second quiet period is maintained to make sure there are no additional class changes.

### Remote Applications

The Spring Boot developer tools are not limited to local development. You can also use several features when running applications remotely. Remote support is opt-in as enabling it can be a security risk. It should only be enabled when running on a trusted network or when secured with SSL. If neither of these options is available to you, you should not use DevTools' remote support. You should never enable support on a production deployment.

To enable it, you need to make sure that devtools is included in the repackaged archive, as shown in the following listing:

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<excludeDevtools>false</excludeDevtools>

</configuration>

</plugin>

</plugins>

</build>

Then you need to set the spring.devtools.remote.secret property. Like any important password or secret, the value should be unique and strong such that it cannot be guessed or brute-forced.

Remote devtools support is provided in two parts: a server-side endpoint that accepts connections and a client application that you run in your IDE. The server component is automatically enabled when the spring.devtools.remote.secret property is set. The client component must be launched manually.

#### Running the Remote Client Application

The remote client application is designed to be run from within your IDE. You need to run org.springframework.boot.devtools.RemoteSpringApplication with the same classpath as the remote project that you connect to. The application’s single required argument is the remote URL to which it connects.

For example, if you are using Eclipse or STS and you have a project named my-app that you have deployed to Cloud Foundry, you would do the following:

* Select Run Configurations… from the Run menu.
* Create a new Java Application “launch configuration”.
* Browse for the my-app project.
* Use org.springframework.boot.devtools.RemoteSpringApplication as the main class.
* Add https://myapp.cfapps.io to the Program arguments (or whatever your remote URL is).

A running remote client might resemble the following listing:

. \_ \_ \_

/\\ / '\_ \_ \_(\_)\_ \_ \_ \ \ \ \

( ( )\ | '\_ | '\_| | '\_ \/ \_` | | \_ \ \_ | |\_ \ \ \ \

\\/ )| |\_)| | | | | || (\_| []::::::[] / -\_) ' \/ \_ \ \_/ -\_) ) ) ) ) ' | | . |\_| |\_|\_| |\_\ , | |\_|\_\ |\_|\_|\_\ /\ \ |/ / / /

=========|\_|==============| /===================================/\_/\_/\_/

:: Spring Boot Remote :: 2.3.4.RELEASE

2015-06-10 18:25:06.632 INFO 14938 --- [ main]

o.s.b.devtools.RemoteSpringApplication : Starting RemoteSpringApplication on pwmbp with PID 14938 (/Users/pwebb/projects/spring-boot/code/spring-boot-project/spring- boot-devtools/target/classes started by pwebb in /Users/pwebb/projects/spring- boot/code)

2015-06-10 18:25:06.671 INFO 14938 --- [ main]

s.c.a.AnnotationConfigApplicationContext : Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@2a17b7b6: startup date [Wed Jun 10 18:25:06 PDT 2015]; root of context hierarchy

2015-06-10 18:25:07.043 WARN 14938 --- [ main]

o.s.b.d.r.c.RemoteClientConfiguration : The connection to http://localhost:8080 is insecure. You should use a URL starting with 'https://'.

2015-06-10 18:25:07.074 INFO 14938 --- [ main]

o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729 2015-06-10 18:25:07.130 INFO 14938 --- [ main]

o.s.b.devtools.RemoteSpringApplication : Started RemoteSpringApplication in 0.74 seconds (JVM running for 1.105)

Because the remote client is using the same classpath as the real application it can directly read application properties. This is how the spring.devtools.remote.secret property is read and passed to the server for authentication.

It is always advisable to use https:// as the connection protocol, so that traffic is encrypted and passwords cannot be intercepted.

If you need to use a proxy to access the remote application, configure the spring.devtools.remote.proxy.host and spring.devtools.remote.proxy.port properties.

#### Remote Update

The remote client monitors your application classpath for changes in the same way as the [local](#_bookmark67) [restart](#_bookmark67). Any updated resource is pushed to the remote application and (*if required*) triggers a restart. This can be helpful if you iterate on a feature that uses a cloud service that you do not have locally. Generally, remote updates and restarts are much quicker than a full rebuild and deploy cycle.

On a slower development environment, it may happen that the quiet period is not enough, and the changes in the classes may be split into batches. The server is restarted after the first batch of class changes is uploaded. The next batch can’t be sent to the application, since the server is restarting.

This is typically manifested by a warning in the RemoteSpringApplication logs about failing to upload some of the classes, and a consequent retry. But it may also lead to application code inconsistency and failure to restart after the first batch of changes is uploaded. If you observe such problems constantly, try increasing the spring.devtools.restart.poll-interval and spring.devtools.restart.quiet-period parameters to the values that fit your development environment. See the [Configuring File System Watcher](#_bookmark78) section for configuring these properties.

Files are only monitored when the remote client is running. If you change a file before starting the remote client, it is not pushed to the remote server.

## Packaging Your Application for Production

Executable jars can be used for production deployment. As they are self-contained, they are also ideally suited for cloud-based deployment.

For additional “production ready” features, such as health, auditing, and metric REST or JMX end- points, consider adding spring-boot-actuator. See [*Spring Boot Actuator: Production-ready Features*](#_bookmark378)for details.

## What to Read Next

You should now understand how you can use Spring Boot and some best practices that you should follow. You can now go on to learn about specific [*Spring Boot features*](#_bookmark84) in depth, or you could skip ahead and read about the “[production ready](#_bookmark378)” aspects of Spring Boot.

**Chapter 4. Spring Boot Features**

This section dives into the details of Spring Boot. Here you can learn about the key features that you may want to use and customize. If you have not already done so, you might want to read the "[Getting Started](#_bookmark9)" and "[Using Spring Boot](#_bookmark36)" sections, so that you have a good grounding of the basics.

## SpringApplication

The SpringApplication class provides a convenient way to bootstrap a Spring application that is started from a main() method. In many situations, you can delegate to the static SpringApplication.run method, as shown in the following example:

public static void main(String[] args) { SpringApplication.run(MySpringConfiguration.class, args);

}

When your application starts, you should see something similar to the following output:

. \_ \_ \_

/\\ / '\_ \_ \_(\_)\_ \_ \ \ \ \ ( ( )\ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ )| |\_)| | | | | || (\_| | ) ) ) ) ' | | . |\_| |\_|\_| |\_\ , | / / / /

=========|\_|==============| /=/\_/\_/\_/

:: Spring Boot :: v2.3.4.RELEASE

2019-04-31 13:09:54.117 INFO 56603 --- [ main]

o.s.b.s.app.SampleApplication : Starting SampleApplication v0.1.0 on mycomputer with PID 56603 (/apps/myapp.jar started by pwebb)

2019-04-31 13:09:54.166 INFO 56603 --- [ main]

ationConfigServletWebServerApplicationContext : Refreshing org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicati onContext@6e5a8246: startup date [Wed Jul 31 00:08:16 PDT 2013]; root of context hierarchy

2019-04-01 13:09:56.912 INFO 41370 --- [ main]

.t.TomcatServletWebServerFactory : Server initialized with port: 8080 2019-04-01 13:09:57.501 INFO 41370 --- [ main]

o.s.b.s.app.SampleApplication : Started SampleApplication in 2.992 seconds (JVM running for 3.658)

By default, INFO logging messages are shown, including some relevant startup details, such as the user that launched the application. If you need a log level other than INFO, you can set it, as described in [Log Levels](#_bookmark135). The application version is determined using the implementation version from the main application class’s package. Startup information logging can be turned off by setting spring.main.log-startup-info to false. This will also turn off logging of the application’s active profiles.

To add additional logging during startup, you can override

logStartupInfo(boolean) in a subclass of SpringApplication.

### Startup Failure

If your application fails to start, registered FailureAnalyzers get a chance to provide a dedicated error message and a concrete action to fix the problem. For instance, if you start a web application on port 8080 and that port is already in use, you should see something similar to the following message:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* APPLICATION FAILED TO START

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Description:

Embedded servlet container failed to start. Port 8080 was already in use. Action:

Identify and stop the process that's listening on port 8080 or configure this application to listen on another port.

Spring Boot provides numerous FailureAnalyzer implementations, and you can [add your own](#_bookmark527).

If no failure analyzers are able to handle the exception, you can still display the full conditions report to better understand what went wrong. To do so, you need to [enable the](#_bookmark101) [debug property](#_bookmark101) or [enable](#_bookmark135) [DEBUG](#_bookmark135) [logging](#_bookmark135) for org.springframework.boot.autoconfigure.logging.ConditionEvaluationReportLoggingListener.

For instance, if you are running your application by using java -jar, you can enable the debug

property as follows:

$ java -jar myproject-0.0.1-SNAPSHOT.jar --debug

### Lazy Initialization

SpringApplication allows an application to be initialized lazily. When lazy initialization is enabled, beans are created as they are needed rather than during application startup. As a result, enabling lazy initialization can reduce the time that it takes your application to start. In a web application, enabling lazy initialization will result in many web-related beans not being initialized until an HTTP request is received.

A downside of lazy initialization is that it can delay the discovery of a problem with the application. If a misconfigured bean is initialized lazily, a failure will no longer occur during startup and the

problem will only become apparent when the bean is initialized. Care must also be taken to ensure that the JVM has sufficient memory to accommodate all of the application’s beans and not just those that are initialized during startup. For these reasons, lazy initialization is not enabled by default and it is recommended that fine-tuning of the JVM’s heap size is done before enabling lazy initialization.

Lazy initialization can be enabled programmatically using the lazyInitialization method on SpringApplicationBuilder or the setLazyInitialization method on SpringApplication. Alternatively, it can be enabled using the spring.main.lazy-initialization property as shown in the following example:

spring.main.lazy-initialization=true

If you want to disable lazy initialization for certain beans while using lazy initialization for the rest of the application, you can explicitly set their lazy attribute to false using the @Lazy(false) annotation.

### Customizing the Banner

The banner that is printed on start up can be changed by adding a banner.txt file to your classpath or by setting the spring.banner.location property to the location of such a file. If the file has an encoding other than UTF-8, you can set spring.banner.charset. In addition to a text file, you can also add a banner.gif, banner.jpg, or banner.png image file to your classpath or set the spring.banner.image.location property. Images are converted into an ASCII art representation and printed above any text banner.

Inside your banner.txt file, you can use any of the following placeholders:

*Table 4. Banner variables*

|  |  |
| --- | --- |
| **Variable** | **Description** |
| ${application.version} | The version number of your application, as declared in MANIFEST.MF. For example, Implementation-Version: 1.0 is printed as 1.0. |
| ${application.formatted-version} | The version number of your application, as declared in MANIFEST.MF and formatted for display (surrounded with brackets and prefixed with v). For example (v1.0). |
| ${spring-boot.version} | The Spring Boot version that you are using. For example 2.3.4.RELEASE. |
| ${spring-boot.formatted-version} | The Spring Boot version that you are using, formatted for display (surrounded with brackets and prefixed with v). For example (v2.3.4.RELEASE). |
| ${Ansi.NAME} (or ${AnsiColor.NAME},  ${AnsiBackground.NAME}, ${AnsiStyle.NAME}) | Where NAME is the name of an ANSI escape code. See [AnsiPropertySource](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/ansi/AnsiPropertySource.java) for details. |

|  |  |
| --- | --- |
| **Variable** | **Description** |
| ${application.title} | The title of your application, as declared in MANIFEST.MF. For example Implementation-Title: MyApp is printed as MyApp. |

The SpringApplication.setBanner(…) method can be used if you want to generate a banner programmatically. Use the org.springframework.boot.Banner interface and implement your own printBanner() method.

You can also use the spring.main.banner-mode property to determine if the banner has to be printed on System.out (console), sent to the configured logger (log), or not produced at all (off).

The printed banner is registered as a singleton bean under the following name: springBootBanner.

### Customizing SpringApplication

If the SpringApplication defaults are not to your taste, you can instead create a local instance and customize it. For example, to turn off the banner, you could write:

public static void main(String[] args) {

SpringApplication app = new SpringApplication(MySpringConfiguration.class); app.setBannerMode(Banner.Mode.OFF);

app.run(args);

}

The constructor arguments passed to SpringApplication are configuration sources for Spring beans. In most cases, these are references to @Configuration classes, but they could also be references to XML configuration or to packages that should be scanned.

It is also possible to configure the SpringApplication by using an application.properties file. See

[*Externalized Configuration*](#_bookmark101) for details.

For a complete list of the configuration options, see the [SpringApplication Javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/SpringApplication.html).

### Fluent Builder API

If you need to build an ApplicationContext hierarchy (multiple contexts with a parent/child relationship) or if you prefer using a “fluent” builder API, you can use the SpringApplicationBuilder.

The SpringApplicationBuilder lets you chain together multiple method calls and includes parent and

child methods that let you create a hierarchy, as shown in the following example:

new SpringApplicationBuilder()

.sources(Parent.class)

.child(Application.class)

.bannerMode(Banner.Mode.OFF)

.run(args);

There are some restrictions when creating an ApplicationContext hierarchy. For example, Web components **must** be contained within the child context, and the same Environment is used for both parent and child contexts. See the [SpringApplicationBuilder Javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/builder/SpringApplicationBuilder.html) for full details.

### Application Availability

When deployed on platforms, applications can provide information about their availability to the platform using infrastructure such as [Kubernetes Probes](https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/). Spring Boot includes out-of-the box support for the commonly used “liveness” and “readiness” availability states. If you are using Spring Boot’s “actuator” support then these states are exposed as health endpoint groups.

In addition, you can also obtain availability states by injecting the ApplicationAvailability interface into your own beans.

#### Liveness State

The “Liveness” state of an application tells whether its internal state allows it to work correctly, or recover by itself if it’s currently failing. A broken “Liveness” state means that the application is in a state that it cannot recover from, and the infrastructure should restart the application.

In general, the "Liveness" state should not be based on external checks, such as [Health checks](#_bookmark392). If it did, a failing external system (a database, a Web API, an external cache) would trigger massive restarts and cascading failures across the platform.

The internal state of Spring Boot applications is mostly represented by the Spring ApplicationContext. If the application context has started successfully, Spring Boot assumes that the application is in a valid state. An application is considered live as soon as the context has been refreshed, see [Spring Boot application lifecycle and related Application Events](#_bookmark95).

#### Readiness State

The “Readiness” state of an application tells whether the application is ready to handle traffic. A failing “Readiness” state tells the platform that it should not route traffic to the application for now. This typically happens during startup, while CommandLineRunner and ApplicationRunner components are being processed, or at any time if the application decides that it’s too busy for additional traffic.

An application is considered ready as soon as application and command-line runners have been called, see [Spring Boot application lifecycle and related Application Events](#_bookmark95).

Tasks expected to run during startup should be executed by CommandLineRunner and ApplicationRunner components instead of using Spring component lifecycle callbacks such as @PostConstruct.

#### Managing the Application Availability State

Application components can retrieve the current availability state at any time, by injecting the ApplicationAvailability interface and calling methods on it. More often, applications will want to listen to state updates or update the state of the application.

For example, we can export the "Readiness" state of the application to a file so that a Kubernetes "exec Probe" can look at this file:

@Component

public class ReadinessStateExporter {

@EventListener

public void onStateChange(AvailabilityChangeEvent<ReadinessState> event) { switch (event.getState()) {

case ACCEPTING\_TRAFFIC:

// create file /tmp/healthy break;

case REFUSING\_TRAFFIC:

// remove file /tmp/healthy break;

}

}

}

We can also update the state of the application, when the application breaks and cannot recover:

@Component

public class LocalCacheVerifier {

private final ApplicationEventPublisher eventPublisher;

public LocalCacheVerifier(ApplicationEventPublisher eventPublisher) { this.eventPublisher = eventPublisher;

}

public void checkLocalCache() { try {

//...

}

catch (CacheCompletelyBrokenException ex) { AvailabilityChangeEvent.publish(this.eventPublisher, ex,

LivenessState.BROKEN);

}

}

}

Spring Boot provides [Kubernetes HTTP probes for "Liveness" and "Readiness" with Actuator Health](#_bookmark398) [Endpoints](#_bookmark398). You can get more guidance about [deploying Spring Boot applications on Kubernetes in](#_bookmark474) [the dedicated section](#_bookmark474).

### Application Events and Listeners

In addition to the usual Spring Framework events, such as [ContextRefreshedEvent](https://docs.spring.io/spring/docs/5.2.9.RELEASE/javadoc-api/org/springframework/context/event/ContextRefreshedEvent.html), a

SpringApplication sends some additional application events.

Some events are actually triggered before the ApplicationContext is created, so you cannot register a listener on those as a @Bean. You can register them with the SpringApplication.addListeners(…) method or the SpringApplicationBuilder.listeners(…) method.

If you want those listeners to be registered automatically, regardless of the way the application is created, you can add a META-INF/spring.factories file to your project and reference your listener(s) by using the org.springframework.context.ApplicationListener key, as shown in the following example:

org.springframework.context.ApplicationListener=com.example.project.MyL istener

Application events are sent in the following order, as your application runs:

* + - 1. An ApplicationStartingEvent is sent at the start of a run but before any processing, except for

the registration of listeners and initializers.

* + - 1. An ApplicationEnvironmentPreparedEvent is sent when the Environment to be used in the context is known but before the context is created.
      2. An ApplicationContextInitializedEvent is sent when the ApplicationContext is prepared and ApplicationContextInitializers have been called but before any bean definitions are loaded.
      3. An ApplicationPreparedEvent is sent just before the refresh is started but after bean definitions have been loaded.
      4. An ApplicationStartedEvent is sent after the context has been refreshed but before any application and command-line runners have been called.
      5. An AvailabilityChangeEvent is sent right after with LivenessState.CORRECT to indicate that the application is considered as live.
      6. An ApplicationReadyEvent is sent after any [application and command-line runners](#_bookmark98) have been called.
      7. An AvailabilityChangeEvent is sent right after with ReadinessState.ACCEPTING\_TRAFFIC to indicate that the application is ready to service requests.
      8. An ApplicationFailedEvent is sent if there is an exception on startup.

The above list only includes SpringApplicationEvents that are tied to a SpringApplication. In addition to these, the following events are also published after ApplicationPreparedEvent and before ApplicationStartedEvent:

* A WebServerInitializedEvent is sent after the WebServer is ready. ServletWebServerInitializedEvent and ReactiveWebServerInitializedEvent are the servlet and reactive variants respectively.
* A ContextRefreshedEvent is sent when an ApplicationContext is refreshed.

You often need not use application events, but it can be handy to know that they exist. Internally, Spring Boot uses events to handle a variety of tasks.

Event listeners should not run potentially lengthy tasks as they execute in the same thread by default. Consider using [application and command-line runners](#_bookmark98) instead.

Application events are sent by using Spring Framework’s event publishing mechanism. Part of this mechanism ensures that an event published to the listeners in a child context is also published to the listeners in any ancestor contexts. As a result of this, if your application uses a hierarchy of SpringApplication instances, a listener may receive multiple instances of the same type of application event.

To allow your listener to distinguish between an event for its context and an event for a descendant context, it should request that its application context is injected and then compare the injected context with the context of the event. The context can be injected by implementing ApplicationContextAware or, if the listener is a bean, by using @Autowired.

### Web Environment

A SpringApplication attempts to create the right type of ApplicationContext on your behalf. The algorithm used to determine a WebApplicationType is the following:

* If Spring MVC is present, an AnnotationConfigServletWebServerApplicationContext is used
* If Spring MVC is not present and Spring WebFlux is present, an

AnnotationConfigReactiveWebServerApplicationContext is used

* Otherwise, AnnotationConfigApplicationContext is used

This means that if you are using Spring MVC and the new WebClient from Spring WebFlux in the same application, Spring MVC will be used by default. You can override that easily by calling setWebApplicationType(WebApplicationType).

It is also possible to take complete control of the ApplicationContext type that is used by calling

setApplicationContextClass(…).

It is often desirable to call setWebApplicationType(WebApplicationType.NONE) when using SpringApplication within a JUnit test.

### Accessing Application Arguments

If you need to access the application arguments that were passed to SpringApplication.run(…), you can inject a org.springframework.boot.ApplicationArguments bean. The ApplicationArguments interface provides access to both the raw String[] arguments as well as parsed option and non- option arguments, as shown in the following example:

import org.springframework.boot.\*;

import org.springframework.beans.factory.annotation.\*; import org.springframework.stereotype.\*;

@Component

public class MyBean {

@Autowired

public MyBean(ApplicationArguments args) { boolean debug = args.containsOption("debug"); List<String> files = args.getNonOptionArgs();

// if run with "--debug logfile.txt" debug=true, files=["logfile.txt"]

}

}

Spring Boot also registers a CommandLinePropertySource with the Spring Environment. This lets you also inject single application arguments by using the @Value annotation.

### Using the ApplicationRunner or CommandLineRunner

If you need to run some specific code once the SpringApplication has started, you can implement the ApplicationRunner or CommandLineRunner interfaces. Both interfaces work in the same way and offer a single run method, which is called just before SpringApplication.run(…) completes.

This contract is well suited for tasks that should run after application startup but before it starts accepting traffic.

The CommandLineRunner interfaces provides access to application arguments as a string array, whereas the ApplicationRunner uses the ApplicationArguments interface discussed earlier. The following example shows a CommandLineRunner with a run method:

import org.springframework.boot.\*; import org.springframework.stereotype.\*;

@Component

public class MyBean implements CommandLineRunner {

public void run(String... args) {

// Do something...

}

}

If several CommandLineRunner or ApplicationRunner beans are defined that must be called in a specific order, you can additionally implement the org.springframework.core.Ordered interface or use the org.springframework.core.annotation.Order annotation.

### Application Exit

Each SpringApplication registers a shutdown hook with the JVM to ensure that the ApplicationContext closes gracefully on exit. All the standard Spring lifecycle callbacks (such as the DisposableBean interface or the @PreDestroy annotation) can be used.

In addition, beans may implement the org.springframework.boot.ExitCodeGenerator interface if they wish to return a specific exit code when SpringApplication.exit() is called. This exit code can then be passed to System.exit() to return it as a status code, as shown in the following example:

@SpringBootApplication

public class ExitCodeApplication {

@Bean

public ExitCodeGenerator exitCodeGenerator() { return () -> 42;

}

public static void main(String[] args) {

System.exit(SpringApplication.exit(SpringApplication.run(ExitCodeApplication.class, args)));

}

}

Also, the ExitCodeGenerator interface may be implemented by exceptions. When such an exception is encountered, Spring Boot returns the exit code provided by the implemented getExitCode() method.

### Admin Features

It is possible to enable admin-related features for the application by specifying the spring.application.admin.enabled property. This exposes the [SpringApplicationAdminMXBean](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/admin/SpringApplicationAdminMXBean.java) on the platform MBeanServer. You could use this feature to administer your Spring Boot application remotely. This feature could also be useful for any service wrapper implementation.

If you want to know on which HTTP port the application is running, get the property with a key of local.server.port.

## Externalized Configuration

Spring Boot lets you externalize your configuration so that you can work with the same application code in different environments. You can use properties files, YAML files, environment variables, and command-line arguments to externalize configuration. Property values can be injected directly into your beans by using the @Value annotation, accessed through Spring’s Environment abstraction, or be [bound to structured objects](#_bookmark114) through @ConfigurationProperties.

Spring Boot uses a very particular PropertySource order that is designed to allow sensible overriding of values. Properties are considered in the following order:

1. [Devtools global settings properties](#_bookmark77) in the $HOME/.config/spring-boot directory when devtools is active.
2. [@TestPropertySource](https://docs.spring.io/spring/docs/5.2.9.RELEASE/javadoc-api/org/springframework/test/context/TestPropertySource.html) annotations on your tests.
3. properties attribute on your tests. Available on [@SpringBootTest](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/test/context/SpringBootTest.html) and the [test annotations for](#_bookmark316) [testing a particular slice of your application](#_bookmark316).
4. Command line arguments.
5. Properties from SPRING\_APPLICATION\_JSON (inline JSON embedded in an environment variable or system property).
6. ServletConfig init parameters.
7. ServletContext init parameters.
8. JNDI attributes from java:comp/env.
9. Java System properties (System.getProperties()).
10. OS environment variables.
11. A RandomValuePropertySource that has properties only in random.\*.
12. [Profile-specific application properties](#_bookmark106) outside of your packaged jar (application-

{profile}.properties and YAML variants).

1. [Profile-specific application properties](#_bookmark106) packaged inside your jar (application-

{profile}.properties and YAML variants).

1. [Application properties](#_bookmark105) outside of your packaged jar (application.properties and YAML variants).
2. [Application properties](#_bookmark105) packaged inside your jar (application.properties and YAML variants).
3. [@PropertySource](https://docs.spring.io/spring/docs/5.2.9.RELEASE/javadoc-api/org/springframework/context/annotation/PropertySource.html) annotations on your @Configuration classes. Please note that such property sources are not added to the Environment until the application context is being refreshed. This is too late to configure certain properties such as logging.\* and spring.main.\* which are read before refresh begins.
4. Default properties (specified by setting SpringApplication.setDefaultProperties).

To provide a concrete example, suppose you develop a @Component that uses a name property, as shown in the following example:

import org.springframework.stereotype.\*;

import org.springframework.beans.factory.annotation.\*;

@Component

public class MyBean {

@Value("${name}") private String name;

// ...

}

On your application classpath (for example, inside your jar) you can have an application.properties file that provides a sensible default property value for name. When running in a new environment, an application.properties file can be provided outside of your jar that overrides the name. For one-off testing, you can launch with a specific command line switch (for example, java -jar app.jar --name="Spring").

Spring Boot also supports wildcard locations when loading configuration files. By default, a wildcard location of config/\*/ outside of your jar is supported. Wildcard locations are also supported when specifying spring.config.additional-location and spring.config.location.

Wildcard locations are particularly useful in an environment such as Kubernetes when there are multiple sources of config properties. For example, if you have some Redis configuration and some MySQL configuration, you might want to keep those two pieces of configuration separate, while requiring that both those are present in an application.properties that the app can bind to. This might result in two separate application.properties files mounted at different locations such as

/config/redis/application.properties and /config/mysql/application.properties. In such a case,

having a wildcard location of config/\*/, will result in both files being processed.

A wildcard location must contain only one and end with / for search locations that are directories or \*/<filename> for search locations that are files. Locations with wildcards are sorted alphabetically based on the absolute path of the file names.

The SPRING\_APPLICATION\_JSON properties can be supplied on the command line with an environment variable. For example, you could use the following line in a UN\*X shell:

$ SPRING\_APPLICATION\_JSON='{"acme":{"name":"test"}}' java -jar myapp.jar

In the preceding example, you end up with acme.name=test in the Spring Environment. You can also supply the JSON as spring.application.json in a System property, as shown in the following example:

$ java -Dspring.application.json='{"name":"test"}' -jar myapp.jar

You can also supply the JSON by using a command line argument, as shown in the following example:

$ java -jar myapp.jar --spring.application.json='{"name":"test"}'

You can also supply the JSON as a JNDI variable, as follows:

java:comp/env/spring.application.json.

Although null values from the JSON will be added to the resulting property source, the PropertySourcesPropertyResolver treats null properties as missing values. This means that the JSON cannot override properties from lower order property sources with a null value.

### Configuring Random Values

The RandomValuePropertySource is useful for injecting random values (for example, into secrets or test cases). It can produce integers, longs, uuids, or strings, as shown in the following example:

my.secret=${random.value} my.number=${random.int} my.bignumber=${random.long} my.uuid=${random.uuid} my.number.less.than.ten=${random.int(10)} my.number.in.range=${random.int[1024,65536]}

The random.int\* syntax is OPEN value (,max) CLOSE where the OPEN,CLOSE are any character and value,max are integers. If max is provided, then value is the minimum value and max is the maximum value (exclusive).

### Accessing Command Line Properties

By default, SpringApplication converts any command line option arguments (that is, arguments starting with --, such as --server.port=9000) to a property and adds them to the Spring Environment. As mentioned previously, command line properties always take precedence over other property sources.

If you do not want command line properties to be added to the Environment, you can disable them by using SpringApplication.setAddCommandLineProperties(false).

### Application Property Files

SpringApplication loads properties from application.properties files in the following locations and adds them to the Spring Environment:

* + - 1. A /config subdirectory of the current directory
      2. The current directory
      3. A classpath /config package
      4. The classpath root

The list is ordered by precedence (properties defined in locations higher in the list override those defined in lower locations).

 You can also [use YAML ('.yml') files](#_bookmark109) as an alternative to '.properties'.

If you do not like application.properties as the configuration file name, you can switch to another file name by specifying a spring.config.name environment property. You can also refer to an explicit location by using the spring.config.location environment property (which is a comma-separated list of directory locations or file paths). The following example shows how to specify a different file name:

$ java -jar myproject.jar --spring.config.name=myproject

The following example shows how to specify two locations:

$ java -jar myproject.jar

--spring.config.location=classpath:/default.properties,classpath:/override.properties

spring.config.name and spring.config.location are used very early to determine which files have to be loaded. They must be defined as an environment property (typically an OS environment variable, a system property, or a command-line argument).

If spring.config.location contains directories (as opposed to files), they should end in / (and, at runtime, be appended with the names generated from spring.config.name before being loaded, including profile-specific file names). Files specified in spring.config.location are used as-is, with no support for profile-specific variants, and are overridden by any profile-specific properties. Whether specified directly or contained in a directory, configuration files must include a file extension in their name. Typical extensions that are supported out-of-the-box are .properties,

.yaml, and .yml.

Config locations are searched in reverse order. By default, the configured locations are classpath:/,classpath:/config/,file:./,file:./config/\*/,file:./config/. The resulting search order is the following:

1. file:./config/
2. file:./config/\*/
3. file:./
4. classpath:/config/
5. classpath:/

When custom config locations are configured by using spring.config.location, they replace the default locations. For example, if spring.config.location is configured with the value classpath:/custom-config/,file:./custom-config/, the search order becomes the following:

1. file:./custom-config/
2. classpath:custom-config/

Alternatively, when custom config locations are configured by using spring.config.additional- location, they are used in addition to the default locations. Additional locations are searched before the default locations. For example, if additional locations of classpath:/custom- config/,file:./custom-config/ are configured, the search order becomes the following:

1. file:./custom-config/
2. classpath:custom-config/
3. file:./config/
4. file:./config/\*/
5. file:./
6. classpath:/config/
7. classpath:/

This search ordering lets you specify default values in one configuration file and then selectively override those values in another. You can provide default values for your application in application.properties (or whatever other basename you choose with spring.config.name) in one of the default locations. These default values can then be overridden at runtime with a different file located in one of the custom locations.

If you use environment variables rather than system properties, most operating systems disallow period-separated key names, but you can use underscores instead (for example, SPRING\_CONFIG\_NAME instead of spring.config.name). See [Binding from Environment Variables](#_bookmark121) for details.

If your application runs in a container, then JNDI properties (in java:comp/env) or servlet context initialization parameters can be used instead of, or as well as, environment variables or system properties.

### Profile-specific Properties

In addition to application.properties files, profile-specific properties can also be defined by using the following naming convention: application-{profile}.properties. The Environment has a set of default profiles (by default, [default]) that are used if no active profiles are set. In other words, if no profiles are explicitly activated, then properties from application-default.properties are loaded.

Profile-specific properties are loaded from the same locations as standard application.properties, with profile-specific files always overriding the non-specific ones, whether or not the profile- specific files are inside or outside your packaged jar.

If several profiles are specified, a last-wins strategy applies. For example, profiles specified by the spring.profiles.active property are added after those configured through the SpringApplication API and therefore take precedence.

If you have specified any files in spring.config.location, profile-specific variants of those files are not considered. Use directories in spring.config.location if you want to also use profile-specific properties.

### Placeholders in Properties

The values in application.properties are filtered through the existing Environment when they are used, so you can refer back to previously defined values (for example, from System properties).

app.name=MyApp

app.description=${app.name} is a Spring Boot application

You can also use this technique to create “short” variants of existing Spring Boot properties. See the [*Use ‘Short’ Command Line Arguments*](#_bookmark538) how-to for details.

### Encrypting Properties

Spring Boot does not provide any built in support for encrypting property values, however, it does provide the hook points necessary to modify values contained in the Spring Environment. The EnvironmentPostProcessor interface allows you to manipulate the Environment before the application starts. See [Customize the Environment or ApplicationContext Before It Starts](#_bookmark529) for details.

If you’re looking for a secure way to store credentials and passwords, the [Spring Cloud Vault](https://cloud.spring.io/spring-cloud-vault/) project provides support for storing externalized configuration in [HashiCorp Vault](https://www.vaultproject.io/).

### Using YAML Instead of Properties

[YAML](https://yaml.org/) is a superset of JSON and, as such, is a convenient format for specifying hierarchical configuration data. The SpringApplication class automatically supports YAML as an alternative to properties whenever you have the [SnakeYAML](https://bitbucket.org/asomov/snakeyaml) library on your classpath.

If you use “Starters”, SnakeYAML is automatically provided by spring-boot- starter.

#### Loading YAML

Spring Framework provides two convenient classes that can be used to load YAML documents. The YamlPropertiesFactoryBean loads YAML as Properties and the YamlMapFactoryBean loads YAML as a Map.

For example, consider the following YAML document:

environments: dev:

url: https://dev.example.com name: Developer Setup

prod:

url: https://another.example.com name: My Cool App

The preceding example would be transformed into the following properties:

environments.dev.url=https://dev.example.com environments.dev.name=Developer Setup environments.prod.url=https://another.example.com environments.prod.name=My Cool App

YAML lists are represented as property keys with [index] dereferencers. For example, consider the following YAML:

my:

servers:

* dev.example.com
* another.example.com

The preceding example would be transformed into these properties:

my.servers[0]=dev.example.com my.servers[1]=another.example.com

To bind to properties like that by using Spring Boot’s Binder utilities (which is what @ConfigurationProperties does), you need to have a property in the target bean of type java.util.List (or Set) and you either need to provide a setter or initialize it with a mutable value. For example, the following example binds to the properties shown previously:

@ConfigurationProperties(prefix="my") public class Config {

private List<String> servers = new ArrayList<String>(); public List<String> getServers() {

return this.servers;

}

}

#### Exposing YAML as Properties in the Spring Environment

The YamlPropertySourceLoader class can be used to expose YAML as a PropertySource in the Spring Environment. Doing so lets you use the @Value annotation with placeholders syntax to access YAML properties.

#### Multi-profile YAML Documents

You can specify multiple profile-specific YAML documents in a single file by using a spring.profiles

key to indicate when the document applies, as shown in the following example:

server:

address: 192.168.1.100

---

spring:

profiles: development server:

address: 127.0.0.1

---

spring:

profiles: production & eu-central server:

address: 192.168.1.120

In the preceding example, if the development profile is active, the server.address property is

127.0.0.1. Similarly, if the production **and** eu-central profiles are active, the server.address property is 192.168.1.120. If the development, production and eu-central profiles are **not** enabled, then the value for the property is 192.168.1.100.

spring.profiles can therefore contain a profile name (for example production) or a profile expression. A profile expression allows for more complicated profile logic to be expressed, for example production & (eu-central | eu-west). Check the [reference guide](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/core.html#beans-definition-profiles-java) for more details.

If none are explicitly active when the application context starts, the default profiles are activated. So, in the following YAML, we set a value for spring.security.user.password that is available **only** in the "default" profile:

server: port: 8000

---

spring:

profiles: default security:

user:

password: weak

Whereas, in the following example, the password is always set because it is not attached to any profile, and it would have to be explicitly reset in all other profiles as necessary:

server: port: 8000

spring: security:

user:

password: weak

Spring profiles designated by using the spring.profiles element may optionally be negated by using the ! character. If both negated and non-negated profiles are specified for a single document, at least one non-negated profile must match, and no negated profiles may match.

#### YAML Shortcomings

YAML files cannot be loaded by using the @PropertySource annotation. So, in the case that you need to load values that way, you need to use a properties file.

Using the multi YAML document syntax in profile-specific YAML files can lead to unexpected behavior. For example, consider the following config in a file:

*application-dev.yml*

server: port: 8000

---

spring:

profiles: "!test" security:

user:

password: "secret"

If you run the application with the argument --spring.profiles.active=dev you might expect

security.user.password to be set to “secret”, but this is not the case.

The nested document will be filtered because the main file is named application-dev.yml. It is already considered to be profile-specific, and nested documents will be ignored.

We recommend that you don’t mix profile-specific YAML files and multiple YAML documents. Stick to using only one of them.

### Type-safe Configuration Properties

Using the @Value("${property}") annotation to inject configuration properties can sometimes be cumbersome, especially if you are working with multiple properties or your data is hierarchical in nature. Spring Boot provides an alternative method of working with properties that lets strongly typed beans govern and validate the configuration of your application.

 See also the [differences between @Value and type-safe configuration properties](#_bookmark125).

#### JavaBean properties binding

It is possible to bind a bean declaring standard JavaBean properties as shown in the following example:

package com.example;

import java.net.InetAddress; import java.util.ArrayList; import java.util.Collections; import java.util.List;

import org.springframework.boot.context.properties.ConfigurationProperties; @ConfigurationProperties("acme")

public class AcmeProperties {

private boolean enabled;

private InetAddress remoteAddress;

private final Security security = new Security(); public boolean isEnabled() { ... }

public void setEnabled(boolean enabled) { ... } public InetAddress getRemoteAddress() { ... }

public void setRemoteAddress(InetAddress remoteAddress) { ... } public Security getSecurity() { ... }

public static class Security { private String username; private String password;

private List<String> roles = new ArrayList<>(Collections.singleton("USER")); public String getUsername() { ... }

public void setUsername(String username) { ... } public String getPassword() { ... }

public void setPassword(String password) { ... } public List<String> getRoles() { ... }

public void setRoles(List<String> roles) { ... }

}

}

The preceding POJO defines the following properties:

* acme.enabled, with a value of false by default.
* acme.remote-address, with a type that can be coerced from String.
* acme.security.username, with a nested "security" object whose name is determined by the name of the property. In particular, the return type is not used at all there and could have been SecurityProperties.
* acme.security.password.
* acme.security.roles, with a collection of String that defaults to USER.

The properties that map to @ConfigurationProperties classes available in Spring Boot, which are configured via properties files, YAML files, environment variables etc., are public API but the accessors (getters/setters) of the class itself are not meant to be used directly.

Such arrangement relies on a default empty constructor and getters and setters are usually mandatory, since binding is through standard Java Beans property descriptors, just like in Spring MVC. A setter may be omitted in the following cases:

* + Maps, as long as they are initialized, need a getter but not necessarily a setter, since they can be mutated by the binder.
  + Collections and arrays can be accessed either through an index (typically with YAML) or by using a single comma-separated value (properties). In the latter case, a setter is mandatory. We recommend to always add a setter for such types. If you initialize a collection, make sure it is not immutable (as in the preceding example).
  + If nested POJO properties are initialized (like the Security field in the preceding example), a setter is not required. If you want the binder to create the instance on the fly by using its default constructor, you need a setter.

Some people use Project Lombok to add getters and setters automatically. Make sure that Lombok does not generate any particular constructor for such a type, as it is used automatically by the container to instantiate the object.

Finally, only standard Java Bean properties are considered and binding on static properties is not supported.

#### Constructor binding

The example in the previous section can be rewritten in an immutable fashion as shown in the following example:

package com.example;

import java.net.InetAddress; import java.util.List;

import org.springframework.boot.context.properties.ConfigurationProperties; import org.springframework.boot.context.properties.ConstructorBinding; import org.springframework.boot.context.properties.bind.DefaultValue;

@ConstructorBinding @ConfigurationProperties("acme") public class AcmeProperties {

private final boolean enabled;

private final InetAddress remoteAddress; private final Security security;

public AcmeProperties(boolean enabled, InetAddress remoteAddress, Security security) {

this.enabled = enabled; this.remoteAddress = remoteAddress; this.security = security;

}

public boolean isEnabled() { ... }

public InetAddress getRemoteAddress() { ... } public Security getSecurity() { ... }

public static class Security { private final String username; private final String password; private final List<String> roles;

public Security(String username, String password, @DefaultValue("USER") List<String> roles) {

this.username = username; this.password = password; this.roles = roles;

}

public String getUsername() { ... } public String getPassword() { ... } public List<String> getRoles() { ... }

}

}

In this setup, the @ConstructorBinding annotation is used to indicate that constructor binding should be used. This means that the binder will expect to find a constructor with the parameters that you wish to have bound.

Nested members of a @ConstructorBinding class (such as Security in the example above) will also be bound via their constructor.

Default values can be specified using @DefaultValue and the same conversion service will be applied to coerce the String value to the target type of a missing property. By default, if no properties are bound to Security, the AcmeProperties instance will contain a null value for security. If you wish you return a non-null instance of Security even when no properties are bound to it, you can use an empty @DefaultValue annotation to do so:

package com.example;

import java.net.InetAddress; import java.util.List;

import org.springframework.boot.context.properties.ConfigurationProperties; import org.springframework.boot.context.properties.ConstructorBinding; import org.springframework.boot.context.properties.bind.DefaultValue;

@ConstructorBinding @ConfigurationProperties("acme") public class AcmeProperties {

private final boolean enabled;

private final InetAddress remoteAddress; private final Security security;

public AcmeProperties(boolean enabled, InetAddress remoteAddress, @DefaultValue Security security) {

this.enabled = enabled; this.remoteAddress = remoteAddress; this.security = security;

}

}

To use constructor binding the class must be enabled using @EnableConfigurationProperties or configuration property scanning. You cannot use constructor binding with beans that are created by the regular Spring mechanisms (e.g. @Component beans, beans created via @Bean methods or beans loaded using @Import)

If you have more than one constructor for your class you can also use

@ConstructorBinding directly on the constructor that should be bound.

The use of java.util.Optional with @ConfigurationProperties is not recommended as it is primarily intended for use as a return type. As such, it is not well-suited to configuration property injection. For consistency with properties of other types, if you do declare an Optional property and it has no value, null rather than an empty Optional will be bound.

#### Enabling @ConfigurationProperties-annotated types

Spring Boot provides infrastructure to bind @ConfigurationProperties types and register them as beans. You can either enable configuration properties on a class-by-class basis or enable configuration property scanning that works in a similar manner to component scanning.

Sometimes, classes annotated with @ConfigurationProperties might not be suitable for scanning, for example, if you’re developing your own auto-configuration or you want to enable them conditionally. In these cases, specify the list of types to process using the @EnableConfigurationProperties annotation. This can be done on any @Configuration class, as shown in the following example:

@Configuration(proxyBeanMethods = false) @EnableConfigurationProperties(AcmeProperties.class) public class MyConfiguration {

}

To use configuration property scanning, add the @ConfigurationPropertiesScan annotation to your application. Typically, it is added to the main application class that is annotated with @SpringBootApplication but it can be added to any @Configuration class. By default, scanning will occur from the package of the class that declares the annotation. If you want to define specific packages to scan, you can do so as shown in the following example:

@SpringBootApplication

@ConfigurationPropertiesScan({ "com.example.app", "org.acme.another" }) public class MyApplication {

}

When the @ConfigurationProperties bean is registered using configuration property scanning or via @EnableConfigurationProperties, the bean has a conventional name: <prefix>-<fqn>, where <prefix> is the environment key prefix specified in the @ConfigurationProperties annotation and <fqn> is the fully qualified name of the bean. If the annotation does not provide any prefix, only the fully qualified name of the bean is used.

The bean name in the example above is acme-com.example.AcmeProperties.

We recommend that @ConfigurationProperties only deal with the environment and, in particular, does not inject other beans from the context. For corner cases, setter injection can be used or any of the \*Aware interfaces provided by the framework (such as EnvironmentAware if you need access to the Environment). If you still want to inject other beans using the constructor, the configuration properties bean must be annotated with @Component and use JavaBean-based property binding.

#### Using @ConfigurationProperties-annotated types

This style of configuration works particularly well with the SpringApplication external YAML configuration, as shown in the following example:

# application.yml

acme:

remote-address: 192.168.1.1 security:

username: admin roles:

* USER
* ADMIN

# additional configuration as required

To work with @ConfigurationProperties beans, you can inject them in the same way as any other bean, as shown in the following example:

@Service

public class MyService {

private final AcmeProperties properties; @Autowired

public MyService(AcmeProperties properties) { this.properties = properties;

}

//... @PostConstruct

public void openConnection() {

Server server = new Server(this.properties.getRemoteAddress());

// ...

}

}

Using @ConfigurationProperties also lets you generate metadata files that can be used by IDEs to offer auto-completion for your own keys. See the [appendix](#_bookmark673) for details.

#### Third-party Configuration

As well as using @ConfigurationProperties to annotate a class, you can also use it on public @Bean methods. Doing so can be particularly useful when you want to bind properties to third-party components that are outside of your control.

To configure a bean from the Environment properties, add @ConfigurationProperties to its bean registration, as shown in the following example:

@ConfigurationProperties(prefix = "another") @Bean

public AnotherComponent anotherComponent() {

...

}

Any JavaBean property defined with the another prefix is mapped onto that AnotherComponent bean in manner similar to the preceding AcmeProperties example.

#### Relaxed Binding

Spring Boot uses some relaxed rules for binding Environment properties to @ConfigurationProperties beans, so there does not need to be an exact match between the Environment property name and the bean property name. Common examples where this is useful include dash-separated environment properties (for example, context-path binds to contextPath), and capitalized environment properties (for example, PORT binds to port).

As an example, consider the following @ConfigurationProperties class:

@ConfigurationProperties(prefix="acme.my-project.person") public class OwnerProperties {

private String firstName; public String getFirstName() {

return this.firstName;

}

public void setFirstName(String firstName) { this.firstName = firstName;

}

}

With the preceding code, the following properties names can all be used:

*Table 5. relaxed binding*

|  |  |
| --- | --- |
| **Property** | **Note** |
| acme.my- project.person.fir st-name | Kebab case, which is recommended for use in .properties and .yml files. |
| acme.myProject.per son.firstName | Standard camel case syntax. |
| acme.my\_project.pe rson.first\_name | Underscore notation, which is an alternative format for use in .properties and  .yml files. |
| ACME\_MYPROJECT\_PER SON\_FIRSTNAME | Upper case format, which is recommended when using system environment variables. |

The prefix value for the annotation *must* be in kebab case (lowercase and separated by -, such as acme.my-project.person).

*Table 6. relaxed binding rules per property source*

|  |  |  |
| --- | --- | --- |
| **Property Source** | **Simple** | **List** |
| Properties Files | Camel case, kebab case, or underscore notation | Standard list syntax using [ ] or comma-separated values |
| YAML Files | Camel case, kebab case, or underscore notation | Standard YAML list syntax or comma- separated values |
| Environment Variables | Upper case format with underscore as the delimiter (see [Binding from](#_bookmark121) [Environment Variables](#_bookmark121)). | Numeric values surrounded by underscores (see [Binding from](#_bookmark121) [Environment Variables](#_bookmark121))` |
| System properties | Camel case, kebab case, or underscore notation | Standard list syntax using [ ] or comma-separated values |

We recommend that, when possible, properties are stored in lower-case kebab format, such as my.property-name=acme.

**Binding Maps**

When binding to Map properties, if the key contains anything other than lowercase alpha-numeric characters or -, you need to use the bracket notation so that the original value is preserved. If the key is not surrounded by [], any characters that are not alpha-numeric or - are removed. For example, consider binding the following properties to a Map:

acme:

map:

"[/key1]": value1

"[/key2]": value2

/key3: value3

The properties above will bind to a Map with /key1, /key2 and key3 as the keys in the map.

For YAML files, the brackets need to be surrounded by quotes for the keys to be parsed properly.

**Binding from Environment Variables**

Most operating systems impose strict rules around the names that can be used for environment variables. For example, Linux shell variables can contain only letters (a to z or A to Z), numbers (0 to 9) or the underscore character (\_). By convention, Unix shell variables will also have their names in UPPERCASE.

Spring Boot’s relaxed binding rules are, as much as possible, designed to be compatible with these naming restrictions.

To convert a property name in the canonical-form to an environment variable name you can follow these rules:

* Replace dots (.) with underscores (\_).
* Remove any dashes (-).
* Convert to uppercase.

For example, the configuration property spring.main.log-startup-info would be an environment variable named SPRING\_MAIN\_LOGSTARTUPINFO.

Underscores cannot be used to replace the dashes in property names. If you attempt to use SPRING\_MAIN\_LOG\_STARTUP\_INFO with the example above, no value will be bound.

Environment variables can also be used when binding to object lists. To bind to a List, the element number should be surrounded with underscores in the variable name.

For example, the configuration property my.acme[0].other would use an environment variable named MY\_ACME\_0\_OTHER.

#### Merging Complex Types

When lists are configured in more than one place, overriding works by replacing the entire list.

For example, assume a MyPojo object with name and description attributes that are null by default. The following example exposes a list of MyPojo objects from AcmeProperties:

@ConfigurationProperties("acme") public class AcmeProperties {

private final List<MyPojo> list = new ArrayList<>(); public List<MyPojo> getList() {

return this.list;

}

}

Consider the following configuration:

acme:

list:

* name: my name

description: my description

---

spring: profiles: dev

acme:

list:

* name: my another name

If the dev profile is not active, AcmeProperties.list contains one MyPojo entry, as previously defined. If the dev profile is enabled, however, the list *still* contains only one entry (with a name of my another name and a description of null). This configuration *does not* add a second MyPojo instance to the list, and it does not merge the items.

When a List is specified in multiple profiles, the one with the highest priority (and only that one) is used. Consider the following example:

acme:

list:

* name: my name

description: my description

* name: another name

description: another description

---

spring: profiles: dev

acme:

list:

* name: my another name

In the preceding example, if the dev profile is active, AcmeProperties.list contains *one* MyPojo entry (with a name of my another name and a description of null). For YAML, both comma-separated lists

and YAML lists can be used for completely overriding the contents of the list.

For Map properties, you can bind with property values drawn from multiple sources. However, for the same property in multiple sources, the one with the highest priority is used. The following example exposes a Map<String, MyPojo> from AcmeProperties:

@ConfigurationProperties("acme") public class AcmeProperties {

private final Map<String, MyPojo> map = new HashMap<>(); public Map<String, MyPojo> getMap() {

return this.map;

}

}

Consider the following configuration:

acme:

map:

key1:

name: my name 1

description: my description 1

---

spring: profiles: dev

acme:

map:

key1:

name: dev name 1 key2:

name: dev name 2

description: dev description 2

If the dev profile is not active, AcmeProperties.map contains one entry with key key1 (with a name of my name 1 and a description of my description 1). If the dev profile is enabled, however, map contains two entries with keys key1 (with a name of dev name 1 and a description of my description 1) and key2 (with a name of dev name 2 and a description of dev description 2).

The preceding merging rules apply to properties from all property sources and not just YAML files.

#### Properties Conversion

Spring Boot attempts to coerce the external application properties to the right type when it binds to the @ConfigurationProperties beans. If you need custom type conversion, you can provide a ConversionService bean (with a bean named conversionService) or custom property editors (through

a CustomEditorConfigurer bean) or custom Converters (with bean definitions annotated as

@ConfigurationPropertiesBinding).

As this bean is requested very early during the application lifecycle, make sure to limit the dependencies that your ConversionService is using. Typically, any dependency that you require may not be fully initialized at creation time. You may want to rename your custom ConversionService if it is not required for configuration keys coercion and only rely on custom converters qualified with @ConfigurationPropertiesBinding.

**Converting durations**

Spring Boot has dedicated support for expressing durations. If you expose a java.time.Duration

property, the following formats in application properties are available:

* A regular long representation (using milliseconds as the default unit unless a @DurationUnit has been specified)
* The standard ISO-8601 format [used by java.time.Duration](https://docs.oracle.com/javase/8/docs/api/java/time/Duration.html#parse-java.lang.CharSequence-)
* A more readable format where the value and the unit are coupled (e.g. 10s means 10 seconds) Consider the following example:

@ConfigurationProperties("app.system") public class AppSystemProperties {

@DurationUnit(ChronoUnit.SECONDS)

private Duration sessionTimeout = Duration.ofSeconds(30);

private Duration readTimeout = Duration.ofMillis(1000); public Duration getSessionTimeout() {

return this.sessionTimeout;

}

public void setSessionTimeout(Duration sessionTimeout) { this.sessionTimeout = sessionTimeout;

}

public Duration getReadTimeout() { return this.readTimeout;

}

public void setReadTimeout(Duration readTimeout) { this.readTimeout = readTimeout;

}

}

To specify a session timeout of 30 seconds, 30, PT30S and 30s are all equivalent. A read timeout of 500ms can be specified in any of the following form: 500, PT0.5S and 500ms.

You can also use any of the supported units. These are:

* ns for nanoseconds
* us for microseconds
* ms for milliseconds
* s for seconds
* m for minutes
* h for hours
* d for days

The default unit is milliseconds and can be overridden using @DurationUnit as illustrated in the sample above. Note that @DurationUnit is only supported with JavaBean-style property binding using getters and setters. It is not supported with constructor binding.

If you are upgrading a Long property, make sure to define the unit (using @DurationUnit) if it isn’t milliseconds. Doing so gives a transparent upgrade path while supporting a much richer format.

**Converting periods**

In addition to durations, Spring Boot can also work with java.time.Period type. The following formats can be used in application properties:

* An regular int representation (using days as the default unit unless a @PeriodUnit has been specified)
* The standard ISO-8601 format [used by java.time.Period](https://docs.oracle.com/javase/8/docs/api/java/time/Period.html#parse-java.lang.CharSequence-)
* A simpler format where the value and the unit pairs are coupled (e.g. 1y3d means 1 year and 3 days)

The following units are supported with the simple format:

* y for years
* m for months
* w for weeks
* d for days

The java.time.Period type never actually stores the number of weeks, it is a shortcut that means “7 days”.

**Converting Data Sizes**

Spring Framework has a DataSize value type that expresses a size in bytes. If you expose a DataSize

property, the following formats in application properties are available:

* A regular long representation (using bytes as the default unit unless a @DataSizeUnit has been specified)
* A more readable format where the value and the unit are coupled (e.g. 10MB means 10 megabytes)

Consider the following example:

@ConfigurationProperties("app.io") public class AppIoProperties {

@DataSizeUnit(DataUnit.MEGABYTES)

private DataSize bufferSize = DataSize.ofMegabytes(2); private DataSize sizeThreshold = DataSize.ofBytes(512);

public DataSize getBufferSize() { return this.bufferSize;

}

public void setBufferSize(DataSize bufferSize) { this.bufferSize = bufferSize;

}

public DataSize getSizeThreshold() { return this.sizeThreshold;

}

public void setSizeThreshold(DataSize sizeThreshold) { this.sizeThreshold = sizeThreshold;

}

}

To specify a buffer size of 10 megabytes, 10 and 10MB are equivalent. A size threshold of 256 bytes can be specified as 256 or 256B.

You can also use any of the supported units. These are:

* B for bytes
* KB for kilobytes
* MB for megabytes
* GB for gigabytes
* TB for terabytes

The default unit is bytes and can be overridden using @DataSizeUnit as illustrated in the sample above.

If you are upgrading a Long property, make sure to define the unit (using @DataSizeUnit) if it isn’t bytes. Doing so gives a transparent upgrade path while supporting a much richer format.

#### @ConfigurationProperties Validation

Spring Boot attempts to validate @ConfigurationProperties classes whenever they are annotated with Spring’s @Validated annotation. You can use JSR-303 javax.validation constraint annotations directly on your configuration class. To do so, ensure that a compliant JSR-303 implementation is on your classpath and then add constraint annotations to your fields, as shown in the following example:

@ConfigurationProperties(prefix="acme") @Validated

public class AcmeProperties {

@NotNull

private InetAddress remoteAddress;

// ... getters and setters

}

You can also trigger validation by annotating the @Bean method that creates the configuration properties with @Validated.

To ensure that validation is always triggered for nested properties, even when no properties are found, the associated field must be annotated with @Valid. The following example builds on the preceding AcmeProperties example:

@ConfigurationProperties(prefix="acme") @Validated

public class AcmeProperties {

@NotNull

private InetAddress remoteAddress;

@Valid

private final Security security = new Security();

// ... getters and setters public static class Security {

@NotEmpty

public String username;

// ... getters and setters

}

}

You can also add a custom Spring Validator by creating a bean definition called configurationPropertiesValidator. The @Bean method should be declared static. The configuration properties validator is created very early in the application’s lifecycle, and declaring the @Bean method as static lets the bean be created without having to instantiate the @Configuration class. Doing so avoids any problems that may be caused by early instantiation.

The spring-boot-actuator module includes an endpoint that exposes all @ConfigurationProperties beans. Point your web browser to /actuator/configprops or use the equivalent JMX endpoint. See the "[Production ready features](#_bookmark380)" section for details.

#### @ConfigurationProperties vs. @Value

The @Value annotation is a core container feature, and it does not provide the same features as type- safe configuration properties. The following table summarizes the features that are supported by @ConfigurationProperties and @Value:

|  |  |  |
| --- | --- | --- |
| **Feature** | @ConfigurationProperti es | @Value |
| [Relaxed binding](#_bookmark120) | Yes | Limited (see note below) |
| [Meta-data support](#_bookmark673) | Yes | No |
| SpEL evaluation | No | Yes |

If you define a set of configuration keys for your own components, we recommend you group them in a POJO annotated with @ConfigurationProperties. Doing so will provide you with structured, type-safe object that you can inject into your own beans.

If you do want to use @Value, we recommend that you refer to property names using their canonical form (kebab-case using only lowercase letters). This will allow Spring Boot to use the same logic as it does when relaxed binding @ConfigurationProperties. For example, @Value("{demo.item-price}") will pick up demo.item-price and demo.itemPrice forms from the application.properties file, as well as DEMO\_ITEMPRICE from the system environment. If you used @Value("{demo.itemPrice}") instead, demo.item-price and DEMO\_ITEMPRICE would not be considered.

Finally, while you can write a SpEL expression in @Value, such expressions are not processed from [application property files](#_bookmark105).

## Profiles

Spring Profiles provide a way to segregate parts of your application configuration and make it be available only in certain environments. Any @Component, @Configuration or @ConfigurationProperties can be marked with @Profile to limit when it is loaded, as shown in the following example:

@Configuration(proxyBeanMethods = false) @Profile("production")

public class ProductionConfiguration {

// ...

}

If @ConfigurationProperties beans are registered via @EnableConfigurationProperties instead of automatic scanning, the @Profile annotation needs to be specified on the @Configuration class that has the @EnableConfigurationProperties annotation. In the case where @ConfigurationProperties are scanned, @Profile can be specified on the @ConfigurationProperties class itself.

You can use a spring.profiles.active Environment property to specify which profiles are active. You can specify the property in any of the ways described earlier in this chapter. For example, you could include it in your application.properties, as shown in the following example:

spring.profiles.active=dev,hsqldb

You could also specify it on the command line by using the following switch:

--spring.profiles.active=dev,hsqldb.

### Adding Active Profiles

The spring.profiles.active property follows the same ordering rules as other properties: The highest PropertySource wins. This means that you can specify active profiles in application.properties and then **replace** them by using the command line switch.

Sometimes, it is useful to have profile-specific properties that **add** to the active profiles rather than replace them. The spring.profiles.include property can be used to unconditionally add active profiles. The SpringApplication entry point also has a Java API for setting additional profiles (that is, on top of those activated by the spring.profiles.active property). See the setAdditionalProfiles() method in [SpringApplication](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/SpringApplication.html).

For example, when an application with the following properties is run by using the switch,

--spring.profiles.active=prod, the proddb and prodmq profiles are also activated:

---

my.property: fromyamlfile

---

spring.profiles: prod spring.profiles.include:

* proddb
* prodmq

Remember that the spring.profiles property can be defined in a YAML document to determine when this particular document is included in the configuration. See [Change Configuration Depending on the Environment](#_bookmark541) for more details.

### Programmatically Setting Profiles

You can programmatically set active profiles by calling SpringApplication.setAdditionalProfiles(…

) before your application runs. It is also possible to activate profiles by using Spring’s

ConfigurableEnvironment interface.

### Profile-specific Configuration Files

Profile-specific variants of both application.properties (or application.yml) and files referenced through @ConfigurationProperties are considered as files and loaded. See "[Profile-specific](#_bookmark106) [Properties](#_bookmark106)" for details.

## Logging

Spring Boot uses [Commons Logging](https://commons.apache.org/logging) for all internal logging but leaves the underlying log implementation open. Default configurations are provided for [Java Util Logging](https://docs.oracle.com/javase/8/docs/api/java/util/logging/package-summary.html), [Log4J2](https://logging.apache.org/log4j/2.x/), and [Logback](https://logback.qos.ch/). In each case, loggers are pre-configured to use console output with optional file output also available.

By default, if you use the “Starters”, Logback is used for logging. Appropriate Logback routing is

also included to ensure that dependent libraries that use Java Util Logging, Commons Logging, Log4J, or SLF4J all work correctly.

There are a lot of logging frameworks available for Java. Do not worry if the above list seems confusing. Generally, you do not need to change your logging dependencies and the Spring Boot defaults work just fine.

When you deploy your application to a servlet container or application server, logging performed via the Java Util Logging API is not routed into your application’s logs. This prevents logging performed by the container or other applications that have been deployed to it from appearing in your application’s logs.

### Log Format

The default log output from Spring Boot resembles the following example:

2019-03-05 10:57:51.112 INFO 45469 --- [

main]

org.apache.catalina.core.StandardEngine : Starting Servlet Engine: Apache Tomcat/7.0.52

2019-03-05 10:57:51.253 INFO 45469 --- [ost-startStop-1]

o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext

2019-03-05 10:57:51.253 INFO 45469 --- [ost-startStop-1]

o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 1358 ms

2019-03-05 10:57:51.698 INFO 45469 --- [ost-startStop-1]

o.s.b.c.e.ServletRegistrationBean : Mapping servlet: 'dispatcherServlet' to [/] 2019-03-05 10:57:51.702 INFO 45469 --- [ost-startStop-1]

o.s.b.c.embedded.FilterRegistrationBean : Mapping filter: 'hiddenHttpMethodFilter' to: [/\*]

The following items are output:

* Date and Time: Millisecond precision and easily sortable.
* Log Level: ERROR, WARN, INFO, DEBUG, or TRACE.
* Process ID.
* A --- separator to distinguish the start of actual log messages.
* Thread name: Enclosed in square brackets (may be truncated for console output).
* Logger name: This is usually the source class name (often abbreviated).
* The log message.

 Logback does not have a FATAL level. It is mapped to ERROR.

### Console Output

The default log configuration echoes messages to the console as they are written. By default, ERROR

-level, WARN-level, and INFO-level messages are logged. You can also enable a “debug” mode by starting your application with a --debug flag.

$ java -jar myapp.jar --debug

 You can also specify debug=true in your application.properties.

When the debug mode is enabled, a selection of core loggers (embedded container, Hibernate, and Spring Boot) are configured to output more information. Enabling the debug mode does *not* configure your application to log all messages with DEBUG level.

Alternatively, you can enable a “trace” mode by starting your application with a --trace flag (or trace=true in your application.properties). Doing so enables trace logging for a selection of core loggers (embedded container, Hibernate schema generation, and the whole Spring portfolio).

#### Color-coded Output

If your terminal supports ANSI, color output is used to aid readability. You can set

spring.output.ansi.enabled to a [supported value](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/ansi/AnsiOutput.Enabled.html) to override the auto-detection.

Color coding is configured by using the %clr conversion word. In its simplest form, the converter colors the output according to the log level, as shown in the following example:

%clr(%5p)

The following table describes the mapping of log levels to colors:

|  |  |
| --- | --- |
| **Level** | **Color** |
| FATAL | Red |
| ERROR | Red |
| WARN | Yellow |
| INFO | Green |
| DEBUG | Green |
| TRACE | Green |

Alternatively, you can specify the color or style that should be used by providing it as an option to the conversion. For example, to make the text yellow, use the following setting:

%clr(%d{yyyy-MM-dd HH:mm:ss.SSS}){yellow}

The following colors and styles are supported:

* blue
* cyan
* faint
* green
* magenta
* red
* yellow

### File Output

By default, Spring Boot logs only to the console and does not write log files. If you want to write log files in addition to the console output, you need to set a logging.file.name or logging.file.path property (for example, in your application.properties).

The following table shows how the logging.\* properties can be used together:

*Table 7. Logging properties*

|  |  |  |  |
| --- | --- | --- | --- |
| logging.file  .name | logging.file  .path | **Example** | **Description** |
| *(none)* | *(none)* |  | Console only logging. |
| Specific file | *(none)* | my.log | Writes to the specified log file. Names can be an exact location or relative to the current directory. |
| *(none)* | Specific directory | /var/log | Writes spring.log to the specified directory. Names can be an exact location or relative to the current directory. |

Log files rotate when they reach 10 MB and, as with console output, ERROR-level, WARN-level, and INFO

-level messages are logged by default. Size limits can be changed using the logging.file.max-size property. Rotated log files of the last 7 days are kept by default unless the logging.file.max-history property has been set. The total size of log archives can be capped using logging.file.total-size- cap. When the total size of log archives exceeds that threshold, backups will be deleted. To force log archive cleanup on application startup, use the logging.file.clean-history-on-start property.

Logging properties are independent of the actual logging infrastructure. As a result, specific configuration keys (such as logback.configurationFile for Logback) are not managed by spring Boot.

### Log Levels

All the supported logging systems can have the logger levels set in the Spring Environment (for example, in application.properties) by using logging.level.<logger-name>=<level> where level is one of TRACE, DEBUG, INFO, WARN, ERROR, FATAL, or OFF. The root logger can be configured by using logging.level.root.

The following example shows potential logging settings in application.properties:

logging.level.root=warn logging.level.org.springframework.web=debug logging.level.org.hibernate=error

It’s also possible to set logging levels using environment variables. For example,

LOGGING\_LEVEL\_ORG\_SPRINGFRAMEWORK\_WEB=DEBUG will set org.springframework.web to DEBUG.

The above approach will only work for package level logging. Since relaxed binding always converts environment variables to lowercase, it’s not possible to configure logging for an individual class in this way. If you need to configure logging for a class, you can use [the SPRING\_APPLICATION\_JSON](#_bookmark102) variable.

### Log Groups

It’s often useful to be able to group related loggers together so that they can all be configured at the same time. For example, you might commonly change the logging levels for *all* Tomcat related loggers, but you can’t easily remember top level packages.

To help with this, Spring Boot allows you to define logging groups in your Spring Environment. For example, here’s how you could define a “tomcat” group by adding it to your application.properties:

logging.group.tomcat=org.apache.catalina, org.apache.coyote, org.apache.tomcat

Once defined, you can change the level for all the loggers in the group with a single line:

logging.level.tomcat=TRACE

Spring Boot includes the following pre-defined logging groups that can be used out-of-the-box:

|  |  |
| --- | --- |
| **Name** | **Loggers** |
| web | org.springframework.core.codec, org.springframework.http,  org.springframework.web, org.springframework.boot.actuate.endpoint.web, org.springframework.boot.web.servlet.ServletContextInitializerBeans |
| sql | org.springframework.jdbc.core, org.hibernate.SQL, org.jooq.tools.LoggerListener |

### Custom Log Configuration

The various logging systems can be activated by including the appropriate libraries on the classpath and can be further customized by providing a suitable configuration file in the root of the classpath or in a location specified by the following Spring Environment property: logging.config.

You can force Spring Boot to use a particular logging system by using the org.springframework.boot.logging.LoggingSystem system property. The value should be the fully qualified class name of a LoggingSystem implementation. You can also disable Spring Boot’s logging

configuration entirely by using a value of none.

Since logging is initialized **before** the ApplicationContext is created, it is not possible to control logging from @PropertySources in Spring @Configuration files. The only way to change the logging system or disable it entirely is via System properties.

Depending on your logging system, the following files are loaded:

|  |  |
| --- | --- |
| **Logging System** | **Customization** |
| Logback | logback-spring.xml, logback-spring.groovy, logback.xml, or logback.groovy |
| Log4j2 | log4j2-spring.xml or log4j2.xml |
| JDK (Java Util Logging) | logging.properties |

When possible, we recommend that you use the -spring variants for your logging configuration (for example, logback-spring.xml rather than logback.xml). If you use standard configuration locations, Spring cannot completely control log initialization.

There are known classloading issues with Java Util Logging that cause problems when running from an 'executable jar'. We recommend that you avoid it when running from an 'executable jar' if at all possible.

To help with the customization, some other properties are transferred from the Spring Environment

to System properties, as described in the following table:

|  |  |  |
| --- | --- | --- |
| **Spring Environment** | **System Property** | **Comments** |
| logging.exception-conversion- word | LOG\_EXCEPTION\_CONVERSION\_WORD | The conversion word used when logging exceptions. |
| logging.file.clean-history-on- start | LOG\_FILE\_CLEAN\_HISTORY\_ON\_STAR T | Whether to clean the archive log files on startup (if LOG\_FILE enabled). (Only supported with the default Logback setup.) |
| logging.file.name | LOG\_FILE | If defined, it is used in the default log configuration. |
| logging.file.max-size | LOG\_FILE\_MAX\_SIZE | Maximum log file size (if LOG\_FILE enabled). (Only supported with the default Logback setup.) |
| logging.file.max-history | LOG\_FILE\_MAX\_HISTORY | Maximum number of archive log files to keep (if LOG\_FILE enabled). (Only supported with the default Logback setup.) |

|  |  |  |
| --- | --- | --- |
| **Spring Environment** | **System Property** | **Comments** |
| logging.file.path | LOG\_PATH | If defined, it is used in the default log configuration. |
| logging.file.total-size-cap | LOG\_FILE\_TOTAL\_SIZE\_CAP | Total size of log backups to be kept (if LOG\_FILE enabled). (Only supported with the default Logback setup.) |
| logging.pattern.console | CONSOLE\_LOG\_PATTERN | The log pattern to use on the console (stdout). (Only supported with the default Logback setup.) |
| logging.pattern.dateformat | LOG\_DATEFORMAT\_PATTERN | Appender pattern for log date format. (Only supported with the default Logback setup.) |
| logging.pattern.file | FILE\_LOG\_PATTERN | The log pattern to use in a file (if LOG\_FILE is enabled). (Only supported with the default Logback setup.) |
| logging.pattern.level | LOG\_LEVEL\_PATTERN | The format to use when rendering the log level (default  %5p). (Only supported with the default Logback setup.) |
| logging.pattern.rolling-file- name | ROLLING\_FILE\_NAME\_PATTERN | Pattern for rolled-over log file names (default  ${LOG\_FILE}.%d{yyyy-MM- dd}.%i.gz). (Only supported with the default Logback setup.) |
| PID | PID | The current process ID (discovered if possible and when not already defined as an OS environment variable). |

All the supported logging systems can consult System properties when parsing their configuration files. See the default configurations in spring-boot.jar for examples:

* [Logback](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot/src/main/resources/org/springframework/boot/logging/logback/defaults.xml)
* [Log4j 2](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot/src/main/resources/org/springframework/boot/logging/log4j2/log4j2.xml)
* [Java Util logging](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot/src/main/resources/org/springframework/boot/logging/java/logging-file.properties)

If you want to use a placeholder in a logging property, you should use [Spring](#_bookmark107) [Boot’s syntax](#_bookmark107) and not the syntax of the underlying framework. Notably, if you use Logback, you should use : as the delimiter between a property name and its default value and not use :-.

You can add MDC and other ad-hoc content to log lines by overriding only the LOG\_LEVEL\_PATTERN (or logging.pattern.level with Logback). For example, if you use logging.pattern.level=user:%X{user} %5p, then the default log format contains an MDC entry for "user", if it exists, as shown in the following example.

2019-08-30 12:30:04.031 user:someone INFO 22174 --- [ nio-8080-exec-0]

demo.Controller

Handling authenticated request

### Logback Extensions

Spring Boot includes a number of extensions to Logback that can help with advanced configuration. You can use these extensions in your logback-spring.xml configuration file.

Because the standard logback.xml configuration file is loaded too early, you cannot use extensions in it. You need to either use logback-spring.xml or define a logging.config property.

The extensions cannot be used with Logback’s [configuration scanning](https://logback.qos.ch/manual/configuration.html#autoScan). If you attempt to do so, making changes to the configuration file results in an error similar to one of the following being logged:

ERROR in ch.qos.logback.core.joran.spi.Interpreter@4:71 - no applicable action for [springProperty], current ElementPath is [[configuration][springProperty]]

ERROR in ch.qos.logback.core.joran.spi.Interpreter@4:71 - no applicable action for [springProfile], current ElementPath is [[configuration][springProfile]]

#### Profile-specific Configuration

The <springProfile> tag lets you optionally include or exclude sections of configuration based on the active Spring profiles. Profile sections are supported anywhere within the <configuration> element. Use the name attribute to specify which profile accepts the configuration. The

<springProfile> tag can contain a profile name (for example staging) or a profile expression. A profile expression allows for more complicated profile logic to be expressed, for example production & (eu-central | eu-west). Check the [reference guide](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/core.html#beans-definition-profiles-java) for more details. The following listing shows three sample profiles:

<springProfile name="staging">

<!-- configuration to be enabled when the "staging" profile is active -->

</springProfile>

<springProfile name="dev | staging">

<!-- configuration to be enabled when the "dev" or "staging" profiles are active

-->

</springProfile>

<springProfile name="!production">

<!-- configuration to be enabled when the "production" profile is not active -->

</springProfile>

#### Environment Properties

The <springProperty> tag lets you expose properties from the Spring Environment for use within Logback. Doing so can be useful if you want to access values from your application.properties file in your Logback configuration. The tag works in a similar way to Logback’s standard <property> tag. However, rather than specifying a direct value, you specify the source of the property (from the Environment). If you need to store the property somewhere other than in local scope, you can use the scope attribute. If you need a fallback value (in case the property is not set in the Environment), you can use the defaultValue attribute. The following example shows how to expose properties for use within Logback:

<springProperty scope="context" name="fluentHost" source="myapp.fluentd.host" defaultValue="localhost"/>

<appender name="FLUENT" class="ch.qos.logback.more.appenders.DataFluentAppender">

<remoteHost>${fluentHost}</remoteHost>

...

</appender>

The source must be specified in kebab case (such as my.property-name). However, properties can be added to the Environment by using the relaxed rules.

## Internationalization

Spring Boot supports localized messages so that your application can cater to users of different language preferences. By default, Spring Boot looks for the presence of a messages resource bundle at the root of the classpath.

The auto-configuration applies when the default properties file for the configured resource bundle is available (i.e. messages.properties by default). If your resource bundle contains only language-specific properties files, you are required to add the default. If no properties file is found that matches any of the configured base names, there will be no auto-configured MessageSource.

The basename of the resource bundle as well as several other attributes can be configured using the spring.messages namespace, as shown in the following example:

spring.messages.basename=messages,config.i18n.messages spring.messages.fallback-to-system-locale=false

spring.messages.basename supports comma-separated list of locations, either a package qualifier or a resource resolved from the classpath root.

See [MessageSourceProperties](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/context/MessageSourceProperties.java) for more supported options.

## JSON

Spring Boot provides integration with three JSON mapping libraries:

* Gson
* Jackson
* JSON-B

Jackson is the preferred and default library.

### Jackson

Auto-configuration for Jackson is provided and Jackson is part of spring-boot-starter-json. When Jackson is on the classpath an ObjectMapper bean is automatically configured. Several configuration properties are provided for [customizing the configuration of the ObjectMapper](#_bookmark571).

### Gson

Auto-configuration for Gson is provided. When Gson is on the classpath a Gson bean is automatically configured. Several spring.gson.\* configuration properties are provided for customizing the configuration. To take more control, one or more GsonBuilderCustomizer beans can be used.

### JSON-B

Auto-configuration for JSON-B is provided. When the JSON-B API and an implementation are on the classpath a Jsonb bean will be automatically configured. The preferred JSON-B implementation is Apache Johnzon for which dependency management is provided.

## Developing Web Applications

Spring Boot is well suited for web application development. You can create a self-contained HTTP server by using embedded Tomcat, Jetty, Undertow, or Netty. Most web applications use the spring- boot-starter-web module to get up and running quickly. You can also choose to build reactive web applications by using the spring-boot-starter-webflux module.

If you have not yet developed a Spring Boot web application, you can follow the "Hello World!" example in the [*Getting started*](#_bookmark26) section.

### The “Spring Web MVC Framework”

The [Spring Web MVC framework](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web.html#mvc) (often referred to as “Spring MVC”) is a rich “model view controller” web framework. Spring MVC lets you create special @Controller or @RestController beans to handle incoming HTTP requests. Methods in your controller are mapped to HTTP by using @RequestMapping annotations.

The following code shows a typical @RestController that serves JSON data:

@RestController @RequestMapping(value="/users") public class MyRestController {

@RequestMapping(value="/{user}", method=RequestMethod.GET) public User getUser(@PathVariable Long user) {

// ...

}

@RequestMapping(value="/{user}/customers", method=RequestMethod.GET) List<Customer> getUserCustomers(@PathVariable Long user) {

// ...

}

@RequestMapping(value="/{user}", method=RequestMethod.DELETE) public User deleteUser(@PathVariable Long user) {

// ...

}

}

Spring MVC is part of the core Spring Framework, and detailed information is available in the [reference documentation](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web.html#mvc). There are also several guides that cover Spring MVC available at [spring.io/guides](https://spring.io/guides).

#### Spring MVC Auto-configuration

Spring Boot provides auto-configuration for Spring MVC that works well with most applications. The auto-configuration adds the following features on top of Spring’s defaults:

* Inclusion of ContentNegotiatingViewResolver and BeanNameViewResolver beans.
* Support for serving static resources, including support for WebJars (covered [later in this](#_bookmark152) [document](#_bookmark152))).
* Automatic registration of Converter, GenericConverter, and Formatter beans.
* Support for HttpMessageConverters (covered [later in this document](#_bookmark149)).
* Automatic registration of MessageCodesResolver (covered [later in this document](#_bookmark151)).
* Static index.html support.
* Custom Favicon support (covered [later in this document](#_bookmark154)).
* Automatic use of a ConfigurableWebBindingInitializer bean (covered [later in this document](#_bookmark156)).

If you want to keep those Spring Boot MVC customizations and make more [MVC customizations](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web.html#mvc) (interceptors, formatters, view controllers, and other features), you can add your own @Configuration class of type WebMvcConfigurer but **without** @EnableWebMvc.

If you want to provide custom instances of RequestMappingHandlerMapping, RequestMappingHandlerAdapter, or ExceptionHandlerExceptionResolver, and still keep the Spring Boot MVC customizations, you can declare a bean of type WebMvcRegistrations and use it to provide custom instances of those components.

If you want to take complete control of Spring MVC, you can add your own @Configuration annotated with @EnableWebMvc, or alternatively add your own @Configuration-annotated DelegatingWebMvcConfiguration as described in the Javadoc of @EnableWebMvc.

#### HttpMessageConverters

Spring MVC uses the HttpMessageConverter interface to convert HTTP requests and responses. Sensible defaults are included out of the box. For example, objects can be automatically converted to JSON (by using the Jackson library) or XML (by using the Jackson XML extension, if available, or by using JAXB if the Jackson XML extension is not available). By default, strings are encoded in UTF- 8.

If you need to add or customize converters, you can use Spring Boot’s HttpMessageConverters class, as shown in the following listing:

import org.springframework.boot.autoconfigure.http.HttpMessageConverters; import org.springframework.context.annotation.\*;

import org.springframework.http.converter.\*;

@Configuration(proxyBeanMethods = false) public class MyConfiguration {

@Bean

public HttpMessageConverters customConverters() { HttpMessageConverter<?> additional = ...

HttpMessageConverter<?> another = ...

return new HttpMessageConverters(additional, another);

}

}

Any HttpMessageConverter bean that is present in the context is added to the list of converters. You can also override default converters in the same way.

#### Custom JSON Serializers and Deserializers

If you use Jackson to serialize and deserialize JSON data, you might want to write your own JsonSerializer and JsonDeserializer classes. Custom serializers are usually [registered with Jackson](https://github.com/FasterXML/jackson-docs/wiki/JacksonHowToCustomSerializers) [through a module](https://github.com/FasterXML/jackson-docs/wiki/JacksonHowToCustomSerializers), but Spring Boot provides an alternative @JsonComponent annotation that makes it easier to directly register Spring Beans.

You can use the @JsonComponent annotation directly on JsonSerializer, JsonDeserializer or KeyDeserializer implementations. You can also use it on classes that contain serializers/deserializers as inner classes, as shown in the following example:

import java.io.\*;

import com.fasterxml.jackson.core.\*; import com.fasterxml.jackson.databind.\*; import org.springframework.boot.jackson.\*;

@JsonComponent

public class Example {

public static class Serializer extends JsonSerializer<SomeObject> {

// ...

}

public static class Deserializer extends JsonDeserializer<SomeObject> {

// ...

}

}

All @JsonComponent beans in the ApplicationContext are automatically registered with Jackson. Because @JsonComponent is meta-annotated with @Component, the usual component-scanning rules apply.

Spring Boot also provides [JsonObjectSerializer](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/jackson/JsonObjectSerializer.java) and [JsonObjectDeserializer](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/jackson/JsonObjectDeserializer.java) base classes that provide useful alternatives to the standard Jackson versions when serializing objects. See [JsonObjectSerializer](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/jackson/JsonObjectSerializer.html) and [JsonObjectDeserializer](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/jackson/JsonObjectDeserializer.html) in the Javadoc for details.

#### MessageCodesResolver

Spring MVC has a strategy for generating error codes for rendering error messages from binding errors: MessageCodesResolver. If you set the spring.mvc.message-codes-resolver-format property PREFIX\_ERROR\_CODE or POSTFIX\_ERROR\_CODE, Spring Boot creates one for you (see the enumeration in [DefaultMessageCodesResolver.Format](https://docs.spring.io/spring/docs/5.2.9.RELEASE/javadoc-api/org/springframework/validation/DefaultMessageCodesResolver.Format.html)).

#### Static Content

By default, Spring Boot serves static content from a directory called /static (or /public or

/resources or /META-INF/resources) in the classpath or from the root of the ServletContext. It uses the ResourceHttpRequestHandler from Spring MVC so that you can modify that behavior by adding

your own WebMvcConfigurer and overriding the addResourceHandlers method.

In a stand-alone web application, the default servlet from the container is also enabled and acts as a fallback, serving content from the root of the ServletContext if Spring decides not to handle it. Most of the time, this does not happen (unless you modify the default MVC configuration), because Spring can always handle requests through the DispatcherServlet.

By default, resources are mapped on /\*\*, but you can tune that with the spring.mvc.static-path- pattern property. For instance, relocating all resources to /resources/\*\* can be achieved as follows:

spring.mvc.static-path-pattern=/resources/\*\*

You can also customize the static resource locations by using the spring.resources.static-locations property (replacing the default values with a list of directory locations). The root Servlet context path, "/", is automatically added as a location as well.

In addition to the “standard” static resource locations mentioned earlier, a special case is made for [Webjars content](https://www.webjars.org/). Any resources with a path in /webjars/\*\* are served from jar files if they are packaged in the Webjars format.

Do not use the src/main/webapp directory if your application is packaged as a jar. Although this directory is a common standard, it works **only** with war packaging, and it is silently ignored by most build tools if you generate a jar.

Spring Boot also supports the advanced resource handling features provided by Spring MVC, allowing use cases such as cache-busting static resources or using version agnostic URLs for Webjars.

To use version agnostic URLs for Webjars, add the webjars-locator-core dependency. Then declare your Webjar. Using jQuery as an example, adding "/webjars/jquery/jquery.min.js" results in "/webjars/jquery/x.y.z/jquery.min.js" where x.y.z is the Webjar version.

If you use JBoss, you need to declare the webjars-locator-jboss-vfs dependency instead of the webjars-locator-core. Otherwise, all Webjars resolve as a 404.

To use cache busting, the following configuration configures a cache busting solution for all static resources, effectively adding a content hash, such as <link href="/css/spring- 2a2d595e6ed9a0b24f027f2b63b134d6.css"/>, in URLs:

spring.resources.chain.strategy.content.enabled=true spring.resources.chain.strategy.content.paths=/\*\*

Links to resources are rewritten in templates at runtime, thanks to a ResourceUrlEncodingFilter that is auto-configured for Thymeleaf and FreeMarker. You should manually declare this filter when using JSPs. Other template engines are currently not automatically supported but can be with custom template macros/helpers and the use of the [ResourceUrlProvider](https://docs.spring.io/spring/docs/5.2.9.RELEASE/javadoc-api/org/springframework/web/servlet/resource/ResourceUrlProvider.html).

When loading resources dynamically with, for example, a JavaScript module loader, renaming files is not an option. That is why other strategies are also supported and can be combined. A "fixed" strategy adds a static version string in the URL without changing the file name, as shown in the following example:

spring.resources.chain.strategy.content.enabled=true spring.resources.chain.strategy.content.paths=/\*\* spring.resources.chain.strategy.fixed.enabled=true spring.resources.chain.strategy.fixed.paths=/js/lib/ spring.resources.chain.strategy.fixed.version=v12

With this configuration, JavaScript modules located under "/js/lib/" use a fixed versioning strategy ("/v12/js/lib/mymodule.js"), while other resources still use the content one (<link href="/css/spring-2a2d595e6ed9a0b24f027f2b63b134d6.css"/>).

See [ResourceProperties](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/ResourceProperties.java) for more supported options.

This feature has been thoroughly described in a dedicated [blog post](https://spring.io/blog/2014/07/24/spring-framework-4-1-handling-static-web-resources) and in Spring Framework’s [reference documentation](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web.html#mvc-config-static-resources).

#### Welcome Page

Spring Boot supports both static and templated welcome pages. It first looks for an index.html file in the configured static content locations. If one is not found, it then looks for an index template. If either is found, it is automatically used as the welcome page of the application.

#### Custom Favicon

As with other static resources, Spring Boot looks for a favicon.ico in the configured static content locations. If such a file is present, it is automatically used as the favicon of the application.

#### Path Matching and Content Negotiation

Spring MVC can map incoming HTTP requests to handlers by looking at the request path and matching it to the mappings defined in your application (for example, @GetMapping annotations on Controller methods).

Spring Boot chooses to disable suffix pattern matching by default, which means that requests like "GET /projects/spring-boot.json" won’t be matched to @GetMapping("/projects/spring-boot") mappings. This is considered as a [best practice for Spring MVC applications](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web.html#mvc-ann-requestmapping-suffix-pattern-match). This feature was mainly useful in the past for HTTP clients which did not send proper "Accept" request headers; we needed to make sure to send the correct Content Type to the client. Nowadays, Content Negotiation

is much more reliable.

There are other ways to deal with HTTP clients that don’t consistently send proper "Accept" request headers. Instead of using suffix matching, we can use a query parameter to ensure that requests like "GET /projects/spring-boot?format=json" will be mapped to @GetMapping("/projects/spring- boot"):

spring.mvc.contentnegotiation.favor-parameter=true

# We can change the parameter name, which is "format" by default: # spring.mvc.contentnegotiation.parameter-name=myparam

# We can also register additional file extensions/media types with: spring.mvc.contentnegotiation.media-types.markdown=text/markdown

Suffix pattern matching is deprecated and will be removed in a future release. If you understand the caveats and would still like your application to use suffix pattern matching, the following configuration is required:

spring.mvc.contentnegotiation.favor-path-extension=true spring.mvc.pathmatch.use-suffix-pattern=true

Alternatively, rather than open all suffix patterns, it’s more secure to only support registered suffix patterns:

spring.mvc.contentnegotiation.favor-path-extension=true spring.mvc.pathmatch.use-registered-suffix-pattern=true

# You can also register additional file extensions/media types with: # spring.mvc.contentnegotiation.media-types.adoc=text/asciidoc

#### ConfigurableWebBindingInitializer

Spring MVC uses a WebBindingInitializer to initialize a WebDataBinder for a particular request. If you create your own ConfigurableWebBindingInitializer @Bean, Spring Boot automatically configures Spring MVC to use it.

#### Template Engines

As well as REST web services, you can also use Spring MVC to serve dynamic HTML content. Spring MVC supports a variety of templating technologies, including Thymeleaf, FreeMarker, and JSPs. Also, many other templating engines include their own Spring MVC integrations.

Spring Boot includes auto-configuration support for the following templating engines:

* [FreeMarker](https://freemarker.apache.org/docs/)
* [Groovy](https://docs.groovy-lang.org/docs/next/html/documentation/template-engines.html#_the_markuptemplateengine)
* [Thymeleaf](https://www.thymeleaf.org/)
* [Mustache](https://mustache.github.io/)

If possible, JSPs should be avoided. There are several [known limitations](#_bookmark178) when using them with embedded servlet containers.

When you use one of these templating engines with the default configuration, your templates are picked up automatically from src/main/resources/templates.

Depending on how you run your application, IntelliJ IDEA orders the classpath differently. Running your application in the IDE from its main method results in a different ordering than when you run your application by using Maven or Gradle or from its packaged jar. This can cause Spring Boot to fail to find the templates on the classpath. If you have this problem, you can reorder the classpath in the IDE to place the module’s classes and resources first. Alternatively, you can configure the template prefix to search every templates directory on the classpath, as follows: classpath\*:/templates/.

#### Error Handling

By default, Spring Boot provides an /error mapping that handles all errors in a sensible way, and it is registered as a “global” error page in the servlet container. For machine clients, it produces a JSON response with details of the error, the HTTP status, and the exception message. For browser clients, there is a “whitelabel” error view that renders the same data in HTML format (to customize it, add a View that resolves to error). To replace the default behavior completely, you can implement ErrorController and register a bean definition of that type or add a bean of type ErrorAttributes to use the existing mechanism but replace the contents.

The BasicErrorController can be used as a base class for a custom ErrorController. This is particularly useful if you want to add a handler for a new content type (the default is to handle text/html specifically and provide a fallback for everything else). To do so, extend BasicErrorController, add a public method with a @RequestMapping that has a produces attribute, and create a bean of your new type.

You can also define a class annotated with @ControllerAdvice to customize the JSON document to return for a particular controller and/or exception type, as shown in the following example:

@ControllerAdvice(basePackageClasses = AcmeController.class)

public class AcmeControllerAdvice extends ResponseEntityExceptionHandler {

@ExceptionHandler(YourException.class) @ResponseBody

ResponseEntity<?> handleControllerException(HttpServletRequest request, Throwable ex) {

HttpStatus status = getStatus(request);

return new ResponseEntity<>(new CustomErrorType(status.value(), ex.getMessage()), status);

}

private HttpStatus getStatus(HttpServletRequest request) { Integer statusCode = (Integer)

request.getAttribute("javax.servlet.error.status\_code"); if (statusCode == null) {

return HttpStatus.INTERNAL\_SERVER\_ERROR;

}

return HttpStatus.valueOf(statusCode);

}

}

In the preceding example, if YourException is thrown by a controller defined in the same package as AcmeController, a JSON representation of the CustomErrorType POJO is used instead of the ErrorAttributes representation.

**Custom Error Pages**

If you want to display a custom HTML error page for a given status code, you can add a file to an

/error directory. Error pages can either be static HTML (that is, added under any of the static resource directories) or be built by using templates. The name of the file should be the exact status code or a series mask.

For example, to map 404 to a static HTML file, your directory structure would be as follows:

src/

+- main/

+- java/

| + <source code>

+- resources/

+- public/

+- error/

| +- 404.html

+- <other public assets>

To map all 5xx errors by using a FreeMarker template, your directory structure would be as follows:

src/

+- main/

+- java/

| + <source code>

+- resources/

+- templates/

+- error/

| +- 5xx.ftlh

+- <other templates>

For more complex mappings, you can also add beans that implement the ErrorViewResolver

interface, as shown in the following example:

public class MyErrorViewResolver implements ErrorViewResolver {

@Override

public ModelAndView resolveErrorView(HttpServletRequest request, HttpStatus status, Map<String, Object> model) {

// Use the request or status to optionally return a ModelAndView return ...

}

}

You can also use regular Spring MVC features such as [@ExceptionHandler](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web.html#mvc-exceptionhandlers) [methods](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web.html#mvc-exceptionhandlers) and

[@ControllerAdvice](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web.html#mvc-ann-controller-advice). The ErrorController then picks up any unhandled exceptions.

**Mapping Error Pages outside of Spring MVC**

For applications that do not use Spring MVC, you can use the ErrorPageRegistrar interface to directly register ErrorPages. This abstraction works directly with the underlying embedded servlet container and works even if you do not have a Spring MVC DispatcherServlet.

@Bean

public ErrorPageRegistrar errorPageRegistrar(){ return new MyErrorPageRegistrar();

}

// ...

private static class MyErrorPageRegistrar implements ErrorPageRegistrar { @Override

public void registerErrorPages(ErrorPageRegistry registry) { registry.addErrorPages(new ErrorPage(HttpStatus.BAD\_REQUEST, "/400"));

}

}

If you register an ErrorPage with a path that ends up being handled by a Filter (as is common with some non-Spring web frameworks, like Jersey and Wicket), then the Filter has to be explicitly registered as an ERROR dispatcher, as shown in the following example:

@Bean

public FilterRegistrationBean myFilter() {

FilterRegistrationBean registration = new FilterRegistrationBean(); registration.setFilter(new MyFilter());

...

registration.setDispatcherTypes(EnumSet.allOf(DispatcherType.class)); return registration;

}

Note that the default FilterRegistrationBean does not include the ERROR dispatcher type.

**Error handling in a war deployment**

When deployed to a servlet container, Spring Boot uses its error page filter to forward a request with an error status to the appropriate error page. This is necessary as the Servlet specification does not provide an API for registering error pages. Depending on the container that you are deploying your war file to and the technologies that your application uses, some additional configuration may be required.

The error page filter can only forward the request to the correct error page if the response has not already been committed. By default, WebSphere Application Server 8.0 and later commits the response upon successful completion of a servlet’s service method. You should disable this behavior by setting com.ibm.ws.webcontainer.invokeFlushAfterService to false.

If you are using Spring Security and want to access the principal in an error page, you must configure Spring Security’s filter to be invoked on error dispatches. To do so, set the

spring.security.filter.dispatcher-types property to async, error, forward, request.

#### Spring HATEOAS

If you develop a RESTful API that makes use of hypermedia, Spring Boot provides auto- configuration for Spring HATEOAS that works well with most applications. The auto-configuration replaces the need to use @EnableHypermediaSupport and registers a number of beans to ease building hypermedia-based applications, including a LinkDiscoverers (for client side support) and an ObjectMapper configured to correctly marshal responses into the desired representation. The ObjectMapper is customized by setting the various spring.jackson.\* properties or, if one exists, by a Jackson2ObjectMapperBuilder bean.

You can take control of Spring HATEOAS’s configuration by using @EnableHypermediaSupport. Note that doing so disables the ObjectMapper customization described earlier.

#### CORS Support

[Cross-origin resource sharing](https://en.wikipedia.org/wiki/Cross-origin_resource_sharing) (CORS) is a [W3C specification](https://www.w3.org/TR/cors/) implemented by [most browsers](https://caniuse.com/#feat%3Dcors) that lets you specify in a flexible way what kind of cross-domain requests are authorized., instead of using some less secure and less powerful approaches such as IFRAME or JSONP.

As of version 4.2, Spring MVC [supports CORS](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web.html#mvc-cors). Using [controller method CORS configuration](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web.html#mvc-cors-controller) with [@CrossOrigin](https://docs.spring.io/spring/docs/5.2.9.RELEASE/javadoc-api/org/springframework/web/bind/annotation/CrossOrigin.html) annotations in your Spring Boot application does not require any specific configuration. [Global CORS configuration](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web.html#mvc-cors-global) can be defined by registering a WebMvcConfigurer bean with a customized addCorsMappings(CorsRegistry) method, as shown in the following example:

@Configuration(proxyBeanMethods = false) public class MyConfiguration {

@Bean

public WebMvcConfigurer corsConfigurer() { return new WebMvcConfigurer() {

@Override

public void addCorsMappings(CorsRegistry registry) { registry.addMapping("/api/\*\*");

}

};

}

}

### The “Spring WebFlux Framework”

Spring WebFlux is the new reactive web framework introduced in Spring Framework 5.0. Unlike Spring MVC, it does not require the Servlet API, is fully asynchronous and non-blocking, and implements the [Reactive Streams](https://www.reactive-streams.org/) specification through [the Reactor project](https://projectreactor.io/).

Spring WebFlux comes in two flavors: functional and annotation-based. The annotation-based one is quite close to the Spring MVC model, as shown in the following example:

@RestController @RequestMapping("/users") public class MyRestController {

@GetMapping("/{user}")

public Mono<User> getUser(@PathVariable Long user) {

// ...

}

@GetMapping("/{user}/customers")

public Flux<Customer> getUserCustomers(@PathVariable Long user) {

// ...

}

@DeleteMapping("/{user}")

public Mono<User> deleteUser(@PathVariable Long user) {

// ...

}

}

“WebFlux.fn”, the functional variant, separates the routing configuration from the actual handling of the requests, as shown in the following example:

@Configuration(proxyBeanMethods = false) public class RoutingConfiguration {

@Bean

public RouterFunction<ServerResponse> monoRouterFunction(UserHandler userHandler)

{

return route(GET("/{user}").and(accept(APPLICATION\_JSON)), userHandler::getUser)

.andRoute(GET("/{user}/customers").and(accept(APPLICATION\_JSON)), userHandler::getUserCustomers)

.andRoute(DELETE("/{user}").and(accept(APPLICATION\_JSON)), userHandler::deleteUser);

}

}

@Component

public class UserHandler {

public Mono<ServerResponse> getUser(ServerRequest request) {

// ...

}

public Mono<ServerResponse> getUserCustomers(ServerRequest request) {

// ...

}

public Mono<ServerResponse> deleteUser(ServerRequest request) {

// ...

}

}

WebFlux is part of the Spring Framework and detailed information is available in its [reference](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web-reactive.html#webflux-fn) [documentation](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web-reactive.html#webflux-fn).

You can define as many RouterFunction beans as you like to modularize the definition of the router. Beans can be ordered if you need to apply a precedence.

To get started, add the spring-boot-starter-webflux module to your application.

Adding both spring-boot-starter-web and spring-boot-starter-webflux modules in your application results in Spring Boot auto-configuring Spring MVC, not WebFlux. This behavior has been chosen because many Spring developers add spring-boot- starter-webflux to their Spring MVC application to use the reactive WebClient. You can still enforce your choice by setting the chosen application type to SpringApplication.setWebApplicationType(WebApplicationType.REACTIVE).

#### Spring WebFlux Auto-configuration

Spring Boot provides auto-configuration for Spring WebFlux that works well with most applications.

The auto-configuration adds the following features on top of Spring’s defaults:

* Configuring codecs for HttpMessageReader and HttpMessageWriter instances (described [later in](#_bookmark164) [this document](#_bookmark164)).
* Support for serving static resources, including support for WebJars (described [later in this](#_bookmark152) [document](#_bookmark152)).

If you want to keep Spring Boot WebFlux features and you want to add additional [WebFlux](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web-reactive.html#webflux-config) [configuration](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web-reactive.html#webflux-config), you can add your own @Configuration class of type WebFluxConfigurer but **without** @EnableWebFlux.

If you want to take complete control of Spring WebFlux, you can add your own @Configuration

annotated with @EnableWebFlux.

#### HTTP Codecs with HttpMessageReaders and HttpMessageWriters

Spring WebFlux uses the HttpMessageReader and HttpMessageWriter interfaces to convert HTTP requests and responses. They are configured with CodecConfigurer to have sensible defaults by looking at the libraries available in your classpath.

Spring Boot provides dedicated configuration properties for codecs, spring.codec.\*. It also applies further customization by using CodecCustomizer instances. For example, spring.jackson.\* configuration keys are applied to the Jackson codec.

If you need to add or customize codecs, you can create a custom CodecCustomizer component, as shown in the following example:

import org.springframework.boot.web.codec.CodecCustomizer;

@Configuration(proxyBeanMethods = false) public class MyConfiguration {

@Bean

public CodecCustomizer myCodecCustomizer() { return codecConfigurer -> {

// ...

};

}

}

You can also leverage [Boot’s custom JSON serializers and deserializers](#_bookmark150).

#### Static Content

By default, Spring Boot serves static content from a directory called /static (or /public or

/resources or /META-INF/resources) in the classpath. It uses the ResourceWebHandler from Spring WebFlux so that you can modify that behavior by adding your own WebFluxConfigurer and overriding the addResourceHandlers method.

By default, resources are mapped on /\*\*, but you can tune that by setting the spring.webflux.static-path-pattern property. For instance, relocating all resources to /resources/\*\* can be achieved as follows:

spring.webflux.static-path-pattern=/resources/\*\*

You can also customize the static resource locations by using spring.resources.static-locations. Doing so replaces the default values with a list of directory locations. If you do so, the default welcome page detection switches to your custom locations. So, if there is an index.html in any of your locations on startup, it is the home page of the application.

In addition to the “standard” static resource locations listed earlier, a special case is made for [Webjars content](https://www.webjars.org/). Any resources with a path in /webjars/\*\* are served from jar files if they are packaged in the Webjars format.

Spring WebFlux applications do not strictly depend on the Servlet API, so they cannot be deployed as war files and do not use the src/main/webapp directory.

#### Welcome Page

Spring Boot supports both static and templated welcome pages. It first looks for an index.html file in the configured static content locations. If one is not found, it then looks for an index template. If either is found, it is automatically used as the welcome page of the application.

#### Template Engines

As well as REST web services, you can also use Spring WebFlux to serve dynamic HTML content. Spring WebFlux supports a variety of templating technologies, including Thymeleaf, FreeMarker, and Mustache.

Spring Boot includes auto-configuration support for the following templating engines:

* [FreeMarker](https://freemarker.apache.org/docs/)
* [Thymeleaf](https://www.thymeleaf.org/)
* [Mustache](https://mustache.github.io/)

When you use one of these templating engines with the default configuration, your templates are picked up automatically from src/main/resources/templates.

#### Error Handling

Spring Boot provides a WebExceptionHandler that handles all errors in a sensible way. Its position in the processing order is immediately before the handlers provided by WebFlux, which are considered last. For machine clients, it produces a JSON response with details of the error, the HTTP status, and the exception message. For browser clients, there is a “whitelabel” error handler that renders the same data in HTML format. You can also provide your own HTML templates to display errors (see the [next section](#_bookmark169)).

The first step to customizing this feature often involves using the existing mechanism but replacing or augmenting the error contents. For that, you can add a bean of type ErrorAttributes.

To change the error handling behavior, you can implement ErrorWebExceptionHandler and register a bean definition of that type. Because a WebExceptionHandler is quite low-level, Spring Boot also provides a convenient AbstractErrorWebExceptionHandler to let you handle errors in a WebFlux functional way, as shown in the following example:

public class CustomErrorWebExceptionHandler extends AbstractErrorWebExceptionHandler {

// Define constructor here @Override

protected RouterFunction<ServerResponse> getRoutingFunction(ErrorAttributes errorAttributes) {

return RouterFunctions

.route(aPredicate, aHandler)

.andRoute(anotherPredicate, anotherHandler);

}

}

For a more complete picture, you can also subclass DefaultErrorWebExceptionHandler directly and override specific methods.

**Custom Error Pages**

If you want to display a custom HTML error page for a given status code, you can add a file to an

/error directory. Error pages can either be static HTML (that is, added under any of the static resource directories) or built with templates. The name of the file should be the exact status code or a series mask.

For example, to map 404 to a static HTML file, your directory structure would be as follows:

src/

+- main/

+- java/

| + <source code>

+- resources/

+- public/

+- error/

| +- 404.html

+- <other public assets>

To map all 5xx errors by using a Mustache template, your directory structure would be as follows:

src/

+- main/

+- java/

| + <source code>

+- resources/

+- templates/

+- error/

| +- 5xx.mustache

+- <other templates>

#### Web Filters

Spring WebFlux provides a WebFilter interface that can be implemented to filter HTTP request- response exchanges. WebFilter beans found in the application context will be automatically used to filter each exchange.

Where the order of the filters is important they can implement Ordered or be annotated with @Order. Spring Boot auto-configuration may configure web filters for you. When it does so, the orders shown in the following table will be used:

|  |  |
| --- | --- |
| **Web Filter** | **Order** |
| MetricsWebFilter | Ordered.HIGHEST\_PRECEDENCE + 1 |
| WebFilterChainProxy (Spring Security) | -100 |
| HttpTraceWebFilter | Ordered.LOWEST\_PRECEDENCE - 10 |

### JAX-RS and Jersey

If you prefer the JAX-RS programming model for REST endpoints, you can use one of the available implementations instead of Spring MVC. [Jersey](https://jersey.github.io/) and [Apache CXF](https://cxf.apache.org/) work quite well out of the box. CXF requires you to register its Servlet or Filter as a @Bean in your application context. Jersey has some native Spring support, so we also provide auto-configuration support for it in Spring Boot, together with a starter.

To get started with Jersey, include the spring-boot-starter-jersey as a dependency and then you

need one @Bean of type ResourceConfig in which you register all the endpoints, as shown in the following example:

@Component

public class JerseyConfig extends ResourceConfig {

public JerseyConfig() { register(Endpoint.class);

}

}

Jersey’s support for scanning executable archives is rather limited. For example, it cannot scan for endpoints in a package found in a [fully executable jar file](#_bookmark483) or in WEB-INF/classes when running an executable war file. To avoid this limitation, the packages method should not be used, and endpoints should be registered individually by using the register method, as shown in the preceding example.

For more advanced customizations, you can also register an arbitrary number of beans that implement ResourceConfigCustomizer.

All the registered endpoints should be @Components with HTTP resource annotations (@GET and others), as shown in the following example:

@Component @Path("/hello")

public class Endpoint {

@GET

public String message() { return "Hello";

}

}

Since the Endpoint is a Spring @Component, its lifecycle is managed by Spring and you can use the @Autowired annotation to inject dependencies and use the @Value annotation to inject external configuration. By default, the Jersey servlet is registered and mapped to /\*. You can change the mapping by adding @ApplicationPath to your ResourceConfig.

By default, Jersey is set up as a Servlet in a @Bean of type ServletRegistrationBean named jerseyServletRegistration. By default, the servlet is initialized lazily, but you can customize that behavior by setting spring.jersey.servlet.load-on-startup. You can disable or override that bean by creating one of your own with the same name. You can also use a filter instead of a servlet by setting spring.jersey.type=filter (in which case, the @Bean to replace or override is jerseyFilterRegistration). The filter has an @Order, which you can set with spring.jersey.filter.order. Both the servlet and the filter registrations can be given init

parameters by using spring.jersey.init.\* to specify a map of properties.

### Embedded Servlet Container Support

Spring Boot includes support for embedded [Tomcat](https://tomcat.apache.org/), [Jetty](https://www.eclipse.org/jetty/), and [Undertow](https://github.com/undertow-io/undertow) servers. Most developers use the appropriate “Starter” to obtain a fully configured instance. By default, the embedded server listens for HTTP requests on port 8080.

#### Servlets, Filters, and listeners

When using an embedded servlet container, you can register servlets, filters, and all the listeners (such as HttpSessionListener) from the Servlet spec, either by using Spring beans or by scanning for Servlet components.

**Registering Servlets, Filters, and Listeners as Spring Beans**

Any Servlet, Filter, or servlet \*Listener instance that is a Spring bean is registered with the embedded container. This can be particularly convenient if you want to refer to a value from your application.properties during configuration.

By default, if the context contains only a single Servlet, it is mapped to /. In the case of multiple servlet beans, the bean name is used as a path prefix. Filters map to /\*.

If convention-based mapping is not flexible enough, you can use the ServletRegistrationBean, FilterRegistrationBean, and ServletListenerRegistrationBean classes for complete control.

It is usually safe to leave Filter beans unordered. If a specific order is required, you should annotate the Filter with @Order or make it implement Ordered. You cannot configure the order of a Filter by annotating its bean method with @Order. If you cannot change the Filter class to add @Order or implement Ordered, you must define a FilterRegistrationBean for the Filter and set the registration bean’s order using the setOrder(int) method. Avoid configuring a Filter that reads the request body at Ordered.HIGHEST\_PRECEDENCE, since it might go against the character encoding configuration of your application. If a Servlet filter wraps the request, it should be configured with an order that is less than or equal to OrderedFilter.REQUEST\_WRAPPER\_FILTER\_MAX\_ORDER.

To see the order of every Filter in your application, enable debug level logging for the web [logging group](#_bookmark136) (logging.level.web=debug). Details of the registered filters, including their order and URL patterns, will then be logged at startup.

Take care when registering Filter beans since they are initialized very early in the application lifectyle. If you need to register a Filter that interacts with other beans, consider using a [DelegatingFilterProxyRegistrationBean](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/web/servlet/DelegatingFilterProxyRegistrationBean.html) instead.

#### Servlet Context Initialization

Embedded servlet containers do not directly execute the Servlet 3.0+ javax.servlet.ServletContainerInitializer interface or Spring’s org.springframework.web.WebApplicationInitializer interface. This is an intentional design decision intended to reduce the risk that third party libraries designed to run inside a war may break Spring

Boot applications.

If you need to perform servlet context initialization in a Spring Boot application, you should register a bean that implements the org.springframework.boot.web.servlet.ServletContextInitializer interface. The single onStartup method provides access to the ServletContext and, if necessary, can easily be used as an adapter to an existing WebApplicationInitializer.

**Scanning for Servlets, Filters, and listeners**

When using an embedded container, automatic registration of classes annotated with @WebServlet, @WebFilter, and @WebListener can be enabled by using @ServletComponentScan.

@ServletComponentScan has no effect in a standalone container, where the container’s built-in discovery mechanisms are used instead.

#### The ServletWebServerApplicationContext

Under the hood, Spring Boot uses a different type of ApplicationContext for embedded servlet container support. The ServletWebServerApplicationContext is a special type of WebApplicationContext that bootstraps itself by searching for a single ServletWebServerFactory bean. Usually a TomcatServletWebServerFactory, JettyServletWebServerFactory, or UndertowServletWebServerFactory has been auto-configured.

You usually do not need to be aware of these implementation classes. Most applications are auto-configured, and the appropriate ApplicationContext and ServletWebServerFactory are created on your behalf.

#### Customizing Embedded Servlet Containers

Common servlet container settings can be configured by using Spring Environment properties. Usually, you would define the properties in your application.properties file.

Common server settings include:

* Network settings: Listen port for incoming HTTP requests (server.port), interface address to bind to server.address, and so on.
* Session settings: Whether the session is persistent (server.servlet.session.persistent), session timeout (server.servlet.session.timeout), location of session data (server.servlet.session.store-dir), and session-cookie configuration (server.servlet.session.cookie.\*).
* Error management: Location of the error page (server.error.path) and so on.
* [SSL](#_bookmark550)
* [HTTP compression](#_bookmark549)

Spring Boot tries as much as possible to expose common settings, but this is not always possible. For those cases, dedicated namespaces offer server-specific customizations (see server.tomcat and server.undertow). For instance, [access logs](#_bookmark560) can be configured with specific features of the embedded

servlet container.

 See the [ServerProperties](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/ServerProperties.java) class for a complete list.

**Programmatic Customization**

If you need to programmatically configure your embedded servlet container, you can register a Spring bean that implements the WebServerFactoryCustomizer interface. WebServerFactoryCustomizer provides access to the ConfigurableServletWebServerFactory, which includes numerous customization setter methods. The following example shows programmatically setting the port:

import org.springframework.boot.web.server.WebServerFactoryCustomizer; import

org.springframework.boot.web.servlet.server.ConfigurableServletWebServerFactory; import org.springframework.stereotype.Component;

@Component

public class CustomizationBean implements WebServerFactoryCustomizer<ConfigurableServletWebServerFactory> {

@Override

public void customize(ConfigurableServletWebServerFactory server) { server.setPort(9000);

}

}

TomcatServletWebServerFactory, JettyServletWebServerFactory and UndertowServletWebServerFactory are dedicated variants of ConfigurableServletWebServerFactory that have additional customization setter methods for Tomcat, Jetty and Undertow respectively.

**Customizing ConfigurableServletWebServerFactory Directly**

If the preceding customization techniques are too limited, you can register the TomcatServletWebServerFactory, JettyServletWebServerFactory, or UndertowServletWebServerFactory bean yourself.

@Bean

public ConfigurableServletWebServerFactory webServerFactory() { TomcatServletWebServerFactory factory = new TomcatServletWebServerFactory(); factory.setPort(9000);

factory.setSessionTimeout(10, TimeUnit.MINUTES);

factory.addErrorPages(new ErrorPage(HttpStatus.NOT\_FOUND, "/notfound.html")); return factory;

}

Setters are provided for many configuration options. Several protected method “hooks” are also provided should you need to do something more exotic. See the [source code documentation](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/web/servlet/server/ConfigurableServletWebServerFactory.html) for details.

#### JSP Limitations

When running a Spring Boot application that uses an embedded servlet container (and is packaged as an executable archive), there are some limitations in the JSP support.

* With Jetty and Tomcat, it should work if you use war packaging. An executable war will work when launched with java -jar, and will also be deployable to any standard container. JSPs are not supported when using an executable jar.
* Undertow does not support JSPs.
* Creating a custom error.jsp page does not override the default view for [error handling](#_bookmark158). [Custom](#_bookmark159) [error pages](#_bookmark159) should be used instead.

### Embedded Reactive Server Support

Spring Boot includes support for the following embedded reactive web servers: Reactor Netty, Tomcat, Jetty, and Undertow. Most developers use the appropriate “Starter” to obtain a fully configured instance. By default, the embedded server listens for HTTP requests on port 8080.

### Reactive Server Resources Configuration

When auto-configuring a Reactor Netty or Jetty server, Spring Boot will create specific beans that will provide HTTP resources to the server instance: ReactorResourceFactory or JettyResourceFactory.

By default, those resources will be also shared with the Reactor Netty and Jetty clients for optimal performances, given:

* the same technology is used for server and client
* the client instance is built using the WebClient.Builder bean auto-configured by Spring Boot

Developers can override the resource configuration for Jetty and Reactor Netty by providing a custom ReactorResourceFactory or JettyResourceFactory bean - this will be applied to both clients and servers.

You can learn more about the resource configuration on the client side in the [WebClient Runtime](#_bookmark287) [section](#_bookmark287).

## Graceful shutdown

Graceful shutdown is supported with all four embedded web servers (Jetty, Reactor Netty, Tomcat, and Undertow) and with both reactive and Servlet-based web applications. It occurs as part of closing the application context and is performed in the earliest phase of stopping SmartLifecycle beans. This stop processing uses a timeout which provides a grace period during which existing requests will be allowed to complete but no new requests will be permitted. The exact way in which new requests are not permitted varies depending on the web server that is being used. Jetty,

Reactor Netty, and Tomcat will stop accepting requests at the network layer. Undertow will accept requests but respond immediately with a service unavailable (503) response.

 Graceful shutdown with Tomcat requires Tomcat 9.0.33 or later.

To enable graceful shutdown, configure the server.shutdown property, as shown in the following example:

server.shutdown=graceful

To configure the timeout period, configure the spring.lifecycle.timeout-per-shutdown-phase

property, as shown in the following example:

spring.lifecycle.timeout-per-shutdown-phase=20s

Using graceful shutdown with your IDE may not work properly if it does not send a proper SIGTERM signal. Refer to the documentation of your IDE for more details.

## RSocket

[RSocket](https://rsocket.io/) is a binary protocol for use on byte stream transports. It enables symmetric interaction models via async message passing over a single connection.

The spring-messaging module of the Spring Framework provides support for RSocket requesters and responders, both on the client and on the server side. See the [RSocket section](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web-reactive.html#rsocket-spring) of the Spring Framework reference for more details, including an overview of the RSocket protocol.

### RSocket Strategies Auto-configuration

Spring Boot auto-configures an RSocketStrategies bean that provides all the required infrastructure for encoding and decoding RSocket payloads. By default, the auto-configuration will try to configure the following (in order):

* + - 1. [CBOR](https://cbor.io/) codecs with Jackson
      2. JSON codecs with Jackson

The spring-boot-starter-rsocket starter provides both dependencies. Check out the [Jackson support](#_bookmark143) [section](#_bookmark143) to know more about customization possibilities.

Developers can customize the RSocketStrategies component by creating beans that implement the RSocketStrategiesCustomizer interface. Note that their @Order is important, as it determines the order of codecs.

### RSocket server Auto-configuration

Spring Boot provides RSocket server auto-configuration. The required dependencies are provided

by the spring-boot-starter-rsocket.

Spring Boot allows exposing RSocket over WebSocket from a WebFlux server, or standing up an independent RSocket server. This depends on the type of application and its configuration.

For WebFlux application (i.e. of type WebApplicationType.REACTIVE), the RSocket server will be plugged into the Web Server only if the following properties match:

spring.rsocket.server.mapping-path=/rsocket # a mapping path is defined spring.rsocket.server.transport=websocket # websocket is chosen as a transport #spring.rsocket.server.port= # no port is defined

Plugging RSocket into a web server is only supported with Reactor Netty, as RSocket itself is built with that library.

Alternatively, an RSocket TCP or websocket server is started as an independent, embedded server. Besides the dependency requirements, the only required configuration is to define a port for that server:

spring.rsocket.server.port=9898 # the only required configuration spring.rsocket.server.transport=tcp # you're free to configure other properties

### Spring Messaging RSocket support

Spring Boot will auto-configure the Spring Messaging infrastructure for RSocket.

This means that Spring Boot will create a RSocketMessageHandler bean that will handle RSocket requests to your application.

### Calling RSocket Services with RSocketRequester

Once the RSocket channel is established between server and client, any party can send or receive requests to the other.

As a server, you can get injected with an RSocketRequester instance on any handler method of an RSocket @Controller. As a client, you need to configure and establish an RSocket connection first. Spring Boot auto-configures an RSocketRequester.Builder for such cases with the expected codecs.

The RSocketRequester.Builder instance is a prototype bean, meaning each injection point will provide you with a new instance . This is done on purpose since this builder is stateful and you shouldn’t create requesters with different setups using the same instance.

The following code shows a typical example:

@Service

public class MyService {

private final Mono<RSocketRequester> rsocketRequester;

public MyService(RSocketRequester.Builder rsocketRequesterBuilder) { this.rsocketRequester = rsocketRequesterBuilder

.connectTcp("example.org", 9898).cache();

}

public Mono<User> someRSocketCall(String name) { return this.rsocketRequester.flatMap(req ->

req.route("user").data(name).retrieveMono(User.class));

}

}

## Security

If [Spring Security](https://spring.io/projects/spring-security) is on the classpath, then web applications are secured by default. Spring Boot relies on Spring Security’s content-negotiation strategy to determine whether to use httpBasic or formLogin. To add method-level security to a web application, you can also add @EnableGlobalMethodSecurity with your desired settings. Additional information can be found in the [Spring Security Reference Guide](https://docs.spring.io/spring-security/site/docs/5.3.4.RELEASE/reference/html5/#jc-method).

The default UserDetailsService has a single user. The user name is user, and the password is random and is printed at INFO level when the application starts, as shown in the following example:

Using generated security password: 78fa095d-3f4c-48b1-ad50-e24c31d5cf35

If you fine-tune your logging configuration, ensure that the org.springframework.boot.autoconfigure.security category is set to log INFO-level messages. Otherwise, the default password is not printed.

You can change the username and password by providing a spring.security.user.name and

spring.security.user.password.

The basic features you get by default in a web application are:

* A UserDetailsService (or ReactiveUserDetailsService in case of a WebFlux application) bean with in-memory store and a single user with a generated password (see [SecurityProperties.User](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/security/SecurityProperties.User.html) for the properties of the user).
* Form-based login or HTTP Basic security (depending on the Accept header in the request) for the entire application (including actuator endpoints if actuator is on the classpath).
* A DefaultAuthenticationEventPublisher for publishing authentication events.

You can provide a different AuthenticationEventPublisher by adding a bean for it.

### MVC Security

The default security configuration is implemented in SecurityAutoConfiguration and UserDetailsServiceAutoConfiguration. SecurityAutoConfiguration imports SpringBootWebSecurityConfiguration for web security and UserDetailsServiceAutoConfiguration configures authentication, which is also relevant in non-web applications. To switch off the default web application security configuration completely or to combine multiple Spring Security components such as OAuth 2 Client and Resource Server, add a bean of type WebSecurityConfigurerAdapter (doing so does not disable the UserDetailsService configuration or Actuator’s security).

To also switch off the UserDetailsService configuration, you can add a bean of type

UserDetailsService, AuthenticationProvider, or AuthenticationManager.

Access rules can be overridden by adding a custom WebSecurityConfigurerAdapter. Spring Boot provides convenience methods that can be used to override access rules for actuator endpoints and static resources. EndpointRequest can be used to create a RequestMatcher that is based on the management.endpoints.web.base-path property. PathRequest can be used to create a RequestMatcher for resources in commonly used locations.

### WebFlux Security

Similar to Spring MVC applications, you can secure your WebFlux applications by adding the spring-boot-starter-security dependency. The default security configuration is implemented in ReactiveSecurityAutoConfiguration and UserDetailsServiceAutoConfiguration. ReactiveSecurityAutoConfiguration imports WebFluxSecurityConfiguration for web security and UserDetailsServiceAutoConfiguration configures authentication, which is also relevant in non-web applications. To switch off the default web application security configuration completely, you can add a bean of type WebFilterChainProxy (doing so does not disable the UserDetailsService configuration or Actuator’s security).

To also switch off the UserDetailsService configuration, you can add a bean of type

ReactiveUserDetailsService or ReactiveAuthenticationManager.

Access rules and the use of multiple Spring Security components such as OAuth 2 Client and Resource Server can be configured by adding a custom SecurityWebFilterChain bean. Spring Boot provides convenience methods that can be used to override access rules for actuator endpoints and static resources. EndpointRequest can be used to create a ServerWebExchangeMatcher that is based on the management.endpoints.web.base-path property.

PathRequest can be used to create a ServerWebExchangeMatcher for resources in commonly used locations.

For example, you can customize your security configuration by adding something like:

@Bean

public SecurityWebFilterChain springSecurityFilterChain(ServerHttpSecurity http) { return http

.authorizeExchange()

.matchers(PathRequest.toStaticResources().atCommonLocations()).permitAll()

.pathMatchers("/foo", "/bar")

.authenticated().and()

.formLogin().and()

.build();

}

### OAuth2

[OAuth2](https://oauth.net/2/) is a widely used authorization framework that is supported by Spring.

#### Client

If you have spring-security-oauth2-client on your classpath, you can take advantage of some auto- configuration to set up an OAuth2/Open ID Connect clients. This configuration makes use of the properties under OAuth2ClientProperties. The same properties are applicable to both servlet and reactive applications.

You can register multiple OAuth2 clients and providers under the spring.security.oauth2.client

prefix, as shown in the following example:

spring.security.oauth2.client.registration.my-client-1.client-id=abcd spring.security.oauth2.client.registration.my-client-1.client-secret=password spring.security.oauth2.client.registration.my-client-1.client-name=Client for user scope

spring.security.oauth2.client.registration.my-client-1.provider=my-oauth-provider spring.security.oauth2.client.registration.my-client-1.scope=user spring.security.oauth2.client.registration.my-client-1.redirect-uri=https://my- redirect-uri.com

spring.security.oauth2.client.registration.my-client-1.client-authentication- method=basic

spring.security.oauth2.client.registration.my-client-1.authorization-grant- type=authorization\_code

spring.security.oauth2.client.registration.my-client-2.client-id=abcd spring.security.oauth2.client.registration.my-client-2.client-secret=password spring.security.oauth2.client.registration.my-client-2.client-name=Client for email scope

spring.security.oauth2.client.registration.my-client-2.provider=my-oauth-provider spring.security.oauth2.client.registration.my-client-2.scope=email spring.security.oauth2.client.registration.my-client-2.redirect-uri=https://my- redirect-uri.com

spring.security.oauth2.client.registration.my-client-2.client-authentication- method=basic

spring.security.oauth2.client.registration.my-client-2.authorization-grant- type=authorization\_code

spring.security.oauth2.client.provider.my-oauth-provider.authorization-uri=https://my- auth-server/oauth/authorize

spring.security.oauth2.client.provider.my-oauth-provider.token-uri=https://my-auth- server/oauth/token

spring.security.oauth2.client.provider.my-oauth-provider.user-info-uri=https://my- auth-server/userinfo

spring.security.oauth2.client.provider.my-oauth-provider.user-info-authentication- method=header

spring.security.oauth2.client.provider.my-oauth-provider.jwk-set-uri=https://my-auth- server/token\_keys

spring.security.oauth2.client.provider.my-oauth-provider.user-name-attribute=name

For OpenID Connect providers that support [OpenID Connect discovery](https://openid.net/specs/openid-connect-discovery-1_0.html), the configuration can be further simplified. The provider needs to be configured with an issuer-uri which is the URI that the it asserts as its Issuer Identifier. For example, if the issuer-uri provided is "https://example.com", then an OpenID Provider Configuration Request will be made to "https://example.com/.well- known/openid-configuration". The result is expected to be an OpenID Provider Configuration Response. The following example shows how an OpenID Connect Provider can be configured with the issuer-uri:

spring.security.oauth2.client.provider.oidc-provider.issuer-uri=https://dev- 123456.oktapreview.com/oauth2/default/

By default, Spring Security’s OAuth2LoginAuthenticationFilter only processes URLs matching

/login/oauth2/code/\*. If you want to customize the redirect-uri to use a different pattern, you need to provide configuration to process that custom pattern. For example, for servlet applications, you can add your own WebSecurityConfigurerAdapter that resembles the following:

public class OAuth2LoginSecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception { http

.authorizeRequests()

.anyRequest().authenticated()

.and()

.oauth2Login()

.redirectionEndpoint()

.baseUri("/custom-callback");

}

}

**OAuth2 client registration for common providers**

For common OAuth2 and OpenID providers, including Google, Github, Facebook, and Okta, we provide a set of provider defaults (google, github, facebook, and okta, respectively).

If you do not need to customize these providers, you can set the provider attribute to the one for which you need to infer defaults. Also, if the key for the client registration matches a default supported provider, Spring Boot infers that as well.

In other words, the two configurations in the following example use the Google provider:

spring.security.oauth2.client.registration.my-client.client-id=abcd spring.security.oauth2.client.registration.my-client.client-secret=password spring.security.oauth2.client.registration.my-client.provider=google

spring.security.oauth2.client.registration.google.client-id=abcd spring.security.oauth2.client.registration.google.client-secret=password

#### Resource Server

If you have spring-security-oauth2-resource-server on your classpath, Spring Boot can set up an OAuth2 Resource Server. For JWT configuration, a JWK Set URI or OIDC Issuer URI needs to be specified, as shown in the following examples:

spring.security.oauth2.resourceserver.jwt.jwk-set- uri=https://example.com/oauth2/default/v1/keys

spring.security.oauth2.resourceserver.jwt.issuer-uri=https://dev- 123456.oktapreview.com/oauth2/default/

If the authorization server does not support a JWK Set URI, you can configure the resource server with the Public Key used for verifying the signature of the JWT. This can be done using the spring.security.oauth2.resourceserver.jwt.public-key- location property, where the value needs to point to a file containing the public key in the PEM-encoded x509 format.

The same properties are applicable for both servlet and reactive applications.

Alternatively, you can define your own JwtDecoder bean for servlet applications or a

ReactiveJwtDecoder for reactive applications.

In cases where opaque tokens are used instead of JWTs, you can configure the following properties to validate tokens via introspection:

spring.security.oauth2.resourceserver.opaquetoken.introspection- uri=https://example.com/check-token spring.security.oauth2.resourceserver.opaquetoken.client-id=my-client-id spring.security.oauth2.resourceserver.opaquetoken.client-secret=my-client-secret

Again, the same properties are applicable for both servlet and reactive applications.

Alternatively, you can define your own OpaqueTokenIntrospector bean for servlet applications or a

ReactiveOpaqueTokenIntrospector for reactive applications.

#### Authorization Server

Currently, Spring Security does not provide support for implementing an OAuth 2.0 Authorization Server. However, this functionality is available from the [Spring Security OAuth](https://spring.io/projects/spring-security-oauth) project, which will eventually be superseded by Spring Security completely. Until then, you can use the spring- security-oauth2-autoconfigure module to easily set up an OAuth 2.0 authorization server; see its [documentation](https://docs.spring.io/spring-security-oauth2-boot/) for instructions.

### SAML 2.0

#### Relying Party

If you have spring-security-saml2-service-provider on your classpath, you can take advantage of some auto-configuration to set up a SAML 2.0 Relying Party. This configuration makes use of the properties under Saml2RelyingPartyProperties.

A relying party registration represents a paired configuration between an Identity Provider, IDP, and a Service Provider, SP. You can register multiple relying parties under the spring.security.saml2.relyingparty prefix, as shown in the following example:

spring.security.saml2.relyingparty.registration.my-relying- party1.signing.credentials[0].private-key-location=path-to-private-key spring.security.saml2.relyingparty.registration.my-relying- party1.signing.credentials[0].certificate-location=path-to-certificate spring.security.saml2.relyingparty.registration.my-relying- party1.identityprovider.verification.credentials[0].certificate-location=path-to- verification-cert

spring.security.saml2.relyingparty.registration.my-relying- party1.identityprovider.entity-id=remote-idp-entity-id1 spring.security.saml2.relyingparty.registration.my-relying- party1.identityprovider.sso-url=https://remoteidp1.sso.url

spring.security.saml2.relyingparty.registration.my-relying- party2.signing.credentials[0].private-key-location=path-to-private-key spring.security.saml2.relyingparty.registration.my-relying- party2.signing.credentials[0].certificate-location=path-to-certificate spring.security.saml2.relyingparty.registration.my-relying- party2.identityprovider.verification.credentials[0].certificate-location=path-to- other-verification-cert

spring.security.saml2.relyingparty.registration.my-relying- party2.identityprovider.entity-id=remote-idp-entity-id2 spring.security.saml2.relyingparty.registration.my-relying- party2.identityprovider.sso-url=https://remoteidp2.sso.url

### Actuator Security

For security purposes, all actuators other than /health and /info are disabled by default. The

management.endpoints.web.exposure.include property can be used to enable the actuators.

If Spring Security is on the classpath and no other WebSecurityConfigurerAdapter is present, all actuators other than /health and /info are secured by Spring Boot auto-configuration. If you define a custom WebSecurityConfigurerAdapter, Spring Boot auto-configuration will back off and you will be in full control of actuator access rules.

Before setting the management.endpoints.web.exposure.include, ensure that the exposed actuators do not contain sensitive information and/or are secured by placing them behind a firewall or by something like Spring Security.

#### Cross Site Request Forgery Protection

Since Spring Boot relies on Spring Security’s defaults, CSRF protection is turned on by default. This means that the actuator endpoints that require a POST (shutdown and loggers endpoints), PUT or DELETE will get a 403 forbidden error when the default security configuration is in use.

We recommend disabling CSRF protection completely only if you are creating a service that is used by non-browser clients.

Additional information about CSRF protection can be found in the [Spring Security Reference Guide](https://docs.spring.io/spring-security/site/docs/5.3.4.RELEASE/reference/html5/#csrf).

## Working with SQL Databases

The [Spring Framework](https://spring.io/projects/spring-framework) provides extensive support for working with SQL databases, from direct JDBC access using JdbcTemplate to complete “object relational mapping” technologies such as Hibernate. [Spring Data](https://spring.io/projects/spring-data) provides an additional level of functionality: creating Repository implementations directly from interfaces and using conventions to generate queries from your method names.

### Configure a DataSource

Java’s javax.sql.DataSource interface provides a standard method of working with database connections. Traditionally, a 'DataSource' uses a URL along with some credentials to establish a database connection.

See [the “How-to” section](#_bookmark590) for more advanced examples, typically to take full control over the configuration of the DataSource.

#### Embedded Database Support

It is often convenient to develop applications by using an in-memory embedded database. Obviously, in-memory databases do not provide persistent storage. You need to populate your database when your application starts and be prepared to throw away data when your application ends.

 The “How-to” section includes a [section on how to initialize a database](#_bookmark606).

Spring Boot can auto-configure embedded [H2](https://www.h2database.com/), [HSQL](http://hsqldb.org/), and [Derby](https://db.apache.org/derby/) databases. You need not provide any connection URLs. You need only include a build dependency to the embedded database that you want to use.

If you are using this feature in your tests, you may notice that the same database is reused by your whole test suite regardless of the number of application contexts that you use. If you want to make sure that each context has a separate embedded database, you should set spring.datasource.generate-unique-name to true.

For example, the typical POM dependencies would be as follows:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.hsqldb</groupId>

<artifactId>hsqldb</artifactId>

<scope>runtime</scope>

</dependency>

You need a dependency on spring-jdbc for an embedded database to be auto- configured. In this example, it is pulled in transitively through spring-boot- starter-data-jpa.

If, for whatever reason, you do configure the connection URL for an embedded database, take care to ensure that the database’s automatic shutdown is disabled. If you use H2, you should use DB\_CLOSE\_ON\_EXIT=FALSE to do so. If you use HSQLDB, you should ensure that shutdown=true is not used. Disabling the database’s automatic shutdown lets Spring Boot control when the database is closed, thereby ensuring that it happens once access to the database is no longer needed.

#### Connection to a Production Database

Production database connections can also be auto-configured by using a pooling DataSource. Spring Boot uses the following algorithm for choosing a specific implementation:

* + - 1. We prefer [HikariCP](https://github.com/brettwooldridge/HikariCP) for its performance and concurrency. If HikariCP is available, we always choose it.
      2. Otherwise, if the Tomcat pooling DataSource is available, we use it.
      3. If neither HikariCP nor the Tomcat pooling datasource are available and if [Commons DBCP2](https://commons.apache.org/proper/commons-dbcp/) is available, we use it.

If you use the spring-boot-starter-jdbc or spring-boot-starter-data-jpa “starters”, you automatically get a dependency to HikariCP.

You can bypass that algorithm completely and specify the connection pool to use by setting the spring.datasource.type property. This is especially important if you run your application in a Tomcat container, as tomcat-jdbc is provided by default.

Additional connection pools can always be configured manually. If you define your own DataSource bean, auto-configuration does not occur.

DataSource configuration is controlled by external configuration properties in spring.datasource.\*. For example, you might declare the following section in application.properties:

spring.datasource.url=jdbc:mysql://localhost/test spring.datasource.username=dbuser spring.datasource.password=dbpass spring.datasource.driver-class-name=com.mysql.jdbc.Driver

You should at least specify the URL by setting the spring.datasource.url property. Otherwise, Spring Boot tries to auto-configure an embedded database.

You often do not need to specify the driver-class-name, since Spring Boot can deduce it for most databases from the url.

For a pooling DataSource to be created, we need to be able to verify that a valid Driver class is available, so we check for that before doing anything. In other words, if you set spring.datasource.driver-class-name=com.mysql.jdbc.Driver, then that class has to be loadable.

See [DataSourceProperties](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/DataSourceProperties.java) for more of the supported options. These are the standard options that work regardless of the actual implementation. It is also possible to fine-tune implementation- specific settings by using their respective prefix (spring.datasource.hikari.\*, spring.datasource.tomcat.\*, and spring.datasource.dbcp2.\*). Refer to the documentation of the connection pool implementation you are using for more details.

For instance, if you use the [Tomcat connection pool](https://tomcat.apache.org/tomcat-9.0-doc/jdbc-pool.html#Common_Attributes), you could customize many additional settings, as shown in the following example:

# Number of ms to wait before throwing an exception if no connection is available. spring.datasource.tomcat.max-wait=10000

# Maximum number of active connections that can be allocated from this pool at the same time.

spring.datasource.tomcat.max-active=50

# Validate the connection before borrowing it from the pool. spring.datasource.tomcat.test-on-borrow=true

#### Connection to a JNDI DataSource

If you deploy your Spring Boot application to an Application Server, you might want to configure and manage your DataSource by using your Application Server’s built-in features and access it by using JNDI.

The spring.datasource.jndi-name property can be used as an alternative to the spring.datasource.url, spring.datasource.username, and spring.datasource.password properties to access the DataSource from a specific JNDI location. For example, the following section in application.properties shows how you can access a JBoss AS defined DataSource:

spring.datasource.jndi-name=java:jboss/datasources/customers

### Using JdbcTemplate

Spring’s JdbcTemplate and NamedParameterJdbcTemplate classes are auto-configured, and you can

@Autowire them directly into your own beans, as shown in the following example:

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.stereotype.Component;

@Component

public class MyBean {

private final JdbcTemplate jdbcTemplate; @Autowired

public MyBean(JdbcTemplate jdbcTemplate) { this.jdbcTemplate = jdbcTemplate;

}

// ...

}

You can customize some properties of the template by using the spring.jdbc.template.\* properties, as shown in the following example:

spring.jdbc.template.max-rows=500

The NamedParameterJdbcTemplate reuses the same JdbcTemplate instance behind the scenes. If more than one JdbcTemplate is defined and no primary candidate exists, the NamedParameterJdbcTemplate is not auto-configured.

### JPA and Spring Data JPA

The Java Persistence API is a standard technology that lets you “map” objects to relational databases. The spring-boot-starter-data-jpa POM provides a quick way to get started. It provides the following key dependencies:

* Hibernate: One of the most popular JPA implementations.
* Spring Data JPA: Helps you to implement JPA-based repositories.
* Spring ORM: Core ORM support from the Spring Framework.

We do not go into too many details of JPA or [Spring Data](https://spring.io/projects/spring-data) here. You can follow the [“Accessing Data with JPA”](https://spring.io/guides/gs/accessing-data-jpa/) guide from [spring.io](https://spring.io/) and read the [Spring Data JPA](https://spring.io/projects/spring-data-jpa) and [Hibernate](https://hibernate.org/orm/documentation/) reference documentation.

#### Entity Classes

Traditionally, JPA “Entity” classes are specified in a persistence.xml file. With Spring Boot, this file is not necessary and “Entity Scanning” is used instead. By default, all packages below your main configuration class (the one annotated with @EnableAutoConfiguration or @SpringBootApplication)

are searched.

Any classes annotated with @Entity, @Embeddable, or @MappedSuperclass are considered. A typical entity class resembles the following example:

package com.example.myapp.domain;

import java.io.Serializable; import javax.persistence.\*;

@Entity

public class City implements Serializable {

@Id @GeneratedValue private Long id;

@Column(nullable = false) private String name;

@Column(nullable = false) private String state;

// ... additional members, often include @OneToMany mappings protected City() {

// no-args constructor required by JPA spec

// this one is protected since it shouldn't be used directly

}

public City(String name, String state) { this.name = name;

this.state = state;

}

public String getName() { return this.name;

}

public String getState() { return this.state;

}

// ... etc

}

You can customize entity scanning locations by using the @EntityScan annotation. See the “[Separate @Entity Definitions from Spring Configuration](#_bookmark593)” how-to.

#### Spring Data JPA Repositories

[Spring Data JPA](https://spring.io/projects/spring-data-jpa) repositories are interfaces that you can define to access data. JPA queries are created automatically from your method names. For example, a CityRepository interface might declare a findAllByState(String state) method to find all the cities in a given state.

For more complex queries, you can annotate your method with Spring Data’s [Query](https://docs.spring.io/spring-data/jpa/docs/2.3.4.RELEASE/api/org/springframework/data/jpa/repository/Query.html) annotation.

Spring Data repositories usually extend from the [Repository](https://docs.spring.io/spring-data/commons/docs/2.3.4.RELEASE/api/org/springframework/data/repository/Repository.html) or [CrudRepository](https://docs.spring.io/spring-data/commons/docs/2.3.4.RELEASE/api/org/springframework/data/repository/CrudRepository.html) interfaces. If you use auto-configuration, repositories are searched from the package containing your main configuration class (the one annotated with @EnableAutoConfiguration or @SpringBootApplication) down.

The following example shows a typical Spring Data repository interface definition:

package com.example.myapp.domain;

import org.springframework.data.domain.\*; import org.springframework.data.repository.\*;

public interface CityRepository extends Repository<City, Long> { Page<City> findAll(Pageable pageable);

City findByNameAndStateAllIgnoringCase(String name, String state);

}

Spring Data JPA repositories support three different modes of bootstrapping: default, deferred, and lazy. To enable deferred or lazy bootstrapping, set the spring.data.jpa.repositories.bootstrap-mode property to deferred or lazy respectively. When using deferred or lazy bootstrapping, the auto- configured EntityManagerFactoryBuilder will use the context’s AsyncTaskExecutor, if any, as the bootstrap executor. If more than one exists, the one named applicationTaskExecutor will be used.

We have barely scratched the surface of Spring Data JPA. For complete details, see the [Spring Data JPA reference documentation](https://docs.spring.io/spring-data/jdbc/docs/2.0.4.RELEASE/reference/html/).

#### Creating and Dropping JPA Databases

By default, JPA databases are automatically created **only** if you use an embedded database (H2, HSQL, or Derby). You can explicitly configure JPA settings by using spring.jpa.\* properties. For example, to create and drop tables you can add the following line to your application.properties:

spring.jpa.hibernate.ddl-auto=create-drop

Hibernate’s own internal property name for this (if you happen to remember it better) is hibernate.hbm2ddl.auto. You can set it, along with other Hibernate native properties, by using spring.jpa.properties.\* (the prefix is stripped before adding them to the entity manager). The following line shows an example of setting JPA properties for Hibernate:

spring.jpa.properties.hibernate.globally\_quoted\_identifiers=true

The line in the preceding example passes a value of true for the

hibernate.globally\_quoted\_identifiers property to the Hibernate entity manager.

By default, the DDL execution (or validation) is deferred until the ApplicationContext has started. There is also a spring.jpa.generate-ddl flag, but it is not used if Hibernate auto-configuration is active, because the ddl-auto settings are more fine-grained.

#### Open EntityManager in View

If you are running a web application, Spring Boot by default registers [OpenEntityManagerInViewInterceptor](https://docs.spring.io/spring/docs/5.2.9.RELEASE/javadoc-api/org/springframework/orm/jpa/support/OpenEntityManagerInViewInterceptor.html) to apply the “Open EntityManager in View” pattern, to allow for lazy loading in web views. If you do not want this behavior, you should set spring.jpa.open-in- view to false in your application.properties.

### Spring Data JDBC

Spring Data includes repository support for JDBC and will automatically generate SQL for the methods on CrudRepository. For more advanced queries, a @Query annotation is provided.

Spring Boot will auto-configure Spring Data’s JDBC repositories when the necessary dependencies are on the classpath. They can be added to your project with a single dependency on spring-boot- starter-data-jdbc. If necessary, you can take control of Spring Data JDBC’s configuration by adding the @EnableJdbcRepositories annotation or a JdbcConfiguration subclass to your application.

For complete details of Spring Data JDBC, please refer to the [reference](https://docs.spring.io/spring-data/jdbc/docs/2.0.4.RELEASE/reference/html/) [documentation](https://docs.spring.io/spring-data/jdbc/docs/2.0.4.RELEASE/reference/html/).

### Using H2’s Web Console

The [H2 database](https://www.h2database.com/) provides a [browser-based console](https://www.h2database.com/html/quickstart.html#h2_console) that Spring Boot can auto-configure for you. The console is auto-configured when the following conditions are met:

* You are developing a servlet-based web application.
* com.h2database:h2 is on the classpath.
* You are using [Spring Boot’s developer tools](#_bookmark65).

If you are not using Spring Boot’s developer tools but would still like to make use of H2’s console, you can configure the spring.h2.console.enabled property with a value of true.

The H2 console is only intended for use during development, so you should take care to ensure that spring.h2.console.enabled is not set to true in production.

#### Changing the H2 Console’s Path

By default, the console is available at /h2-console. You can customize the console’s path by using the

spring.h2.console.path property.

### Using jOOQ

jOOQ Object Oriented Querying ([jOOQ](https://www.jooq.org/)) is a popular product from [Data Geekery](https://www.datageekery.com/) which generates Java code from your database and lets you build type-safe SQL queries through its fluent API. Both the commercial and open source editions can be used with Spring Boot.

#### Code Generation

In order to use jOOQ type-safe queries, you need to generate Java classes from your database schema. You can follow the instructions in the [jOOQ user manual](https://www.jooq.org/doc/3.13.4/manual-single-page/#jooq-in-7-steps-step3). If you use the jooq-codegen-maven plugin and you also use the spring-boot-starter-parent “parent POM”, you can safely omit the plugin’s <version> tag. You can also use Spring Boot-defined version variables (such as h2.version) to declare the plugin’s database dependency. The following listing shows an example:

<plugin>

<groupId>org.jooq</groupId>

<artifactId>jooq-codegen-maven</artifactId>

<executions>

...

</executions>

<dependencies>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>${h2.version}</version>

</dependency>

</dependencies>

<configuration>

<jdbc>

<driver>org.h2.Driver</driver>

<url>jdbc:h2:~/yourdatabase</url>

</jdbc>

<generator>

...

</generator>

</configuration>

</plugin>

#### Using DSLContext

The fluent API offered by jOOQ is initiated through the org.jooq.DSLContext interface. Spring Boot auto-configures a DSLContext as a Spring Bean and connects it to your application DataSource. To use the DSLContext, you can @Autowire it, as shown in the following example:

@Component

public class JooqExample implements CommandLineRunner { private final DSLContext create;

@Autowired

public JooqExample(DSLContext dslContext) { this.create = dslContext;

}

}

 The jOOQ manual tends to use a variable named create to hold the DSLContext. You can then use the DSLContext to construct your queries, as shown in the following example:

public List<GregorianCalendar> authorsBornAfter1980() { return this.create.selectFrom(AUTHOR)

.where(AUTHOR.DATE\_OF\_BIRTH.greaterThan(new GregorianCalendar(1980, 0, 1)))

.fetch(AUTHOR.DATE\_OF\_BIRTH);

}

#### jOOQ SQL Dialect

Unless the spring.jooq.sql-dialect property has been configured, Spring Boot determines the SQL dialect to use for your datasource. If Spring Boot could not detect the dialect, it uses DEFAULT.

Spring Boot can only auto-configure dialects supported by the open source version of jOOQ.

#### Customizing jOOQ

More advanced customizations can be achieved by defining your own @Bean definitions, which is used when the jOOQ Configuration is created. You can define beans for the following jOOQ Types:

* ConnectionProvider
* ExecutorProvider
* TransactionProvider
* RecordMapperProvider
* RecordUnmapperProvider
* Settings
* RecordListenerProvider
* ExecuteListenerProvider
* VisitListenerProvider
* TransactionListenerProvider

You can also create your own org.jooq.Configuration @Bean if you want to take complete control of the jOOQ configuration.

### Using R2DBC

The Reactive Relational Database Connectivity ([R2DBC](https://r2dbc.io/)) project brings reactive programming APIs to relational databases. R2DBC’s io.r2dbc.spi.Connection provides a standard method of working with non-blocking database connections. Connections are provided via a ConnectionFactory, similar to a DataSource with jdbc.

ConnectionFactory configuration is controlled by external configuration properties in

spring.r2dbc.\*. For example, you might declare the following section in application.properties:

spring.r2dbc.url=r2dbc:postgresql://localhost/test spring.r2dbc.username=dbuser spring.r2dbc.password=dbpass

You do not need to specify a driver class name, since Spring Boot obtains the driver from R2DBC’s Connection Factory discovery.

At least the url should be provided. Information specified in the URL takes  precedence over individual properties, i.e. name, username, password and pooling

options.

 The “How-to” section includes a [section on how to initialize a database](#_bookmark610).

To customize the connections created by a ConnectionFactory, i.e., set specific parameters that you do not want (or cannot) configure in your central database configuration, you can use a ConnectionFactoryOptionsBuilderCustomizer @Bean. The following example shows how to manually override the database port while the rest of the options is taken from the application configuration:

@Bean

public ConnectionFactoryOptionsBuilderCustomizer connectionFactoryPortCustomizer() { return (builder) -> builder.option(PORT, 5432);

}

The following examples shows how to set some PostgreSQL connection options:

@Bean

public ConnectionFactoryOptionsBuilderCustomizer postgresCustomizer() { Map<String, String> options = new HashMap<>(); options.put("lock\_timeout", "30s"); options.put("statement\_timeout", "60s");

return (builder) -> builder.option(OPTIONS, options);

}

When a ConnectionFactory bean is available, the regular JDBC DataSource auto-configuration backs off. If you want to retain the JDBC DataSource auto-configuration, and are comfortable with the risk of using the blocking JDBC API in a reactive application, add @Import(DataSourceAutoConfiguration.class) on a @Configuration class in your application to re- enable it.

#### Embedded Database Support

Similarly to [the JDBC support](#_bookmark200), Spring Boot can automatically configure an embedded database for reactive usage. You need not provide any connection URLs. You need only include a build dependency to the embedded database that you want to use, as shown in the following example:

<dependency>

<groupId>io.r2dbc</groupId>

<artifactId>r2dbc-h2</artifactId>

<scope>runtime</scope>

</dependency>

If you are using this feature in your tests, you may notice that the same database is reused by your whole test suite regardless of the number of application contexts that you use. If you want to make sure that each context has a separate embedded database, you should set spring.r2dbc.generate-unique-name to true.

#### Using DatabaseClient

Spring Data’s DatabaseClient class is auto-configured, and you can @Autowire it directly into your own beans, as shown in the following example:

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.data.r2dbc.function.DatabaseClient; import org.springframework.stereotype.Component;

@Component

public class MyBean {

private final DatabaseClient databaseClient; @Autowired

public MyBean(DatabaseClient databaseClient) { this.databaseClient = databaseClient;

}

// ...

}

#### Spring Data R2DBC Repositories

[Spring Data R2DBC](https://spring.io/projects/spring-data-r2dbc) repositories are interfaces that you can define to access data. Queries are created automatically from your method names. For example, a CityRepository interface might declare a findAllByState(String state) method to find all the cities in a given state.

For more complex queries, you can annotate your method with Spring Data’s [Query](https://docs.spring.io/spring-data/r2dbc/docs/1.1.4.RELEASE/api/org/springframework/data/r2dbc/repository/Query.html) annotation.

Spring Data repositories usually extend from the [Repository](https://docs.spring.io/spring-data/commons/docs/2.3.4.RELEASE/api/org/springframework/data/repository/Repository.html) or [CrudRepository](https://docs.spring.io/spring-data/commons/docs/2.3.4.RELEASE/api/org/springframework/data/repository/CrudRepository.html) interfaces. If you use auto-configuration, repositories are searched from the package containing your main configuration class (the one annotated with @EnableAutoConfiguration or @SpringBootApplication) down.

The following example shows a typical Spring Data repository interface definition:

package com.example.myapp.domain;

import org.springframework.data.domain.\*; import org.springframework.data.repository.\*; import reactor.core.publisher.Mono;

public interface CityRepository extends Repository<City, Long> {

Mono<City> findByNameAndStateAllIgnoringCase(String name, String state);

}

We have barely scratched the surface of Spring Data R2DBC. For complete details, see the [Spring Data R2DBC reference documentation](https://docs.spring.io/spring-data/r2dbc/docs/1.1.4.RELEASE/reference/html/).

## Working with NoSQL Technologies

Spring Data provides additional projects that help you access a variety of NoSQL technologies, including:

* [MongoDB](https://spring.io/projects/spring-data-mongodb)
* [Neo4J](https://spring.io/projects/spring-data-neo4j)
* [Elasticsearch](https://spring.io/projects/spring-data-elasticsearch)
* [Solr](https://spring.io/projects/spring-data-solr)
* [Redis](https://spring.io/projects/spring-data-redis)
* [GemFire](https://spring.io/projects/spring-data-gemfire) or [Geode](https://spring.io/projects/spring-data-geode)
* [Cassandra](https://spring.io/projects/spring-data-cassandra)
* [Couchbase](https://spring.io/projects/spring-data-couchbase)
* [LDAP](https://spring.io/projects/spring-data-ldap)

Spring Boot provides auto-configuration for Redis, MongoDB, Neo4j, Elasticsearch, Solr Cassandra,

Couchbase, and LDAP. You can make use of the other projects, but you must configure them yourself. Refer to the appropriate reference documentation at [spring.io/projects/spring-data](https://spring.io/projects/spring-data).

### Redis

[Redis](https://redis.io/) is a cache, message broker, and richly-featured key-value store. Spring Boot offers basic auto- configuration for the [Lettuce](https://github.com/lettuce-io/lettuce-core/) and [Jedis](https://github.com/xetorthio/jedis/) client libraries and the abstractions on top of them provided by [Spring Data Redis](https://github.com/spring-projects/spring-data-redis).

There is a spring-boot-starter-data-redis “Starter” for collecting the dependencies in a convenient way. By default, it uses [Lettuce](https://github.com/lettuce-io/lettuce-core/). That starter handles both traditional and reactive applications.

we also provide a spring-boot-starter-data-redis-reactive “Starter” for consistency with the other stores with reactive support.

#### Connecting to Redis

You can inject an auto-configured RedisConnectionFactory, StringRedisTemplate, or vanilla RedisTemplate instance as you would any other Spring Bean. By default, the instance tries to connect to a Redis server at localhost:6379. The following listing shows an example of such a bean:

@Component

public class MyBean {

private StringRedisTemplate template; @Autowired

public MyBean(StringRedisTemplate template) { this.template = template;

}

// ...

}

You can also register an arbitrary number of beans that implement LettuceClientConfigurationBuilderCustomizer for more advanced customizations. If you use Jedis, JedisClientConfigurationBuilderCustomizer is also available.

If you add your own @Bean of any of the auto-configured types, it replaces the default (except in the case of RedisTemplate, when the exclusion is based on the bean name, redisTemplate, not its type). By default, if commons-pool2 is on the classpath, you get a pooled connection factory.

### MongoDB

[MongoDB](https://www.mongodb.com/) is an open-source NoSQL document database that uses a JSON-like schema instead of traditional table-based relational data. Spring Boot offers several conveniences for working with MongoDB, including the spring-boot-starter-data-mongodb and spring-boot-starter-data-mongodb- reactive “Starters”.

#### Connecting to a MongoDB Database

To access MongoDB databases, you can inject an auto-configured org.springframework.data.mongodb.MongoDatabaseFactory. By default, the instance tries to connect to a MongoDB server at mongodb://localhost/test. The following example shows how to connect to a MongoDB database:

import org.springframework.data.mongodb.MongoDatabaseFactory; import com.mongodb.client.MongoDatabase;

@Component

public class MyBean {

private final MongoDatabaseFactory mongo; @Autowired

public MyBean(MongoDatabaseFactory mongo) { this.mongo = mongo;

}

// ...

public void example() {

MongoDatabase db = mongo.getMongoDatabase();

// ...

}

}

You can set the spring.data.mongodb.uri property to change the URL and configure additional settings such as the *replica set*, as shown in the following example:

spring.data.mongodb.uri=mongodb://user:secret@mongo1.example.com:12345,mongo2.example. com:23456/test

Alternatively, you can specify connection details using discrete properties. For example, you might declare the following settings in your application.properties:

spring.data.mongodb.host=mongoserver.example.com spring.data.mongodb.port=27017 spring.data.mongodb.database=test spring.data.mongodb.username=user spring.data.mongodb.password=secret

If you have defined your own MongoClient, it will be used to auto-configure a suitable

MongoDatabaseFactory.

The auto-configured MongoClient is created using MongoClientSettings. To fine-tune its configuration, declare one or more MongoClientSettingsBuilderCustomizer beans. Each will be called in order with the MongoClientSettings.Builder that is used to build the MongoClientSettings.

If spring.data.mongodb.port is not specified, the default of 27017 is used. You could delete this line from the example shown earlier.

If you do not use Spring Data MongoDB, you can inject a MongoClient bean instead of using MongoDatabaseFactory. If you want to take complete control of establishing the MongoDB connection, you can also declare your own MongoDatabaseFactory or MongoClient bean.

If you are using the reactive driver, Netty is required for SSL. The auto- configuration configures this factory automatically if Netty is available and the factory to use hasn’t been customized already.

#### MongoTemplate

[Spring Data MongoDB](https://spring.io/projects/spring-data-mongodb) provides a [MongoTemplate](https://docs.spring.io/spring-data/mongodb/docs/3.0.4.RELEASE/api/org/springframework/data/mongodb/core/MongoTemplate.html) class that is very similar in its design to Spring’s JdbcTemplate. As with JdbcTemplate, Spring Boot auto-configures a bean for you to inject the template, as follows:

import org.springframework.data.mongodb.core.MongoTemplate; import org.springframework.stereotype.Component;

@Component

public class MyBean {

private final MongoTemplate mongoTemplate; public MyBean(MongoTemplate mongoTemplate) {

this.mongoTemplate = mongoTemplate;

}

// ...

}

See the [MongoOperations Javadoc](https://docs.spring.io/spring-data/mongodb/docs/3.0.4.RELEASE/api/org/springframework/data/mongodb/core/MongoOperations.html) for complete details.

#### Spring Data MongoDB Repositories

Spring Data includes repository support for MongoDB. As with the JPA repositories discussed earlier, the basic principle is that queries are constructed automatically, based on method names.

In fact, both Spring Data JPA and Spring Data MongoDB share the same common infrastructure. You could take the JPA example from earlier and, assuming that City is now a MongoDB data class rather than a JPA @Entity, it works in the same way, as shown in the following example:

package com.example.myapp.domain;

import org.springframework.data.domain.\*; import org.springframework.data.repository.\*;

public interface CityRepository extends Repository<City, Long> { Page<City> findAll(Pageable pageable);

City findByNameAndStateAllIgnoringCase(String name, String state);

}

You can customize document scanning locations by using the @EntityScan

annotation.

For complete details of Spring Data MongoDB, including its rich object mapping technologies, refer to its [reference documentation](https://spring.io/projects/spring-data-mongodb).

#### Embedded Mongo

Spring Boot offers auto-configuration for [Embedded Mongo](https://github.com/flapdoodle-oss/de.flapdoodle.embed.mongo). To use it in your Spring Boot application, add a dependency on de.flapdoodle.embed:de.flapdoodle.embed.mongo.

The port that Mongo listens on can be configured by setting the spring.data.mongodb.port property. To use a randomly allocated free port, use a value of 0. The MongoClient created by MongoAutoConfiguration is automatically configured to use the randomly allocated port.

If you do not configure a custom port, the embedded support uses a random port (rather than 27017) by default.

If you have SLF4J on the classpath, the output produced by Mongo is automatically routed to a logger named org.springframework.boot.autoconfigure.mongo.embedded.EmbeddedMongo.

You can declare your own IMongodConfig and IRuntimeConfig beans to take control of the Mongo instance’s configuration and logging routing. The download configuration can be customized by declaring a DownloadConfigBuilderCustomizer bean.

### Neo4j

[Neo4j](https://neo4j.com/) is an open-source NoSQL graph database that uses a rich data model of nodes connected by first class relationships, which is better suited for connected big data than traditional RDBMS approaches. Spring Boot offers several conveniences for working with Neo4j, including the spring- boot-starter-data-neo4j “Starter”.

#### Connecting to a Neo4j Database

To access a Neo4j server, you can inject an auto-configured org.neo4j.ogm.session.Session. By

default, the instance tries to connect to a Neo4j server at localhost:7687 using the Bolt protocol. The following example shows how to inject a Neo4j Session:

@Component

public class MyBean {

private final Session session; @Autowired

public MyBean(Session session) { this.session = session;

}

// ...

}

You can configure the uri and credentials to use by setting the spring.data.neo4j.\* properties, as shown in the following example:

spring.data.neo4j.uri=bolt://my-server:7687 spring.data.neo4j.username=neo4j spring.data.neo4j.password=secret

You can take full control over the session creation by adding either an

org.neo4j.ogm.config.Configuration bean or an org.neo4j.ogm.session.SessionFactory bean.

#### Using the Embedded Mode

If you add org.neo4j:neo4j-ogm-embedded-driver to the dependencies of your application, Spring Boot automatically configures an in-process embedded instance of Neo4j that does not persist any data when your application shuts down.

As the embedded Neo4j OGM driver does not provide the Neo4j kernel itself, you have to declare org.neo4j:neo4j as dependency yourself. Refer to [the Neo4j OGM](https://neo4j.com/docs/ogm-manual/current/reference/#reference%3Agetting-started) [documentation](https://neo4j.com/docs/ogm-manual/current/reference/#reference%3Agetting-started) for a list of compatible versions.

The embedded driver takes precedence over the other drivers when there are multiple drivers on the classpath. You can explicitly disable the embedded mode by setting spring.data.neo4j.embedded.enabled=false.

[Data Neo4j Tests](#_bookmark325) automatically make use of an embedded Neo4j instance if the embedded driver and Neo4j kernel are on the classpath as described above.

You can enable persistence for the embedded mode by providing a path to a database file in your configuration, e.g. spring.data.neo4j.uri=file://var/tmp/graph.db.

#### Using Native Types

Neo4j-OGM can map some types, like those in java.time.\*, to String-based properties or to one of the native types that Neo4j provides. For backwards compatibility reasons the default for Neo4j- OGM is to use a String-based representation. To use native types, add a dependency on either org.neo4j:neo4j-ogm-bolt-native-types or org.neo4j:neo4j-ogm-embedded-native-types, and configure the spring.data.neo4j.use-native-types property as shown in the following example:

spring.data.neo4j.use-native-types=true

#### Neo4jSession

By default, if you are running a web application, the session is bound to the thread for the entire processing of the request (that is, it uses the "Open Session in View" pattern). If you do not want this behavior, add the following line to your application.properties file:

spring.data.neo4j.open-in-view=false

#### Spring Data Neo4j Repositories

Spring Data includes repository support for Neo4j.

Spring Data Neo4j shares the common infrastructure with Spring Data JPA as many other Spring Data modules do. You could take the JPA example from earlier and define City as Neo4j OGM @NodeEntity rather than JPA @Entity and the repository abstraction works in the same way, as shown in the following example:

package com.example.myapp.domain; import java.util.Optional;

import org.springframework.data.neo4j.repository.\*;

public interface CityRepository extends Neo4jRepository<City, Long> { Optional<City> findOneByNameAndState(String name, String state);

}

The spring-boot-starter-data-neo4j “Starter” enables the repository support as well as transaction management. You can customize the locations to look for repositories and entities by using @EnableNeo4jRepositories and @EntityScan respectively on a @Configuration-bean.

For complete details of Spring Data Neo4j, including its object mapping technologies, refer to the [reference documentation](https://docs.spring.io/spring-data/neo4j/docs/5.3.4.RELEASE/reference/html/).

### Solr

[Apache Solr](https://lucene.apache.org/solr/) is a search engine. Spring Boot offers basic auto-configuration for the Solr 5 client library and the abstractions on top of it provided by [Spring Data Solr](https://github.com/spring-projects/spring-data-solr). There is a spring-boot- starter-data-solr “Starter” for collecting the dependencies in a convenient way.

#### Connecting to Solr

You can inject an auto-configured SolrClient instance as you would any other Spring bean. By default, the instance tries to connect to a server at localhost:8983/solr. The following example shows how to inject a Solr bean:

@Component

public class MyBean { private SolrClient solr;

@Autowired

public MyBean(SolrClient solr) { this.solr = solr;

}

// ...

}

If you add your own @Bean of type SolrClient, it replaces the default.

#### Spring Data Solr Repositories

Spring Data includes repository support for Apache Solr. As with the JPA repositories discussed earlier, the basic principle is that queries are automatically constructed for you based on method names.

In fact, both Spring Data JPA and Spring Data Solr share the same common infrastructure. You could take the JPA example from earlier and, assuming that City is now a @SolrDocument class rather than a JPA @Entity, it works in the same way.

IP: For complete details of Spring Data Solr, refer to the [reference documentation](https://docs.spring.io/spring-data/solr/docs/4.2.4.RELEASE/reference/html/).

### Elasticsearch

[Elasticsearch](https://www.elastic.co/products/elasticsearch) is an open source, distributed, RESTful search and analytics engine. Spring Boot offers basic auto-configuration for Elasticsearch.

Spring Boot supports several clients:

* The official Java "Low Level" and "High Level" REST clients
* The ReactiveElasticsearchClient provided by Spring Data Elasticsearch

Spring Boot provides a dedicated “Starter”, spring-boot-starter-data-elasticsearch.

#### Connecting to Elasticsearch using REST clients

Elasticsearch ships [two different REST clients](https://www.elastic.co/guide/en/elasticsearch/client/java-rest/current/index.html) that you can use to query a cluster: the "Low Level" client and the "High Level" client.

If you have the org.elasticsearch.client:elasticsearch-rest-client dependency on the classpath, Spring Boot will auto-configure and register a RestClient bean that by default targets localhost:9200. You can further tune how RestClient is configured, as shown in the following example:

spring.elasticsearch.rest.uris=https://search.example.com:9200 spring.elasticsearch.rest.read-timeout=10s spring.elasticsearch.rest.username=user spring.elasticsearch.rest.password=secret

You can also register an arbitrary number of beans that implement RestClientBuilderCustomizer for more advanced customizations. To take full control over the registration, define a RestClientBuilder bean.

If you have the org.elasticsearch.client:elasticsearch-rest-high-level-client dependency on the classpath, Spring Boot will auto-configure a RestHighLevelClient, which leverages any existing RestClientBuilder bean, reusing its HTTP configuration.

#### Connecting to Elasticsearch using Reactive REST clients

[Spring Data Elasticsearch](https://spring.io/projects/spring-data-elasticsearch) ships ReactiveElasticsearchClient for querying Elasticsearch instances in a reactive fashion. It is built on top of WebFlux’s WebClient, so both spring-boot-starter- elasticsearch and spring-boot-starter-webflux dependencies are useful to enable this support.

By default, Spring Boot will auto-configure and register a ReactiveElasticsearchClient bean that targets localhost:9200. You can further tune how it is configured, as shown in the following example:

spring.data.elasticsearch.client.reactive.endpoints=search.example.com:9200 spring.data.elasticsearch.client.reactive.use-ssl=true spring.data.elasticsearch.client.reactive.socket-timeout=10s spring.data.elasticsearch.client.reactive.username=user spring.data.elasticsearch.client.reactive.password=secret

If the configuration properties are not enough and you’d like to fully control the client configuration, you can register a custom ClientConfiguration bean.

#### Connecting to Elasticsearch by Using Spring Data

To connect to Elasticsearch, a RestHighLevelClient bean must be defined, auto-configured by Spring Boot or manually provided by the application (see previous sections). With this configuration in

place, an ElasticsearchRestTemplate can be injected like any other Spring bean, as shown in the following example:

@Component

public class MyBean {

private final ElasticsearchRestTemplate template; public MyBean(ElasticsearchRestTemplate template) {

this.template = template;

}

// ...

}

In the presence of spring-data-elasticsearch and the required dependencies for using a WebClient (typically spring-boot-starter-webflux), Spring Boot can also auto-configure a [ReactiveElasticsearchClient](#_bookmark240) and a ReactiveElasticsearchTemplate as beans. They are the reactive equivalent of the other REST clients.

#### Spring Data Elasticsearch Repositories

Spring Data includes repository support for Elasticsearch. As with the JPA repositories discussed earlier, the basic principle is that queries are constructed for you automatically based on method names.

In fact, both Spring Data JPA and Spring Data Elasticsearch share the same common infrastructure. You could take the JPA example from earlier and, assuming that City is now an Elasticsearch @Document class rather than a JPA @Entity, it works in the same way.

For complete details of Spring Data Elasticsearch, refer to the [reference](https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/) [documentation](https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/).

Spring Boot supports both classic and reactive Elasticsearch repositories, using the ElasticsearchRestTemplate or ReactiveElasticsearchTemplate beans. Most likely those beans are auto-configured by Spring Boot given the required dependencies are present.

If you wish to use your own template for backing the Elasticsearch repositories, you can add your own ElasticsearchRestTemplate or ElasticsearchOperations @Bean, as long as it is named "elasticsearchTemplate". Same applies to ReactiveElasticsearchTemplate and ReactiveElasticsearchOperations, with the bean name "reactiveElasticsearchTemplate".

You can choose to disable the repositories support with the following property:

spring.data.elasticsearch.repositories.enabled=false

### Cassandra

[Cassandra](https://cassandra.apache.org/) is an open source, distributed database management system designed to handle large amounts of data across many commodity servers. Spring Boot offers auto-configuration for Cassandra and the abstractions on top of it provided by [Spring Data Cassandra](https://github.com/spring-projects/spring-data-cassandra). There is a spring- boot-starter-data-cassandra “Starter” for collecting the dependencies in a convenient way.

#### Connecting to Cassandra

You can inject an auto-configured CassandraTemplate or a Cassandra CqlSession instance as you would with any other Spring Bean. The spring.data.cassandra.\* properties can be used to customize the connection. Generally, you provide keyspace-name and contact-points as well the local datacenter name, as shown in the following example:

spring.data.cassandra.keyspace-name=mykeyspace

spring.data.cassandra.contact-points=cassandrahost1:9042,cassandrahost2:9042 spring.data.cassandra.local-datacenter=datacenter1

If the port is the same for all your contact points you can use a shortcut and only specify the host names, as shown in the following example:

spring.data.cassandra.keyspace-name=mykeyspace spring.data.cassandra.contact-points=cassandrahost1,cassandrahost2 spring.data.cassandra.local-datacenter=datacenter1

Those two examples are identical as the port default to 9042. If you need to configure the port, use spring.data.cassandra.port.

The Cassandra driver has its own configuration infrastructure that loads an

application.conf at the root of the classpath.

Spring Boot does not look for such a file and rather provides a number of configuration properties via the spring.data.cassandra.\* namespace. For more advanced driver customizations, you can register an arbitrary number of beans that implement DriverConfigLoaderBuilderCustomizer. The CqlSession can be customized with a bean of type CqlSessionBuilderCustomizer.

If you’re using CqlSessionBuilder to create multiple CqlSession beans, keep in mind the builder is mutable so make sure to inject a fresh copy for each session.

The following code listing shows how to inject a Cassandra bean:

@Component

public class MyBean {

private final CassandraTemplate template; public MyBean(CassandraTemplate template) {

this.template = template;

}

// ...

}

If you add your own @Bean of type CassandraTemplate, it replaces the default.

#### Spring Data Cassandra Repositories

Spring Data includes basic repository support for Cassandra. Currently, this is more limited than the JPA repositories discussed earlier and needs to annotate finder methods with @Query.

For complete details of Spring Data Cassandra, refer to the [reference](https://docs.spring.io/spring-data/cassandra/docs/) [documentation](https://docs.spring.io/spring-data/cassandra/docs/).

### Couchbase

[Couchbase](https://www.couchbase.com/) is an open-source, distributed, multi-model NoSQL document-oriented database that is optimized for interactive applications. Spring Boot offers auto-configuration for Couchbase and the abstractions on top of it provided by [Spring Data Couchbase](https://github.com/spring-projects/spring-data-couchbase). There are spring-boot-starter-data- couchbase and spring-boot-starter-data-couchbase-reactive “Starters” for collecting the dependencies in a convenient way.

#### Connecting to Couchbase

You can get a Cluster by adding the Couchbase SDK and some configuration. The spring.couchbase.\* properties can be used to customize the connection. Generally, you provide the [connection string](https://github.com/couchbaselabs/sdk-rfcs/blob/master/rfc/0011-connection-string.md), username, and password, as shown in the following example:

spring.couchbase.connection-string=couchbase://192.168.1.123 spring.couchbase.username=user spring.couchbase.password=secret

It is also possible to customize some of the ClusterEnvironment settings. For instance, the following configuration changes the timeout to use to open a new Bucket and enables SSL support:

spring.couchbase.env.timeouts.connect=3000 spring.couchbase.env.ssl.key-store=/location/of/keystore.jks spring.couchbase.env.ssl.key-store-password=secret

Check the spring.couchbase.env.\* properties for more details. To take more control, one or more ClusterEnvironmentBuilderCustomizer beans can be used.

#### Spring Data Couchbase Repositories

Spring Data includes repository support for Couchbase. For complete details of Spring Data Couchbase, refer to the [reference documentation](https://docs.spring.io/spring-data/couchbase/docs/4.0.4.RELEASE/reference/html/).

You can inject an auto-configured CouchbaseTemplate instance as you would with any other Spring Bean, provided a CouchbaseClientFactory bean is available. This happens when a Cluster is available, as described above, and a bucket name has been specified:

spring.data.couchbase.bucket-name=my-bucket

The following examples shows how to inject a CouchbaseTemplate bean:

@Component

public class MyBean {

private final CouchbaseTemplate template; @Autowired

public MyBean(CouchbaseTemplate template) { this.template = template;

}

// ...

}

There are a few beans that you can define in your own configuration to override those provided by the auto-configuration:

* A CouchbaseMappingContext @Bean with a name of couchbaseMappingContext.
* A CustomConversions @Bean with a name of couchbaseCustomConversions.
* A CouchbaseTemplate @Bean with a name of couchbaseTemplate.

To avoid hard-coding those names in your own config, you can reuse BeanNames provided by Spring Data Couchbase. For instance, you can customize the converters to use, as follows:

@Configuration(proxyBeanMethods = false) public class SomeConfiguration {

@Bean(BeanNames.COUCHBASE\_CUSTOM\_CONVERSIONS)

public CustomConversions myCustomConversions() { return new CustomConversions(...);

}

// ...

}

### LDAP

[LDAP](https://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol) (Lightweight Directory Access Protocol) is an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over an IP network. Spring Boot offers auto-configuration for any compliant LDAP server as well as support for the embedded in-memory LDAP server from [UnboundID](https://ldap.com/unboundid-ldap-sdk-for-java/).

LDAP abstractions are provided by [Spring Data LDAP](https://github.com/spring-projects/spring-data-ldap). There is a spring-boot-starter-data-ldap

“Starter” for collecting the dependencies in a convenient way.

#### Connecting to an LDAP Server

To connect to an LDAP server, make sure you declare a dependency on the spring-boot-starter- data-ldap “Starter” or spring-ldap-core and then declare the URLs of your server in your application.properties, as shown in the following example:

spring.ldap.urls=ldap://myserver:1235 spring.ldap.username=admin spring.ldap.password=secret

If you need to customize connection settings, you can use the spring.ldap.base and

spring.ldap.base-environment properties.

An LdapContextSource is auto-configured based on these settings. If a DirContextAuthenticationStrategy bean is available, it is associated to the auto-configured LdapContextSource. If you need to customize it, for instance to use a PooledContextSource, you can still inject the auto-configured LdapContextSource. Make sure to flag your customized ContextSource as @Primary so that the auto-configured LdapTemplate uses it.

#### Spring Data LDAP Repositories

Spring Data includes repository support for LDAP. For complete details of Spring Data LDAP, refer to the [reference documentation](https://docs.spring.io/spring-data/ldap/docs/1.0.x/reference/html/).

You can also inject an auto-configured LdapTemplate instance as you would with any other Spring Bean, as shown in the following example:

@Component

public class MyBean {

private final LdapTemplate template; @Autowired

public MyBean(LdapTemplate template) { this.template = template;

}

// ...

}

#### Embedded In-memory LDAP Server

For testing purposes, Spring Boot supports auto-configuration of an in-memory LDAP server from [UnboundID](https://ldap.com/unboundid-ldap-sdk-for-java/). To configure the server, add a dependency to com.unboundid:unboundid-ldapsdk and declare a spring.ldap.embedded.base-dn property, as follows:

spring.ldap.embedded.base-dn=dc=spring,dc=io

It is possible to define multiple base-dn values, however, since distinguished names usually contain commas, they must be defined using the correct notation.

In yaml files, you can use the yaml list notation:



spring.ldap.embedded.base-dn:

* dc=spring,dc=io
* dc=pivotal,dc=io

In properties files, you must include the index as part of the property name:

spring.ldap.embedded.base-dn[0]=dc=spring,dc=io spring.ldap.embedded.base-dn[1]=dc=pivotal,dc=io

By default, the server starts on a random port and triggers the regular LDAP support. There is no need to specify a spring.ldap.urls property.

If there is a schema.ldif file on your classpath, it is used to initialize the server. If you want to load the initialization script from a different resource, you can also use the spring.ldap.embedded.ldif property.

By default, a standard schema is used to validate LDIF files. You can turn off validation altogether by setting the spring.ldap.embedded.validation.enabled property. If you have custom attributes, you

can use spring.ldap.embedded.validation.schema to define your custom attribute types or object classes.

### InfluxDB

[InfluxDB](https://www.influxdata.com/) is an open-source time series database optimized for fast, high-availability storage and retrieval of time series data in fields such as operations monitoring, application metrics, Internet- of-Things sensor data, and real-time analytics.

#### Connecting to InfluxDB

Spring Boot auto-configures an InfluxDB instance, provided the influxdb-java client is on the classpath and the URL of the database is set, as shown in the following example:

spring.influx.url=https://172.0.0.1:8086

If the connection to InfluxDB requires a user and password, you can set the spring.influx.user and

spring.influx.password properties accordingly.

InfluxDB relies on OkHttp. If you need to tune the http client InfluxDB uses behind the scenes, you can register an InfluxDbOkHttpClientBuilderProvider bean.

## Caching

The Spring Framework provides support for transparently adding caching to an application. At its core, the abstraction applies caching to methods, thus reducing the number of executions based on the information available in the cache. The caching logic is applied transparently, without any interference to the invoker. Spring Boot auto-configures the cache infrastructure as long as caching support is enabled via the @EnableCaching annotation.

 Check the [relevant section](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/integration.html#cache) of the Spring Framework reference for more details.

In a nutshell, to add caching to an operation of your service add the relevant annotation to its method, as shown in the following example:

import org.springframework.cache.annotation.Cacheable; import org.springframework.stereotype.Component;

@Component

public class MathService {

@Cacheable("piDecimals")

public int computePiDecimal(int i) {

// ...

}

}

This example demonstrates the use of caching on a potentially costly operation. Before invoking computePiDecimal, the abstraction looks for an entry in the piDecimals cache that matches the i argument. If an entry is found, the content in the cache is immediately returned to the caller, and the method is not invoked. Otherwise, the method is invoked, and the cache is updated before returning the value.

You can also use the standard JSR-107 (JCache) annotations (such as @CacheResult) transparently. However, we strongly advise you to not mix and match the Spring Cache and JCache annotations.

If you do not add any specific cache library, Spring Boot auto-configures a [simple provider](#_bookmark265) that uses concurrent maps in memory. When a cache is required (such as piDecimals in the preceding example), this provider creates it for you. The simple provider is not really recommended for production usage, but it is great for getting started and making sure that you understand the features. When you have made up your mind about the cache provider to use, please make sure to read its documentation to figure out how to configure the caches that your application uses. Nearly all providers require you to explicitly configure every cache that you use in the application. Some offer a way to customize the default caches defined by the spring.cache.cache-names property.

 It is also possible to transparently [update](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/integration.html#cache-annotations-put) or [evict](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/integration.html#cache-annotations-evict) data from the cache.

### Supported Cache Providers

The cache abstraction does not provide an actual store and relies on abstraction materialized by the

org.springframework.cache.Cache and org.springframework.cache.CacheManager interfaces.

If you have not defined a bean of type CacheManager or a CacheResolver named cacheResolver (see

[CachingConfigurer](https://docs.spring.io/spring/docs/5.2.9.RELEASE/javadoc-api/org/springframework/cache/annotation/CachingConfigurer.html)), Spring Boot tries to detect the following providers (in the indicated order):

* + - 1. [Generic](#_bookmark257)
      2. [JCache (JSR-107)](#_bookmark258) (EhCache 3, Hazelcast, Infinispan, and others)
      3. [EhCache 2.x](#_bookmark259)
      4. [Hazelcast](#_bookmark260)
      5. [Infinispan](#_bookmark261)
      6. [Couchbase](#_bookmark262)
      7. [Redis](#_bookmark263)
      8. [Caffeine](#_bookmark264)
      9. [Simple](#_bookmark265)

It is also possible to *force* a particular cache provider by setting the spring.cache.type property. Use this property if you need to [disable caching](#_bookmark266) [altogether](#_bookmark266) in certain environment (such as tests).

Use the spring-boot-starter-cache “Starter” to quickly add basic caching dependencies. The starter brings in spring-context-support. If you add dependencies manually, you must include spring-context-support in order to use the JCache, EhCache 2.x, or Caffeine support.

If the CacheManager is auto-configured by Spring Boot, you can further tune its configuration before it is fully initialized by exposing a bean that implements the CacheManagerCustomizer interface. The following example sets a flag to say that null values should be passed down to the underlying map:

@Bean

public CacheManagerCustomizer<ConcurrentMapCacheManager> cacheManagerCustomizer() { return new CacheManagerCustomizer<ConcurrentMapCacheManager>() {

@Override

public void customize(ConcurrentMapCacheManager cacheManager) { cacheManager.setAllowNullValues(false);

}

};

}

In the preceding example, an auto-configured ConcurrentMapCacheManager is expected. If that is not the case (either you provided your own config or a different cache provider was auto-configured), the customizer is not invoked at all. You can have as many customizers as you want, and you can also order them by using @Order or Ordered.

#### Generic

Generic caching is used if the context defines *at least* one org.springframework.cache.Cache bean. A

CacheManager wrapping all beans of that type is created.

#### JCache (JSR-107)

[JCache](https://jcp.org/en/jsr/detail?id=107) is bootstrapped through the presence of a javax.cache.spi.CachingProvider on the classpath (that is, a JSR-107 compliant caching library exists on the classpath), and the JCacheCacheManager is provided by the spring-boot-starter-cache “Starter”. Various compliant libraries are available, and Spring Boot provides dependency management for Ehcache 3, Hazelcast, and Infinispan. Any other compliant library can be added as well.

It might happen that more than one provider is present, in which case the provider must be explicitly specified. Even if the JSR-107 standard does not enforce a standardized way to define the location of the configuration file, Spring Boot does its best to accommodate setting a cache with implementation details, as shown in the following example:

# Only necessary if more than one provider is present spring.cache.jcache.provider=com.acme.MyCachingProvider spring.cache.jcache.config=classpath:acme.xml

When a cache library offers both a native implementation and JSR-107 support, Spring Boot prefers the JSR-107 support, so that the same features are available if you switch to a different JSR-107 implementation.

Spring Boot has [general support for Hazelcast](#_bookmark297). If a single HazelcastInstance is available, it is automatically reused for the CacheManager as well, unless the spring.cache.jcache.config property is specified.

There are two ways to customize the underlying javax.cache.cacheManager:

* Caches can be created on startup by setting the spring.cache.cache-names property. If a custom

javax.cache.configuration.Configuration bean is defined, it is used to customize them.

* org.springframework.boot.autoconfigure.cache.JCacheManagerCustomizer beans are invoked with the reference of the CacheManager for full customization.

If a standard javax.cache.CacheManager bean is defined, it is wrapped automatically in an org.springframework.cache.CacheManager implementation that the abstraction expects. No further customization is applied to it.

#### EhCache 2.x

[EhCache](https://www.ehcache.org/) 2.x is used if a file named ehcache.xml can be found at the root of the classpath. If EhCache

2.x is found, the EhCacheCacheManager provided by the spring-boot-starter-cache “Starter” is used to bootstrap the cache manager. An alternate configuration file can be provided as well, as shown in the following example:

spring.cache.ehcache.config=classpath:config/another-config.xml

#### Hazelcast

Spring Boot has [general support for Hazelcast](#_bookmark297). If a HazelcastInstance has been auto-configured, it is automatically wrapped in a CacheManager.

#### Infinispan

[Infinispan](https://infinispan.org/) has no default configuration file location, so it must be specified explicitly. Otherwise, the default bootstrap is used.

spring.cache.infinispan.config=infinispan.xml

Caches can be created on startup by setting the spring.cache.cache-names property. If a custom

ConfigurationBuilder bean is defined, it is used to customize the caches.

The support of Infinispan in Spring Boot is restricted to the embedded mode and is quite basic. If you want more options, you should use the official Infinispan Spring Boot starter instead. See [Infinispan’s documentation](https://github.com/infinispan/infinispan-spring-boot) for more details.

#### Couchbase

If Spring Data Couchbase is available and Couchbase is [configured](#_bookmark246), a CouchbaseCacheManager is auto- configured. It is possible to create additional caches on startup by setting the spring.cache.cache- names property and cache defaults can be configured by using spring.cache.couchbase.\* properties. For instance, the following configuration creates cache1 and cache2 caches with an entry *expiration* of 10 minutes:

spring.cache.cache-names=cache1,cache2 spring.cache.couchbase.expiration=10m

If you need more control over the configuration, consider registering a CouchbaseCacheManagerBuilderCustomizer bean. The following example shows a customizer that configures a specific entry expiration for cache1 and cache2:

@Bean

public CouchbaseCacheManagerBuilderCustomizer myCouchbaseCacheManagerBuilderCustomizer() {

return (builder) -> builder

.withCacheConfiguration("cache1",

CouchbaseCacheConfiguration.defaultCacheConfig().entryExpiry(Duration.ofSeconds(10)))

.withCacheConfiguration("cache2", CouchbaseCacheConfiguration.defaultCacheConfig().entryExpiry(Duration.ofMinutes(1)));

}

#### Redis

If [Redis](https://redis.io/) is available and configured, a RedisCacheManager is auto-configured. It is possible to create additional caches on startup by setting the spring.cache.cache-names property and cache defaults can be configured by using spring.cache.redis.\* properties. For instance, the following configuration creates cache1 and cache2 caches with a *time to live* of 10 minutes:

spring.cache.cache-names=cache1,cache2 spring.cache.redis.time-to-live=600000

By default, a key prefix is added so that, if two separate caches use the same key, Redis does not have overlapping keys and cannot return invalid values. We strongly recommend keeping this setting enabled if you create your own RedisCacheManager.

You can take full control of the default configuration by adding a RedisCacheConfiguration @Bean of your own. This can be useful if you’re looking for customizing the default serialization strategy.

If you need more control over the configuration, consider registering a RedisCacheManagerBuilderCustomizer bean. The following example shows a customizer that configures a specific time to live for cache1 and cache2:

@Bean

public RedisCacheManagerBuilderCustomizer myRedisCacheManagerBuilderCustomizer() { return (builder) -> builder

.withCacheConfiguration("cache1",

RedisCacheConfiguration.defaultCacheConfig().entryTtl(Duration.ofSeconds(10)))

.withCacheConfiguration("cache2", RedisCacheConfiguration.defaultCacheConfig().entryTtl(Duration.ofMinutes(1)));

}

#### Caffeine

[Caffeine](https://github.com/ben-manes/caffeine) is a Java 8 rewrite of Guava’s cache that supersedes support for Guava. If Caffeine is present, a CaffeineCacheManager (provided by the spring-boot-starter-cache “Starter”) is auto- configured. Caches can be created on startup by setting the spring.cache.cache-names property and can be customized by one of the following (in the indicated order):

1. A cache spec defined by spring.cache.caffeine.spec
2. A com.github.benmanes.caffeine.cache.CaffeineSpec bean is defined
3. A com.github.benmanes.caffeine.cache.Caffeine bean is defined

For instance, the following configuration creates cache1 and cache2 caches with a maximum size of 500 and a *time to live* of 10 minutes

spring.cache.cache-names=cache1,cache2 spring.cache.caffeine.spec=maximumSize=500,expireAfterAccess=600s

If a com.github.benmanes.caffeine.cache.CacheLoader bean is defined, it is automatically associated to the CaffeineCacheManager. Since the CacheLoader is going to be associated with *all* caches managed by the cache manager, it must be defined as CacheLoader<Object, Object>. The auto-configuration ignores any other generic type.

#### Simple

If none of the other providers can be found, a simple implementation using a ConcurrentHashMap as the cache store is configured. This is the default if no caching library is present in your application. By default, caches are created as needed, but you can restrict the list of available caches by setting the cache-names property. For instance, if you want only cache1 and cache2 caches, set the cache- names property as follows:

spring.cache.cache-names=cache1,cache2

If you do so and your application uses a cache not listed, then it fails at runtime when the cache is needed, but not on startup. This is similar to the way the "real" cache providers behave if you use an undeclared cache.

#### None

When @EnableCaching is present in your configuration, a suitable cache configuration is expected as well. If you need to disable caching altogether in certain environments, force the cache type to none to use a no-op implementation, as shown in the following example:

spring.cache.type=none

## Messaging

The Spring Framework provides extensive support for integrating with messaging systems, from simplified use of the JMS API using JmsTemplate to a complete infrastructure to receive messages asynchronously. Spring AMQP provides a similar feature set for the Advanced Message Queuing Protocol. Spring Boot also provides auto-configuration options for RabbitTemplate and RabbitMQ. Spring WebSocket natively includes support for STOMP messaging, and Spring Boot has support for that through starters and a small amount of auto-configuration. Spring Boot also has support for Apache Kafka.

### JMS

The javax.jms.ConnectionFactory interface provides a standard method of creating a javax.jms.Connection for interacting with a JMS broker. Although Spring needs a ConnectionFactory to work with JMS, you generally need not use it directly yourself and can instead rely on higher level messaging abstractions. (See the [relevant section](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/integration.html#jms) of the Spring Framework reference documentation for details.) Spring Boot also auto-configures the necessary infrastructure to send and receive messages.

#### ActiveMQ Support

When [ActiveMQ](https://activemq.apache.org/) is available on the classpath, Spring Boot can also configure a ConnectionFactory. If the broker is present, an embedded broker is automatically started and configured (provided no broker URL is specified through configuration).

If you use spring-boot-starter-activemq, the necessary dependencies to connect or embed an ActiveMQ instance are provided, as is the Spring infrastructure to integrate with JMS.

ActiveMQ configuration is controlled by external configuration properties in spring.activemq.\*. For example, you might declare the following section in application.properties:

spring.activemq.broker-url=tcp://192.168.1.210:9876 spring.activemq.user=admin spring.activemq.password=secret

By default, a CachingConnectionFactory wraps the native ConnectionFactory with sensible settings that you can control by external configuration properties in spring.jms.\*:

spring.jms.cache.session-cache-size=5

If you’d rather use native pooling, you can do so by adding a dependency to org.messaginghub:pooled-jms and configuring the JmsPoolConnectionFactory accordingly, as shown in the following example:

spring.activemq.pool.enabled=true spring.activemq.pool.max-connections=50

See [ActiveMQProperties](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/activemq/ActiveMQProperties.java) for more of the supported options. You can also register an arbitrary number of beans that implement ActiveMQConnectionFactoryCustomizer for more advanced customizations.

By default, ActiveMQ creates a destination if it does not yet exist so that destinations are resolved against their provided names.

#### Artemis Support

Spring Boot can auto-configure a ConnectionFactory when it detects that [Artemis](https://activemq.apache.org/components/artemis/) is available on the classpath. If the broker is present, an embedded broker is automatically started and configured (unless the mode property has been explicitly set). The supported modes are embedded (to make explicit that an embedded broker is required and that an error should occur if the broker is not available on the classpath) and native (to connect to a broker using the netty transport protocol). When the latter is configured, Spring Boot configures a ConnectionFactory that connects to a broker running on the local machine with the default settings.

If you use spring-boot-starter-artemis, the necessary dependencies to connect to an existing Artemis instance are provided, as well as the Spring infrastructure to integrate with JMS. Adding org.apache.activemq:artemis-jms-server to your application lets you use embedded mode.

Artemis configuration is controlled by external configuration properties in spring.artemis.\*. For example, you might declare the following section in application.properties:

spring.artemis.mode=native spring.artemis.host=192.168.1.210 spring.artemis.port=9876 spring.artemis.user=admin spring.artemis.password=secret

When embedding the broker, you can choose if you want to enable persistence and list the destinations that should be made available. These can be specified as a comma-separated list to create them with the default options, or you can define bean(s) of type org.apache.activemq.artemis.jms.server.config.JMSQueueConfiguration or org.apache.activemq.artemis.jms.server.config.TopicConfiguration, for advanced queue and topic configurations, respectively.

By default, a CachingConnectionFactory wraps the native ConnectionFactory with sensible settings that you can control by external configuration properties in spring.jms.\*:

spring.jms.cache.session-cache-size=5

If you’d rather use native pooling, you can do so by adding a dependency to org.messaginghub:pooled-jms and configuring the JmsPoolConnectionFactory accordingly, as shown in the following example:

spring.artemis.pool.enabled=true spring.artemis.pool.max-connections=50

See [ArtemisProperties](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/artemis/ArtemisProperties.java) for more supported options.

No JNDI lookup is involved, and destinations are resolved against their names, using either the name

attribute in the Artemis configuration or the names provided through configuration.

#### Using a JNDI ConnectionFactory

If you are running your application in an application server, Spring Boot tries to locate a JMS ConnectionFactory by using JNDI. By default, the java:/JmsXA and java:/XAConnectionFactory location are checked. You can use the spring.jms.jndi-name property if you need to specify an alternative location, as shown in the following example:

spring.jms.jndi-name=java:/MyConnectionFactory

#### Sending a Message

Spring’s JmsTemplate is auto-configured, and you can autowire it directly into your own beans, as

shown in the following example:

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.jms.core.JmsTemplate;

import org.springframework.stereotype.Component;

@Component

public class MyBean {

private final JmsTemplate jmsTemplate; @Autowired

public MyBean(JmsTemplate jmsTemplate) { this.jmsTemplate = jmsTemplate;

}

// ...

}

[JmsMessagingTemplate](https://docs.spring.io/spring/docs/5.2.9.RELEASE/javadoc-api/org/springframework/jms/core/JmsMessagingTemplate.html) can be injected in a similar manner. If a DestinationResolver or a MessageConverter bean is defined, it is associated automatically to the auto- configured JmsTemplate.

#### Receiving a Message

When the JMS infrastructure is present, any bean can be annotated with @JmsListener to create a listener endpoint. If no JmsListenerContainerFactory has been defined, a default one is configured automatically. If a DestinationResolver or a MessageConverter beans is defined, it is associated automatically to the default factory.

By default, the default factory is transactional. If you run in an infrastructure where a JtaTransactionManager is present, it is associated to the listener container by default. If not, the sessionTransacted flag is enabled. In that latter scenario, you can associate your local data store transaction to the processing of an incoming message by adding @Transactional on your listener method (or a delegate thereof). This ensures that the incoming message is acknowledged, once the local transaction has completed. This also includes sending response messages that have been performed on the same JMS session.

The following component creates a listener endpoint on the someQueue destination:

@Component

public class MyBean {

@JmsListener(destination = "someQueue") public void processMessage(String content) {

// ...

}

}

 See [the Javadoc of @EnableJms](https://docs.spring.io/spring/docs/5.2.9.RELEASE/javadoc-api/org/springframework/jms/annotation/EnableJms.html) for more details.

If you need to create more JmsListenerContainerFactory instances or if you want to override the default, Spring Boot provides a DefaultJmsListenerContainerFactoryConfigurer that you can use to

initialize a DefaultJmsListenerContainerFactory with the same settings as the one that is auto- configured.

For instance, the following example exposes another factory that uses a specific MessageConverter:

@Configuration(proxyBeanMethods = false) static class JmsConfiguration {

@Bean

public DefaultJmsListenerContainerFactory myFactory( DefaultJmsListenerContainerFactoryConfigurer configurer) {

DefaultJmsListenerContainerFactory factory =

new DefaultJmsListenerContainerFactory(); configurer.configure(factory, connectionFactory()); factory.setMessageConverter(myMessageConverter()); return factory;

}

}

Then you can use the factory in any @JmsListener-annotated method as follows:

@Component

public class MyBean {

@JmsListener(destination = "someQueue", containerFactory="myFactory") public void processMessage(String content) {

// ...

}

}

### AMQP

The Advanced Message Queuing Protocol (AMQP) is a platform-neutral, wire-level protocol for message-oriented middleware. The Spring AMQP project applies core Spring concepts to the development of AMQP-based messaging solutions. Spring Boot offers several conveniences for working with AMQP through RabbitMQ, including the spring-boot-starter-amqp “Starter”.

#### RabbitMQ support

[RabbitMQ](https://www.rabbitmq.com/) is a lightweight, reliable, scalable, and portable message broker based on the AMQP protocol. Spring uses RabbitMQ to communicate through the AMQP protocol.

RabbitMQ configuration is controlled by external configuration properties in spring.rabbitmq.\*. For example, you might declare the following section in application.properties:

spring.rabbitmq.host=localhost spring.rabbitmq.port=5672 spring.rabbitmq.username=admin spring.rabbitmq.password=secret

Alternatively, you could configure the same connection using the addresses attribute:

spring.rabbitmq.addresses=amqp://admin:secret@localhost

When specifying addresses that way, the host and port properties are ignored. If the address uses the amqps protocol, SSL support is enabled automatically.

If a ConnectionNameStrategy bean exists in the context, it will be automatically used to name connections created by the auto-configured ConnectionFactory. See [RabbitProperties](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/amqp/RabbitProperties.java) for more of the supported options.

 See [Understanding AMQP, the protocol used by RabbitMQ](https://spring.io/blog/2010/06/14/understanding-amqp-the-protocol-used-by-rabbitmq/) for more details.

#### Sending a Message

Spring’s AmqpTemplate and AmqpAdmin are auto-configured, and you can autowire them directly into your own beans, as shown in the following example:

import org.springframework.amqp.core.AmqpAdmin; import org.springframework.amqp.core.AmqpTemplate;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Component;

@Component

public class MyBean {

private final AmqpAdmin amqpAdmin; private final AmqpTemplate amqpTemplate;

@Autowired

public MyBean(AmqpAdmin amqpAdmin, AmqpTemplate amqpTemplate) { this.amqpAdmin = amqpAdmin;

this.amqpTemplate = amqpTemplate;

}

// ...

}

[RabbitMessagingTemplate](https://docs.spring.io/spring-amqp/docs/2.2.11.RELEASE/api/org/springframework/amqp/rabbit/core/RabbitMessagingTemplate.html) can be injected in a similar manner. If a MessageConverter

bean is defined, it is associated automatically to the auto-configured AmqpTemplate.

If necessary, any org.springframework.amqp.core.Queue that is defined as a bean is automatically used to declare a corresponding queue on the RabbitMQ instance.

To retry operations, you can enable retries on the AmqpTemplate (for example, in the event that the broker connection is lost):

spring.rabbitmq.template.retry.enabled=true spring.rabbitmq.template.retry.initial-interval=2s

Retries are disabled by default. You can also customize the RetryTemplate programmatically by declaring a RabbitRetryTemplateCustomizer bean.

If you need to create more RabbitTemplate instances or if you want to override the default, Spring Boot provides a RabbitTemplateConfigurer bean that you can use to initialize a RabbitTemplate with the same settings as the factories used by the auto-configuration.

#### Receiving a Message

When the Rabbit infrastructure is present, any bean can be annotated with @RabbitListener to create a listener endpoint. If no RabbitListenerContainerFactory has been defined, a default SimpleRabbitListenerContainerFactory is automatically configured and you can switch to a direct container using the spring.rabbitmq.listener.type property. If a MessageConverter or a MessageRecoverer bean is defined, it is automatically associated with the default factory.

The following sample component creates a listener endpoint on the someQueue queue:

@Component

public class MyBean {

@RabbitListener(queues = "someQueue")

public void processMessage(String content) {

// ...

}

}

 See [the Javadoc of @EnableRabbit](https://docs.spring.io/spring-amqp/docs/2.2.11.RELEASE/api/org/springframework/amqp/rabbit/annotation/EnableRabbit.html) for more details.

If you need to create more RabbitListenerContainerFactory instances or if you want to override the default, Spring Boot provides a SimpleRabbitListenerContainerFactoryConfigurer and a DirectRabbitListenerContainerFactoryConfigurer that you can use to initialize a SimpleRabbitListenerContainerFactory and a DirectRabbitListenerContainerFactory with the same settings as the factories used by the auto-configuration.

It does not matter which container type you chose. Those two beans are exposed by the auto-configuration.

For instance, the following configuration class exposes another factory that uses a specific

MessageConverter:

@Configuration(proxyBeanMethods = false) static class RabbitConfiguration {

@Bean

public SimpleRabbitListenerContainerFactory myFactory( SimpleRabbitListenerContainerFactoryConfigurer configurer) {

SimpleRabbitListenerContainerFactory factory =

new SimpleRabbitListenerContainerFactory(); configurer.configure(factory, connectionFactory); factory.setMessageConverter(myMessageConverter()); return factory;

}

}

Then you can use the factory in any @RabbitListener-annotated method, as follows:

@Component

public class MyBean {

@RabbitListener(queues = "someQueue", containerFactory="myFactory") public void processMessage(String content) {

// ...

}

}

You can enable retries to handle situations where your listener throws an exception. By default, RejectAndDontRequeueRecoverer is used, but you can define a MessageRecoverer of your own. When retries are exhausted, the message is rejected and either dropped or routed to a dead-letter exchange if the broker is configured to do so. By default, retries are disabled. You can also customize the RetryTemplate programmatically by declaring a RabbitRetryTemplateCustomizer bean.

By default, if retries are disabled and the listener throws an exception, the delivery is retried indefinitely. You can modify this behavior in two ways: Set the defaultRequeueRejected property to false so that zero re-deliveries are attempted or throw an AmqpRejectAndDontRequeueException to signal the message should be rejected. The latter is the mechanism used when retries are enabled and the maximum number of delivery attempts is reached.

### Apache Kafka Support

[Apache Kafka](https://kafka.apache.org/) is supported by providing auto-configuration of the spring-kafka project.

Kafka configuration is controlled by external configuration properties in spring.kafka.\*. For example, you might declare the following section in application.properties:

spring.kafka.bootstrap-servers=localhost:9092 spring.kafka.consumer.group-id=myGroup

To create a topic on startup, add a bean of type NewTopic. If the topic already exists, the bean is ignored.

See [KafkaProperties](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/kafka/KafkaProperties.java) for more supported options.

#### Sending a Message

Spring’s KafkaTemplate is auto-configured, and you can autowire it directly in your own beans, as shown in the following example:

@Component

public class MyBean {

private final KafkaTemplate kafkaTemplate; @Autowired

public MyBean(KafkaTemplate kafkaTemplate) { this.kafkaTemplate = kafkaTemplate;

}

// ...

}

If the property spring.kafka.producer.transaction-id-prefix is defined, a KafkaTransactionManager is automatically configured. Also, if a RecordMessageConverter bean is defined, it is automatically associated to the auto- configured KafkaTemplate.

#### Receiving a Message

When the Apache Kafka infrastructure is present, any bean can be annotated with @KafkaListener to create a listener endpoint. If no KafkaListenerContainerFactory has been defined, a default one is automatically configured with keys defined in spring.kafka.listener.\*.

The following component creates a listener endpoint on the someTopic topic:

@Component

public class MyBean {

@KafkaListener(topics = "someTopic")

public void processMessage(String content) {

// ...

}

}

If a KafkaTransactionManager bean is defined, it is automatically associated to the container factory. Similarly, if a ErrorHandler, AfterRollbackProcessor or ConsumerAwareRebalanceListener bean is defined, it is automatically associated to the default factory.

Depending on the listener type, a RecordMessageConverter or BatchMessageConverter bean is associated to the default factory. If only a RecordMessageConverter bean is present for a batch listener, it is wrapped in a BatchMessageConverter.

A custom ChainedKafkaTransactionManager must be marked @Primary as it usually references the auto-configured KafkaTransactionManager bean.

#### Kafka Streams

Spring for Apache Kafka provides a factory bean to create a StreamsBuilder object and manage the lifecycle of its streams. Spring Boot auto-configures the required KafkaStreamsConfiguration bean as long as kafka-streams is on the classpath and Kafka Streams is enabled via the @EnableKafkaStreams annotation.

Enabling Kafka Streams means that the application id and bootstrap servers must be set. The former can be configured using spring.kafka.streams.application-id, defaulting to spring.application.name if not set. The latter can be set globally or specifically overridden only for streams.

Several additional properties are available using dedicated properties; other arbitrary Kafka properties can be set using the spring.kafka.streams.properties namespace. See also [Additional](#_bookmark282) [Kafka Properties](#_bookmark282) for more information.

To use the factory bean, wire StreamsBuilder into your @Bean as shown in the following example:

@Configuration(proxyBeanMethods = false) @EnableKafkaStreams

public static class KafkaStreamsExampleConfiguration {

@Bean

public KStream<Integer, String> kStream(StreamsBuilder streamsBuilder) { KStream<Integer, String> stream = streamsBuilder.stream("ks1In"); stream.map((k, v) -> new KeyValue<>(k, v.toUpperCase())).to("ks1Out",

Produced.with(Serdes.Integer(), new JsonSerde<>())); return stream;

}

}

By default, the streams managed by the StreamBuilder object it creates are started automatically. You can customize this behaviour using the spring.kafka.streams.auto-startup property.

#### Additional Kafka Properties

The properties supported by auto configuration are shown in [Common Application properties](#_bookmark656). Note that, for the most part, these properties (hyphenated or camelCase) map directly to the Apache Kafka dotted properties. Refer to the Apache Kafka documentation for details.

The first few of these properties apply to all components (producers, consumers, admins, and streams) but can be specified at the component level if you wish to use different values. Apache Kafka designates properties with an importance of HIGH, MEDIUM, or LOW. Spring Boot auto- configuration supports all HIGH importance properties, some selected MEDIUM and LOW properties, and any properties that do not have a default value.

Only a subset of the properties supported by Kafka are available directly through the

KafkaProperties class. If you wish to configure the producer or consumer with additional properties

that are not directly supported, use the following properties:

spring.kafka.properties.prop.one=first spring.kafka.admin.properties.prop.two=second spring.kafka.consumer.properties.prop.three=third spring.kafka.producer.properties.prop.four=fourth spring.kafka.streams.properties.prop.five=fifth

This sets the common prop.one Kafka property to first (applies to producers, consumers and admins), the prop.two admin property to second, the prop.three consumer property to third, the prop.four producer property to fourth and the prop.five streams property to fifth.

You can also configure the Spring Kafka JsonDeserializer as follows:

spring.kafka.consumer.value- deserializer=org.springframework.kafka.support.serializer.JsonDeserializer spring.kafka.consumer.properties.spring.json.value.default.type=com.example.Invoice spring.kafka.consumer.properties.spring.json.trusted.packages=com.example,org.acme

Similarly, you can disable the JsonSerializer default behavior of sending type information in headers:

spring.kafka.producer.value- serializer=org.springframework.kafka.support.serializer.JsonSerializer spring.kafka.producer.properties.spring.json.add.type.headers=false

Properties set in this way override any configuration item that Spring Boot explicitly supports.

#### Testing with Embedded Kafka

Spring for Apache Kafka provides a convenient way to test projects with an embedded Apache Kafka broker. To use this feature, annotate a test class with @EmbeddedKafka from the spring-kafka- test module. For more information, please see the Spring for Apache Kafka [reference manual](https://docs.spring.io/spring-kafka/docs/2.5.6.RELEASE/reference/html/#embedded-kafka-annotation).

To make Spring Boot auto-configuration work with the aforementioned embedded Apache Kafka broker, you need to remap a system property for embedded broker addresses (populated by the EmbeddedKafkaBroker) into the Spring Boot configuration property for Apache Kafka. There are several ways to do that:

* Provide a system property to map embedded broker addresses into spring.kafka.bootstrap- servers in the test class:

static {

System.setProperty(EmbeddedKafkaBroker.BROKER\_LIST\_PROPERTY, "spring.kafka.bootstrap-servers");

}

* Configure a property name on the @EmbeddedKafka annotation:

@EmbeddedKafka(topics = "someTopic",

bootstrapServersProperty = "spring.kafka.bootstrap-servers")

* Use a placeholder in configuration properties:

spring.kafka.bootstrap-servers=${spring.embedded.kafka.brokers}

## Calling REST Services with RestTemplate

If you need to call remote REST services from your application, you can use the Spring Framework’s [RestTemplate](https://docs.spring.io/spring/docs/5.2.9.RELEASE/javadoc-api/org/springframework/web/client/RestTemplate.html) class. Since RestTemplate instances often need to be customized before being used, Spring Boot does not provide any single auto-configured RestTemplate bean. It does, however, auto- configure a RestTemplateBuilder, which can be used to create RestTemplate instances when needed. The auto-configured RestTemplateBuilder ensures that sensible HttpMessageConverters are applied to RestTemplate instances.

The following code shows a typical example:

@Service

public class MyService {

private final RestTemplate restTemplate;

public MyService(RestTemplateBuilder restTemplateBuilder) { this.restTemplate = restTemplateBuilder.build();

}

public Details someRestCall(String name) {

return this.restTemplate.getForObject("/{name}/details", Details.class, name);

}

}

RestTemplateBuilder includes a number of useful methods that can be used to quickly configure a RestTemplate. For example, to add BASIC auth support, you can use builder.basicAuthentication("user", "password").build().

### RestTemplate Customization

There are three main approaches to RestTemplate customization, depending on how broadly you want the customizations to apply.

To make the scope of any customizations as narrow as possible, inject the auto-configured RestTemplateBuilder and then call its methods as required. Each method call returns a new RestTemplateBuilder instance, so the customizations only affect this use of the builder.

To make an application-wide, additive customization, use a RestTemplateCustomizer bean. All such beans are automatically registered with the auto-configured RestTemplateBuilder and are applied to any templates that are built with it.

The following example shows a customizer that configures the use of a proxy for all hosts except

192.168.0.5:

static class ProxyCustomizer implements RestTemplateCustomizer {

@Override

public void customize(RestTemplate restTemplate) { HttpHost proxy = new HttpHost("proxy.example.com");

HttpClient httpClient = HttpClientBuilder.create().setRoutePlanner(new DefaultProxyRoutePlanner(proxy) {

@Override

public HttpHost determineProxy(HttpHost target, HttpRequest request, HttpContext context)

throws HttpException {

if (target.getHostName().equals("192.168.0.5")) { return null;

}

return super.determineProxy(target, request, context);

}

}).build(); restTemplate.setRequestFactory(new

HttpComponentsClientHttpRequestFactory(httpClient));

}

}

Finally, the most extreme (and rarely used) option is to create your own RestTemplateBuilder bean. Doing so switches off the auto-configuration of a RestTemplateBuilder and prevents any RestTemplateCustomizer beans from being used.

## Calling REST Services with WebClient

If you have Spring WebFlux on your classpath, you can also choose to use WebClient to call remote REST services. Compared to RestTemplate, this client has a more functional feel and is fully reactive.

You can learn more about the WebClient in the dedicated [section in the Spring Framework docs](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web-reactive.html#webflux-client).

Spring Boot creates and pre-configures a WebClient.Builder for you; it is strongly advised to inject it in your components and use it to create WebClient instances. Spring Boot is configuring that builder to share HTTP resources, reflect codecs setup in the same fashion as the server ones (see [WebFlux](#_bookmark164) [HTTP codecs auto-configuration](#_bookmark164)), and more.

The following code shows a typical example:

@Service

public class MyService {

private final WebClient webClient;

public MyService(WebClient.Builder webClientBuilder) {

this.webClient = webClientBuilder.baseUrl("https://example.org").build();

}

public Mono<Details> someRestCall(String name) {

return this.webClient.get().uri("/{name}/details", name)

.retrieve().bodyToMono(Details.class);

}

}

### WebClient Runtime

Spring Boot will auto-detect which ClientHttpConnector to use to drive WebClient, depending on the libraries available on the application classpath. For now, Reactor Netty and Jetty RS client are supported.

The spring-boot-starter-webflux starter depends on io.projectreactor.netty:reactor-netty by default, which brings both server and client implementations. If you choose to use Jetty as a reactive server instead, you should add a dependency on the Jetty Reactive HTTP client library, org.eclipse.jetty:jetty-reactive-httpclient. Using the same technology for server and client has it advantages, as it will automatically share HTTP resources between client and server.

Developers can override the resource configuration for Jetty and Reactor Netty by providing a custom ReactorResourceFactory or JettyResourceFactory bean - this will be applied to both clients and servers.

If you wish to override that choice for the client, you can define your own ClientHttpConnector bean and have full control over the client configuration.

You can learn more about the [WebClient configuration options in the Spring Framework reference](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web-reactive.html#webflux-client-builder) [documentation](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web-reactive.html#webflux-client-builder).

### WebClient Customization

There are three main approaches to WebClient customization, depending on how broadly you want the customizations to apply.

To make the scope of any customizations as narrow as possible, inject the auto-configured WebClient.Builder and then call its methods as required. WebClient.Builder instances are stateful: Any change on the builder is reflected in all clients subsequently created with it. If you want to create several clients with the same builder, you can also consider cloning the builder with WebClient.Builder other = builder.clone();.

To make an application-wide, additive customization to all WebClient.Builder instances, you can declare WebClientCustomizer beans and change the WebClient.Builder locally at the point of injection.

Finally, you can fall back to the original API and use WebClient.create(). In that case, no auto- configuration or WebClientCustomizer is applied.

## Validation

The method validation feature supported by Bean Validation 1.1 is automatically enabled as long as a JSR-303 implementation (such as Hibernate validator) is on the classpath. This lets bean methods be annotated with javax.validation constraints on their parameters and/or on their return value. Target classes with such annotated methods need to be annotated with the @Validated annotation at the type level for their methods to be searched for inline constraint annotations.

For instance, the following service triggers the validation of the first argument, making sure its size is between 8 and 10:

@Service @Validated

public class MyBean {

public Archive findByCodeAndAuthor(@Size(min = 8, max = 10) String code, Author author) {

...

}

}

## Sending Email

The Spring Framework provides an abstraction for sending email by using the JavaMailSender

interface, and Spring Boot provides auto-configuration for it as well as a starter module.

See the [reference documentation](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/integration.html#mail) for a detailed explanation of how you can use

JavaMailSender.

If spring.mail.host and the relevant libraries (as defined by spring-boot-starter-mail) are available, a default JavaMailSender is created if none exists. The sender can be further customized by configuration items from the spring.mail namespace. See [MailProperties](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mail/MailProperties.java) for more details.

In particular, certain default timeout values are infinite, and you may want to change that to avoid having a thread blocked by an unresponsive mail server, as shown in the following example:

spring.mail.properties.mail.smtp.connectiontimeout=5000 spring.mail.properties.mail.smtp.timeout=3000 spring.mail.properties.mail.smtp.writetimeout=5000

It is also possible to configure a JavaMailSender with an existing Session from JNDI:

spring.mail.jndi-name=mail/Session

When a jndi-name is set, it takes precedence over all other Session-related settings.

## Distributed Transactions with JTA

Spring Boot supports distributed JTA transactions across multiple XA resources by using an [Atomikos](https://www.atomikos.com/) embedded transaction manager. Deprecated support for using a [Bitronix](https://github.com/bitronix/btm) embedded transaction manager is also provided but it will be removed in a future release. JTA transactions are also supported when deploying to a suitable Java EE Application Server.

When a JTA environment is detected, Spring’s JtaTransactionManager is used to manage transactions. Auto-configured JMS, DataSource, and JPA beans are upgraded to support XA transactions. You can use standard Spring idioms, such as @Transactional, to participate in a distributed transaction. If you are within a JTA environment and still want to use local transactions, you can set the spring.jta.enabled property to false to disable the JTA auto-configuration.

### Using an Atomikos Transaction Manager

[Atomikos](https://www.atomikos.com/) is a popular open source transaction manager which can be embedded into your Spring Boot application. You can use the spring-boot-starter-jta-atomikos starter to pull in the appropriate Atomikos libraries. Spring Boot auto-configures Atomikos and ensures that appropriate depends-on settings are applied to your Spring beans for correct startup and shutdown ordering.

By default, Atomikos transaction logs are written to a transaction-logs directory in your application’s home directory (the directory in which your application jar file resides). You can customize the location of this directory by setting a spring.jta.log-dir property in your application.properties file. Properties starting with spring.jta.atomikos.properties can also be used to customize the Atomikos UserTransactionServiceImp. See the [AtomikosProperties Javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/jta/atomikos/AtomikosProperties.html) for complete details.

To ensure that multiple transaction managers can safely coordinate the same resource managers, each Atomikos instance must be configured with a unique ID. By default, this ID is the IP address of the machine on which Atomikos is running. To ensure uniqueness in production, you should configure the spring.jta.transaction-manager-id property with a different value for each instance of your application.

### Using a Bitronix Transaction Manager

As of Spring Boot 2.3, support for Bitronix has been deprecated and will be removed in a future release.

You can use the spring-boot-starter-jta-bitronix starter to add the appropriate Bitronix dependencies to your project. As with Atomikos, Spring Boot automatically configures Bitronix and post-processes your beans to ensure that startup and shutdown ordering is correct.

By default, Bitronix transaction log files (part1.btm and part2.btm) are written to a transaction-logs directory in your application home directory. You can customize the location of this directory by setting the spring.jta.log-dir property. Properties starting with spring.jta.bitronix.properties are also bound to the bitronix.tm.Configuration bean, allowing for complete customization. See the [Bitronix documentation](https://github.com/bitronix/btm/wiki/Transaction-manager-configuration) for details.

To ensure that multiple transaction managers can safely coordinate the same resource managers, each Bitronix instance must be configured with a unique ID. By default, this ID is the IP address of the machine on which Bitronix is running. To ensure uniqueness in production, you should configure the spring.jta.transaction-manager-id property with a different value for each instance of your application.

### Using a Java EE Managed Transaction Manager

If you package your Spring Boot application as a war or ear file and deploy it to a Java EE application server, you can use your application server’s built-in transaction manager. Spring Boot tries to auto-configure a transaction manager by looking at common JNDI locations (java:comp/UserTransaction, java:comp/TransactionManager, and so on). If you use a transaction service provided by your application server, you generally also want to ensure that all resources are managed by the server and exposed over JNDI. Spring Boot tries to auto-configure JMS by looking for a ConnectionFactory at the JNDI path (java:/JmsXA or java:/XAConnectionFactory), and you can use the [spring.datasource.jndi-name property](#_bookmark202) to configure your DataSource.

### Mixing XA and Non-XA JMS Connections

When using JTA, the primary JMS ConnectionFactory bean is XA-aware and participates in distributed transactions. In some situations, you might want to process certain JMS messages by using a non-XA ConnectionFactory. For example, your JMS processing logic might take longer than the XA timeout.

If you want to use a non-XA ConnectionFactory, you can inject the nonXaJmsConnectionFactory bean rather than the @Primary jmsConnectionFactory bean. For consistency, the jmsConnectionFactory bean is also provided by using the bean alias xaJmsConnectionFactory.

The following example shows how to inject ConnectionFactory instances:

// Inject the primary (XA aware) ConnectionFactory @Autowired

private ConnectionFactory defaultConnectionFactory;

// Inject the XA aware ConnectionFactory (uses the alias and injects the same as above)

@Autowired @Qualifier("xaJmsConnectionFactory")

private ConnectionFactory xaConnectionFactory;

// Inject the non-XA aware ConnectionFactory @Autowired @Qualifier("nonXaJmsConnectionFactory")

private ConnectionFactory nonXaConnectionFactory;

### Supporting an Alternative Embedded Transaction Manager

The [XAConnectionFactoryWrapper](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/jms/XAConnectionFactoryWrapper.java) and [XADataSourceWrapper](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/jdbc/XADataSourceWrapper.java) interfaces can be used to support alternative embedded transaction managers. The interfaces are responsible for wrapping XAConnectionFactory and XADataSource beans and exposing them as regular ConnectionFactory and DataSource beans, which transparently enroll in the distributed transaction. DataSource and JMS auto-configuration use JTA variants, provided you have a JtaTransactionManager bean and appropriate XA wrapper beans registered within your ApplicationContext.

The [AtomikosXAConnectionFactoryWrapper](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/jta/atomikos/AtomikosXAConnectionFactoryWrapper.java) and [AtomikosXADataSourceWrapper](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/jta/atomikos/AtomikosXADataSourceWrapper.java) provide good examples of how to write XA wrappers.

## Hazelcast

If [Hazelcast](https://hazelcast.com/) is on the classpath and a suitable configuration is found, Spring Boot auto-configures a

HazelcastInstance that you can inject in your application.

If you define a com.hazelcast.config.Config bean, Spring Boot uses that. If your configuration defines an instance name, Spring Boot tries to locate an existing instance rather than creating a new one.

You could also specify the Hazelcast configuration file to use through configuration, as shown in the following example:

spring.hazelcast.config=classpath:config/my-hazelcast.xml

Otherwise, Spring Boot tries to find the Hazelcast configuration from the default locations: hazelcast.xml in the working directory or at the root of the classpath, or a .yaml counterpart in the same locations. We also check if the hazelcast.config system property is set. See the [Hazelcast](https://docs.hazelcast.org/docs/latest/manual/html-single/) [documentation](https://docs.hazelcast.org/docs/latest/manual/html-single/) for more details.

If hazelcast-client is present on the classpath, Spring Boot first attempts to create a client by checking the following configuration options:

* The presence of a com.hazelcast.client.config.ClientConfig bean.
* A configuration file defined by the spring.hazelcast.config property.
* The presence of the hazelcast.client.config system property.
* A hazelcast-client.xml in the working directory or at the root of the classpath.
* A hazelcast-client.yaml in the working directory or at the root of the classpath.

Spring Boot also has [explicit caching support for Hazelcast](#_bookmark260). If caching is enabled, the HazelcastInstance is automatically wrapped in a CacheManager implementation.

## Quartz Scheduler

Spring Boot offers several conveniences for working with the [Quartz scheduler](https://www.quartz-scheduler.org/), including the spring-boot-starter-quartz “Starter”. If Quartz is available, a Scheduler is auto-configured (through the SchedulerFactoryBean abstraction).

Beans of the following types are automatically picked up and associated with the Scheduler:

* JobDetail: defines a particular Job. JobDetail instances can be built with the JobBuilder API.
* Calendar.
* Trigger: defines when a particular job is triggered.

By default, an in-memory JobStore is used. However, it is possible to configure a JDBC-based store if a DataSource bean is available in your application and if the spring.quartz.job-store-type property is configured accordingly, as shown in the following example:

spring.quartz.job-store-type=jdbc

When the JDBC store is used, the schema can be initialized on startup, as shown in the following example:

spring.quartz.jdbc.initialize-schema=always

By default, the database is detected and initialized by using the standard scripts provided with the Quartz library. These scripts drop existing tables, deleting all triggers on every restart. It is also possible to provide a custom script by setting the spring.quartz.jdbc.schema property.

To have Quartz use a DataSource other than the application’s main DataSource, declare a DataSource bean, annotating its @Bean method with @QuartzDataSource. Doing so ensures that the Quartz-specific DataSource is used by both the SchedulerFactoryBean and for schema initialization.

By default, jobs created by configuration will not overwrite already registered jobs that have been read from a persistent job store. To enable overwriting existing job definitions set the spring.quartz.overwrite-existing-jobs property.

Quartz Scheduler configuration can be customized using spring.quartz properties and SchedulerFactoryBeanCustomizer beans, which allow programmatic SchedulerFactoryBean customization. Advanced Quartz configuration properties can be customized using spring.quartz.properties.\*.

In particular, an Executor bean is not associated with the scheduler as Quartz offers a way to configure the scheduler via spring.quartz.properties. If you need to customize the task executor, consider implementing SchedulerFactoryBeanCustomizer.

Jobs can define setters to inject data map properties. Regular beans can also be injected in a similar manner, as shown in the following example:

public class SampleJob extends QuartzJobBean { private MyService myService;

private String name;

// Inject "MyService" bean

public void setMyService(MyService myService) { ... }

// Inject the "name" job data property public void setName(String name) { ... }

@Override

protected void executeInternal(JobExecutionContext context) throws JobExecutionException {

...

}

}

## Task Execution and Scheduling

In the absence of an Executor bean in the context, Spring Boot auto-configures a ThreadPoolTaskExecutor with sensible defaults that can be automatically associated to asynchronous task execution (@EnableAsync) and Spring MVC asynchronous request processing.

If you have defined a custom Executor in the context, regular task execution (i.e. @EnableAsync) will use it transparently but the Spring MVC support will not be configured as it requires an AsyncTaskExecutor implementation (named applicationTaskExecutor). Depending on your target arrangement, you could change your Executor into a ThreadPoolTaskExecutor or define both a ThreadPoolTaskExecutor and an AsyncConfigurer wrapping your custom Executor.

The auto-configured TaskExecutorBuilder allows you to easily create instances that reproduce what the auto-configuration does by default.

The thread pool uses 8 core threads that can grow and shrink according to the load. Those default settings can be fine-tuned using the spring.task.execution namespace as shown in the following example:

spring.task.execution.pool.max-size=16 spring.task.execution.pool.queue-capacity=100 spring.task.execution.pool.keep-alive=10s

This changes the thread pool to use a bounded queue so that when the queue is full (100 tasks), the thread pool increases to maximum 16 threads. Shrinking of the pool is more aggressive as threads are reclaimed when they are idle for 10 seconds (rather than 60 seconds by default).

A ThreadPoolTaskScheduler can also be auto-configured if need to be associated to scheduled task execution (@EnableScheduling). The thread pool uses one thread by default and those settings can be fine-tuned using the spring.task.scheduling namespace.

Both a TaskExecutorBuilder bean and a TaskSchedulerBuilder bean are made available in the context if a custom executor or scheduler needs to be created.

## Spring Integration

Spring Boot offers several conveniences for working with [Spring Integration](https://spring.io/projects/spring-integration), including the spring- boot-starter-integration “Starter”. Spring Integration provides abstractions over messaging and also other transports such as HTTP, TCP, and others. If Spring Integration is available on your classpath, it is initialized through the @EnableIntegration annotation.

Spring Boot also configures some features that are triggered by the presence of additional Spring Integration modules. If spring-integration-jmx is also on the classpath, message processing statistics are published over JMX. If spring-integration-jdbc is available, the default database schema can be created on startup, as shown in the following line:

spring.integration.jdbc.initialize-schema=always

If spring-integration-rsocket is available, developers can configure an RSocket server using "spring.rsocket.server.\*" properties and let it use IntegrationRSocketEndpoint or RSocketOutboundGateway components to handle incoming RSocket messages. This infrastructure can

handle Spring Integration RSocket channel adapters and @MessageMapping handlers (given

"spring.integration.rsocket.server.message-mapping-enabled" is configured).

Spring Boot can also auto-configure an ClientRSocketConnector using configuration properties:

# Connecting to a RSocket server over TCP spring.integration.rsocket.client.host=example.org spring.integration.rsocket.client.port=9898

# Connecting to a RSocket Server over WebSocket spring.integration.rsocket.client.uri=ws://example.org

See the [IntegrationAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/integration/IntegrationAutoConfiguration.java) and [IntegrationProperties](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/integration/IntegrationProperties.java) classes for more details.

By default, if a Micrometer meterRegistry bean is present, Spring Integration metrics will be managed by Micrometer. If you wish to use legacy Spring Integration metrics, add a DefaultMetricsFactory bean to the application context.

## Spring Session

Spring Boot provides [Spring Session](https://spring.io/projects/spring-session) auto-configuration for a wide range of data stores. When building a Servlet web application, the following stores can be auto-configured:

* JDBC
* Redis
* Hazelcast
* MongoDB

When building a reactive web application, the following stores can be auto-configured:

* Redis
* MongoDB

If a single Spring Session module is present on the classpath, Spring Boot uses that store implementation automatically. If you have more than one implementation, you must choose the [StoreType](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/session/StoreType.java) that you wish to use to store the sessions. For instance, to use JDBC as the back-end store, you can configure your application as follows:

spring.session.store-type=jdbc

 You can disable Spring Session by setting the store-type to none.

Each store has specific additional settings. For instance, it is possible to customize the name of the table for the JDBC store, as shown in the following example:

spring.session.jdbc.table-name=SESSIONS

For setting the timeout of the session you can use the spring.session.timeout property. If that property is not set, the auto-configuration falls back to the value of server.servlet.session.timeout.

## Monitoring and Management over JMX

Java Management Extensions (JMX) provide a standard mechanism to monitor and manage applications. Spring Boot exposes the most suitable MBeanServer as a bean with an ID of mbeanServer. Any of your beans that are annotated with Spring JMX annotations (@ManagedResource, @ManagedAttribute, or @ManagedOperation) are exposed to it.

If your platform provides a standard MBeanServer, Spring Boot will use that and default to the VM

MBeanServer if necessary. If all that fails, a new MBeanServer will be created. See the [JmxAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jmx/JmxAutoConfiguration.java) class for more details.

## Testing

Spring Boot provides a number of utilities and annotations to help when testing your application. Test support is provided by two modules: spring-boot-test contains core items, and spring-boot- test-autoconfigure supports auto-configuration for tests.

Most developers use the spring-boot-starter-test “Starter”, which imports both Spring Boot test modules as well as JUnit Jupiter, AssertJ, Hamcrest, and a number of other useful libraries.

The starter also brings the vintage engine so that you can run both JUnit 4 and JUnit 5 tests. If you have migrated your tests to JUnit 5, you should exclude JUnit 4 support, as shown in the following example:



<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</dependency>

### Test Scope Dependencies

The spring-boot-starter-test “Starter” (in the test scope) contains the following provided libraries:

* [JUnit 5](https://junit.org/junit5/) (including the vintage engine for backward compatibility with JUnit 4): The de-facto standard for unit testing Java applications.
* [Spring Test](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/testing.html#integration-testing) & Spring Boot Test: Utilities and integration test support for Spring Boot applications.
* [AssertJ](https://assertj.github.io/doc/): A fluent assertion library.
* [Hamcrest](https://github.com/hamcrest/JavaHamcrest): A library of matcher objects (also known as constraints or predicates).
* [Mockito](https://site.mockito.org/): A Java mocking framework.
* [JSONassert](https://github.com/skyscreamer/JSONassert): An assertion library for JSON.
* [JsonPath](https://github.com/jayway/JsonPath): XPath for JSON.

We generally find these common libraries to be useful when writing tests. If these libraries do not suit your needs, you can add additional test dependencies of your own.

### Testing Spring Applications

One of the major advantages of dependency injection is that it should make your code easier to unit test. You can instantiate objects by using the new operator without even involving Spring. You can also use *mock objects* instead of real dependencies.

Often, you need to move beyond unit testing and start integration testing (with a Spring ApplicationContext). It is useful to be able to perform integration testing without requiring deployment of your application or needing to connect to other infrastructure.

The Spring Framework includes a dedicated test module for such integration testing. You can declare a dependency directly to org.springframework:spring-test or use the spring-boot-starter- test “Starter” to pull it in transitively.

If you have not used the spring-test module before, you should start by reading the [relevant](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/testing.html#testing) [section](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/testing.html#testing) of the Spring Framework reference documentation.

### Testing Spring Boot Applications

A Spring Boot application is a Spring ApplicationContext, so nothing very special has to be done to test it beyond what you would normally do with a vanilla Spring context.

External properties, logging, and other features of Spring Boot are installed in the context by default only if you use SpringApplication to create it.

Spring Boot provides a @SpringBootTest annotation, which can be used as an alternative to the standard spring-test @ContextConfiguration annotation when you need Spring Boot features. The annotation works by [creating the ApplicationContext used in your tests through](#_bookmark308) [SpringApplication](#_bookmark308). In addition to @SpringBootTest a number of other annotations are also provided for [testing more](#_bookmark316) [specific slices](#_bookmark316) of an application.

If you are using JUnit 4, don’t forget to also add @RunWith(SpringRunner.class) to your test, otherwise the annotations will be ignored. If you are using JUnit 5, there’s no need to add the equivalent @ExtendWith(SpringExtension.class) as @SpringBootTest and the other @…Test annotations are already annotated with it.

By default, @SpringBootTest will not start a server. You can use the webEnvironment attribute of

@SpringBootTest to further refine how your tests run:

* MOCK(Default) : Loads a web ApplicationContext and provides a mock web environment. Embedded servers are not started when using this annotation. If a web environment is not available on your classpath, this mode transparently falls back to creating a regular non-web ApplicationContext. It can be used in conjunction with [@AutoConfigureMockMvc](#_bookmark311) [or](#_bookmark311) [@AutoConfigureWebTestClient](#_bookmark311) for mock-based testing of your web application.
* RANDOM\_PORT: Loads a WebServerApplicationContext and provides a real web environment. Embedded servers are started and listen on a random port.
* DEFINED\_PORT: Loads a WebServerApplicationContext and provides a real web environment. Embedded servers are started and listen on a defined port (from your application.properties) or on the default port of 8080.
* NONE: Loads an ApplicationContext by using SpringApplication but does not provide *any* web environment (mock or otherwise).

If your test is @Transactional, it rolls back the transaction at the end of each test method by default. However, as using this arrangement with either RANDOM\_PORT or DEFINED\_PORT implicitly provides a real servlet environment, the HTTP client and server run in separate threads and, thus, in separate transactions. Any transaction initiated on the server does not roll back in this case.

@SpringBootTest with webEnvironment = WebEnvironment.RANDOM\_PORT will also start the management server on a separate random port if your application uses a different port for the management server.

#### Detecting Web Application Type

If Spring MVC is available, a regular MVC-based application context is configured. If you have only Spring WebFlux, we’ll detect that and configure a WebFlux-based application context instead.

If both are present, Spring MVC takes precedence. If you want to test a reactive web application in this scenario, you must set the spring.main.web-application-type property:

@SpringBootTest(properties = "spring.main.web-application-type=reactive") class MyWebFluxTests { ... }

#### Detecting Test Configuration

If you are familiar with the Spring Test Framework, you may be used to using

@ContextConfiguration(classes=…) in order to specify which Spring @Configuration to load.

Alternatively, you might have often used nested @Configuration classes within your test.

When testing Spring Boot applications, this is often not required. Spring Boot’s @\*Test annotations search for your primary configuration automatically whenever you do not explicitly define one.

The search algorithm works up from the package that contains the test until it finds a class annotated with @SpringBootApplication or @SpringBootConfiguration. As long as you [structured your](#_bookmark48) [code](#_bookmark48) in a sensible way, your main configuration is usually found.

If you use a [test annotation to test a more specific slice of your application](#_bookmark316), you should avoid adding configuration settings that are specific to a particular area on the [main method’s application class](#_bookmark332).

The underlying component scan configuration of @SpringBootApplication defines exclude filters that are used to make sure slicing works as expected. If you are using an explicit @ComponentScan directive on your @SpringBootApplication

-annotated class, be aware that those filters will be disabled. If you are using slicing, you should define them again.

If you want to customize the primary configuration, you can use a nested @TestConfiguration class. Unlike a nested @Configuration class, which would be used instead of your application’s primary configuration, a nested @TestConfiguration class is used in addition to your application’s primary configuration.

Spring’s test framework caches application contexts between tests. Therefore, as long as your tests share the same configuration (no matter how it is discovered), the potentially time-consuming process of loading the context happens only once.

#### Excluding Test Configuration

If your application uses component scanning (for example, if you use @SpringBootApplication or @ComponentScan), you may find top-level configuration classes that you created only for specific tests accidentally get picked up everywhere.

As we [have seen earlier](#_bookmark308), @TestConfiguration can be used on an inner class of a test to customize the primary configuration. When placed on a top-level class, @TestConfiguration indicates that classes in src/test/java should not be picked up by scanning. You can then import that class explicitly where it is required, as shown in the following example:

@SpringBootTest @Import(MyTestsConfiguration.class) class MyTests {

@Test

void exampleTest() {

...

}

}

If you directly use @ComponentScan (that is, not through @SpringBootApplication) you need to register the TypeExcludeFilter with it. See [the Javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/context/TypeExcludeFilter.html) for details.

#### Using Application Arguments

If your application expects [arguments](#_bookmark97), you can have @SpringBootTest inject them using the args

attribute.

@SpringBootTest(args = "--app.test=one") class ApplicationArgumentsExampleTests {

@Test

void applicationArgumentsPopulated(@Autowired ApplicationArguments args) { assertThat(args.getOptionNames()).containsOnly("app.test"); assertThat(args.getOptionValues("app.test")).containsOnly("one");

}

}

#### Testing with a mock environment

By default, @SpringBootTest does not start the server. If you have web endpoints that you want to test against this mock environment, you can additionally configure [MockMvc](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/testing.html#spring-mvc-test-framework) as shown in the following example:

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc; import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.web.servlet.MockMvc;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get; import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@SpringBootTest @AutoConfigureMockMvc

class MockMvcExampleTests {

@Test

void exampleTest(@Autowired MockMvc mvc) throws Exception {

mvc.perform(get("/")).andExpect(status().isOk()).andExpect(content().string("Hello World"));

}

}

If you want to focus only on the web layer and not start a complete

ApplicationContext, consider [using @WebMvcTest instead](#_bookmark318).

Alternatively, you can configure a [WebTestClient](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/testing.html#webtestclient-tests) as shown in the following example:

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired; import

org.springframework.boot.test.autoconfigure.web.reactive.AutoConfigureWebTestClient; import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.web.reactive.server.WebTestClient;

@SpringBootTest @AutoConfigureWebTestClient

class MockWebTestClientExampleTests {

@Test

void exampleTest(@Autowired WebTestClient webClient) {

webClient.get().uri("/").exchange().expectStatus().isOk().expectBody(String.class).isE qualTo("Hello World");

}

}

Testing within a mocked environment is usually faster than running with a full Servlet container. However, since mocking occurs at the Spring MVC layer, code that relies on lower-level Servlet container behavior cannot be directly tested with MockMvc.

 For example, Spring Boot’s error handling is based on the “error page” support provided by the Servlet container. This means that, whilst you can test your MVC layer throws and handles exceptions as expected, you cannot directly test that a specific [custom error page](#_bookmark159) is rendered. If you need to test these lower-level concerns, you can start a fully running server as described in the next section.

#### Testing with a running server

If you need to start a full running server, we recommend that you use random ports. If you use @SpringBootTest(webEnvironment=WebEnvironment.RANDOM\_PORT), an available port is picked at random each time your test runs.

The @LocalServerPort annotation can be used to [inject the actual port used](#_bookmark548) into your test. For convenience, tests that need to make REST calls to the started server can additionally @Autowire a [WebTestClient](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/testing.html#webtestclient-tests), which resolves relative links to the running server and comes with a dedicated API for verifying responses, as shown in the following example:

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.boot.test.context.SpringBootTest.WebEnvironment; import org.springframework.test.web.reactive.server.WebTestClient;

@SpringBootTest(webEnvironment = WebEnvironment.RANDOM\_PORT) class RandomPortWebTestClientExampleTests {

@Test

void exampleTest(@Autowired WebTestClient webClient) {

webClient.get().uri("/").exchange().expectStatus().isOk().expectBody(String.class).isE qualTo("Hello World");

}

}

This setup requires spring-webflux on the classpath. If you can’t or won’t add webflux, Spring Boot also provides a TestRestTemplate facility:

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.boot.test.context.SpringBootTest.WebEnvironment; import org.springframework.boot.test.web.client.TestRestTemplate;

import static org.assertj.core.api.Assertions.assertThat;

@SpringBootTest(webEnvironment = WebEnvironment.RANDOM\_PORT) class RandomPortTestRestTemplateExampleTests {

@Test

void exampleTest(@Autowired TestRestTemplate restTemplate) { String body = restTemplate.getForObject("/", String.class); assertThat(body).isEqualTo("Hello World");

}

}

#### Customizing WebTestClient

To customize the WebTestClient bean, configure a WebTestClientBuilderCustomizer bean. Any such beans are called with the WebTestClient.Builder that is used to create the WebTestClient.

#### Using JMX

As the test context framework caches context, JMX is disabled by default to prevent identical components to register on the same domain. If such test needs access to an MBeanServer, consider marking it dirty as well:

@ExtendWith(SpringExtension.class) @SpringBootTest(properties = "spring.jmx.enabled=true") @DirtiesContext

class SampleJmxTests {

@Autowired

private MBeanServer mBeanServer;

@Test

void exampleTest() {

// ...

}

}

#### Mocking and Spying Beans

When running tests, it is sometimes necessary to mock certain components within your application context. For example, you may have a facade over some remote service that is unavailable during development. Mocking can also be useful when you want to simulate failures that might be hard to trigger in a real environment.

Spring Boot includes a @MockBean annotation that can be used to define a Mockito mock for a bean inside your ApplicationContext. You can use the annotation to add new beans or replace a single existing bean definition. The annotation can be used directly on test classes, on fields within your test, or on @Configuration classes and fields. When used on a field, the instance of the created mock is also injected. Mock beans are automatically reset after each test method.

If your test uses one of Spring Boot’s test annotations (such as @SpringBootTest), this feature is automatically enabled. To use this feature with a different arrangement, a listener must be explicitly added, as shown in the following example:

@TestExecutionListeners(MockitoTestExecutionListener.class)

The following example replaces an existing RemoteService bean with a mock implementation:

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.\*; import org.springframework.boot.test.context.\*;

import org.springframework.boot.test.mock.mockito.\*;

import static org.assertj.core.api.Assertions.\*; import static org.mockito.BDDMockito.\*;

@SpringBootTest class MyTests {

@MockBean

private RemoteService remoteService;

@Autowired

private Reverser reverser;

@Test

void exampleTest() {

// RemoteService has been injected into the reverser bean given(this.remoteService.someCall()).willReturn("mock"); String reverse = reverser.reverseSomeCall(); assertThat(reverse).isEqualTo("kcom");

}

}

@MockBean cannot be used to mock the behavior of a bean that’s exercised during application context refresh. By the time the test is executed, the application context refresh has completed and it is too late to configure the mocked behavior. We recommend using a @Bean method to create and configure the mock in this situation.

Additionally, you can use @SpyBean to wrap any existing bean with a Mockito spy. See the [Javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/test/mock/mockito/SpyBean.html) for full details.

CGLib proxies, such as those created for scoped beans, declare the proxied methods as final. This stops Mockito from functioning correctly as it cannot mock or spy on final methods in its default configuration. If you want to mock or spy on such a bean, configure Mockito to use its inline mock maker by adding org.mockito:mockito-inline to your application’s test dependencies. This allows Mockito to mock and spy on final methods.

While Spring’s test framework caches application contexts between tests and reuses a context for tests sharing the same configuration, the use of @MockBean or @SpyBean influences the cache key, which will most likely increase the number of contexts.

If you are using @SpyBean to spy on a bean with @Cacheable methods that refer to parameters by name, your application must be compiled with -parameters. This ensures that the parameter names are available to the caching infrastructure once the bean has been spied upon.

When you are using @SpyBean to spy on a bean that is proxied by Spring, you may need to remove Spring’s proxy in some situations, for example when setting expectations using given or when. Use AopTestUtils.getTargetObject(yourProxiedSpy) to do so.

#### Auto-configured Tests

Spring Boot’s auto-configuration system works well for applications but can sometimes be a little too much for tests. It often helps to load only the parts of the configuration that are required to test a “slice” of your application. For example, you might want to test that Spring MVC controllers are mapping URLs correctly, and you do not want to involve database calls in those tests, or you might want to test JPA entities, and you are not interested in the web layer when those tests run.

The spring-boot-test-autoconfigure module includes a number of annotations that can be used to automatically configure such “slices”. Each of them works in a similar way, providing a @…Test annotation that loads the ApplicationContext and one or more @AutoConfigure… annotations that can be used to customize auto-configuration settings.

Each slice restricts component scan to appropriate components and loads a very restricted set of auto-configuration classes. If you need to exclude one of them, most @…Test annotations provide an excludeAutoConfiguration attribute. Alternatively, you can use @ImportAutoConfiguration#exclude.

Including multiple “slices” by using several @…Test annotations in one test is not supported. If you need multiple “slices”, pick one of the @…Test annotations and include the @AutoConfigure… annotations of the other “slices” by hand.

It is also possible to use the @AutoConfigure… annotations with the standard @SpringBootTest annotation. You can use this combination if you are not interested in “slicing” your application but you want some of the auto-configured test beans.

#### Auto-configured JSON Tests

To test that object JSON serialization and deserialization is working as expected, you can use the @JsonTest annotation. @JsonTest auto-configures the available supported JSON mapper, which can be one of the following libraries:

* Jackson ObjectMapper, any @JsonComponent beans and any Jackson Modules
* Gson
* Jsonb

A list of the auto-configurations that are enabled by @JsonTest can be [found in the](#_bookmark689) [appendix](#_bookmark689).

If you need to configure elements of the auto-configuration, you can use the

@AutoConfigureJsonTesters annotation.

Spring Boot includes AssertJ-based helpers that work with the JSONAssert and JsonPath libraries to check that JSON appears as expected. The JacksonTester, GsonTester, JsonbTester, and BasicJsonTester classes can be used for Jackson, Gson, Jsonb, and Strings respectively. Any helper fields on the test class can be @Autowired when using @JsonTest. The following example shows a test class for Jackson:

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.\*; import org.springframework.boot.test.autoconfigure.json.\*; import org.springframework.boot.test.context.\*;

import org.springframework.boot.test.json.\*; import static org.assertj.core.api.Assertions.\*;

@JsonTest

class MyJsonTests {

@Autowired

private JacksonTester<VehicleDetails> json;

@Test

void testSerialize() throws Exception {

VehicleDetails details = new VehicleDetails("Honda", "Civic");

// Assert against a `.json` file in the same package as the test assertThat(this.json.write(details)).isEqualToJson("expected.json");

// Or use JSON path based assertions assertThat(this.json.write(details)).hasJsonPathStringValue("@.make"); assertThat(this.json.write(details)).extractingJsonPathStringValue("@.make")

.isEqualTo("Honda");

}

@Test

void testDeserialize() throws Exception {

String content = "{\"make\":\"Ford\",\"model\":\"Focus\"}"; assertThat(this.json.parse(content))

.isEqualTo(new VehicleDetails("Ford", "Focus")); assertThat(this.json.parseObject(content).getMake()).isEqualTo("Ford");

}

}

JSON helper classes can also be used directly in standard unit tests. To do so, call the initFields method of the helper in your @Before method if you do not use @JsonTest.

If you’re using Spring Boot’s AssertJ-based helpers to assert on a number value at a given JSON path, you might not be able to use isEqualTo depending on the type. Instead, you can use AssertJ’s satisfies to assert that the value matches the given condition. For instance, the following example asserts that the actual number is a float value close to 0.15 within an offset of 0.01.

assertThat(json.write(message))

.extractingJsonPathNumberValue("@.test.numberValue")

.satisfies((number) -> assertThat(number.floatValue()).isCloseTo(0.15f, within(0.01f)));

#### Auto-configured Spring MVC Tests

To test whether Spring MVC controllers are working as expected, use the @WebMvcTest annotation. @WebMvcTest auto-configures the Spring MVC infrastructure and limits scanned beans to @Controller, @ControllerAdvice, @JsonComponent, Converter, GenericConverter, Filter, HandlerInterceptor, WebMvcConfigurer, and HandlerMethodArgumentResolver. Regular @Component and @ConfigurationProperties beans are not scanned when the @WebMvcTest annotation is used. @EnableConfigurationProperties can be used to include @ConfigurationProperties beans.

A list of the auto-configuration settings that are enabled by @WebMvcTest can be [found in the appendix](#_bookmark689).

If you need to register extra components, such as the Jackson Module, you can import additional configuration classes by using @Import on your test.

Often, @WebMvcTest is limited to a single controller and is used in combination with @MockBean to provide mock implementations for required collaborators.

@WebMvcTest also auto-configures MockMvc. Mock MVC offers a powerful way to quickly test MVC controllers without needing to start a full HTTP server.

You can also auto-configure MockMvc in a non-@WebMvcTest (such as @SpringBootTest) by annotating it with @AutoConfigureMockMvc. The following example uses MockMvc:

import org.junit.jupiter.api.\*;

import org.springframework.beans.factory.annotation.\*;

import org.springframework.boot.test.autoconfigure.web.servlet.\*; import org.springframework.boot.test.mock.mockito.\*;

import static org.assertj.core.api.Assertions.\*; import static org.mockito.BDDMockito.\*;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*; import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@WebMvcTest(UserVehicleController.class) class MyControllerTests {

@Autowired

private MockMvc mvc;

@MockBean

private UserVehicleService userVehicleService;

@Test

void testExample() throws Exception { given(this.userVehicleService.getVehicleDetails("sboot"))

.willReturn(new VehicleDetails("Honda", "Civic")); this.mvc.perform(get("/sboot/vehicle").accept(MediaType.TEXT\_PLAIN))

.andExpect(status().isOk()).andExpect(content().string("Honda

Civic"));

}

}

If you need to configure elements of the auto-configuration (for example, when servlet filters should be applied) you can use attributes in the @AutoConfigureMockMvc annotation.

If you use HtmlUnit or Selenium, auto-configuration also provides an HtmlUnit WebClient bean and/or a Selenium WebDriver bean. The following example uses HtmlUnit:

import com.gargoylesoftware.htmlunit.\*; import org.junit.jupiter.api.\*;

import org.springframework.beans.factory.annotation.\*;

import org.springframework.boot.test.autoconfigure.web.servlet.\*; import org.springframework.boot.test.mock.mockito.\*;

import static org.assertj.core.api.Assertions.\*; import static org.mockito.BDDMockito.\*;

@WebMvcTest(UserVehicleController.class) class MyHtmlUnitTests {

@Autowired

private WebClient webClient;

@MockBean

private UserVehicleService userVehicleService;

@Test

void testExample() throws Exception { given(this.userVehicleService.getVehicleDetails("sboot"))

.willReturn(new VehicleDetails("Honda", "Civic")); HtmlPage page = this.webClient.getPage("/sboot/vehicle.html");

assertThat(page.getBody().getTextContent()).isEqualTo("Honda Civic");

}

}

By default, Spring Boot puts WebDriver beans in a special “scope” to ensure that the driver exits after each test and that a new instance is injected. If you do not want this behavior, you can add @Scope("singleton") to your WebDriver @Bean definition.

The webDriver scope created by Spring Boot will replace any user defined scope of the same name. If you define your own webDriver scope you may find it stops working when you use @WebMvcTest.

If you have Spring Security on the classpath, @WebMvcTest will also scan WebSecurityConfigurer beans. Instead of disabling security completely for such tests, you can use Spring Security’s test support. More details on how to use Spring Security’s MockMvc support can be found in this [*Testing*](#_bookmark577)[*With Spring Security*](#_bookmark577) how-to section.

Sometimes writing Spring MVC tests is not enough; Spring Boot can help you run [full end-to-end tests with an actual server](#_bookmark312).

#### Auto-configured Spring WebFlux Tests

To test that [Spring WebFlux](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web-reactive.html) controllers are working as expected, you can use the @WebFluxTest

annotation. @WebFluxTest auto-configures the Spring WebFlux infrastructure and limits scanned

beans to @Controller, @ControllerAdvice, @JsonComponent, Converter, GenericConverter, WebFilter, and WebFluxConfigurer. Regular @Component and @ConfigurationProperties beans are not scanned when the @WebFluxTest annotation is used. @EnableConfigurationProperties can be used to include @ConfigurationProperties beans.

A list of the auto-configurations that are enabled by @WebFluxTest can be [found in](#_bookmark689) [the appendix](#_bookmark689).

If you need to register extra components, such as Jackson Module, you can import additional configuration classes using @Import on your test.

Often, @WebFluxTest is limited to a single controller and used in combination with the @MockBean

annotation to provide mock implementations for required collaborators.

@WebFluxTest also auto-configures [WebTestClient](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/testing.html#webtestclient), which offers a powerful way to quickly test WebFlux controllers without needing to start a full HTTP server.

You can also auto-configure WebTestClient in a non-@WebFluxTest (such as @SpringBootTest) by annotating it with @AutoConfigureWebTestClient. The following example shows a class that uses both @WebFluxTest and a WebTestClient:

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.reactive.WebFluxTest; import org.springframework.http.MediaType;

import org.springframework.test.web.reactive.server.WebTestClient;

@WebFluxTest(UserVehicleController.class) class MyControllerTests {

@Autowired

private WebTestClient webClient;

@MockBean

private UserVehicleService userVehicleService;

@Test

void testExample() throws Exception { given(this.userVehicleService.getVehicleDetails("sboot"))

.willReturn(new VehicleDetails("Honda", "Civic")); this.webClient.get().uri("/sboot/vehicle").accept(MediaType.TEXT\_PLAIN)

.exchange()

.expectStatus().isOk()

.expectBody(String.class).isEqualTo("Honda Civic");

}

}

This setup is only supported by WebFlux applications as using WebTestClient in a mocked web application only works with WebFlux at the moment.

@WebFluxTest cannot detect routes registered via the functional web framework. For testing RouterFunction beans in the context, consider importing your RouterFunction yourself via @Import or using @SpringBootTest.

@WebFluxTest cannot detect custom security configuration registered via a @Bean of type SecurityWebFilterChain. To include that in your test, you will need to import the configuration that registers the bean via @Import or use @SpringBootTest.

Sometimes writing Spring WebFlux tests is not enough; Spring Boot can help you run [full end-to-end tests with an actual server](#_bookmark312).

#### Auto-configured Data JPA Tests

You can use the @DataJpaTest annotation to test JPA applications. By default, it scans for @Entity classes and configures Spring Data JPA repositories. If an embedded database is available on the classpath, it configures one as well. Regular @Component and @ConfigurationProperties beans are not scanned when the @DataJpaTest annotation is used. @EnableConfigurationProperties can be used to include @ConfigurationProperties beans.

A list of the auto-configuration settings that are enabled by @DataJpaTest can be [found in the appendix](#_bookmark689).

By default, data JPA tests are transactional and roll back at the end of each test. See the [relevant](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/testing.html#testcontext-tx-enabling-transactions) [section](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/testing.html#testcontext-tx-enabling-transactions) in the Spring Framework Reference Documentation for more details. If that is not what you want, you can disable transaction management for a test or for the whole class as follows:

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest; import org.springframework.transaction.annotation.Propagation;

import org.springframework.transaction.annotation.Transactional;

@DataJpaTest

@Transactional(propagation = Propagation.NOT\_SUPPORTED) class ExampleNonTransactionalTests {

}

Data JPA tests may also inject a [TestEntityManager](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-test-autoconfigure/src/main/java/org/springframework/boot/test/autoconfigure/orm/jpa/TestEntityManager.java) bean, which provides an alternative to the standard JPA EntityManager that is specifically designed for tests. If you want to use TestEntityManager outside of @DataJpaTest instances, you can also use the @AutoConfigureTestEntityManager annotation. A JdbcTemplate is also available if you need that. The following example shows the @DataJpaTest annotation in use:

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.autoconfigure.orm.jpa.\*; import static org.assertj.core.api.Assertions.\*;

@DataJpaTest

class ExampleRepositoryTests {

@Autowired

private TestEntityManager entityManager;

@Autowired

private UserRepository repository;

@Test

void testExample() throws Exception { this.entityManager.persist(new User("sboot", "1234")); User user = this.repository.findByUsername("sboot"); assertThat(user.getUsername()).isEqualTo("sboot"); assertThat(user.getVin()).isEqualTo("1234");

}

}

In-memory embedded databases generally work well for tests, since they are fast and do not require any installation. If, however, you prefer to run tests against a real database you can use the @AutoConfigureTestDatabase annotation, as shown in the following example:

@DataJpaTest @AutoConfigureTestDatabase(replace=Replace.NONE) class ExampleRepositoryTests {

// ...

}

#### Auto-configured JDBC Tests

@JdbcTest is similar to @DataJpaTest but is for tests that only require a DataSource and do not use Spring Data JDBC. By default, it configures an in-memory embedded database and a JdbcTemplate. Regular @Component and @ConfigurationProperties beans are not scanned when the @JdbcTest annotation is used. @EnableConfigurationProperties can be used to include @ConfigurationProperties beans.

A list of the auto-configurations that are enabled by @JdbcTest can be [found in the](#_bookmark689) [appendix](#_bookmark689).

By default, JDBC tests are transactional and roll back at the end of each test. See the [relevant section](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/testing.html#testcontext-tx-enabling-transactions)

in the Spring Framework Reference Documentation for more details. If that is not what you want, you can disable transaction management for a test or for the whole class, as follows:

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.autoconfigure.jdbc.JdbcTest; import org.springframework.transaction.annotation.Propagation; import org.springframework.transaction.annotation.Transactional;

@JdbcTest

@Transactional(propagation = Propagation.NOT\_SUPPORTED) class ExampleNonTransactionalTests {

}

If you prefer your test to run against a real database, you can use the @AutoConfigureTestDatabase

annotation in the same way as for DataJpaTest. (See "[Auto-configured Data JPA Tests](#_bookmark320)".)

#### Auto-configured Data JDBC Tests

@DataJdbcTest is similar to @JdbcTest but is for tests that use Spring Data JDBC repositories. By default, it configures an in-memory embedded database, a JdbcTemplate, and Spring Data JDBC repositories. Regular @Component and @ConfigurationProperties beans are not scanned when the @DataJdbcTest annotation is used. @EnableConfigurationProperties can be used to include @ConfigurationProperties beans.

A list of the auto-configurations that are enabled by @DataJdbcTest can be [found in](#_bookmark689) [the appendix](#_bookmark689).

By default, Data JDBC tests are transactional and roll back at the end of each test. See the [relevant](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/testing.html#testcontext-tx-enabling-transactions) [section](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/testing.html#testcontext-tx-enabling-transactions) in the Spring Framework Reference Documentation for more details. If that is not what you want, you can disable transaction management for a test or for the whole test class as [shown in the](#_bookmark321) [JDBC example](#_bookmark321).

If you prefer your test to run against a real database, you can use the @AutoConfigureTestDatabase

annotation in the same way as for DataJpaTest. (See "[Auto-configured Data JPA Tests](#_bookmark320)".)

#### Auto-configured jOOQ Tests

You can use @JooqTest in a similar fashion as @JdbcTest but for jOOQ-related tests. As jOOQ relies heavily on a Java-based schema that corresponds with the database schema, the existing DataSource is used. If you want to replace it with an in-memory database, you can use @AutoConfigureTestDatabase to override those settings. (For more about using jOOQ with Spring Boot, see "[Using jOOQ](#_bookmark212)", earlier in this chapter.) Regular @Component and @ConfigurationProperties beans are not scanned when the @JooqTest annotation is used. @EnableConfigurationProperties can be used to include @ConfigurationProperties beans.

A list of the auto-configurations that are enabled by @JooqTest can be [found in the](#_bookmark689)

[appendix](#_bookmark689).

@JooqTest configures a DSLContext. The following example shows the @JooqTest annotation in use:

import org.jooq.DSLContext;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.autoconfigure.jooq.JooqTest;

@JooqTest

class ExampleJooqTests {

@Autowired

private DSLContext dslContext;

}

JOOQ tests are transactional and roll back at the end of each test by default. If that is not what you want, you can disable transaction management for a test or for the whole test class as [shown in the](#_bookmark321) [JDBC example](#_bookmark321).

#### Auto-configured Data MongoDB Tests

You can use @DataMongoTest to test MongoDB applications. By default, it configures an in-memory embedded MongoDB (if available), configures a MongoTemplate, scans for @Document classes, and configures Spring Data MongoDB repositories. Regular @Component and @ConfigurationProperties beans are not scanned when the @DataMongoTest annotation is used. @EnableConfigurationProperties can be used to include @ConfigurationProperties beans. (For more about using MongoDB with Spring Boot, see "[MongoDB](#_bookmark224)", earlier in this chapter.)

A list of the auto-configuration settings that are enabled by @DataMongoTest can be [found in the appendix](#_bookmark689).

The following class shows the @DataMongoTest annotation in use:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.data.mongo.DataMongoTest; import org.springframework.data.mongodb.core.MongoTemplate;

@DataMongoTest

class ExampleDataMongoTests {

@Autowired

private MongoTemplate mongoTemplate;

//

}

In-memory embedded MongoDB generally works well for tests, since it is fast and does not require any developer installation. If, however, you prefer to run tests against a real MongoDB server, you should exclude the embedded MongoDB auto-configuration, as shown in the following example:

import org.springframework.boot.autoconfigure.mongo.embedded.EmbeddedMongoAutoConfiguration; import org.springframework.boot.test.autoconfigure.data.mongo.DataMongoTest;

@DataMongoTest(excludeAutoConfiguration = EmbeddedMongoAutoConfiguration.class) class ExampleDataMongoNonEmbeddedTests {

}

#### Auto-configured Data Neo4j Tests

You can use @DataNeo4jTest to test Neo4j applications. By default, it uses an in-memory embedded Neo4j (if the embedded driver is available), scans for @NodeEntity classes, and configures Spring Data Neo4j repositories. Regular @Component and @ConfigurationProperties beans are not scanned when the @DataNeo4jTest annotation is used. @EnableConfigurationProperties can be used to include @ConfigurationProperties beans. (For more about using Neo4J with Spring Boot, see "[Neo4j](#_bookmark229)", earlier in this chapter.)

A list of the auto-configuration settings that are enabled by @DataNeo4jTest can be [found in the appendix](#_bookmark689).

The following example shows a typical setup for using Neo4J tests in Spring Boot:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.data.neo4j.DataNeo4jTest;

@DataNeo4jTest

class ExampleDataNeo4jTests {

@Autowired

private YourRepository repository;

//

}

By default, Data Neo4j tests are transactional and roll back at the end of each test. See the [relevant](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/testing.html#testcontext-tx-enabling-transactions) [section](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/testing.html#testcontext-tx-enabling-transactions) in the Spring Framework Reference Documentation for more details. If that is not what you want, you can disable transaction management for a test or for the whole class, as follows:

import org.springframework.boot.test.autoconfigure.data.neo4j.DataNeo4jTest; import org.springframework.transaction.annotation.Propagation;

import org.springframework.transaction.annotation.Transactional;

@DataNeo4jTest

@Transactional(propagation = Propagation.NOT\_SUPPORTED) class ExampleNonTransactionalTests {

}

#### Auto-configured Data Redis Tests

You can use @DataRedisTest to test Redis applications. By default, it scans for @RedisHash classes and configures Spring Data Redis repositories. Regular @Component and @ConfigurationProperties beans are not scanned when the @DataRedisTest annotation is used. @EnableConfigurationProperties can be used to include @ConfigurationProperties beans. (For more about using Redis with Spring Boot, see "[Redis](#_bookmark222)", earlier in this chapter.)

A list of the auto-configuration settings that are enabled by @DataRedisTest can be [found in the appendix](#_bookmark689).

The following example shows the @DataRedisTest annotation in use:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.data.redis.DataRedisTest;

@DataRedisTest

class ExampleDataRedisTests {

@Autowired

private YourRepository repository;

//

}

#### Auto-configured Data LDAP Tests

You can use @DataLdapTest to test LDAP applications. By default, it configures an in-memory embedded LDAP (if available), configures an LdapTemplate, scans for @Entry classes, and configures Spring Data LDAP repositories. Regular @Component and @ConfigurationProperties beans are not scanned when the @DataLdapTest annotation is used. @EnableConfigurationProperties can be used to include @ConfigurationProperties beans. (For more about using LDAP with Spring Boot, see "[LDAP](#_bookmark249)", earlier in this chapter.)

A list of the auto-configuration settings that are enabled by @DataLdapTest can be

[found in the appendix](#_bookmark689).

The following example shows the @DataLdapTest annotation in use:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.data.ldap.DataLdapTest; import org.springframework.ldap.core.LdapTemplate;

@DataLdapTest

class ExampleDataLdapTests {

@Autowired

private LdapTemplate ldapTemplate;

//

}

In-memory embedded LDAP generally works well for tests, since it is fast and does not require any developer installation. If, however, you prefer to run tests against a real LDAP server, you should exclude the embedded LDAP auto-configuration, as shown in the following example:

import org.springframework.boot.autoconfigure.ldap.embedded.EmbeddedLdapAutoConfiguration; import org.springframework.boot.test.autoconfigure.data.ldap.DataLdapTest;

@DataLdapTest(excludeAutoConfiguration = EmbeddedLdapAutoConfiguration.class) class ExampleDataLdapNonEmbeddedTests {

}

#### Auto-configured REST Clients

You can use the @RestClientTest annotation to test REST clients. By default, it auto-configures Jackson, GSON, and Jsonb support, configures a RestTemplateBuilder, and adds support for MockRestServiceServer. Regular @Component and @ConfigurationProperties beans are not scanned when the @RestClientTest annotation is used. @EnableConfigurationProperties can be used to include @ConfigurationProperties beans.

A list of the auto-configuration settings that are enabled by @RestClientTest can be [found in the appendix](#_bookmark689).

The specific beans that you want to test should be specified by using the value or components

attribute of @RestClientTest, as shown in the following example:

@RestClientTest(RemoteVehicleDetailsService.class) class ExampleRestClientTest {

@Autowired

private RemoteVehicleDetailsService service;

@Autowired

private MockRestServiceServer server;

@Test

void getVehicleDetailsWhenResultIsSuccessShouldReturnDetails() throws Exception {

this.server.expect(requestTo("/greet/details"))

.andRespond(withSuccess("hello", MediaType.TEXT\_PLAIN)); String greeting = this.service.callRestService(); assertThat(greeting).isEqualTo("hello");

}

}

#### Auto-configured Spring REST Docs Tests

You can use the @AutoConfigureRestDocs annotation to use [Spring REST Docs](https://spring.io/projects/spring-restdocs) in your tests with Mock MVC, REST Assured, or WebTestClient. It removes the need for the JUnit extension in Spring REST Docs.

@AutoConfigureRestDocs can be used to override the default output directory (target/generated- snippets if you are using Maven or build/generated-snippets if you are using Gradle). It can also be used to configure the host, scheme, and port that appears in any documented URIs.

**Auto-configured Spring REST Docs Tests with Mock MVC**

@AutoConfigureRestDocs customizes the MockMvc bean to use Spring REST Docs when testing Servlet- based web applications. You can inject it by using @Autowired and use it in your tests as you normally would when using Mock MVC and Spring REST Docs, as shown in the following example:

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest; import org.springframework.http.MediaType;

import org.springframework.test.web.servlet.MockMvc;

import static org.springframework.restdocs.mockmvc.MockMvcRestDocumentation.document; import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get; import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@WebMvcTest(UserController.class) @AutoConfigureRestDocs

class UserDocumentationTests {

@Autowired

private MockMvc mvc;

@Test

void listUsers() throws Exception { this.mvc.perform(get("/users").accept(MediaType.TEXT\_PLAIN))

.andExpect(status().isOk())

.andDo(document("list-users"));

}

}

If you require more control over Spring REST Docs configuration than offered by the attributes of @AutoConfigureRestDocs, you can use a RestDocsMockMvcConfigurationCustomizer bean, as shown in the following example:

@TestConfiguration

static class CustomizationConfiguration

implements RestDocsMockMvcConfigurationCustomizer {

@Override

public void customize(MockMvcRestDocumentationConfigurer configurer) { configurer.snippets().withTemplateFormat(TemplateFormats.markdown());

}

}

If you want to make use of Spring REST Docs support for a parameterized output directory, you can create a RestDocumentationResultHandler bean. The auto-configuration calls alwaysDo with this result handler, thereby causing each MockMvc call to automatically generate the default snippets. The following example shows a RestDocumentationResultHandler being defined:

@TestConfiguration(proxyBeanMethods = false) static class ResultHandlerConfiguration {

@Bean

public RestDocumentationResultHandler restDocumentation() { return MockMvcRestDocumentation.document("{method-name}");

}

}

**Auto-configured Spring REST Docs Tests with WebTestClient**

@AutoConfigureRestDocs can also be used with WebTestClient when testing reactive web applications. You can inject it by using @Autowired and use it in your tests as you normally would when using @WebFluxTest and Spring REST Docs, as shown in the following example:

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.restdocs.AutoConfigureRestDocs; import org.springframework.boot.test.autoconfigure.web.reactive.WebFluxTest; import org.springframework.test.web.reactive.server.WebTestClient;

import static org.springframework.restdocs.webtestclient.WebTestClientRestDocumentation.document;

@WebFluxTest @AutoConfigureRestDocs

class UsersDocumentationTests {

@Autowired

private WebTestClient webTestClient;

@Test

void listUsers() {

this.webTestClient.get().uri("/").exchange().expectStatus().isOk().expectBody()

.consumeWith(document("list-users"));

}

}

If you require more control over Spring REST Docs configuration than offered by the attributes of @AutoConfigureRestDocs, you can use a RestDocsWebTestClientConfigurationCustomizer bean, as shown in the following example:

@TestConfiguration(proxyBeanMethods = false)

public static class CustomizationConfiguration implements RestDocsWebTestClientConfigurationCustomizer {

@Override

public void customize(WebTestClientRestDocumentationConfigurer configurer) { configurer.snippets().withEncoding("UTF-8");

}

}

**Auto-configured Spring REST Docs Tests with REST Assured**

@AutoConfigureRestDocs makes a RequestSpecification bean, preconfigured to use Spring REST Docs, available to your tests. You can inject it by using @Autowired and use it in your tests as you normally would when using REST Assured and Spring REST Docs, as shown in the following example:

import io.restassured.specification.RequestSpecification; import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.restdocs.AutoConfigureRestDocs; import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.boot.test.context.SpringBootTest.WebEnvironment; import org.springframework.boot.web.server.LocalServerPort;

import static io.restassured.RestAssured.given; import static org.hamcrest.Matchers.is;

import static org.springframework.restdocs.restassured3.RestAssuredRestDocumentation.document;

@SpringBootTest(webEnvironment = WebEnvironment.RANDOM\_PORT) @AutoConfigureRestDocs

class UserDocumentationTests {

@Test

void listUsers(@Autowired RequestSpecification documentationSpec, @LocalServerPort int port) {

given(documentationSpec).filter(document("list- users")).when().port(port).get("/").then().assertThat()

.statusCode(is(200));

}

}

If you require more control over Spring REST Docs configuration than offered by the attributes of @AutoConfigureRestDocs, a RestDocsRestAssuredConfigurationCustomizer bean can be used, as shown in the following example:

@TestConfiguration(proxyBeanMethods = false)

public static class CustomizationConfiguration implements RestDocsRestAssuredConfigurationCustomizer {

@Override

public void customize(RestAssuredRestDocumentationConfigurer configurer) { configurer.snippets().withTemplateFormat(TemplateFormats.markdown());

}

}

#### Auto-configured Spring Web Services Tests

You can use @WebServiceClientTest to test applications that use call web services using the Spring Web Services project. By default, it configures a mock WebServiceServer bean and automatically customizes your WebServiceTemplateBuilder. (For more about using Web Services with Spring Boot, see "[Web Services](#_bookmark340)", earlier in this chapter.)

A list of the auto-configuration settings that are enabled by @WebServiceClientTest

can be [found in the appendix](#_bookmark689).

The following example shows the @WebServiceClientTest annotation in use:

@WebServiceClientTest(ExampleWebServiceClient.class) class WebServiceClientIntegrationTests {

@Autowired

private MockWebServiceServer server;

@Autowired

private ExampleWebServiceClient client;

@Test

void mockServerCall() {

this.server.expect(payload(new StringSource("<request/>"))).andRespond( withPayload(new

StringSource("<response><status>200</status></response>"))); assertThat(this.client.test()).extracting(Response::getStatus).isEqualTo(200);

}

}

#### Additional Auto-configuration and Slicing

Each slice provides one or more @AutoConfigure… annotations that namely defines the auto- configurations that should be included as part of a slice. Additional auto-configurations can be added on a test-by-test basis by creating a custom @AutoConfigure… annotation or by adding @ImportAutoConfiguration to the test as shown in the following example:

@JdbcTest @ImportAutoConfiguration(IntegrationAutoConfiguration.class) class ExampleJdbcTests {

}

Make sure to not use the regular @Import annotation to import auto-configurations as they are handled in a specific way by Spring Boot.

Alternatively, additional auto-configurations can be added for any use of a slice annotation by registering them in META-INF/spring.factories as shown in the following example:

org.springframework.boot.test.autoconfigure.jdbc.JdbcTest=com.example.IntegrationAutoC onfiguration

A slice or @AutoConfigure… annotation can be customized this way as long as it is meta-annotated with @ImportAutoConfiguration.

#### User Configuration and Slicing

If you [structure your code](#_bookmark48) in a sensible way, your @SpringBootApplication class is [used by default](#_bookmark308) as the configuration of your tests.

It then becomes important not to litter the application’s main class with configuration settings that are specific to a particular area of its functionality.

Assume that you are using Spring Batch and you rely on the auto-configuration for it. You could define your @SpringBootApplication as follows:

@SpringBootApplication @EnableBatchProcessing

public class SampleApplication { ... }

Because this class is the source configuration for the test, any slice test actually tries to start Spring Batch, which is definitely not what you want to do. A recommended approach is to move that area- specific configuration to a separate @Configuration class at the same level as your application, as shown in the following example:

@Configuration(proxyBeanMethods = false) @EnableBatchProcessing

public class BatchConfiguration { ... }

Depending on the complexity of your application, you may either have a single @Configuration class for your customizations or one class per domain area. The latter approach lets you enable it in one of your tests, if necessary, with the @Import annotation.

Test slices exclude @Configuration classes from scanning. For example, for a @WebMvcTest, the following configuration will not include the given WebMvcConfigurer bean in the application context loaded by the test slice:

@Configuration

public class WebConfiguration { @Bean

public WebMvcConfigurer testConfigurer() { return new WebMvcConfigurer() {

...

};

}

}

The configuration below will, however, cause the custom WebMvcConfigurer to be loaded by the test slice.

@Component

public class TestWebMvcConfigurer implements WebMvcConfigurer {

...

}

Another source of confusion is classpath scanning. Assume that, while you structured your code in a sensible way, you need to scan an additional package. Your application may resemble the following code:

@SpringBootApplication

@ComponentScan({ "com.example.app", "org.acme.another" }) public class SampleApplication { ... }

Doing so effectively overrides the default component scan directive with the side effect of scanning those two packages regardless of the slice that you chose. For instance, a @DataJpaTest seems to suddenly scan components and user configurations of your application. Again, moving the custom directive to a separate class is a good way to fix this issue.

If this is not an option for you, you can create a @SpringBootConfiguration somewhere in the hierarchy of your test so that it is used instead. Alternatively, you can specify a source for your test, which disables the behavior of finding a default one.

#### Using Spock to Test Spring Boot Applications

If you wish to use Spock to test a Spring Boot application, you should add a dependency on Spock’s spock-spring module to your application’s build. spock-spring integrates Spring’s test framework into Spock. It is recommended that you use Spock 1.2 or later to benefit from a number of improvements to Spock’s Spring Framework and Spring Boot integration. See [the documentation](http://spockframework.org/spock/docs/1.2/modules.html#_spring_module) [for Spock’s Spring module](http://spockframework.org/spock/docs/1.2/modules.html#_spring_module) for further details.

### Test Utilities

A few test utility classes that are generally useful when testing your application are packaged as part of spring-boot.

#### ConfigFileApplicationContextInitializer

ConfigFileApplicationContextInitializer is an ApplicationContextInitializer that you can apply to your tests to load Spring Boot application.properties files. You can use it when you do not need the full set of features provided by @SpringBootTest, as shown in the following example:

@ContextConfiguration(classes = Config.class,

initializers = ConfigFileApplicationContextInitializer.class)

Using ConfigFileApplicationContextInitializer alone does not provide support for @Value("${…}") injection. Its only job is to ensure that application.properties files are loaded into Spring’s Environment. For @Value support, you need to either additionally configure a PropertySourcesPlaceholderConfigurer or use @SpringBootTest, which auto-configures one for you.

#### TestPropertyValues

TestPropertyValues lets you quickly add properties to a ConfigurableEnvironment or

ConfigurableApplicationContext. You can call it with key=value strings, as follows:

TestPropertyValues.of("org=Spring", "name=Boot").applyTo(env);

#### OutputCapture

OutputCapture is a JUnit Extension that you can use to capture System.out and System.err output. To use add @ExtendWith(OutputCaptureExtension.class) and inject CapturedOutput as an argument to your test class constructor or test method as follows:

@ExtendWith(OutputCaptureExtension.class) class OutputCaptureTests {

@Test

void testName(CapturedOutput output) { System.out.println("Hello World!"); assertThat(output).contains("World");

}

}

#### TestRestTemplate

TestRestTemplate is a convenience alternative to Spring’s RestTemplate that is useful in integration tests. You can get a vanilla template or one that sends Basic HTTP authentication (with a username and password). In either case, the template behaves in a test-friendly way by not throwing exceptions on server-side errors.

Spring Framework 5.0 provides a new WebTestClient that works for [WebFlux](#_bookmark319) [integration tests](#_bookmark319) and both [WebFlux and MVC end-to-end testing](#_bookmark312). It provides a fluent API for assertions, unlike TestRestTemplate.

It is recommended, but not mandatory, to use the Apache HTTP Client (version 4.3.2 or better). If you have that on your classpath, the TestRestTemplate responds by configuring the client appropriately. If you do use Apache’s HTTP client, some additional test-friendly features are enabled:

* Redirects are not followed (so you can assert the response location).
* Cookies are ignored (so the template is stateless).

TestRestTemplate can be instantiated directly in your integration tests, as shown in the following example:

public class MyTest {

private TestRestTemplate template = new TestRestTemplate(); @Test

public void testRequest() throws Exception { HttpHeaders headers = this.template.getForEntity(

"https://myhost.example.com/example", String.class).getHeaders(); assertThat(headers.getLocation()).hasHost("other.example.com");

}

}

Alternatively, if you use the @SpringBootTest annotation with WebEnvironment.RANDOM\_PORT or

WebEnvironment.DEFINED\_PORT, you can inject a fully configured TestRestTemplate and start using it. If necessary, additional customizations can be applied through the RestTemplateBuilder bean. Any URLs that do not specify a host and port automatically connect to the embedded server, as shown in the following example:

@SpringBootTest(webEnvironment = WebEnvironment.RANDOM\_PORT) class SampleWebClientTests {

@Autowired

private TestRestTemplate template;

@Test

void testRequest() {

HttpHeaders headers = this.template.getForEntity("/example", String.class).getHeaders();

assertThat(headers.getLocation()).hasHost("other.example.com");

}

@TestConfiguration(proxyBeanMethods = false) static class Config {

@Bean

RestTemplateBuilder restTemplateBuilder() {

return new RestTemplateBuilder().setConnectTimeout(Duration.ofSeconds(1))

.setReadTimeout(Duration.ofSeconds(1));

}

}

}

## WebSockets

Spring Boot provides WebSockets auto-configuration for embedded Tomcat, Jetty, and Undertow. If you deploy a war file to a standalone container, Spring Boot assumes that the container is responsible for the configuration of its WebSocket support.

Spring Framework provides [rich WebSocket support](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web.html#websocket) for MVC web applications that can be easily accessed through the spring-boot-starter-websocket module.

WebSocket support is also available for [reactive web applications](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web-reactive.html#webflux-websocket) and requires to include the WebSocket API alongside spring-boot-starter-webflux:

<dependency>

<groupId>javax.websocket</groupId>

<artifactId>javax.websocket-api</artifactId>

</dependency>

## Web Services

Spring Boot provides Web Services auto-configuration so that all you must do is define your

Endpoints.

The [Spring Web Services features](https://docs.spring.io/spring-ws/docs/3.0.10.RELEASE/reference/) can be easily accessed with the spring-boot-starter-webservices

module.

SimpleWsdl11Definition and SimpleXsdSchema beans can be automatically created for your WSDLs and XSDs respectively. To do so, configure their location, as shown in the following example:

spring.webservices.wsdl-locations=classpath:/wsdl

### Calling Web Services with WebServiceTemplate

If you need to call remote Web services from your application, you can use the [WebServiceTemplate](https://docs.spring.io/spring-ws/docs/3.0.10.RELEASE/reference/#client-web-service-template) class. Since WebServiceTemplate instances often need to be customized before being used, Spring Boot does not provide any single auto-configured WebServiceTemplate bean. It does, however, auto- configure a WebServiceTemplateBuilder, which can be used to create WebServiceTemplate instances when needed.

The following code shows a typical example:

@Service

public class MyService {

private final WebServiceTemplate webServiceTemplate;

public MyService(WebServiceTemplateBuilder webServiceTemplateBuilder) { this.webServiceTemplate = webServiceTemplateBuilder.build();

}

public DetailsResp someWsCall(DetailsReq detailsReq) { return (DetailsResp)

this.webServiceTemplate.marshalSendAndReceive(detailsReq, new SoapActionCallback(ACTION));

}

}

By default, WebServiceTemplateBuilder detects a suitable HTTP-based WebServiceMessageSender using the available HTTP client libraries on the classpath. You can also customize read and connection timeouts as follows:

@Bean

public WebServiceTemplate webServiceTemplate(WebServiceTemplateBuilder builder) { return builder.messageSenders(new HttpWebServiceMessageSenderBuilder()

.setConnectTimeout(5000).setReadTimeout(2000).build()).build();

}

## Creating Your Own Auto-configuration

If you work in a company that develops shared libraries, or if you work on an open-source or commercial library, you might want to develop your own auto-configuration. Auto-configuration classes can be bundled in external jars and still be picked-up by Spring Boot.

Auto-configuration can be associated to a “starter” that provides the auto-configuration code as well as the typical libraries that you would use with it. We first cover what you need to know to build your own auto-configuration and then we move on to the [typical steps required to create a](#_bookmark355) [custom starter](#_bookmark355).

 A [demo project](https://github.com/snicoll-demos/spring-boot-master-auto-configuration) is available to showcase how you can create a starter step-by-step.

### Understanding Auto-configured Beans

Under the hood, auto-configuration is implemented with standard @Configuration classes. Additional @Conditional annotations are used to constrain when the auto-configuration should apply. Usually, auto-configuration classes use @ConditionalOnClass and @ConditionalOnMissingBean annotations. This ensures that auto-configuration applies only when relevant classes are found and when you have not declared your own @Configuration.

You can browse the source code of [spring-boot-autoconfigure](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure) to see the @Configuration classes that Spring provides (see the [META-INF/spring.factories](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/resources/META-INF/spring.factories) file).

### Locating Auto-configuration Candidates

Spring Boot checks for the presence of a META-INF/spring.factories file within your published jar. The file should list your configuration classes under the EnableAutoConfiguration key, as shown in the following example:

org.springframework.boot.autoconfigure.EnableAutoConfiguration=\ com.mycorp.libx.autoconfigure.LibXAutoConfiguration,\ com.mycorp.libx.autoconfigure.LibXWebAutoConfiguration

Auto-configurations must be loaded that way *only*. Make sure that they are defined in a specific package space and that they are never the target of component scanning. Furthermore, auto-configuration classes should not enable component scanning to find additional components. Specific @Imports should be used instead.

You can use the [@AutoConfigureAfter](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/AutoConfigureAfter.java) or [@AutoConfigureBefore](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/AutoConfigureBefore.java) annotations if your configuration

needs to be applied in a specific order. For example, if you provide web-specific configuration, your class may need to be applied after WebMvcAutoConfiguration.

If you want to order certain auto-configurations that should not have any direct knowledge of each other, you can also use @AutoConfigureOrder. That annotation has the same semantic as the regular @Order annotation but provides a dedicated order for auto-configuration classes.

As with standard @Configuration classes, the order in which auto-configuration classes are applied only affects the order in which their beans are defined. The order in which those beans are subsequently created is unaffected and is determined by each bean’s dependencies and any @DependsOn relationships.

### Condition Annotations

You almost always want to include one or more @Conditional annotations on your auto- configuration class. The @ConditionalOnMissingBean annotation is one common example that is used to allow developers to override auto-configuration if they are not happy with your defaults.

Spring Boot includes a number of @Conditional annotations that you can reuse in your own code by annotating @Configuration classes or individual @Bean methods. These annotations include:

* [Class Conditions](#_bookmark346)
* [Bean Conditions](#_bookmark347)
* [Property Conditions](#_bookmark348)
* [Resource Conditions](#_bookmark349)
* [Web Application Conditions](#_bookmark350)
* [SpEL Expression Conditions](#_bookmark351)

#### Class Conditions

The @ConditionalOnClass and @ConditionalOnMissingClass annotations let @Configuration classes be included based on the presence or absence of specific classes. Due to the fact that annotation metadata is parsed by using [ASM](https://asm.ow2.io/), you can use the value attribute to refer to the real class, even though that class might not actually appear on the running application classpath. You can also use the name attribute if you prefer to specify the class name by using a String value.

This mechanism does not apply the same way to @Bean methods where typically the return type is the target of the condition: before the condition on the method applies, the JVM will have loaded the class and potentially processed method references which will fail if the class is not present.

To handle this scenario, a separate @Configuration class can be used to isolate the condition, as shown in the following example:

@Configuration(proxyBeanMethods = false)

// Some conditions

public class MyAutoConfiguration {

// Auto-configured beans

@Configuration(proxyBeanMethods = false) @ConditionalOnClass(EmbeddedAcmeService.class) static class EmbeddedConfiguration {

@Bean @ConditionalOnMissingBean

public EmbeddedAcmeService embeddedAcmeService() { ... }

}

}

If you use @ConditionalOnClass or @ConditionalOnMissingClass as a part of a meta- annotation to compose your own composed annotations, you must use name as referring to the class in such a case is not handled.

#### Bean Conditions

The @ConditionalOnBean and @ConditionalOnMissingBean annotations let a bean be included based on the presence or absence of specific beans. You can use the value attribute to specify beans by type or name to specify beans by name. The search attribute lets you limit the ApplicationContext hierarchy that should be considered when searching for beans.

When placed on a @Bean method, the target type defaults to the return type of the method, as shown in the following example:

@Configuration(proxyBeanMethods = false) public class MyAutoConfiguration {

@Bean @ConditionalOnMissingBean

public MyService myService() { ... }

}

In the preceding example, the myService bean is going to be created if no bean of type MyService is already contained in the ApplicationContext.

You need to be very careful about the order in which bean definitions are added, as these conditions are evaluated based on what has been processed so far. For this reason, we recommend using only @ConditionalOnBean and @ConditionalOnMissingBean annotations on auto-configuration classes (since these are guaranteed to load after any user-defined bean definitions have been added).

@ConditionalOnBean and @ConditionalOnMissingBean do not prevent @Configuration classes from being created. The only difference between using these conditions at the class level and marking each contained @Bean method with the annotation is that the former prevents registration of the @Configuration class as a bean if the condition does not match.

#### Property Conditions

The @ConditionalOnProperty annotation lets configuration be included based on a Spring Environment property. Use the prefix and name attributes to specify the property that should be checked. By default, any property that exists and is not equal to false is matched. You can also create more advanced checks by using the havingValue and matchIfMissing attributes.

#### Resource Conditions

The @ConditionalOnResource annotation lets configuration be included only when a specific resource is present. Resources can be specified by using the usual Spring conventions, as shown in the following example: file:/home/user/test.dat.

#### Web Application Conditions

The @ConditionalOnWebApplication and @ConditionalOnNotWebApplication annotations let configuration be included depending on whether the application is a “web application”. A servlet- based web application is any application that uses a Spring WebApplicationContext, defines a session scope, or has a ConfigurableWebEnvironment. A reactive web application is any application that uses a ReactiveWebApplicationContext, or has a ConfigurableReactiveWebEnvironment.

The @ConditionalOnWarDeployment annotation lets configuration be included depending on whether the application is a traditional WAR application that is deployed to a container. This condition will not match for applications that are run with an embedded server.

#### SpEL Expression Conditions

The @ConditionalOnExpression annotation lets configuration be included based on the result of a [SpEL expression](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/core.html#expressions).

### Testing your Auto-configuration

An auto-configuration can be affected by many factors: user configuration (@Bean definition and Environment customization), condition evaluation (presence of a particular library), and others. Concretely, each test should create a well defined ApplicationContext that represents a combination of those customizations. ApplicationContextRunner provides a great way to achieve that.

ApplicationContextRunner is usually defined as a field of the test class to gather the base, common configuration. The following example makes sure that UserServiceAutoConfiguration is always invoked:

private final ApplicationContextRunner contextRunner = new ApplicationContextRunner()

.withConfiguration(AutoConfigurations.of(UserServiceAutoConfiguration.class));

If multiple auto-configurations have to be defined, there is no need to order their declarations as they are invoked in the exact same order as when running the application.

Each test can use the runner to represent a particular use case. For instance, the sample below invokes a user configuration (UserConfiguration) and checks that the auto-configuration backs off properly. Invoking run provides a callback context that can be used with AssertJ.

@Test

void defaultServiceBacksOff() { this.contextRunner.withUserConfiguration(UserConfiguration.class).run((context) ->

{

assertThat(context).hasSingleBean(UserService.class);

assertThat(context).getBean("myUserService").isSameAs(context.getBean(UserService.clas s));

});

}

@Configuration(proxyBeanMethods = false) static class UserConfiguration {

@Bean

UserService myUserService() { return new UserService("mine");

}

}

It is also possible to easily customize the Environment, as shown in the following example:

@Test

void serviceNameCanBeConfigured() { this.contextRunner.withPropertyValues("user.name=test123").run((context) -> {

assertThat(context).hasSingleBean(UserService.class); assertThat(context.getBean(UserService.class).getName()).isEqualTo("test123");

});

}

The runner can also be used to display the ConditionEvaluationReport. The report can be printed at

INFO or DEBUG level. The following example shows how to use the

ConditionEvaluationReportLoggingListener to print the report in auto-configuration tests.

@Test

public void autoConfigTest { ConditionEvaluationReportLoggingListener initializer = new

ConditionEvaluationReportLoggingListener( LogLevel.INFO);

ApplicationContextRunner contextRunner = new ApplicationContextRunner()

.withInitializer(initializer).run((context) -> {

// Do something...

});

}

#### Simulating a Web Context

If you need to test an auto-configuration that only operates in a Servlet or Reactive web application context, use the WebApplicationContextRunner or ReactiveWebApplicationContextRunner respectively.

#### Overriding the Classpath

It is also possible to test what happens when a particular class and/or package is not present at runtime. Spring Boot ships with a FilteredClassLoader that can easily be used by the runner. In the following example, we assert that if UserService is not present, the auto-configuration is properly disabled:

@Test

void serviceIsIgnoredIfLibraryIsNotPresent() { this.contextRunner.withClassLoader(new FilteredClassLoader(UserService.class))

.run((context) -> assertThat(context).doesNotHaveBean("userService"));

}

### Creating Your Own Starter

A typical Spring Boot starter contains code to auto-configure and customize the infrastructure of a given technology, let’s call that "acme". To make it easily extensible, a number of configuration keys in a dedicated namespace can be exposed to the environment. Finally, a single "starter" dependency is provided to help users get started as easily as possible.

Concretely, a custom starter can contain the following:

* The autoconfigure module that contains the auto-configuration code for "acme".
* The starter module that provides a dependency to the autoconfigure module as well as "acme" and any additional dependencies that are typically useful. In a nutshell, adding the starter should provide everything needed to start using that library.

This separation in two modules is in no way necessary. If "acme" has several flavours, options or

optional features, then it is better to separate the auto-configuration as you can clearly express the fact some features are optional. Besides, you have the ability to craft a starter that provides an opinion about those optional dependencies. At the same time, others can rely only on the autoconfigure module and craft their own starter with different opinions.

If the auto-configuration is relatively straightforward and does not have optional feature, merging the two modules in the starter is definitely an option.

#### Naming

You should make sure to provide a proper namespace for your starter. Do not start your module names with spring-boot, even if you use a different Maven groupId. We may offer official support for the thing you auto-configure in the future.

As a rule of thumb, you should name a combined module after the starter. For example, assume that you are creating a starter for "acme" and that you name the auto-configure module acme- spring-boot and the starter acme-spring-boot-starter. If you only have one module that combines the two, name it acme-spring-boot-starter.

#### Configuration keys

If your starter provides configuration keys, use a unique namespace for them. In particular, do not include your keys in the namespaces that Spring Boot uses (such as server, management, spring, and so on). If you use the same namespace, we may modify these namespaces in the future in ways that break your modules. As a rule of thumb, prefix all your keys with a namespace that you own (e.g. acme).

Make sure that configuration keys are documented by adding field javadoc for each property, as shown in the following example:

@ConfigurationProperties("acme") public class AcmeProperties {

/\*\*

* Whether to check the location of acme resources.

\*/

private boolean checkLocation = true;

/\*\*

* Timeout for establishing a connection to the acme server.

\*/

private Duration loginTimeout = Duration.ofSeconds(3);

// getters & setters

}

You should only use plain text with @ConfigurationProperties field Javadoc, since they are not processed before being added to the JSON.

Here are some rules we follow internally to make sure descriptions are consistent:

* Do not start the description by "The" or "A".
* For boolean types, start the description with "Whether" or "Enable".
* For collection-based types, start the description with "Comma-separated list"
* Use java.time.Duration rather than long and describe the default unit if it differs from milliseconds, e.g. "If a duration suffix is not specified, seconds will be used".
* Do not provide the default value in the description unless it has to be determined at runtime.

Make sure to [trigger meta-data generation](#_bookmark682) so that IDE assistance is available for your keys as well. You may want to review the generated metadata (META-INF/spring-configuration-metadata.json) to make sure your keys are properly documented. Using your own starter in a compatible IDE is also a good idea to validate that quality of the metadata.

#### The “autoconfigure” Module

The autoconfigure module contains everything that is necessary to get started with the library. It may also contain configuration key definitions (such as @ConfigurationProperties) and any callback interface that can be used to further customize how the components are initialized.

You should mark the dependencies to the library as optional so that you can include the autoconfigure module in your projects more easily. If you do it that way, the library is not provided and, by default, Spring Boot backs off.

Spring Boot uses an annotation processor to collect the conditions on auto-configurations in a metadata file (META-INF/spring-autoconfigure-metadata.properties). If that file is present, it is used to eagerly filter auto-configurations that do not match, which will improve startup time. It is recommended to add the following dependency in a module that contains auto-configurations:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-autoconfigure-processor</artifactId>

<optional>true</optional>

</dependency>

If you have defined auto-configurations directly in your application, make sure to configure the

spring-boot-maven-plugin to prevent the repackage goal from adding the dependency into the fat jar:

<project>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<excludes>

<exclude>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-autoconfigure-

processor</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

</plugins>

</build>

</project>

With Gradle 4.5 and earlier, the dependency should be declared in the compileOnly configuration, as shown in the following example:

dependencies {

compileOnly "org.springframework.boot:spring-boot-autoconfigure-processor"

}

With Gradle 4.6 and later, the dependency should be declared in the annotationProcessor

configuration, as shown in the following example:

dependencies {

annotationProcessor "org.springframework.boot:spring-boot-autoconfigure-processor"

}

#### Starter Module

The starter is really an empty jar. Its only purpose is to provide the necessary dependencies to work with the library. You can think of it as an opinionated view of what is required to get started.

Do not make assumptions about the project in which your starter is added. If the library you are auto-configuring typically requires other starters, mention them as well. Providing a proper set of *default* dependencies may be hard if the number of optional dependencies is high, as you should avoid including dependencies that are unnecessary for a typical usage of the library. In other words, you should not include optional dependencies.

Either way, your starter must reference the core Spring Boot starter (spring-boot- starter) directly or indirectly (i.e. no need to add it if your starter relies on another starter). If a project is created with only your custom starter, Spring Boot’s core features will be honoured by the presence of the core starter.

## Kotlin support

[Kotlin](https://kotlinlang.org/) is a statically-typed language targeting the JVM (and other platforms) which allows writing concise and elegant code while providing [interoperability](https://kotlinlang.org/docs/reference/java-interop.html) with existing libraries written in Java.

Spring Boot provides Kotlin support by leveraging the support in other Spring projects such as Spring Framework, Spring Data, and Reactor. See the [Spring Framework Kotlin support](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/languages.html#kotlin) [documentation](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/languages.html#kotlin) for more information.

The easiest way to start with Spring Boot and Kotlin is to follow [this comprehensive tutorial](https://spring.io/guides/tutorials/spring-boot-kotlin/). You can create new Kotlin projects via [start.spring.io](https://start.spring.io/#!language%3Dkotlin). Feel free to join the #spring channel of [Kotlin](https://slack.kotlinlang.org/) [Slack](https://slack.kotlinlang.org/) or ask a question with the spring and kotlin tags on [Stack Overflow](https://stackoverflow.com/questions/tagged/spring%2Bkotlin) if you need support.

### Requirements

Spring Boot supports Kotlin 1.3.x. To use Kotlin, org.jetbrains.kotlin:kotlin-stdlib and org.jetbrains.kotlin:kotlin-reflect must be present on the classpath. The kotlin-stdlib variants kotlin-stdlib-jdk7 and kotlin-stdlib-jdk8 can also be used.

Since [Kotlin classes are final by default](https://discuss.kotlinlang.org/t/classes-final-by-default/166), you are likely to want to configure [kotlin-spring](https://kotlinlang.org/docs/reference/compiler-plugins.html#spring-support) plugin in order to automatically open Spring-annotated classes so that they can be proxied.

[Jackson’s Kotlin module](https://github.com/FasterXML/jackson-module-kotlin) is required for serializing / deserializing JSON data in Kotlin. It is automatically registered when found on the classpath. A warning message is logged if Jackson and Kotlin are present but the Jackson Kotlin module is not.

These dependencies and plugins are provided by default if one bootstraps a Kotlin project on [start.spring.io](https://start.spring.io/#!language%3Dkotlin).

### Null-safety

One of Kotlin’s key features is [null-safety](https://kotlinlang.org/docs/reference/null-safety.html). It deals with null values at compile time rather than deferring the problem to runtime and encountering a NullPointerException. This helps to eliminate a common source of bugs without paying the cost of wrappers like Optional. Kotlin also allows using functional constructs with nullable values as described in this [comprehensive guide to null-safety](https://www.baeldung.com/kotlin-null-safety) [in Kotlin](https://www.baeldung.com/kotlin-null-safety).

Although Java does not allow one to express null-safety in its type system, Spring Framework, Spring Data, and Reactor now provide null-safety of their API via tooling-friendly annotations. By default, types from Java APIs used in Kotlin are recognized as [platform types](https://kotlinlang.org/docs/reference/java-interop.html#null-safety-and-platform-types) for which null-checks are relaxed. [Kotlin’s support for JSR 305 annotations](https://kotlinlang.org/docs/reference/java-interop.html#jsr-305-support) combined with nullability annotations provide null-safety for the related Spring API in Kotlin.

The JSR 305 checks can be configured by adding the -Xjsr305 compiler flag with the following options: -Xjsr305={strict|warn|ignore}. The default behavior is the same as -Xjsr305=warn. The strict value is required to have null-safety taken in account in Kotlin types inferred from Spring API but should be used with the knowledge that Spring API nullability declaration could evolve even between minor releases and more checks may be added in the future).

Generic type arguments, varargs and array elements nullability are not yet supported. See [SPR-15942](https://jira.spring.io/browse/SPR-15942) for up-to-date information. Also be aware that Spring Boot’s own API is [not yet annotated](https://github.com/spring-projects/spring-boot/issues/10712).

### Kotlin API

#### runApplication

Spring Boot provides an idiomatic way to run an application with

runApplication<MyApplication>(\*args) as shown in the following example:

import org.springframework.boot.autoconfigure.SpringBootApplication import org.springframework.boot.runApplication

@SpringBootApplication class MyApplication

fun main(args: Array<String>) { runApplication<MyApplication>(\*args)

}

This is a drop-in replacement for SpringApplication.run(MyApplication::class.java, \*args). It also allows customization of the application as shown in the following example:

runApplication<MyApplication>(\*args) { setBannerMode(OFF)

}

#### Extensions

Kotlin [extensions](https://kotlinlang.org/docs/reference/extensions.html) provide the ability to extend existing classes with additional functionality. The Spring Boot Kotlin API makes use of these extensions to add new Kotlin specific conveniences to existing APIs.

TestRestTemplate extensions, similar to those provided by Spring Framework for RestOperations in Spring Framework, are provided. Among other things, the extensions make it possible to take advantage of Kotlin reified type parameters.

### Dependency management

In order to avoid mixing different versions of Kotlin dependencies on the classpath, Spring Boot

imports the Kotlin BOM.

With Maven, the Kotlin version can be customized via the kotlin.version property and plugin management is provided for kotlin-maven-plugin. With Gradle, the Spring Boot plugin automatically aligns the kotlin.version with the version of the Kotlin plugin.

Spring Boot also manages the version of Coroutines dependencies by importing the Kotlin Coroutines BOM. The version can be customized via the kotlin-coroutines.version property.

org.jetbrains.kotlinx:kotlinx-coroutines-reactor dependency is provided by default if one bootstraps a Kotlin project with at least one reactive dependency on [start.spring.io](https://start.spring.io/#!language%3Dkotlin).

### @ConfigurationProperties

@ConfigurationProperties when used in combination with [@ConstructorBinding](#_bookmark116) supports classes with immutable val properties as shown in the following example:

@ConstructorBinding @ConfigurationProperties("example.kotlin") data class KotlinExampleProperties(

val name: String,

val description: String, val myService: MyService) {

data class MyService(

val apiToken: String, val uri: URI

)

}

To generate [your own metadata](#_bookmark682) using the annotation processor, [kapt should be](https://kotlinlang.org/docs/reference/kapt.html) [configured](https://kotlinlang.org/docs/reference/kapt.html) with the spring-boot-configuration-processor dependency. Note that some features (such as detecting the default value or deprecated items) are not working due to limitations in the model kapt provides.

### Testing

While it is possible to use JUnit 4 to test Kotlin code, JUnit 5 is provided by default and is recommended. JUnit 5 enables a test class to be instantiated once and reused for all of the class’s tests. This makes it possible to use @BeforeAll and @AfterAll annotations on non-static methods, which is a good fit for Kotlin.

JUnit 5 is the default and the vintage engine is provided for backward compatibility with JUnit 4. If you don’t use it, exclude org.junit.vintage:junit-vintage-engine. You also need to [switch test](https://junit.org/junit5/docs/current/user-guide/#writing-tests-test-instance-lifecycle-changing-default) [instance lifecycle to "per-class"](https://junit.org/junit5/docs/current/user-guide/#writing-tests-test-instance-lifecycle-changing-default).

To mock Kotlin classes, [MockK](https://mockk.io/) is recommended. If you need the Mockk equivalent of the Mockito

specific [@MockBean and](#_bookmark315) [@SpyBean annotations](#_bookmark315), you can use [SpringMockK](https://github.com/Ninja-Squad/springmockk) which provides similar

@MockkBean and @SpykBean annotations.

### Resources

#### Further reading

* [Kotlin language reference](https://kotlinlang.org/docs/reference/)
* [Kotlin Slack](https://kotlinlang.slack.com/) (with a dedicated #spring channel)
* [Stackoverflow with spring and kotlin tags](https://stackoverflow.com/questions/tagged/spring%2Bkotlin)
* [Try Kotlin in your browser](https://try.kotlinlang.org/)
* [Kotlin blog](https://blog.jetbrains.com/kotlin/)
* [Awesome Kotlin](https://kotlin.link/)
* [Tutorial: building web applications with Spring Boot and Kotlin](https://spring.io/guides/tutorials/spring-boot-kotlin/)
* [Developing Spring Boot applications with Kotlin](https://spring.io/blog/2016/02/15/developing-spring-boot-applications-with-kotlin)
* [A Geospatial Messenger with Kotlin, Spring Boot and PostgreSQL](https://spring.io/blog/2016/03/20/a-geospatial-messenger-with-kotlin-spring-boot-and-postgresql)
* [Introducing Kotlin support in Spring Framework 5.0](https://spring.io/blog/2017/01/04/introducing-kotlin-support-in-spring-framework-5-0)
* [Spring Framework 5 Kotlin APIs, the functional way](https://spring.io/blog/2017/08/01/spring-framework-5-kotlin-apis-the-functional-way)

#### Examples

* [spring-boot-kotlin-demo](https://github.com/sdeleuze/spring-boot-kotlin-demo): regular Spring Boot + Spring Data JPA project
* [mixit](https://github.com/mixitconf/mixit): Spring Boot 2 + WebFlux + Reactive Spring Data MongoDB
* [spring-kotlin-fullstack](https://github.com/sdeleuze/spring-kotlin-fullstack): WebFlux Kotlin fullstack example with Kotlin2js for frontend instead of JavaScript or TypeScript
* [spring-petclinic-kotlin](https://github.com/spring-petclinic/spring-petclinic-kotlin): Kotlin version of the Spring PetClinic Sample Application
* [spring-kotlin-deepdive](https://github.com/sdeleuze/spring-kotlin-deepdive): a step by step migration for Boot 1.0 + Java to Boot 2.0 + Kotlin
* [spring-boot-coroutines-demo](https://github.com/sdeleuze/spring-boot-coroutines-demo): Coroutines sample project

## Building Container Images

Spring Boot applications can be containerized either by [packaging them into Docker images](#_bookmark373), or by [using Buildpacks to create docker compatible container images that you can run anywhere](#_bookmark376).

### Building Docker images

A typical Spring Boot fat jar can be converted into a Docker image by adding just a few lines to a Dockerfile that can be used to build the image. However, there are various downsides to copying and running the fat jar as is in the docker image. There’s always a certain amount of overhead when running a fat jar without unpacking it, and in a containerized environment this can be noticeable. The other issue is that putting your application’s code and all its dependencies in one layer in the Docker image is sub-optimal. Since you probably recompile your code more often than

you upgrade the version of Spring Boot you use, it’s often better to separate things a bit more. If you put jar files in the layer before your application classes, Docker often only needs to change the very bottom layer and can pick others up from its cache.

#### Layering Docker Images

To make it easier to create optimized Docker images that can be built with a dockerfile, Spring Boot supports adding a layer index file to the jar. It provides a list of layers and the parts of the jar that should be contained within them. The list of layers in the index is ordered based on the order in which the layers should be added to the Docker/OCI image. Out-of-the-box, the following layers are supported:

* dependencies (for regular released dependencies)
* spring-boot-loader (for everything under org/springframework/boot/loader)
* snapshot-dependencies (for snapshot dependencies)
* application (for application classes and resources)

The following shows an example of a layers.idx file:

* "dependencies":
  + BOOT-INF/lib/library1.jar
  + BOOT-INF/lib/library2.jar
* "spring-boot-loader":
  + org/springframework/boot/loader/JarLauncher.class
  + org/springframework/boot/loader/jar/JarEntry.class
* "snapshot-dependencies":
  + BOOT-INF/lib/library3-SNAPSHOT.jar
* "application":
  + META-INF/MANIFEST.MF
  + BOOT-INF/classes/a/b/C.class

This layering is designed to separate code based on how likely it is to change between application builds. Library code is less likely to change between builds, so it is placed in its own layers to allow tooling to re-use the layers from cache. Application code is more likely to change between builds so it is isolated in a separate layer.

For Maven, refer to the [packaging layered jars section](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/maven-plugin/reference/html/#repackage-layers) for more details on adding a layer index to the jar. For Gradle, refer to the [packaging layered jars section](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/gradle-plugin/reference/html/#packaging-layered-jars) of the Gradle plugin documentation.

#### Writing the Dockerfile

When you create a jar containing the layers index file, the spring-boot-jarmode-layertools jar will be added as a dependency to your jar. With this jar on the classpath, you can launch your application in a special mode which allows the bootstrap code to run something entirely different from your application, for example, something that extracts the layers.

The layertools mode can not be used with a [fully executable Spring Boot archive](#_bookmark483) that includes a launch script. Disable launch script configuration when building a jar file that is intended to be used with layertools.

Here’s how you can launch your jar with a layertools jar mode:

$ java -Djarmode=layertools -jar my-app.jar

This will provide the following output:

Usage:

java -Djarmode=layertools -jar my-app.jar

Available commands:

list List layers from the jar that can be extracted extract Extracts layers from the jar for image creation help Help about any command

The extract command can be used to easily split the application into layers to be added to the dockerfile. Here’s an example of a Dockerfile using jarmode.

FROM adoptopenjdk:11-jre-hotspot as builder WORKDIR application

ARG JAR\_FILE=target/\*.jar

COPY ${JAR\_FILE} application.jar

RUN java -Djarmode=layertools -jar application.jar extract

FROM adoptopenjdk:11-jre-hotspot WORKDIR application

COPY --from=builder application/dependencies/ ./

COPY --from=builder application/spring-boot-loader/ ./ COPY --from=builder application/snapshot-dependencies/ ./ COPY --from=builder application/application/ ./

ENTRYPOINT ["java", "org.springframework.boot.loader.JarLauncher"]

Assuming the above Dockerfile is in the current directory, your docker image can be built with docker build ., or optionally specifying the path to your application jar, as shown in the following example:

docker build --build-arg JAR\_FILE=path/to/myapp.jar .

This is a multi-stage dockerfile. The builder stage extracts the directories that are needed later. Each of the COPY commands relates to the layers extracted by the jarmode.

Of course, a Dockerfile can be written without using the jarmode. You can use some combination of

unzip and mv to move things to the right layer but jarmode simplifies that.

### Buildpacks

Dockerfiles are just one way to build docker images. Another way to build docker images is directly from your Maven or Gradle plugin, using buildpacks. If you’ve ever used an application platform such as Cloud Foundry or Heroku then you’ve probably used a buildpack. Buildpacks are the part of the platform that takes your application and converts it into something that the platform can actually run. For example, Cloud Foundry’s Java buildpack will notice that you’re pushing a .jar file and automatically add a relevant JRE.

With Cloud Native Buildpacks, you can create Docker compatible images that you can run anywhere. Spring Boot includes buildpack support directly for both Maven and Gradle. This means you can just type a single command and quickly get a sensible image into your locally running Docker daemon.

Refer to the individual plugin documentation on how to use buildpacks with [Maven](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/maven-plugin/reference/html/#build-image) and [Gradle](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/gradle-plugin/reference/html/#build-image).

## What to Read Next

If you want to learn more about any of the classes discussed in this section, you can check out the [Spring Boot API documentation](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/) or you can browse the [source code directly](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE). If you have specific questions, take a look at the [how-to](#_bookmark525) section.

If you are comfortable with Spring Boot’s core features, you can continue on and read about [production-ready features](#_bookmark378).

**Chapter 5. Spring Boot Actuator: Production- ready Features**

Spring Boot includes a number of additional features to help you monitor and manage your application when you push it to production. You can choose to manage and monitor your application by using HTTP endpoints or with JMX. Auditing, health, and metrics gathering can also be automatically applied to your application.

## Enabling Production-ready Features

The [spring-boot-actuator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator) module provides all of Spring Boot’s production-ready features. The recommended way to enable the features is to add a dependency on the spring-boot-starter- actuator ‘Starter’.

**Definition of Actuator**

An actuator is a manufacturing term that refers to a mechanical device for moving or controlling something. Actuators can generate a large amount of motion from a small change.

To add the actuator to a Maven based project, add the following ‘Starter’ dependency:

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

</dependencies>

For Gradle, use the following declaration:

dependencies {

implementation 'org.springframework.boot:spring-boot-starter-actuator'

}

## Endpoints

Actuator endpoints let you monitor and interact with your application. Spring Boot includes a number of built-in endpoints and lets you add your own. For example, the health endpoint provides basic application health information.

Each individual endpoint can be [enabled or disabled](#_bookmark381) and [exposed (made remotely accessible) over](#_bookmark382) [HTTP or JMX](#_bookmark382). An endpoint is considered to be available when it is both enabled and exposed. The built-in endpoints will only be auto-configured when they are available. Most applications choose

exposure via HTTP, where the ID of the endpoint along with a prefix of /actuator is mapped to a URL. For example, by default, the health endpoint is mapped to /actuator/health.

The following technology-agnostic endpoints are available:

|  |  |
| --- | --- |
| **ID** | **Description** |
| auditevents | Exposes audit events information for the current application. Requires an AuditEventRepository bean. |
| beans | Displays a complete list of all the Spring beans in your application. |
| caches | Exposes available caches. |
| conditions | Shows the conditions that were evaluated on configuration and auto- configuration classes and the reasons why they did or did not match. |
| configprops | Displays a collated list of all @ConfigurationProperties. |
| env | Exposes properties from Spring’s ConfigurableEnvironment. |
| flyway | Shows any Flyway database migrations that have been applied. Requires one or more Flyway beans. |
| health | Shows application health information. |
| httptrace | Displays HTTP trace information (by default, the last 100 HTTP request-response exchanges). Requires an HttpTraceRepository bean. |
| info | Displays arbitrary application info. |
| integrationgraph | Shows the Spring Integration graph. Requires a dependency on  spring-integration-core. |
| loggers | Shows and modifies the configuration of loggers in the application. |
| liquibase | Shows any Liquibase database migrations that have been applied. Requires one or more Liquibase beans. |
| metrics | Shows ‘metrics’ information for the current application. |
| mappings | Displays a collated list of all @RequestMapping paths. |
| scheduledtasks | Displays the scheduled tasks in your application. |
| sessions | Allows retrieval and deletion of user sessions from a Spring Session- backed session store. Requires a Servlet-based web application using Spring Session. |
| shutdown | Lets the application be gracefully shutdown. Disabled by default. |
| threaddump | Performs a thread dump. |

If your application is a web application (Spring MVC, Spring WebFlux, or Jersey), you can use the following additional endpoints:

|  |  |
| --- | --- |
| **ID** | **Description** |
| heapdump | Returns an hprof heap dump file. |

|  |  |
| --- | --- |
| **ID** | **Description** |
| jolokia | Exposes JMX beans over HTTP (when Jolokia is on the classpath, not available for WebFlux). Requires a dependency on jolokia-core. |
| logfile | Returns the contents of the logfile (if logging.file.name or logging.file.path properties have been set). Supports the use of the HTTP Range header to retrieve part of the log file’s content. |
| prometheus | Exposes metrics in a format that can be scraped by a Prometheus server. Requires a dependency on micrometer-registry-prometheus. |

To learn more about the Actuator’s endpoints and their request and response formats, please refer to the separate API documentation ([HTML](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/actuator-api/html/) or [PDF](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/actuator-api/pdf/spring-boot-actuator-web-api.pdf)).

### Enabling Endpoints

By default, all endpoints except for shutdown are enabled. To configure the enablement of an endpoint, use its management.endpoint.<id>.enabled property. The following example enables the shutdown endpoint:

management.endpoint.shutdown.enabled=true

If you prefer endpoint enablement to be opt-in rather than opt-out, set the management.endpoints.enabled-by-default property to false and use individual endpoint enabled properties to opt back in. The following example enables the info endpoint and disables all other endpoints:

management.endpoints.enabled-by-default=false management.endpoint.info.enabled=true

Disabled endpoints are removed entirely from the application context. If you want to change only the technologies over which an endpoint is exposed, use the [include](#_bookmark382) [and exclude properties](#_bookmark382) instead.

### Exposing Endpoints

Since Endpoints may contain sensitive information, careful consideration should be given about when to expose them. The following table shows the default exposure for the built-in endpoints:

|  |  |  |
| --- | --- | --- |
| **ID** | **JMX** | **Web** |
| auditevents | Yes | No |
| beans | Yes | No |
| caches | Yes | No |
| conditions | Yes | No |

|  |  |  |
| --- | --- | --- |
| **ID** | **JMX** | **Web** |
| configprops | Yes | No |
| env | Yes | No |
| flyway | Yes | No |
| health | Yes | Yes |
| heapdump | N/A | No |
| httptrace | Yes | No |
| info | Yes | Yes |
| integrationgraph | Yes | No |
| jolokia | N/A | No |
| logfile | N/A | No |
| loggers | Yes | No |
| liquibase | Yes | No |
| metrics | Yes | No |
| mappings | Yes | No |
| prometheus | N/A | No |
| scheduledtasks | Yes | No |
| sessions | Yes | No |
| shutdown | Yes | No |
| threaddump | Yes | No |

To change which endpoints are exposed, use the following technology-specific include and exclude

properties:

|  |  |
| --- | --- |
| **Property** | **Default** |
| management.endpoints.jmx.exposure.exclude |  |
| management.endpoints.jmx.exposure.include | \* |
| management.endpoints.web.exposure.exclude |  |
| management.endpoints.web.exposure.include | info, health |

The include property lists the IDs of the endpoints that are exposed. The exclude property lists the IDs of the endpoints that should not be exposed. The exclude property takes precedence over the include property. Both include and exclude properties can be configured with a list of endpoint IDs.

For example, to stop exposing all endpoints over JMX and only expose the health and info

endpoints, use the following property:

management.endpoints.jmx.exposure.include=health,info

\* can be used to select all endpoints. For example, to expose everything over HTTP except the env

and beans endpoints, use the following properties:

management.endpoints.web.exposure.include=\* management.endpoints.web.exposure.exclude=env,beans

\* has a special meaning in YAML, so be sure to add quotes if you want to include (or exclude) all endpoints, as shown in the following example:



management: endpoints:

web:

exposure: include: "\*"

If your application is exposed publicly, we strongly recommend that you also [secure your endpoints](#_bookmark383).

If you want to implement your own strategy for when endpoints are exposed, you can register an EndpointFilter bean.

### Securing HTTP Endpoints

You should take care to secure HTTP endpoints in the same way that you would any other sensitive URL. If Spring Security is present, endpoints are secured by default using Spring Security’s content- negotiation strategy. If you wish to configure custom security for HTTP endpoints, for example, only allow users with a certain role to access them, Spring Boot provides some convenient RequestMatcher objects that can be used in combination with Spring Security.

A typical Spring Security configuration might look something like the following example:

@Configuration(proxyBeanMethods = false)

public class ActuatorSecurity extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.requestMatcher(EndpointRequest.toAnyEndpoint()).authorizeRequests((requests) -> requests.anyRequest().hasRole("ENDPOINT\_ADMIN"));

http.httpBasic();

}

}

The preceding example uses EndpointRequest.toAnyEndpoint() to match a request to any endpoint

and then ensures that all have the ENDPOINT\_ADMIN role. Several other matcher methods are also available on EndpointRequest. See the API documentation ([HTML](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/actuator-api/html) or [PDF](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/actuator-api/pdf/spring-boot-actuator-web-api.pdf)) for details.

If you deploy applications behind a firewall, you may prefer that all your actuator endpoints can be accessed without requiring authentication. You can do so by changing the management.endpoints.web.exposure.include property, as follows:

*application.properties*

management.endpoints.web.exposure.include=\*

Additionally, if Spring Security is present, you would need to add custom security configuration that allows unauthenticated access to the endpoints as shown in the following example:

@Configuration(proxyBeanMethods = false)

public class ActuatorSecurity extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.requestMatcher(EndpointRequest.toAnyEndpoint()).authorizeRequests((requests) -> requests.anyRequest().permitAll());

}

}

### Configuring Endpoints

Endpoints automatically cache responses to read operations that do not take any parameters. To configure the amount of time for which an endpoint will cache a response, use its cache.time-to- live property. The following example sets the time-to-live of the beans endpoint’s cache to 10 seconds:

*application.properties*

management.endpoint.beans.cache.time-to-live=10s

The prefix management.endpoint.<name> is used to uniquely identify the endpoint that is being configured.

### Hypermedia for Actuator Web Endpoints

A “discovery page” is added with links to all the endpoints. The “discovery page” is available on

/actuator by default.

When a custom management context path is configured, the “discovery page” automatically moves from /actuator to the root of the management context. For example, if the management context

path is /management, then the discovery page is available from /management. When the management context path is set to /, the discovery page is disabled to prevent the possibility of a clash with other mappings.

### CORS Support

[Cross-origin resource sharing](https://en.wikipedia.org/wiki/Cross-origin_resource_sharing) (CORS) is a [W3C specification](https://www.w3.org/TR/cors/) that lets you specify in a flexible way what kind of cross-domain requests are authorized. If you use Spring MVC or Spring WebFlux, Actuator’s web endpoints can be configured to support such scenarios.

CORS support is disabled by default and is only enabled once the management.endpoints.web.cors.allowed-origins property has been set. The following configuration permits GET and POST calls from the example.com domain:

management.endpoints.web.cors.allowed-origins=https://example.com management.endpoints.web.cors.allowed-methods=GET,POST

 See [CorsEndpointProperties](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/endpoint/web/CorsEndpointProperties.java) for a complete list of options.

### Implementing Custom Endpoints

If you add a @Bean annotated with @Endpoint, any methods annotated with @ReadOperation, @WriteOperation, or @DeleteOperation are automatically exposed over JMX and, in a web application, over HTTP as well. Endpoints can be exposed over HTTP using Jersey, Spring MVC, or Spring WebFlux. If both Jersey and Spring MVC are available, Spring MVC will be used.

The following example exposes a read operation that returns a custom object:

@ReadOperation

public CustomData getCustomData() { return new CustomData("test", 5);

}

You can also write technology-specific endpoints by using @JmxEndpoint or @WebEndpoint. These endpoints are restricted to their respective technologies. For example, @WebEndpoint is exposed only over HTTP and not over JMX.

You can write technology-specific extensions by using @EndpointWebExtension and @EndpointJmxExtension. These annotations let you provide technology-specific operations to augment an existing endpoint.

Finally, if you need access to web-framework-specific functionality, you can implement Servlet or Spring @Controller and @RestController endpoints at the cost of them not being available over JMX or when using a different web framework.

#### Receiving Input

Operations on an endpoint receive input via their parameters. When exposed via the web, the values for these parameters are taken from the URL’s query parameters and from the JSON request body. When exposed via JMX, the parameters are mapped to the parameters of the MBean’s operations. Parameters are required by default. They can be made optional by annotating them with @org.springframework.lang.Nullable.

Each root property in the JSON request body can be mapped to a parameter of the endpoint. Consider the following JSON request body:

{

"name": "test", "counter": 42

}

This can be used to invoke a write operation that takes String name and int counter parameters, as shown in the following example:

@WriteOperation

public void updateCustomData(String name, int counter) {

// injects "test" and 42

}

Because endpoints are technology agnostic, only simple types can be specified in the method signature. In particular declaring a single parameter with a CustomData type defining a name and counter properties is not supported.

To allow the input to be mapped to the operation method’s parameters, Java code implementing an endpoint should be compiled with -parameters, and Kotlin code implementing an endpoint should be compiled with -java-parameters. This will happen automatically if you are using Spring Boot’s Gradle plugin or if you are using Maven and spring-boot-starter-parent.

**Input type conversion**

The parameters passed to endpoint operation methods are, if necessary, automatically converted to the required type. Before calling an operation method, the input received via JMX or an HTTP request is converted to the required types using an instance of ApplicationConversionService as well as any Converter or GenericConverter beans qualified with @EndpointConverter.

#### Custom Web Endpoints

Operations on an @Endpoint, @WebEndpoint, or @EndpointWebExtension are automatically exposed over HTTP using Jersey, Spring MVC, or Spring WebFlux. If both Jersey and Spring MVC are available, Spring MVC will be used.

**Web Endpoint Request Predicates**

A request predicate is automatically generated for each operation on a web-exposed endpoint.

**Path**

The path of the predicate is determined by the ID of the endpoint and the base path of web-exposed endpoints. The default base path is /actuator. For example, an endpoint with the ID sessions will use /actuator/sessions as its path in the predicate.

The path can be further customized by annotating one or more parameters of the operation method with @Selector. Such a parameter is added to the path predicate as a path variable. The variable’s value is passed into the operation method when the endpoint operation is invoked. If you want to capture all remaining path elements, you can add @Selector(Match=ALL\_REMAINING) to the last parameter and make it a type that is conversion compatible with a String[].

**HTTP method**

The HTTP method of the predicate is determined by the operation type, as shown in the following table:

|  |  |
| --- | --- |
| **Operation** | **HTTP method** |
| @ReadOperation | GET |
| @WriteOperation | POST |
| @DeleteOperation | DELETE |

**Consumes**

For a @WriteOperation (HTTP POST) that uses the request body, the consumes clause of the predicate is application/vnd.spring-boot.actuator.v2+json, application/json. For all other operations the consumes clause is empty.

**Produces**

The produces clause of the predicate can be determined by the produces attribute of the @DeleteOperation, @ReadOperation, and @WriteOperation annotations. The attribute is optional. If it is not used, the produces clause is determined automatically.

If the operation method returns void or Void the produces clause is empty. If the operation method returns a org.springframework.core.io.Resource, the produces clause is application/octet-stream. For all other operations the produces clause is application/vnd.spring-boot.actuator.v2+json, application/json.

**Web Endpoint Response Status**

The default response status for an endpoint operation depends on the operation type (read, write, or delete) and what, if anything, the operation returns.

A @ReadOperation returns a value, the response status will be 200 (OK). If it does not return a value, the response status will be 404 (Not Found).

If a @WriteOperation or @DeleteOperation returns a value, the response status will be 200 (OK). If it does not return a value the response status will be 204 (No Content).

If an operation is invoked without a required parameter, or with a parameter that cannot be converted to the required type, the operation method will not be called and the response status will be 400 (Bad Request).

**Web Endpoint Range Requests**

An HTTP range request can be used to request part of an HTTP resource. When using Spring MVC or Spring Web Flux, operations that return a org.springframework.core.io.Resource automatically support range requests.

 Range requests are not supported when using Jersey.

**Web Endpoint Security**

An operation on a web endpoint or a web-specific endpoint extension can receive the current java.security.Principal or org.springframework.boot.actuate.endpoint.SecurityContext as a method parameter. The former is typically used in conjunction with @Nullable to provide different behavior for authenticated and unauthenticated users. The latter is typically used to perform authorization checks using its isUserInRole(String) method.

#### Servlet endpoints

A Servlet can be exposed as an endpoint by implementing a class annotated with @ServletEndpoint that also implements Supplier<EndpointServlet>. Servlet endpoints provide deeper integration with the Servlet container but at the expense of portability. They are intended to be used to expose an existing Servlet as an endpoint. For new endpoints, the @Endpoint and @WebEndpoint annotations should be preferred whenever possible.

#### Controller endpoints

@ControllerEndpoint and @RestControllerEndpoint can be used to implement an endpoint that is only exposed by Spring MVC or Spring WebFlux. Methods are mapped using the standard annotations for Spring MVC and Spring WebFlux such as @RequestMapping and @GetMapping, with the endpoint’s ID being used as a prefix for the path. Controller endpoints provide deeper integration with Spring’s web frameworks but at the expense of portability. The @Endpoint and @WebEndpoint annotations should be preferred whenever possible.

### Health Information

You can use health information to check the status of your running application. It is often used by monitoring software to alert someone when a production system goes down. The information exposed by the health endpoint depends on the management.endpoint.health.show-details and management.endpoint.health.show-components properties which can be configured with one of the following values:

|  |  |
| --- | --- |
| **Name** | **Description** |
| never | Details are never shown. |
| when-authorized | Details are only shown to authorized users. Authorized roles can be configured using management.endpoint.health.roles. |
| always | Details are shown to all users. |

The default value is never. A user is considered to be authorized when they are in one or more of the endpoint’s roles. If the endpoint has no configured roles (the default) all authenticated users are considered to be authorized. The roles can be configured using the management.endpoint.health.roles property.

If you have secured your application and wish to use always, your security configuration must permit access to the health endpoint for both authenticated and unauthenticated users.

Health information is collected from the content of a [HealthContributorRegistry](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/HealthContributorRegistry.java) (by default all [HealthContributor](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/HealthContributor.java) instances defined in your ApplicationContext). Spring Boot includes a number of auto-configured HealthContributors and you can also write your own.

A HealthContributor can either be a HealthIndicator or a CompositeHealthContributor. A HealthIndicator provides actual health information, including a Status. A CompositeHealthContributor provides a composite of other HealthContributors. Taken together, contributors form a tree structure to represent the overall system health.

By default, the final system health is derived by a StatusAggregator which sorts the statuses from each HealthIndicator based on an ordered list of statuses. The first status in the sorted list is used as the overall health status. If no HealthIndicator returns a status that is known to the StatusAggregator, an UNKNOWN status is used.

The HealthContributorRegistry can be used to register and unregister health indicators at runtime.

#### Auto-configured HealthIndicators

The following HealthIndicators are auto-configured by Spring Boot when appropriate. You can also enable/disable selected indicators by configuring management.health.key.enabled, with the key listed in the table below.

|  |  |  |
| --- | --- | --- |
| **Key** | **Name** | **Description** |
| cassandra | [CassandraHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/cassandra/CassandraHealthIndicator.java) | Checks that a Cassandra database is up. |
| couchbase | [CouchbaseHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/couchbase/CouchbaseHealthIndicator.java) | Checks that a Couchbase cluster is up. |
| datasource | [DataSourceHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/jdbc/DataSourceHealthIndicator.java) | Checks that a connection to DataSource can be obtained. |
| diskspace | [DiskSpaceHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/system/DiskSpaceHealthIndicator.java) | Checks for low disk space. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Name** | **Description** |
| elasticsearch | [ElasticsearchRestHealthIndicat](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/elasticsearch/ElasticsearchRestHealthIndicator.java) [or](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/elasticsearch/ElasticsearchRestHealthIndicator.java) | Checks that an Elasticsearch cluster is up. |
| hazelcast | [HazelcastHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/hazelcast/HazelcastHealthIndicator.java) | Checks that a Hazelcast server is up. |
| influxdb | [InfluxDbHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/influx/InfluxDbHealthIndicator.java) | Checks that an InfluxDB server is up. |
| jms | [JmsHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/jms/JmsHealthIndicator.java) | Checks that a JMS broker is up. |
| ldap | [LdapHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/ldap/LdapHealthIndicator.java) | Checks that an LDAP server is up. |
| mail | [MailHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/mail/MailHealthIndicator.java) | Checks that a mail server is up. |
| mongo | [MongoHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/mongo/MongoHealthIndicator.java) | Checks that a Mongo database is up. |
| neo4j | [Neo4jHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/neo4j/Neo4jHealthIndicator.java) | Checks that a Neo4j database is up. |
| ping | [PingHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/PingHealthIndicator.java) | Always responds with UP. |
| rabbit | [RabbitHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/amqp/RabbitHealthIndicator.java) | Checks that a Rabbit server is up. |
| redis | [RedisHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/redis/RedisHealthIndicator.java) | Checks that a Redis server is up. |
| solr | [SolrHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/solr/SolrHealthIndicator.java) | Checks that a Solr server is up. |

You can disable them all by setting the management.health.defaults.enabled

property.

Additional HealthIndicators are available but not enabled by default:

|  |  |  |
| --- | --- | --- |
| **Key** | **Name** | **Description** |
| livenessstate | [LivenessStateHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/availability/LivenessStateHealthIndicator.java) | Exposes the "Liveness" application availability state. |
| readinessstate | [ReadinessStateHealthIndicato](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/availability/ReadinessStateHealthIndicator.java) [r](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/availability/ReadinessStateHealthIndicator.java) | Exposes the "Readiness" application availability state. |

#### Writing Custom HealthIndicators

To provide custom health information, you can register Spring beans that implement the [HealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/HealthIndicator.java) interface. You need to provide an implementation of the health() method and return a Health response. The Health response should include a status and can optionally include additional details to be displayed. The following code shows a sample HealthIndicator implementation:

import org.springframework.boot.actuate.health.Health;

import org.springframework.boot.actuate.health.HealthIndicator; import org.springframework.stereotype.Component;

@Component

public class MyHealthIndicator implements HealthIndicator {

@Override

public Health health() {

int errorCode = check(); // perform some specific health check if (errorCode != 0) {

return Health.down().withDetail("Error Code", errorCode).build();

}

return Health.up().build();

}

}

The identifier for a given HealthIndicator is the name of the bean without the HealthIndicator suffix, if it exists. In the preceding example, the health information is available in an entry named my.

In addition to Spring Boot’s predefined [Status](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/Status.java) types, it is also possible for Health to return a custom Status that represents a new system state. In such cases, a custom implementation of the [StatusAggregator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/StatusAggregator.java) interface also needs to be provided, or the default implementation has to be configured by using the management.endpoint.health.status.order configuration property.

For example, assume a new Status with code FATAL is being used in one of your HealthIndicator implementations. To configure the severity order, add the following property to your application properties:

management.endpoint.health.status.order=fatal,down,out-of-service,unknown,up

The HTTP status code in the response reflects the overall health status. By default, OUT\_OF\_SERVICE and DOWN map to 503. Any unmapped health statuses, including UP, map to 200. You might also want to register custom status mappings if you access the health endpoint over HTTP. Configuring a custom mapping disables the defaults mappings for DOWN and OUT\_OF\_SERVICE. If you want to retain the default mappings they must be configured explicitly alongside any custom mappings. For example, the following property maps FATAL to 503 (service unavailable) and retains the default mappings for DOWN and OUT\_OF\_SERVICE:

management.endpoint.health.status.http-mapping.down=503 management.endpoint.health.status.http-mapping.fatal=503 management.endpoint.health.status.http-mapping.out-of-service=503

 If you need more control, you can define your own HttpCodeStatusMapper bean.

The following table shows the default status mappings for the built-in statuses:

|  |  |
| --- | --- |
| **Status** | **Mapping** |
| DOWN | SERVICE\_UNAVAILABLE (503) |
| OUT\_OF\_SERVICE | SERVICE\_UNAVAILABLE (503) |
| UP | No mapping by default, so http status is 200 |
| UNKNOWN | No mapping by default, so http status is 200 |

#### Reactive Health Indicators

For reactive applications, such as those using Spring WebFlux, ReactiveHealthContributor provides a non-blocking contract for getting application health. Similar to a traditional HealthContributor, health information is collected from the content of a [ReactiveHealthContributorRegistry](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/ReactiveHealthContributorRegistry.java) (by default all [HealthContributor](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/HealthContributor.java) and [ReactiveHealthContributor](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/ReactiveHealthContributor.java) instances defined in your ApplicationContext). Regular HealthContributors that do not check against a reactive API are executed on the elastic scheduler.

In a reactive application, The ReactiveHealthContributorRegistry should be used to register and unregister health indicators at runtime. If you need to register a regular HealthContributor, you should wrap it using ReactiveHealthContributor#adapt.

To provide custom health information from a reactive API, you can register Spring beans that implement the [ReactiveHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/health/ReactiveHealthIndicator.java) interface. The following code shows a sample ReactiveHealthIndicator implementation:

@Component

public class MyReactiveHealthIndicator implements ReactiveHealthIndicator {

@Override

public Mono<Health> health() {

return doHealthCheck() //perform some specific health check that returns a Mono<Health>

.onErrorResume(ex -> Mono.just(new Health.Builder().down(ex).build()));

}

}

To handle the error automatically, consider extending from

AbstractReactiveHealthIndicator.

#### Auto-configured ReactiveHealthIndicators

The following ReactiveHealthIndicators are auto-configured by Spring Boot when appropriate:

|  |  |
| --- | --- |
| **Name** | **Description** |
| [CassandraReactiveH](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/cassandra/CassandraReactiveHealthIndicator.java) [ealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/cassandra/CassandraReactiveHealthIndicator.java) | Checks that a Cassandra database is up. |
| [CouchbaseReactiveH](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/couchbase/CouchbaseReactiveHealthIndicator.java) [ealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/couchbase/CouchbaseReactiveHealthIndicator.java) | Checks that a Couchbase cluster is up. |
| [ElasticsearchReact](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/elasticsearch/ElasticsearchReactiveHealthIndicator.java) [iveHealthIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/elasticsearch/ElasticsearchReactiveHealthIndicator.java) | Checks that an Elasticsearch cluster is up. |
| [MongoReactiveHealt](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/mongo/MongoReactiveHealthIndicator.java) [hIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/mongo/MongoReactiveHealthIndicator.java) | Checks that a Mongo database is up. |
| [RedisReactiveHealt](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/redis/RedisReactiveHealthIndicator.java) [hIndicator](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/redis/RedisReactiveHealthIndicator.java) | Checks that a Redis server is up. |

If necessary, reactive indicators replace the regular ones. Also, any HealthIndicator

that is not handled explicitly is wrapped automatically.

#### Health Groups

It’s sometimes useful to organize health indicators into groups that can be used for different purposes.

To create a health indicator group you can use the management.endpoint.health.group.<name> property and specify a list of health indicator IDs to include or exclude. For example, to create a group that includes only database indicators you can define the following:

management.endpoint.health.group.custom.include=db

You can then check the result by hitting localhost:8080/actuator/health/custom.

By default groups will inherit the same StatusAggregator and HttpCodeStatusMapper settings as the system health, however, these can also be defined on a per-group basis. It’s also possible to override the show-details and roles properties if required:

management.endpoint.health.group.custom.show-details=when-authorized management.endpoint.health.group.custom.roles=admin management.endpoint.health.group.custom.status.order=fatal,up management.endpoint.health.group.custom.status.http-mapping.fatal=500 management.endpoint.health.group.custom.status.http-mapping.out-of-service=500

You can use @Qualifier("groupname") if you need to register custom

StatusAggregator or HttpCodeStatusMapper beans for use with the group.

### Kubernetes Probes

Applications deployed on Kubernetes can provide information about their internal state with [Container Probes](https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#container-probes). Depending on [your Kubernetes configuration](https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/), the kubelet will call those probes and react to the result.

Spring Boot manages your [Application Availability State](#_bookmark91) out-of-the-box. If deployed in a Kubernetes environment, actuator will gather the "Liveness" and "Readiness" information from the ApplicationAvailability interface and use that information in dedicated [Health Indicators](#_bookmark393): LivenessStateHealthIndicator and ReadinessStateHealthIndicator. These indicators will be shown on the global health endpoint ("/actuator/health"). They will also be exposed as separate HTTP Probes using [Health Groups](#_bookmark397): "/actuator/health/liveness" and "/actuator/health/readiness".

You can then configure your Kubernetes infrastructure with the following endpoint information:

livenessProbe: httpGet:

path: /actuator/health/liveness port: <actuator-port>

failureThreshold: ... periodSeconds: ...

readinessProbe: httpGet:

path: /actuator/health/readiness port: <actuator-port>

failureThreshold: ... periodSeconds: ...

<actuator-port> should be set to the port that the actuator endpoints are available on. It could be the main web server port, or a separate management port if the "management.server.port" property has been set.

These health groups are only enabled automatically if the application is [running in a Kubernetes](#_bookmark474) [environment](#_bookmark474). You can enable them in any environment using the management.endpoint.health.probes.enabled configuration property.

If an application takes longer to start than the configured liveness period, Kubernetes mention the "startupProbe" as a possible solution. The "startupProbe" is not necessarily needed here as the "readinessProbe" fails until all startup tasks are done, see [how Probes behave during the application lifecycle](#_bookmark400).

If your Actuator endpoints are deployed on a separate management context, be aware that endpoints are then not using the same web infrastructure (port, connection pools, framework components) as the main application. In this case, a probe check could be successful even if the main application does not work properly (for example, it cannot accept new connections).

#### Checking external state with Kubernetes Probes

Actuator configures the "liveness" and "readiness" probes as Health Groups; this means that all the [Health Groups features](#_bookmark397) are available for them. You can, for example, configure additional Health Indicators:

management.endpoint.health.group.readiness.include=readinessState,customCheck

By default, Spring Boot does not add other Health Indicators to these groups.

The “liveness” Probe should not depend on health checks for external systems. If the [Liveness State](#_bookmark92) [of an application](#_bookmark92) is broken, Kubernetes will try to solve that problem by restarting the application instance. This means that if an external system fails (e.g. a database, a Web API, an external cache), Kubernetes might restart all application instances and create cascading failures.

As for the “readiness” Probe, the choice of checking external systems must be made carefully by the application developers, i.e. Spring Boot does not include any additional health checks in the readiness probe. If the [Readiness State of an application instance](#_bookmark93) is unready, Kubernetes will not route traffic to that instance. Some external systems might not be shared by application instances, in which case they could quite naturally be included in a readiness probe. Other external systems might not be essential to the application (the application could have circuit breakers and fallbacks), in which case they definitely should not be included. Unfortunately, an external system that is shared by all application instances is common, and you have to make a judgement call: include it in the readiness probe and expect that the application is taken out of service when the external service is down, or leave it out and deal with failures higher up the stack, e.g. using a circuit breaker in the caller.

If all instances of an application are unready, a Kubernetes Service with type=ClusterIP or NodePort will not accept any incoming connections. There is no HTTP error response (503 etc.) since there is no connection. A Service with type=LoadBalancer might or might not accept connections, depending on the provider. A Service that has an explicit [Ingress](https://kubernetes.io/docs/concepts/services-networking/ingress/) will also respond in a way that depends on the implementation - the ingress service itself will have to decide how to handle the "connection refused" from downstream. HTTP 503 is quite likely in the case of both load balancer and ingress.

Also, if an application is using Kubernetes [autoscaling](https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/) it may react differently to applications being taken out of the load-balancer, depending on its autoscaler configuration.

#### Application lifecycle and Probes states

An important aspect of the Kubernetes Probes support is its consistency with the application lifecycle. Spring Boot publishes [Application Events during startup and shutdown](#_bookmark95).

When a Spring Boot application starts:

|  |  |  |  |
| --- | --- | --- | --- |
| **Application startup phase** | **Liveness State** | **Readiness State** | **Notes** |
| Starting | BROKEN | REFUSING\_TRAF FIC | Kubernetes checks the "liveness" Probe and restarts the application if it takes too long. |
| Started | CORRECT | REFUSING\_TRAF FIC | The application context is refreshed. The application performs startup tasks and does not receive traffic yet. |
| Ready | CORRECT | ACCEPTING\_TRA FFIC | Startup tasks are finished. The application is receiving traffic. |

When a Spring Boot application shuts down:

|  |  |  |  |
| --- | --- | --- | --- |
| **Application shutdown phase** | **Liveness State** | **Readiness State** | **Notes** |
| Running | live | ready | Shutdown has been requested. |
| Graceful shutdown | live | unready | If enabled, [graceful shutdown processes in-](#_bookmark181) [flight requests](#_bookmark181). |
| Shutdown complete | broken | unready | The application context is closed and the application cannot serve traffic. |

Check out the [Kubernetes container lifecycle section](#_bookmark475) for more information about Kubernetes deployment.

### Application Information

Application information exposes various information collected from all [InfoContributor](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/info/InfoContributor.java) beans defined in your ApplicationContext. Spring Boot includes a number of auto-configured InfoContributor beans, and you can write your own.

#### Auto-configured InfoContributors

The following InfoContributor beans are auto-configured by Spring Boot, when appropriate:

|  |  |
| --- | --- |
| **Name** | **Description** |
| [EnvironmentInfoCon](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/info/EnvironmentInfoContributor.java) [tributor](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/info/EnvironmentInfoContributor.java) | Exposes any key from the Environment under the info key. |
| [GitInfoContributor](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/info/GitInfoContributor.java) | Exposes git information if a git.properties file is available. |
| [BuildInfoContribut](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/info/BuildInfoContributor.java) [or](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/info/BuildInfoContributor.java) | Exposes build information if a META-INF/build-info.properties file is available. |

It is possible to disable them all by setting the management.info.defaults.enabled

property.

#### Custom Application Information

You can customize the data exposed by the info endpoint by setting info.\* Spring properties. All Environment properties under the info key are automatically exposed. For example, you could add the following settings to your application.properties file:

info.app.encoding=UTF-8 info.app.java.source=1.8 info.app.java.target=1.8

Rather than hardcoding those values, you could also [expand info properties at](#_bookmark533) [build time](#_bookmark533).

Assuming you use Maven, you could rewrite the preceding example as follows:

[info.app.encoding=@project.build.sourceEncoding@](mailto:info.app.encoding%3D@project.build.sourceEncoding@) [info.app.java.source=@java.version@](mailto:info.app.java.source%3D@java.version@) [info.app.java.target=@java.version@](mailto:info.app.java.target%3D@java.version@)

#### Git Commit Information

Another useful feature of the info endpoint is its ability to publish information about the state of your git source code repository when the project was built. If a GitProperties bean is available, the git.branch, git.commit.id, and git.commit.time properties are exposed.

A GitProperties bean is auto-configured if a git.properties file is available at the root of the classpath. See "[Generate git information](#_bookmark641)" for more details.

If you want to display the full git information (that is, the full content of git.properties), use the

management.info.git.mode property, as follows:

management.info.git.mode=full

#### Build Information

If a BuildProperties bean is available, the info endpoint can also publish information about your build. This happens if a META-INF/build-info.properties file is available in the classpath.

The Maven and Gradle plugins can both generate that file. See "[Generate build](#_bookmark640) [information](#_bookmark640)" for more details.

#### Writing Custom InfoContributors

To provide custom application information, you can register Spring beans that implement the

[InfoContributor](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator/src/main/java/org/springframework/boot/actuate/info/InfoContributor.java) interface.

The following example contributes an example entry with a single value:

import java.util.Collections;

import org.springframework.boot.actuate.info.Info;

import org.springframework.boot.actuate.info.InfoContributor; import org.springframework.stereotype.Component;

@Component

public class ExampleInfoContributor implements InfoContributor {

@Override

public void contribute(Info.Builder builder) { builder.withDetail("example",

Collections.singletonMap("key", "value"));

}

}

If you reach the info endpoint, you should see a response that contains the following additional entry:

{

"example": {

"key" : "value"

}

}

## Monitoring and Management over HTTP

If you are developing a web application, Spring Boot Actuator auto-configures all enabled endpoints to be exposed over HTTP. The default convention is to use the id of the endpoint with a prefix of /actuator as the URL path. For example, health is exposed as /actuator/health.

Actuator is supported natively with Spring MVC, Spring WebFlux, and Jersey. If both Jersey and Spring MVC are available, Spring MVC will be used.

Jackson is a required dependency in order to get the correct JSON responses as documented in the API documentation ([HTML](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/actuator-api/html) or [PDF](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/actuator-api/pdf/spring-boot-actuator-web-api.pdf)).

### Customizing the Management Endpoint Paths

Sometimes, it is useful to customize the prefix for the management endpoints. For example, your application might already use /actuator for another purpose. You can use the management.endpoints.web.base-path property to change the prefix for your management endpoint, as shown in the following example:

management.endpoints.web.base-path=/manage

The preceding application.properties example changes the endpoint from /actuator/{id} to

/manage/{id} (for example, /manage/info).

Unless the management port has been configured to [expose endpoints by using a](#_bookmark409) [different HTTP port](#_bookmark409), management.endpoints.web.base-path is relative to server.servlet.context-path. If management.server.port is configured, management.endpoints.web.base-path is relative to management.server.servlet.context-path.

If you want to map endpoints to a different path, you can use the management.endpoints.web.path- mapping property.

The following example remaps /actuator/health to /healthcheck:

*application.properties*

management.endpoints.web.base-path=/ management.endpoints.web.path-mapping.health=healthcheck

### Customizing the Management Server Port

Exposing management endpoints by using the default HTTP port is a sensible choice for cloud- based deployments. If, however, your application runs inside your own data center, you may prefer to expose endpoints by using a different HTTP port.

You can set the management.server.port property to change the HTTP port, as shown in the following example:

management.server.port=8081

On Cloud Foundry, applications only receive requests on port 8080 for both HTTP and TCP routing, by default. If you want to use a custom management port on Cloud Foundry, you will need to explicitly set up the application’s routes to forward traffic to the custom port.

### Configuring Management-specific SSL

When configured to use a custom port, the management server can also be configured with its own SSL by using the various management.server.ssl.\* properties. For example, doing so lets a management server be available over HTTP while the main application uses HTTPS, as shown in the following property settings:

server.port=8443 server.ssl.enabled=true

server.ssl.key-store=classpath:store.jks server.ssl.key-password=secret management.server.port=8080 management.server.ssl.enabled=false

Alternatively, both the main server and the management server can use SSL but with different key stores, as follows:

server.port=8443 server.ssl.enabled=true

server.ssl.key-store=classpath:main.jks server.ssl.key-password=secret management.server.port=8080 management.server.ssl.enabled=true

management.server.ssl.key-store=classpath:management.jks management.server.ssl.key-password=secret

### Customizing the Management Server Address

You can customize the address that the management endpoints are available on by setting the management.server.address property. Doing so can be useful if you want to listen only on an internal or ops-facing network or to listen only for connections from localhost.

You can listen on a different address only when the port differs from the main server port.

The following example application.properties does not allow remote management connections:

management.server.port=8081 management.server.address=127.0.0.1

### Disabling HTTP Endpoints

If you do not want to expose endpoints over HTTP, you can set the management port to -1, as shown in the following example:

management.server.port=-1

This can be achieved using the management.endpoints.web.exposure.exclude property as well, as shown in following example:

management.endpoints.web.exposure.exclude=\*

## Monitoring and Management over JMX

Java Management Extensions (JMX) provide a standard mechanism to monitor and manage applications. By default, this feature is not enabled and can be turned on by setting the configuration property spring.jmx.enabled to true. Spring Boot exposes management endpoints as JMX MBeans under the org.springframework.boot domain by default.

### Customizing MBean Names

The name of the MBean is usually generated from the id of the endpoint. For example, the health

endpoint is exposed as org.springframework.boot:type=Endpoint,name=Health.

If your application contains more than one Spring ApplicationContext, you may find that names clash. To solve this problem, you can set the spring.jmx.unique-names property to true so that MBean names are always unique.

You can also customize the JMX domain under which endpoints are exposed. The following settings show an example of doing so in application.properties:

spring.jmx.unique-names=true management.endpoints.jmx.domain=com.example.myapp

### Disabling JMX Endpoints

If you do not want to expose endpoints over JMX, you can set the

management.endpoints.jmx.exposure.exclude property to \*, as shown in the following example:

management.endpoints.jmx.exposure.exclude=\*

### Using Jolokia for JMX over HTTP

Jolokia is a JMX-HTTP bridge that provides an alternative method of accessing JMX beans. To use Jolokia, include a dependency to org.jolokia:jolokia-core. For example, with Maven, you would add the following dependency:

<dependency>

<groupId>org.jolokia</groupId>

<artifactId>jolokia-core</artifactId>

</dependency>

The Jolokia endpoint can then be exposed by adding jolokia or \* to the

management.endpoints.web.exposure.include property. You can then access it by using

/actuator/jolokia on your management HTTP server.

The Jolokia endpoint exposes Jolokia’s servlet as an actuator endpoint. As a result, it is specific to servlet environments such as Spring MVC and Jersey. The endpoint will not be available in a WebFlux application.

#### Customizing Jolokia

Jolokia has a number of settings that you would traditionally configure by setting servlet parameters. With Spring Boot, you can use your application.properties file. To do so, prefix the parameter with management.endpoint.jolokia.config., as shown in the following example:

management.endpoint.jolokia.config.debug=true

#### Disabling Jolokia

If you use Jolokia but do not want Spring Boot to configure it, set the

management.endpoint.jolokia.enabled property to false, as follows:

management.endpoint.jolokia.enabled=false

## Loggers

Spring Boot Actuator includes the ability to view and configure the log levels of your application at runtime. You can view either the entire list or an individual logger’s configuration, which is made up of both the explicitly configured logging level as well as the effective logging level given to it by the logging framework. These levels can be one of:

* TRACE
* DEBUG
* INFO
* WARN
* ERROR
* FATAL
* OFF
* null

null indicates that there is no explicit configuration.

### Configure a Logger

To configure a given logger, POST a partial entity to the resource’s URI, as shown in the following example:

{

"configuredLevel": "DEBUG"

}

To “reset” the specific level of the logger (and use the default configuration instead), you can pass a value of null as the configuredLevel.

## Metrics

Spring Boot Actuator provides dependency management and auto-configuration for [Micrometer](https://micrometer.io/), an application metrics facade that supports [numerous monitoring systems](https://micrometer.io/docs), including:

* [AppOptics](#_bookmark424)
* [Atlas](#_bookmark425)
* [Datadog](#_bookmark426)
* [Dynatrace](#_bookmark427)
* [Elastic](#_bookmark428)
* [Ganglia](#_bookmark429)
* [Graphite](#_bookmark430)
* [Humio](#_bookmark431)
* [Influx](#_bookmark432)
* [JMX](#_bookmark433)
* [KairosDB](#_bookmark434)
* [New Relic](#_bookmark435)
* [Prometheus](#_bookmark436)
* [SignalFx](#_bookmark437)
* [Simple (in-memory)](#_bookmark438)
* [Stackdriver](#_bookmark439)
* [StatsD](#_bookmark440)
* [Wavefront](#_bookmark441)

To learn more about Micrometer’s capabilities, please refer to its [reference](https://micrometer.io/docs) [documentation](https://micrometer.io/docs), in particular the [concepts section](https://micrometer.io/docs/concepts).

### Getting started

Spring Boot auto-configures a composite MeterRegistry and adds a registry to the composite for each of the supported implementations that it finds on the classpath. Having a dependency on micrometer-registry-{system} in your runtime classpath is enough for Spring Boot to configure the

registry.

Most registries share common features. For instance, you can disable a particular registry even if the Micrometer registry implementation is on the classpath. For example, to disable Datadog:

management.metrics.export.datadog.enabled=false

Spring Boot will also add any auto-configured registries to the global static composite registry on the Metrics class unless you explicitly tell it not to:

management.metrics.use-global-registry=false

You can register any number of MeterRegistryCustomizer beans to further configure the registry, such as applying common tags, before any meters are registered with the registry:

@Bean

MeterRegistryCustomizer<MeterRegistry> metricsCommonTags() {

return registry -> registry.config().commonTags("region", "us-east-1");

}

You can apply customizations to particular registry implementations by being more specific about the generic type:

@Bean

MeterRegistryCustomizer<GraphiteMeterRegistry> graphiteMetricsNamingConvention() { return registry -> registry.config().namingConvention(MY\_CUSTOM\_CONVENTION);

}

Spring Boot also [configures built-in instrumentation](#_bookmark442) that you can control via configuration or dedicated annotation markers.

### Supported monitoring systems

#### AppOptics

By default, the AppOptics registry pushes metrics to [api.appoptics.com/v1/measurements](https://api.appoptics.com/v1/measurements) periodically. To export metrics to SaaS [AppOptics](https://micrometer.io/docs/registry/appOptics), your API token must be provided:

management.metrics.export.appoptics.api-token=YOUR\_TOKEN

#### Atlas

By default, metrics are exported to [Atlas](https://micrometer.io/docs/registry/atlas) running on your local machine. The location of the [Atlas](https://github.com/Netflix/atlas) [server](https://github.com/Netflix/atlas) to use can be provided using:

management.metrics.export.atlas.uri=https://atlas.example.com:7101/api/v1/publish

#### Datadog

Datadog registry pushes metrics to [datadoghq](https://www.datadoghq.com/) periodically. To export metrics to [Datadog](https://micrometer.io/docs/registry/datadog), your API key must be provided:

management.metrics.export.datadog.api-key=YOUR\_KEY

You can also change the interval at which metrics are sent to Datadog:

management.metrics.export.datadog.step=30s

#### Dynatrace

Dynatrace registry pushes metrics to the configured URI periodically. To export metrics to [Dynatrace](https://micrometer.io/docs/registry/dynatrace), your API token, device ID, and URI must be provided:

management.metrics.export.dynatrace.api-token=YOUR\_TOKEN management.metrics.export.dynatrace.device-id=YOUR\_DEVICE\_ID management.metrics.export.dynatrace.uri=YOUR\_URI

You can also change the interval at which metrics are sent to Dynatrace:

management.metrics.export.dynatrace.step=30s

#### Elastic

By default, metrics are exported to [Elastic](https://micrometer.io/docs/registry/elastic) running on your local machine. The location of the Elastic server to use can be provided using the following property:

management.metrics.export.elastic.host=https://elastic.example.com:8086

#### Ganglia

By default, metrics are exported to [Ganglia](https://micrometer.io/docs/registry/ganglia) running on your local machine. The [Ganglia server](http://ganglia.sourceforge.net/) host and port to use can be provided using:

management.metrics.export.ganglia.host=ganglia.example.com management.metrics.export.ganglia.port=9649

#### Graphite

By default, metrics are exported to [Graphite](https://micrometer.io/docs/registry/graphite) running on your local machine. The [Graphite server](https://graphiteapp.org/) host and port to use can be provided using:

management.metrics.export.graphite.host=graphite.example.com management.metrics.export.graphite.port=9004

Micrometer provides a default HierarchicalNameMapper that governs how a dimensional meter id is [mapped to flat hierarchical names](https://micrometer.io/docs/registry/graphite#_hierarchical_name_mapping).

To take control over this behaviour, define your GraphiteMeterRegistry and supply your own HierarchicalNameMapper. An auto-configured GraphiteConfig and Clock beans are provided unless you define your own:

@Bean

public GraphiteMeterRegistry graphiteMeterRegistry(GraphiteConfig config, Clock clock)

{

return new GraphiteMeterRegistry(config, clock, MY\_HIERARCHICAL\_MAPPER);

}

#### Humio

By default, the Humio registry pushes metrics to [cloud.humio.com](https://cloud.humio.com/) periodically. To export metrics to SaaS [Humio](https://micrometer.io/docs/registry/humio), your API token must be provided:

management.metrics.export.humio.api-token=YOUR\_TOKEN

You should also configure one or more tags to identify the data source to which metrics will be pushed:

management.metrics.export.humio.tags.alpha=a management.metrics.export.humio.tags.bravo=b

#### Influx

By default, metrics are exported to [Influx](https://micrometer.io/docs/registry/influx) running on your local machine. The location of the [Influx](https://www.influxdata.com/) [server](https://www.influxdata.com/) to use can be provided using:

management.metrics.export.influx.uri=https://influx.example.com:8086

#### JMX

Micrometer provides a hierarchical mapping to [JMX](https://micrometer.io/docs/registry/jmx), primarily as a cheap and portable way to view

metrics locally. By default, metrics are exported to the metrics JMX domain. The domain to use can be provided using:

management.metrics.export.jmx.domain=com.example.app.metrics

Micrometer provides a default HierarchicalNameMapper that governs how a dimensional meter id is [mapped to flat hierarchical names](https://micrometer.io/docs/registry/jmx#_hierarchical_name_mapping).

To take control over this behaviour, define your JmxMeterRegistry and supply your own HierarchicalNameMapper. An auto-configured JmxConfig and Clock beans are provided unless you define your own:

@Bean

public JmxMeterRegistry jmxMeterRegistry(JmxConfig config, Clock clock) { return new JmxMeterRegistry(config, clock, MY\_HIERARCHICAL\_MAPPER);

}

#### KairosDB

By default, metrics are exported to [KairosDB](https://micrometer.io/docs/registry/kairos) running on your local machine. The location of the [KairosDB server](https://kairosdb.github.io/) to use can be provided using:

management.metrics.export.kairos.uri=https://kairosdb.example.com:8080/api/v1/datapoin ts

#### New Relic

New Relic registry pushes metrics to [New Relic](https://micrometer.io/docs/registry/new-relic) periodically. To export metrics to [New Relic](https://newrelic.com/), your API key and account id must be provided:

management.metrics.export.newrelic.api-key=YOUR\_KEY management.metrics.export.newrelic.account-id=YOUR\_ACCOUNT\_ID

You can also change the interval at which metrics are sent to New Relic:

management.metrics.export.newrelic.step=30s

By default, metrics are published via REST calls but it also possible to use the Java Agent API if you have it on the classpath:

management.metrics.export.newrelic.client-provider-type=insights-agent

Finally, you can take full control by defining your own NewRelicClientProvider bean.

#### Prometheus

[Prometheus](https://micrometer.io/docs/registry/prometheus) expects to scrape or poll individual app instances for metrics. Spring Boot provides an actuator endpoint available at /actuator/prometheus to present a [Prometheus scrape](https://prometheus.io/) with the appropriate format.

The endpoint is not available by default and must be exposed, see [exposing](#_bookmark382) [endpoints](#_bookmark382) for more details.

Here is an example scrape\_config to add to prometheus.yml:

scrape\_configs:

- job\_name: 'spring'

metrics\_path: '/actuator/prometheus' static\_configs:

- targets: ['HOST:PORT']

For ephemeral or batch jobs which may not exist long enough to be scraped, [Prometheus](https://github.com/prometheus/pushgateway) [Pushgateway](https://github.com/prometheus/pushgateway) support can be used to expose their metrics to Prometheus. To enable Prometheus Pushgateway support, add the following dependency to your project:

<dependency>

<groupId>io.prometheus</groupId>

<artifactId>simpleclient\_pushgateway</artifactId>

</dependency>

When the Prometheus Pushgateway dependency is present on the classpath and the management.metrics.export.prometheus.pushgateway.enabled property is set to true, a PrometheusPushGatewayManager bean is auto-configured. This manages the pushing of metrics to a Prometheus Pushgateway.

The PrometheusPushGatewayManager can be tuned using properties under management.metrics.export.prometheus.pushgateway. For advanced configuration, you can also provide your own PrometheusPushGatewayManager bean.

#### SignalFx

SignalFx registry pushes metrics to [SignalFx](https://micrometer.io/docs/registry/signalFx) periodically. To export metrics to [SignalFx](https://www.signalfx.com/), your access token must be provided:

management.metrics.export.signalfx.access-token=YOUR\_ACCESS\_TOKEN

You can also change the interval at which metrics are sent to SignalFx:

management.metrics.export.signalfx.step=30s

#### Simple

Micrometer ships with a simple, in-memory backend that is automatically used as a fallback if no other registry is configured. This allows you to see what metrics are collected in the [metrics](#_bookmark456) [endpoint](#_bookmark456).

The in-memory backend disables itself as soon as you’re using any of the other available backend. You can also disable it explicitly:

management.metrics.export.simple.enabled=false

#### Stackdriver

Stackdriver registry pushes metrics to [Stackdriver](https://cloud.google.com/stackdriver/) periodically. To export metrics to SaaS [Stackdriver](https://micrometer.io/docs/registry/stackdriver), your Google Cloud project id must be provided:

management.metrics.export.stackdriver.project-id=my-project

You can also change the interval at which metrics are sent to Stackdriver:

management.metrics.export.stackdriver.step=30s

#### StatsD

The StatsD registry pushes metrics over UDP to a StatsD agent eagerly. By default, metrics are exported to a [StatsD](https://micrometer.io/docs/registry/statsD) agent running on your local machine. The StatsD agent host and port to use can be provided using:

management.metrics.export.statsd.host=statsd.example.com management.metrics.export.statsd.port=9125

You can also change the StatsD line protocol to use (default to Datadog):

management.metrics.export.statsd.flavor=etsy

#### Wavefront

Wavefront registry pushes metrics to [Wavefront](https://micrometer.io/docs/registry/wavefront) periodically. If you are exporting metrics to [Wavefront](https://www.wavefront.com/) directly, your API token must be provided:

management.metrics.export.wavefront.api-token=YOUR\_API\_TOKEN

Alternatively, you may use a Wavefront sidecar or an internal proxy set up in your environment that forwards metrics data to the Wavefront API host:

management.metrics.export.wavefront.uri=proxy://localhost:2878

If publishing metrics to a Wavefront proxy (as described in [the documentation](https://docs.wavefront.com/proxies_installing.html)), the host must be in the proxy://HOST:PORT format.

You can also change the interval at which metrics are sent to Wavefront:

management.metrics.export.wavefront.step=30s

### Supported Metrics

Spring Boot registers the following core metrics when applicable:

* + - * JVM metrics, report utilization of:
        + Various memory and buffer pools
        + Statistics related to garbage collection
        + Threads utilization
        + Number of classes loaded/unloaded
      * CPU metrics
      * File descriptor metrics
      * Kafka consumer and producer metrics
      * Log4j2 metrics: record the number of events logged to Log4j2 at each level
      * Logback metrics: record the number of events logged to Logback at each level
      * Uptime metrics: report a gauge for uptime and a fixed gauge representing the application’s absolute start time
      * Tomcat metrics (server.tomcat.mbeanregistry.enabled must be set to true for all Tomcat metrics to be registered)
      * [Spring Integration](https://docs.spring.io/spring-integration/docs/5.3.2.RELEASE/reference/html/system-management.html#micrometer-integration) metrics

#### Spring MVC Metrics

Auto-configuration enables the instrumentation of requests handled by Spring MVC. When management.metrics.web.server.request.autotime.enabled is true, this instrumentation occurs for all requests. Alternatively, when set to false, you can enable instrumentation by adding @Timed to a request-handling method:

@RestController @Timed ①

public class MyController {

@GetMapping("/api/people")

@Timed(extraTags = { "region", "us-east-1" }) Ⓒ @Timed(value = "all.people", longTask = true) Ⓒ public List<Person> listPeople() { ... }

}

① A controller class to enable timings on every request handler in the controller.

Ⓒ A method to enable for an individual endpoint. This is not necessary if you have it on the class, but can be used to further customize the timer for this particular endpoint.

Ⓒ A method with longTask = true to enable a long task timer for the method. Long task timers require a separate metric name, and can be stacked with a short task timer.

By default, metrics are generated with the name, http.server.requests. The name can be customized by setting the management.metrics.web.server.request.metric-name property.

By default, Spring MVC-related metrics are tagged with the following information:

|  |  |
| --- | --- |
| **Tag** | **Description** |
| exception | Simple class name of any exception that was thrown while handling the request. |
| method | Request’s method (for example, GET or POST) |
| outcome | Request’s outcome based on the status code of the response. 1xx is INFORMATIONAL, 2xx is SUCCESS, 3xx is REDIRECTION, 4xx CLIENT\_ERROR,  and 5xx is SERVER\_ERROR |
| status | Response’s HTTP status code (for example, 200 or  500) |
| uri | Request’s URI template prior to variable substitution, if possible (for example,  /api/person/{id}) |

To add to the default tags, provide one or more @Beans that implement WebMvcTagsContributor. To replace the default tags, provide a @Bean that implements WebMvcTagsProvider.

#### Spring WebFlux Metrics

Auto-configuration enables the instrumentation of all requests handled by WebFlux controllers and functional handlers.

By default, metrics are generated with the name http.server.requests. You can customize the name by setting the management.metrics.web.server.request.metric-name property.

By default, WebFlux-related metrics are tagged with the following information:

|  |  |
| --- | --- |
| **Tag** | **Description** |
| exception | Simple class name of any exception that was thrown while handling the request. |
| method | Request’s method (for example, GET or POST) |
| outcome | Request’s outcome based on the status code of the response. 1xx is INFORMATIONAL, 2xx is SUCCESS, 3xx is REDIRECTION, 4xx CLIENT\_ERROR,  and 5xx is SERVER\_ERROR |
| status | Response’s HTTP status code (for example, 200 or  500) |
| uri | Request’s URI template prior to variable substitution, if possible (for example,  /api/person/{id}) |

To add to the default tags, provide one or more @Beans that implement WebFluxTagsContributor. To replace the default tags, provide a @Bean that implements WebFluxTagsProvider.

#### Jersey Server Metrics

When Micrometer’s micrometer-jersey2 module is on the classpath, auto-configuration enables the instrumentation of requests handled by the Jersey JAX-RS implementation. When management.metrics.web.server.request.autotime.enabled is true, this instrumentation occurs for all requests. Alternatively, when set to false, you can enable instrumentation by adding @Timed to a request-handling method:

@Component @Path("/api/people") @Timed ①

public class Endpoint { @GET

@Timed(extraTags = { "region", "us-east-1" }) Ⓒ @Timed(value = "all.people", longTask = true) Ⓒ public List<Person> listPeople() { ... }

}

① On a resource class to enable timings on every request handler in the resource.

Ⓒ On a method to enable for an individual endpoint. This is not necessary if you have it on the class, but can be used to further customize the timer for this particular endpoint.

Ⓒ On a method with longTask = true to enable a long task timer for the method. Long task timers require a separate metric name, and can be stacked with a short task timer.

By default, metrics are generated with the name, http.server.requests. The name can be customized by setting the management.metrics.web.server.request.metric-name property.

By default, Jersey server metrics are tagged with the following information:

|  |  |
| --- | --- |
| **Tag** | **Description** |
| exception | Simple class name of any exception that was thrown while handling the request. |
| method | Request’s method (for example, GET or POST) |
| outcome | Request’s outcome based on the status code of the response. 1xx is INFORMATIONAL, 2xx is SUCCESS, 3xx is REDIRECTION, 4xx CLIENT\_ERROR,  and 5xx is SERVER\_ERROR |
| status | Response’s HTTP status code (for example, 200 or  500) |
| uri | Request’s URI template prior to variable substitution, if possible (for example,  /api/person/{id}) |

To customize the tags, provide a @Bean that implements JerseyTagsProvider.

#### HTTP Client Metrics

Spring Boot Actuator manages the instrumentation of both RestTemplate and WebClient. For that, you have to get injected with an auto-configured builder and use it to create instances:

* + - * RestTemplateBuilder for RestTemplate
      * WebClient.Builder for WebClient

It is also possible to apply manually the customizers responsible for this instrumentation, namely

MetricsRestTemplateCustomizer and MetricsWebClientCustomizer.

By default, metrics are generated with the name, http.client.requests. The name can be customized by setting the management.metrics.web.client.request.metric-name property.

By default, metrics generated by an instrumented client are tagged with the following information:

|  |  |
| --- | --- |
| **Tag** | **Description** |
| clientName | Host portion of the URI |
| method | Request’s method (for example, GET or POST) |
| outcome | Request’s outcome based on the status code of the response. 1xx is INFORMATIONAL, 2xx is SUCCESS, 3xx is REDIRECTION, 4xx CLIENT\_ERROR,  and 5xx is SERVER\_ERROR, UNKNOWN otherwise |
| status | Response’s HTTP status code if available (for example, 200 or 500), or IO\_ERROR in case of I/O issues, CLIENT\_ERROR otherwise |

|  |  |
| --- | --- |
| **Tag** | **Description** |
| uri | Request’s URI template prior to variable substitution, if possible (for example,  /api/person/{id}) |

To customize the tags, and depending on your choice of client, you can provide a @Bean that implements RestTemplateExchangeTagsProvider or WebClientExchangeTagsProvider. There are convenience static functions in RestTemplateExchangeTags and WebClientExchangeTags.

#### Cache Metrics

Auto-configuration enables the instrumentation of all available Caches on startup with metrics prefixed with cache. Cache instrumentation is standardized for a basic set of metrics. Additional, cache-specific metrics are also available.

The following cache libraries are supported:

* + - * Caffeine
      * EhCache 2
      * Hazelcast
      * Any compliant JCache (JSR-107) implementation

Metrics are tagged by the name of the cache and by the name of the CacheManager that is derived from the bean name.

Only caches that are configured on startup are bound to the registry. For caches not defined in the cache’s configuration, e.g. caches created on-the-fly or programmatically after the startup phase, an explicit registration is required. A CacheMetricsRegistrar bean is made available to make that process easier.

#### DataSource Metrics

Auto-configuration enables the instrumentation of all available DataSource objects with metrics prefixed with jdbc.connections. Data source instrumentation results in gauges representing the currently active, idle, maximum allowed, and minimum allowed connections in the pool.

Metrics are also tagged by the name of the DataSource computed based on the bean name.

By default, Spring Boot provides metadata for all supported data sources; you can add additional DataSourcePoolMetadataProvider beans if your favorite data source isn’t supported out of the box. See DataSourcePoolMetadataProvidersConfiguration for examples.

Also, Hikari-specific metrics are exposed with a hikaricp prefix. Each metric is tagged by the name of the Pool (can be controlled with spring.datasource.name).

#### Hibernate Metrics

Auto-configuration enables the instrumentation of all available Hibernate EntityManagerFactory

instances that have statistics enabled with a metric named hibernate.

Metrics are also tagged by the name of the EntityManagerFactory that is derived from the bean name.

To enable statistics, the standard JPA property hibernate.generate\_statistics must be set to true. You can enable that on the auto-configured EntityManagerFactory as shown in the following example:

spring.jpa.properties.hibernate.generate\_statistics=true

#### RabbitMQ Metrics

Auto-configuration will enable the instrumentation of all available RabbitMQ connection factories with a metric named rabbitmq.

#### Kafka Metrics

Auto-configuration will register a MicrometerConsumerListener and MicrometerProducerListener for the auto-configured consumer factory and producer factory respectively. For more details refer to [Micrometer Native Metrics](https://docs.spring.io/spring-kafka/docs/2.5.6.RELEASE/reference/html/#micrometer-native) section of the Spring Kafka documentation.

### Registering custom metrics

To register custom metrics, inject MeterRegistry into your component, as shown in the following example:

class Dictionary {

private final List<String> words = new CopyOnWriteArrayList<>(); Dictionary(MeterRegistry registry) {

registry.gaugeCollectionSize("dictionary.size", Tags.empty(), this.words);

}

// …

}

If your metrics depend on other beans, it is recommended that you use a MeterBinder to register them, as shown in the following example:

@Bean

MeterBinder queueSize(Queue queue) {

return (registry) -> Gauge.builder("queueSize", queue::size).register(registry);

}

Using a MeterBinder ensures that the correct dependency relationships are set up and that the bean is available when the metric’s value is retrieved. By default, metrics from all MeterBinder beans will be automatically bound to the Spring-managed MeterRegistry. A MeterBinder implementation can also be useful if you find that you repeatedly instrument a suite of metrics across components or applications.

### Customizing individual metrics

If you need to apply customizations to specific Meter instances you can use the io.micrometer.core.instrument.config.MeterFilter interface. By default, all MeterFilter beans will be automatically applied to the micrometer MeterRegistry.Config.

For example, if you want to rename the mytag.region tag to mytag.area for all meter IDs beginning with com.example, you can do the following:

@Bean

public MeterFilter renameRegionTagMeterFilter() {

return MeterFilter.renameTag("com.example", "mytag.region", "mytag.area");

}

#### Common tags

Common tags are generally used for dimensional drill-down on the operating environment like host, instance, region, stack, etc. Commons tags are applied to all meters and can be configured as shown in the following example:

management.metrics.tags.region=us-east-1 management.metrics.tags.stack=prod

The example above adds region and stack tags to all meters with a value of us-east-1 and prod

respectively.

The order of common tags is important if you are using Graphite. As the order of common tags cannot be guaranteed using this approach, Graphite users are advised to define a custom MeterFilter instead.

#### Per-meter properties

In addition to MeterFilter beans, it’s also possible to apply a limited set of customization on a per- meter basis using properties. Per-meter customizations apply to any all meter IDs that start with the given name. For example, the following will disable any meters that have an ID starting with

example.remote

management.metrics.enable.example.remote=false

The following properties allow per-meter customization:

*Table 8. Per-meter customizations*

|  |  |
| --- | --- |
| **Property** | **Description** |
| management.metrics.enable | Whether to deny meters from emitting any metrics. |
| management.metrics.distribution.percentiles- histogram | Whether to publish a histogram suitable for computing aggregable (across dimension) percentile approximations. |
| management.metrics.distribution.minimum-  expected-value, management.metrics.distribution.maximum- expected-value | Publish less histogram buckets by clamping the range of expected values. |
| management.metrics.distribution.percentiles | Publish percentile values computed in your application |
| management.metrics.distribution.slo | Publish a cumulative histogram with buckets defined by your service-level objectives. |

For more details on concepts behind percentiles-histogram, percentiles and sla refer to the ["Histograms and percentiles" section](https://micrometer.io/docs/concepts#_histograms_and_percentiles) of the micrometer documentation.

### Metrics endpoint

Spring Boot provides a metrics endpoint that can be used diagnostically to examine the metrics collected by an application. The endpoint is not available by default and must be exposed, see [exposing endpoints](#_bookmark382) for more details.

Navigating to /actuator/metrics displays a list of available meter names. You can drill down to view information about a particular meter by providing its name as a selector, e.g.

/actuator/metrics/jvm.memory.max.

The name you use here should match the name used in the code, not the name after it has been naming-convention normalized for a monitoring system it is shipped to. In other words, if jvm.memory.max appears as jvm\_memory\_max in Prometheus because of its snake case naming convention, you should still use jvm.memory.max as the selector when inspecting the meter in the metrics endpoint.

You can also add any number of tag=KEY:VALUE query parameters to the end of the URL to dimensionally drill down on a meter, e.g. /actuator/metrics/jvm.memory.max?tag=area:nonheap.

The reported measurements are the *sum* of the statistics of all meters matching the meter name and any tags that have been applied. So in the example above, the returned "Value" statistic is the sum of the maximum memory footprints of "Code

 Cache", "Compressed Class Space", and "Metaspace" areas of the heap. If you only

wanted to see the maximum size for the "Metaspace", you could add an additional

tag=id:Metaspace, i.e.

/actuator/metrics/jvm.memory.max?tag=area:nonheap&tag=id:Metaspace.

## Auditing

Once Spring Security is in play, Spring Boot Actuator has a flexible audit framework that publishes events (by default, “authentication success”, “failure” and “access denied” exceptions). This feature can be very useful for reporting and for implementing a lock-out policy based on authentication failures.

Auditing can be enabled by providing a bean of type AuditEventRepository in your application’s configuration. For convenience, Spring Boot offers an InMemoryAuditEventRepository. InMemoryAuditEventRepository has limited capabilities and we recommend using it only for development environments. For production environments, consider creating your own alternative AuditEventRepository implementation.

### Custom Auditing

To customize published security events, you can provide your own implementations of

AbstractAuthenticationAuditListener and AbstractAuthorizationAuditListener.

You can also use the audit services for your own business events. To do so, either inject the AuditEventRepository bean into your own components and use that directly or publish an AuditApplicationEvent with the Spring ApplicationEventPublisher (by implementing ApplicationEventPublisherAware).

## HTTP Tracing

HTTP Tracing can be enabled by providing a bean of type HttpTraceRepository in your application’s configuration. For convenience, Spring Boot offers an InMemoryHttpTraceRepository that stores traces for the last 100 request-response exchanges, by default. InMemoryHttpTraceRepository is limited compared to other tracing solutions and we recommend using it only for development environments. For production environments, use of a production-ready tracing or observability solution, such as Zipkin or Spring Cloud Sleuth, is recommended. Alternatively, create your own HttpTraceRepository that meets your needs.

The httptrace endpoint can be used to obtain information about the request-response exchanges that are stored in the HttpTraceRepository.

### Custom HTTP tracing

To customize the items that are included in each trace, use the management.trace.http.include

configuration property. For advanced customization, consider registering your own

HttpExchangeTracer implementation.

## Process Monitoring

In the spring-boot module, you can find two classes to create files that are often useful for process monitoring:

* ApplicationPidFileWriter creates a file containing the application PID (by default, in the application directory with a file name of application.pid).
* WebServerPortFileWriter creates a file (or files) containing the ports of the running web server (by default, in the application directory with a file name of application.port).

By default, these writers are not activated, but you can enable:

* [By Extending Configuration](#_bookmark462)
* [Programmatically](#_bookmark463)

### Extending Configuration

In the META-INF/spring.factories file, you can activate the listener(s) that writes a PID file, as shown in the following example:

org.springframework.context.ApplicationListener=\ org.springframework.boot.context.ApplicationPidFileWriter,\ org.springframework.boot.web.context.WebServerPortFileWriter

### Programmatically

You can also activate a listener by invoking the SpringApplication.addListeners(…) method and passing the appropriate Writer object. This method also lets you customize the file name and path in the Writer constructor.

## Cloud Foundry Support

Spring Boot’s actuator module includes additional support that is activated when you deploy to a compatible Cloud Foundry instance. The /cloudfoundryapplication path provides an alternative secured route to all @Endpoint beans.

The extended support lets Cloud Foundry management UIs (such as the web application that you can use to view deployed applications) be augmented with Spring Boot actuator information. For example, an application status page may include full health information instead of the typical “running” or “stopped” status.

The /cloudfoundryapplication path is not directly accessible to regular users. In order to use the endpoint, a valid UAA token must be passed with the request.

### Disabling Extended Cloud Foundry Actuator Support

If you want to fully disable the /cloudfoundryapplication endpoints, you can add the following setting to your application.properties file:

*application.properties*

management.cloudfoundry.enabled=false

### Cloud Foundry Self-signed Certificates

By default, the security verification for /cloudfoundryapplication endpoints makes SSL calls to various Cloud Foundry services. If your Cloud Foundry UAA or Cloud Controller services use self- signed certificates, you need to set the following property:

*application.properties*

management.cloudfoundry.skip-ssl-validation=true

### Custom context path

If the server’s context-path has been configured to anything other than /, the Cloud Foundry endpoints will not be available at the root of the application. For example, if server.servlet.context-path=/app, Cloud Foundry endpoints will be available at

/app/cloudfoundryapplication/\*.

If you expect the Cloud Foundry endpoints to always be available at /cloudfoundryapplication/\*, regardless of the server’s context-path, you will need to explicitly configure that in your application. The configuration will differ depending on the web server in use. For Tomcat, the following configuration can be added:

@Bean

public TomcatServletWebServerFactory servletWebServerFactory() { return new TomcatServletWebServerFactory() {

@Override

protected void prepareContext(Host host, ServletContextInitializer[] initializers) {

super.prepareContext(host, initializers); StandardContext child = new StandardContext();

child.addLifecycleListener(new Tomcat.FixContextListener()); child.setPath("/cloudfoundryapplication"); ServletContainerInitializer initializer =

getServletContextInitializer(getContextPath()); child.addServletContainerInitializer(initializer, Collections.emptySet()); child.setCrossContext(true);

host.addChild(child);

}

};

}

private ServletContainerInitializer getServletContextInitializer(String contextPath) { return (c, context) -> {

Servlet servlet = new GenericServlet() {

@Override

public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException {

ServletContext context = req.getServletContext().getContext(contextPath);

context.getRequestDispatcher("/cloudfoundryapplication").forward(req,

res);

}

};

context.addServlet("cloudfoundry", servlet).addMapping("/\*");

};

}

## What to Read Next

You might want to read about graphing tools such as [Graphite](https://graphiteapp.org/).

Otherwise, you can continue on, to read about [‘deployment options’](#_bookmark469) or jump ahead for some in- depth information about Spring Boot’s [*build tool plugins*](#_bookmark510).

**Chapter 6. Deploying Spring Boot Applications**

Spring Boot’s flexible packaging options provide a great deal of choice when it comes to deploying your application. You can deploy Spring Boot applications to a variety of cloud platforms, to container images (such as Docker), or to virtual/real machines.

This section covers some of the more common deployment scenarios.

## Deploying to Containers

If you are running your application from a container, you can use an executable jar, but it is also often an advantage to explode it and run it in a different way. Certain PaaS implementations may also choose to unpack archives before they run. For example, Cloud Foundry operates this way. One way to run an unpacked archive is by starting the appropriate launcher, as follows:

$ jar -xf myapp.jar

$ java org.springframework.boot.loader.JarLauncher

This is actually slightly faster on startup (depending on the size of the jar) than running from an unexploded archive. At runtime you shouldn’t expect any differences.

Once you have unpacked the jar file, you can also get an extra boost to startup time by running the app with its "natural" main method instead of the JarLauncher. For example:

$ jar -xf myapp.jar

$ java -cp BOOT-INF/classes:BOOT-INF/lib/\* com.example.MyApplication

Using the JarLauncher over the application’s main method has the added benefit of a predictable classpath order. The jar contains a classpath.idx file which is used by the JarLauncher when constructing the classpath.

More efficient container images can also be created by [creating separate layers](#_bookmark373) for your dependencies and application classes and resources (which normally change more frequently).

## Deploying to the Cloud

Spring Boot’s executable jars are ready-made for most popular cloud PaaS (Platform-as-a-Service) providers. These providers tend to require that you “bring your own container”. They manage application processes (not Java applications specifically), so they need an intermediary layer that adapts *your* application to the *cloud’s* notion of a running process.

Two popular cloud providers, Heroku and Cloud Foundry, employ a “buildpack” approach. The buildpack wraps your deployed code in whatever is needed to *start* your application. It might be a

JDK and a call to java, an embedded web server, or a full-fledged application server. A buildpack is pluggable, but ideally you should be able to get by with as few customizations to it as possible. This reduces the footprint of functionality that is not under your control. It minimizes divergence between development and production environments.

Ideally, your application, like a Spring Boot executable jar, has everything that it needs to run packaged within it.

In this section, we look at what it takes to get the [application that we developed](#_bookmark26) in the “Getting Started” section up and running in the Cloud.

### Cloud Foundry

Cloud Foundry provides default buildpacks that come into play if no other buildpack is specified. The Cloud Foundry [Java buildpack](https://github.com/cloudfoundry/java-buildpack) has excellent support for Spring applications, including Spring Boot. You can deploy stand-alone executable jar applications as well as traditional .war packaged applications.

Once you have built your application (by using, for example, mvn clean package) and have [installed](https://docs.cloudfoundry.org/cf-cli/install-go-cli.html) [the cf command line tool](https://docs.cloudfoundry.org/cf-cli/install-go-cli.html), deploy your application by using the cf push command, substituting the path to your compiled .jar. Be sure to have [logged in with your](https://docs.cloudfoundry.org/cf-cli/getting-started.html#login) [cf command line client](https://docs.cloudfoundry.org/cf-cli/getting-started.html#login) before pushing an application. The following line shows using the cf push command to deploy an application:

$ cf push acloudyspringtime -p target/demo-0.0.1-SNAPSHOT.jar

In the preceding example, we substitute acloudyspringtime for whatever value you give cf as the name of your application.

See the [cf push documentation](https://docs.cloudfoundry.org/cf-cli/getting-started.html#push) for more options. If there is a Cloud Foundry [manifest.yml](https://docs.cloudfoundry.org/devguide/deploy-apps/manifest.html) file present in the same directory, it is considered.

At this point, cf starts uploading your application, producing output similar to the following example:

Uploading acloudyspringtime... OK

Preparing to start acloudyspringtime... OK

-----> Downloaded app package (8.9M)

-----> Java Buildpack Version: v3.12 (offline) | https://github.com/cloudfoundry/java- buildpack.git#6f25b7e

-----> Downloading Open Jdk JRE 1.8.0\_121 from https://java- buildpack.cloudfoundry.org/openjdk/trusty/x86\_64/openjdk-1.8.0\_121.tar.gz (found in cache)

Expanding Open Jdk JRE to .java-buildpack/open\_jdk\_jre (1.6s)

-----> Downloading Open JDK Like Memory Calculator 2.0.2\_RELEASE from https://java- buildpack.cloudfoundry.org/memory-calculator/trusty/x86\_64/memory-calculator- 2.0.2\_RELEASE.tar.gz (found in cache)

Memory Settings: -Xss349K -Xmx681574K -XX:MaxMetaspaceSize=104857K -Xms681574K

-XX:MetaspaceSize=104857K

-----> Downloading Container Certificate Trust Store 1.0.0\_RELEASE from https://java- buildpack.cloudfoundry.org/container-certificate-trust-store/container-certificate- trust-store-1.0.0\_RELEASE.jar (found in cache)

Adding certificates to .java- buildpack/container\_certificate\_trust\_store/truststore.jks (0.6s)

-----> Downloading Spring Auto Reconfiguration 1.10.0\_RELEASE from https://java- buildpack.cloudfoundry.org/auto-reconfiguration/auto-reconfiguration- 1.10.0\_RELEASE.jar (found in cache)

Checking status of app 'acloudyspringtime'...

0 of 1 instances running (1 starting)

...

0 of 1 instances running (1 starting)

...

1. of 1 instances running (1 starting)

...

1. of 1 instances running (1 running) App started

Congratulations! The application is now live!

Once your application is live, you can verify the status of the deployed application by using the cf apps command, as shown in the following example:

acloudyspringtime.cfapps.io

...

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| $ cf apps  Getting applications | in ... |  | | |
| OK |  |
| name | requested state | instances | memory | disk urls |
| ...  acloudyspringtime | started | 1/1 | 512M | 1G |

Once Cloud Foundry acknowledges that your application has been deployed, you should be able to find the application at the URI given. In the preceding example, you could find it at https://acloudyspringtime.cfapps.io/.

#### Binding to Services

By default, metadata about the running application as well as service connection information is exposed to the application as environment variables (for example: $VCAP\_SERVICES). This architecture decision is due to Cloud Foundry’s polyglot (any language and platform can be supported as a buildpack) nature. Process-scoped environment variables are language agnostic.

Environment variables do not always make for the easiest API, so Spring Boot automatically extracts them and flattens the data into properties that can be accessed through Spring’s Environment abstraction, as shown in the following example:

@Component

class MyBean implements EnvironmentAware { private String instanceId;

@Override

public void setEnvironment(Environment environment) {

this.instanceId = environment.getProperty("vcap.application.instance\_id");

}

// ...

}

All Cloud Foundry properties are prefixed with vcap. You can use vcap properties to access application information (such as the public URL of the application) and service information (such as database credentials). See the [‘CloudFoundryVcapEnvironmentPostProcessor’](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/cloud/CloudFoundryVcapEnvironmentPostProcessor.html) Javadoc for complete details.

 The [Java CFEnv](https://github.com/pivotal-cf/java-cfenv/) project is a better fit for tasks such as configuring a DataSource.

### Kubernetes

Spring Boot auto-detects Kubernetes deployment environments by checking the environment for "\*\_SERVICE\_HOST" and "\*\_SERVICE\_PORT" variables. You can override this detection with the spring.main.cloud-platform configuration property.

Spring Boot helps you to [manage the state of your application](#_bookmark91) and export it with [HTTP Kubernetes](#_bookmark398) [Probes using Actuator](#_bookmark398).

#### Kubernetes Container Lifecycle

When Kubernetes deletes an application instance, the shutdown process involves several subsystems concurrently: shutdown hooks, unregistering the service, removing the instance from

the load-balancer… Because this shutdown processing happens in parallel (and due to the nature of distributed systems), there is a window during which traffic can be routed to a pod that has also begun its shutdown processing.

You can configure a sleep execution in a preStop handler to avoid requests being routed to a pod that has already begun shutting down. This sleep should be long enough for new requests to stop being routed to the pod and its duration will vary from deployment to deployment. The preStop handler can be configured via the PodSpec in the pod’s configuration file as follows:

spec:

containers:

- name: example-container image: example-image lifecycle:

preStop: exec:

command: ["sh", "-c", "sleep 10"]

Once the pre-stop hook has completed, SIGTERM will be sent to the container and [graceful](#_bookmark181) [shutdown](#_bookmark181) will begin, allowing any remaining in-flight requests to complete.

### Heroku

Heroku is another popular PaaS platform. To customize Heroku builds, you provide a Procfile, which provides the incantation required to deploy an application. Heroku assigns a port for the Java application to use and then ensures that routing to the external URI works.

You must configure your application to listen on the correct port. The following example shows the

Procfile for our starter REST application:

web: java -Dserver.port=$PORT -jar target/demo-0.0.1-SNAPSHOT.jar

Spring Boot makes -D arguments available as properties accessible from a Spring Environment instance. The server.port configuration property is fed to the embedded Tomcat, Jetty, or Undertow instance, which then uses the port when it starts up. The $PORT environment variable is assigned to us by the Heroku PaaS.

This should be everything you need. The most common deployment workflow for Heroku deployments is to git push the code to production, as shown in the following example:

$ git push heroku master

Initializing repository, done. Counting objects: 95, done.

Delta compression using up to 8 threads. Compressing objects: 100% (78/78), done.

Writing objects: 100% (95/95), 8.66 MiB | 606.00 KiB/s, done. Total 95 (delta 31), reused 0 (delta 0)

-----> Java app detected

-----> Installing OpenJDK 1.8... done

-----> Installing Maven 3.3.1... done

-----> Installing settings.xml... done

-----> Executing: mvn -B -DskipTests=true clean install

[INFO] Scanning for projects... Downloading: https://repo.spring.io/...

Downloaded: https://repo.spring.io/... (818 B at 1.8 KB/sec)

....

Downloaded: https://s3pository.heroku.com/jvm/... (152 KB at 595.3 KB/sec) [INFO] Installing /tmp/build\_0c35a5d2-a067-4abc-a232-14b1fb7a8229/target/... [INFO] Installing /tmp/build\_0c35a5d2-a067-4abc-a232-14b1fb7a8229/pom.xml ... [INFO] [INFO] BUILD SUCCESS

[INFO]

[INFO] Total time: 59.358s

[INFO] Finished at: Fri Mar 07 07:28:25 UTC 2014 [INFO] Final Memory: 20M/493M

[INFO]

-----> Discovering process types Procfile declares types -> web

-----> Compressing... done, 70.4MB

-----> Launching... done, v6

https://agile-sierra-1405.herokuapp.com/ deployed to Heroku

To git@heroku.com:agile-sierra-1405.git

\* [new branch] master -> master

Your application should now be up and running on Heroku. For more details, refer to [Deploying](https://devcenter.heroku.com/articles/deploying-spring-boot-apps-to-heroku) [Spring Boot Applications to Heroku](https://devcenter.heroku.com/articles/deploying-spring-boot-apps-to-heroku).

### OpenShift

[OpenShift](https://www.openshift.com/) has many resources describing how to deploy Spring Boot applications, including:

* + - * [Using the S2I builder](https://blog.openshift.com/using-openshift-enterprise-grade-spring-boot-deployments/)
      * [Architecture guide](https://access.redhat.com/documentation/en-us/reference_architectures/2017/html-single/spring_boot_microservices_on_red_hat_openshift_container_platform_3/)
      * [Running as a traditional web application on Wildfly](https://blog.openshift.com/using-spring-boot-on-openshift/)
      * [OpenShift Commons Briefing](https://blog.openshift.com/openshift-commons-briefing-96-cloud-native-applications-spring-rhoar/)

### Amazon Web Services (AWS)

Amazon Web Services offers multiple ways to install Spring Boot-based applications, either as traditional web applications (war) or as executable jar files with an embedded web server. The options include:

* + - * AWS Elastic Beanstalk
      * AWS Code Deploy
      * AWS OPS Works
      * AWS Cloud Formation
      * AWS Container Registry

Each has different features and pricing models. In this document, we describe to approach using AWS Elastic Beanstalk.

#### AWS Elastic Beanstalk

As described in the official [Elastic Beanstalk Java guide](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_Java.html), there are two main options to deploy a Java application. You can either use the “Tomcat Platform” or the “Java SE platform”.

**Using the Tomcat Platform**

This option applies to Spring Boot projects that produce a war file. No special configuration is required. You need only follow the official guide.

**Using the Java SE Platform**

This option applies to Spring Boot projects that produce a jar file and run an embedded web container. Elastic Beanstalk environments run an nginx instance on port 80 to proxy the actual application, running on port 5000. To configure it, add the following line to your application.properties file:

server.port=5000

*Upload binaries instead of sources*

By default, Elastic Beanstalk uploads sources and compiles them in AWS. However, it is best to upload the binaries instead. To do so, add lines similar to the following

 to your .elasticbeanstalk/config.yml file:

deploy:

artifact: target/demo-0.0.1-SNAPSHOT.jar

*Reduce costs by setting the environment type*

By default an Elastic Beanstalk environment is load balanced. The load balancer has a significant cost. To avoid that cost, set the environment type to “Single instance”, as described in [the Amazon documentation](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/environments-create-wizard.html#environments-create-wizard-capacity). You can also create single instance environments by using the CLI and the following command:

eb create -s

#### Summary

This is one of the easiest ways to get to AWS, but there are more things to cover, such as how to integrate Elastic Beanstalk into any CI / CD tool, use the Elastic Beanstalk Maven plugin instead of the CLI, and others. There is a [blog post](https://exampledriven.wordpress.com/2017/01/09/spring-boot-aws-elastic-beanstalk-example/) covering these topics more in detail.

### Boxfuse and Amazon Web Services

[Boxfuse](https://boxfuse.com/) works by turning your Spring Boot executable jar or war into a minimal VM image that can be deployed unchanged either on VirtualBox or on AWS. Boxfuse comes with deep integration for Spring Boot and uses the information from your Spring Boot configuration file to automatically configure ports and health check URLs. Boxfuse leverages this information both for the images it produces as well as for all the resources it provisions (instances, security groups, elastic load balancers, and so on).

Once you have created a [Boxfuse account](https://console.boxfuse.com/), connected it to your AWS account, installed the latest version of the Boxfuse Client, and ensured that the application has been built by Maven or Gradle (by using, for example, mvn clean package), you can deploy your Spring Boot application to AWS with a command similar to the following:

$ boxfuse run myapp-1.0.jar -env=prod

See the [boxfuse run documentation](https://boxfuse.com/docs/commandline/run.html) for more options. If there is a [boxfuse.conf](https://boxfuse.com/docs/commandline/#configuration) file present in the current directory, it is considered.

By default, Boxfuse activates a Spring profile named boxfuse on startup. If your executable jar or war contains an [application-boxfuse.properties](https://boxfuse.com/docs/payloads/springboot.html#configuration) file, Boxfuse bases its configuration on the properties it contains.

At this point, boxfuse creates an image for your application, uploads it, and configures and starts the necessary resources on AWS, resulting in output similar to the following example:

Fusing Image for myapp-1.0.jar ...

Image fused in 00:06.838s (53937 K) -> axelfontaine/myapp:1.0 Creating axelfontaine/myapp ...

Pushing axelfontaine/myapp:1.0 ... Verifying axelfontaine/myapp:1.0 ... Creating Elastic IP ...

Mapping myapp-axelfontaine.boxfuse.io to 52.28.233.167 ...

Waiting for AWS to create an AMI for axelfontaine/myapp:1.0 in eu-central-1 (this may take up to 50 seconds) ...

AMI created in 00:23.557s -> ami-d23f38cf

Creating security group boxfuse-sg\_axelfontaine/myapp:1.0 ...

Launching t2.micro instance of axelfontaine/myapp:1.0 (ami-d23f38cf) in eu-central-1

...

Instance launched in 00:30.306s -> i-92ef9f53

Waiting for AWS to boot Instance i-92ef9f53 and Payload to start at https://52.28.235.61/ ...

Payload started in 00:29.266s -> https://52.28.235.61/ Remapping Elastic IP 52.28.233.167 to i-92ef9f53 ...

Waiting 15s for AWS to complete Elastic IP Zero Downtime transition ... Deployment completed successfully. axelfontaine/myapp:1.0 is up and running at https://myapp-axelfontaine.boxfuse.io/

Your application should now be up and running on AWS.

See the blog post on [deploying Spring Boot apps on EC2](https://boxfuse.com/blog/spring-boot-ec2.html) as well as the [documentation for the](https://boxfuse.com/docs/payloads/springboot.html) [Boxfuse Spring Boot integration](https://boxfuse.com/docs/payloads/springboot.html) to get started with a Maven build to run the app.

### Google Cloud

Google Cloud has several options that can be used to launch Spring Boot applications. The easiest to get started with is probably App Engine, but you could also find ways to run Spring Boot in a container with Container Engine or on a virtual machine with Compute Engine.

To run in App Engine, you can create a project in the UI first, which sets up a unique identifier for you and also sets up HTTP routes. Add a Java app to the project and leave it empty and then use the [Google Cloud SDK](https://cloud.google.com/sdk/install) to push your Spring Boot app into that slot from the command line or CI build.

App Engine Standard requires you to use WAR packaging. Follow [these steps](https://github.com/GoogleCloudPlatform/java-docs-samples/tree/master/appengine-java8/springboot-helloworld/README.md) to deploy App Engine Standard application to Google Cloud.

Alternatively, App Engine Flex requires you to create an app.yaml file to describe the resources your app requires. Normally, you put this file in src/main/appengine, and it should resemble the following file:

service: default

runtime: java env: flex

runtime\_config: jdk: openjdk8

handlers:

- url: /.\*

script: this field is required, but ignored

manual\_scaling: instances: 1

health\_check: enable\_health\_check: False

env\_variables:

ENCRYPT\_KEY: your\_encryption\_key\_here

You can deploy the app (for example, with a Maven plugin) by adding the project ID to the build configuration, as shown in the following example:

<plugin>

<groupId>com.google.cloud.tools</groupId>

<artifactId>appengine-maven-plugin</artifactId>

<version>1.3.0</version>

<configuration>

<project>myproject</project>

</configuration>

</plugin>

Then deploy with mvn appengine:deploy (if you need to authenticate first, the build fails).

## Installing Spring Boot Applications

In addition to running Spring Boot applications by using java -jar, it is also possible to make fully executable applications for Unix systems. A fully executable jar can be executed like any other executable binary or it can be [registered with](#_bookmark485) [init.d or](#_bookmark485) [systemd](#_bookmark485). This helps when installing and managing Spring Boot applications in common production environments.

Fully executable jars work by embedding an extra script at the front of the file. Currently, some tools do not accept this format, so you may not always be able to use this technique. For example, jar -xf may silently fail to extract a jar or war that has been made fully executable. It is recommended that you make your jar or war fully executable only if you intend to execute it directly, rather than running it with java -jar or deploying it to a servlet container.

A zip64-format jar file cannot be made fully executable. Attempting to do so will result in a jar file that is reported as corrupt when executed directly or with java

-jar. A standard-format jar file that contains one or more zip64-format nested jars

can be fully executable.

To create a ‘fully executable’ jar with Maven, use the following plugin configuration:

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<executable>true</executable>

</configuration>

</plugin>

The following example shows the equivalent Gradle configuration:

bootJar {

launchScript()

}

You can then run your application by typing ./my-application.jar (where my-application is the name of your artifact). The directory containing the jar is used as your application’s working directory.

### Supported Operating Systems

The default script supports most Linux distributions and is tested on CentOS and Ubuntu. Other platforms, such as OS X and FreeBSD, require the use of a custom embeddedLaunchScript.

### Unix/Linux Services

Spring Boot application can be easily started as Unix/Linux services by using either init.d or

systemd.

#### Installation as an init.d Service (System V)

If you configured Spring Boot’s Maven or Gradle plugin to generate a [fully executable jar](#_bookmark483), and you do not use a custom embeddedLaunchScript, your application can be used as an init.d service. To do

so, symlink the jar to init.d to support the standard start, stop, restart, and status commands. The script supports the following features:

* + - * Starts the services as the user that owns the jar file
      * Tracks the application’s PID by using /var/run/<appname>/<appname>.pid
      * Writes console logs to /var/log/<appname>.log

Assuming that you have a Spring Boot application installed in /var/myapp, to install a Spring Boot application as an init.d service, create a symlink, as follows:

$ sudo ln -s /var/myapp/myapp.jar /etc/init.d/myapp

Once installed, you can start and stop the service in the usual way. For example, on a Debian-based system, you could start it with the following command:

$ service myapp start

If your application fails to start, check the log file written to

/var/log/<appname>.log for errors.

You can also flag the application to start automatically by using your standard operating system tools. For example, on Debian, you could use the following command:

$ update-rc.d myapp defaults <priority>

**Securing an init.d Service**

The following is a set of guidelines on how to secure a Spring Boot application that runs as an init.d service. It is not intended to be an exhaustive list of everything that should be done to harden an application and the environment in which it runs.

When executed as root, as is the case when root is being used to start an init.d service, the default executable script runs the application as the user specified in the RUN\_AS\_USER environment variable. When the environment variable is not set, the user who owns the jar file is used instead. You should never run a Spring Boot application as root, so RUN\_AS\_USER should never be root and your application’s jar file should never be owned by root. Instead, create a specific user to run your application and set the RUN\_AS\_USER environment variable or use chown to make it the owner of the jar file, as shown in the following example:

$ chown bootapp:bootapp your-app.jar

In this case, the default executable script runs the application as the bootapp user.

To reduce the chances of the application’s user account being compromised, you should consider preventing it from using a login shell. For example, you can set the account’s shell to /usr/sbin/nologin.

You should also take steps to prevent the modification of your application’s jar file. Firstly, configure its permissions so that it cannot be written and can only be read or executed by its owner, as shown in the following example:

$ chmod 500 your-app.jar

Second, you should also take steps to limit the damage if your application or the account that’s running it is compromised. If an attacker does gain access, they could make the jar file writable and change its contents. One way to protect against this is to make it immutable by using chattr, as shown in the following example:

$ sudo chattr +i your-app.jar

This will prevent any user, including root, from modifying the jar.

If root is used to control the application’s service and you [use a](#_bookmark490) [.conf file](#_bookmark490) to customize its startup, the .conf file is read and evaluated by the root user. It should be secured accordingly. Use chmod so that the file can only be read by the owner and use chown to make root the owner, as shown in the following example:

$ chmod 400 your-app.conf

$ sudo chown root:root your-app.conf

#### Installation as a systemd Service

systemd is the successor of the System V init system and is now being used by many modern Linux distributions. Although you can continue to use init.d scripts with systemd, it is also possible to launch Spring Boot applications by using systemd ‘service’ scripts.

Assuming that you have a Spring Boot application installed in /var/myapp, to install a Spring Boot application as a systemd service, create a script named myapp.service and place it in

/etc/systemd/system directory. The following script offers an example:

[Unit] Description=myapp After=syslog.target

[Service] User=myapp

ExecStart=/var/myapp/myapp.jar SuccessExitStatus=143

[Install]

WantedBy=multi-user.target

Remember to change the Description, User, and ExecStart fields for your application.

The ExecStart field does not declare the script action command, which means that the run command is used by default.

Note that, unlike when running as an init.d service, the user that runs the application, the PID file, and the console log file are managed by systemd itself and therefore must be configured by using appropriate fields in the ‘service’ script. Consult the [service unit configuration man page](https://www.freedesktop.org/software/systemd/man/systemd.service.html) for more details.

To flag the application to start automatically on system boot, use the following command:

$ systemctl enable myapp.service

Refer to man systemctl for more details.

#### Customizing the Startup Script

The default embedded startup script written by the Maven or Gradle plugin can be customized in a number of ways. For most people, using the default script along with a few customizations is usually enough. If you find you cannot customize something that you need to, use the embeddedLaunchScript option to write your own file entirely.

**Customizing the Start Script when It Is Written**

It often makes sense to customize elements of the start script as it is written into the jar file. For example, init.d scripts can provide a “description”. Since you know the description up front (and it need not change), you may as well provide it when the jar is generated.

To customize written elements, use the embeddedLaunchScriptProperties option of the Spring Boot Maven plugin or the [properties property of the Spring Boot Gradle plugin’s launchScript](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/gradle-plugin/reference/html/#packaging-executable-configuring-launch-script).

The following property substitutions are supported with the default script:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **Gradle default** | | | | **Maven default** | | | |
| mode | The script mode. | auto | | | | auto | | | |
| initInfo Provides | The Provides section of “INIT INFO” | ${task.baseName} | | | | ${project.artifactId} | | | |
| initInfo Required Start | Required-Start section of “INIT INFO”. | $remote\_fs $syslog $network | | | | $remote\_fs $syslog $network | | | |
| initInfo Required Stop | Required-Stop section of “INIT INFO”. | $remote\_fs $syslog $network | | | | $remote\_fs $syslog $network | | | |
| initInfo DefaultS tart | Default-Start section of “INIT INFO”. | 2 3 4 5 | | | | 2 3 4 5 | | | |
| initInfo DefaultS top | Default-Stop section of “INIT INFO”. | 0 1 6 | | | | 0 1 6 | | | |
| initInfo ShortDes cription | Short-Description section of “INIT INFO”. | Single-line version of  ${project.description}  (falling back to  ${task.baseName}) | | | | ${project.name} | | | |
| initInfo Descript ion | Description section of “INIT INFO”. | ${project.description}  (falling back to  ${task.baseName}) | | | | ${project.description}  (falling back to  ${project.name}) | | | |
| initInfo Chkconfi g | chkconfig section of “INIT INFO” | 2345 |  | 99 | 01 | 2345 |  | 99 | 01 |
| confFold er | The default value for  CONF\_FOLDER | Folder containing the jar | | | | Folder containing the jar | | | |
| inlinedC onfScrip t | Reference to a file script that should be inlined in the default launch script. This can be used to set environmental variables such as JAVA\_OPTS before any external config files are loaded |  | | | |  | | | |
| logFolde r | Default value for LOG\_FOLDER. Only valid for an init.d service |  | | | |  | | | |
| logFilen ame | Default value for LOG\_FILENAME. Only valid for an init.d service |  | | | |  | | | |
| pidFolde r | Default value for PID\_FOLDER. Only valid for an init.d service |  | | | |  | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Gradle default** | **Maven default** |
| pidFilen ame | Default value for the name of the PID file in PID\_FOLDER. Only valid for an init.d service |  |  |
| useStart StopDaem on | Whether the start-stop- daemon command, when it’s available, should be used to control the process | true | true |
| stopWait Time | Default value for STOP\_WAIT\_TIME in seconds. Only valid for an init.d service | 60 | 60 |

**Customizing a Script When It Runs**

For items of the script that need to be customized *after* the jar has been written, you can use environment variables or a [config file](#_bookmark490).

The following environment properties are supported with the default script:

|  |  |
| --- | --- |
| **Variable** | **Description** |
| MODE | The “mode” of operation. The default depends on the way the jar was built but is usually auto (meaning it tries to guess if it is an init script by checking if it is a symlink in a directory called init.d). You can explicitly set it to service so that the stop|start|status|restart commands work or to run if you want to run the script in the foreground. |
| RUN\_AS\_USER | The user that will be used to run the application. When not set, the user that owns the jar file will be used. |
| USE\_START\_ST OP\_DAEMON | Whether the start-stop-daemon command, when it’s available, should be used to control the process. Defaults to true. |
| PID\_FOLDER | The root name of the pid folder (/var/run by default). |
| LOG\_FOLDER | The name of the folder in which to put log files (/var/log by default). |
| CONF\_FOLDER | The name of the folder from which to read .conf files (same folder as jar-file by default). |
| LOG\_FILENAME | The name of the log file in the LOG\_FOLDER (<appname>.log by default). |
| APP\_NAME | The name of the app. If the jar is run from a symlink, the script guesses the app name. If it is not a symlink or you want to explicitly set the app name, this can be useful. |
| RUN\_ARGS | The arguments to pass to the program (the Spring Boot app). |
| JAVA\_HOME | The location of the java executable is discovered by using the PATH by default, but you can set it explicitly if there is an executable file at $JAVA\_HOME/bin/java. |

|  |  |
| --- | --- |
| **Variable** | **Description** |
| JAVA\_OPTS | Options that are passed to the JVM when it is launched. |
| JARFILE | The explicit location of the jar file, in case the script is being used to launch a jar that it is not actually embedded. |
| DEBUG | If not empty, sets the -x flag on the shell process, allowing you to see the logic in the script. |
| STOP\_WAIT\_TI ME | The time in seconds to wait when stopping the application before forcing a shutdown (60 by default). |

The PID\_FOLDER, LOG\_FOLDER, and LOG\_FILENAME variables are only valid for an init.d service. For systemd, the equivalent customizations are made by using the ‘service’ script. See the [service unit configuration man page](https://www.freedesktop.org/software/systemd/man/systemd.service.html) for more details.

With the exception of JARFILE and APP\_NAME, the settings listed in the preceding section can be configured by using a .conf file. The file is expected to be next to the jar file and have the same name but suffixed with .conf rather than .jar. For example, a jar named /var/myapp/myapp.jar uses the configuration file named /var/myapp/myapp.conf, as shown in the following example:

*myapp.conf*

JAVA\_OPTS=-Xmx1024M

LOG\_FOLDER=/custom/log/folder

If you do not like having the config file next to the jar file, you can set a CONF\_FOLDER

environment variable to customize the location of the config file.

To learn about securing this file appropriately, see [the guidelines for securing an init.d service](#_bookmark487).

### Microsoft Windows Services

A Spring Boot application can be started as a Windows service by using [winsw](https://github.com/kohsuke/winsw).

A ([separately maintained sample](https://github.com/snicoll-scratches/spring-boot-daemon)) describes step-by-step how you can create a Windows service for your Spring Boot application.

## What to Read Next

Check out the [Cloud Foundry](https://www.cloudfoundry.org/), [Heroku](https://www.heroku.com/), [OpenShift](https://www.openshift.com/), and [Boxfuse](https://boxfuse.com/) web sites for more information about the kinds of features that a PaaS can offer. These are just four of the most popular Java PaaS providers. Since Spring Boot is so amenable to cloud-based deployment, you can freely consider other providers as well.

The next section goes on to cover the [*Spring Boot CLI*](#_bookmark493), or you can jump ahead to read about [*build*](#_bookmark510)[*tool plugins*](#_bookmark510).

**Chapter 7. Spring Boot CLI**

The Spring Boot CLI is a command line tool that you can use if you want to quickly develop a Spring application. It lets you run Groovy scripts, which means that you have a familiar Java-like syntax without so much boilerplate code. You can also bootstrap a new project or write your own command for it.

## Installing the CLI

The Spring Boot CLI (Command-Line Interface) can be installed manually by using SDKMAN! (the SDK Manager) or by using Homebrew or MacPorts if you are an OSX user. See [*Installing the Spring*](#_bookmark17)[*Boot CLI*](#_bookmark17) in the “Getting started” section for comprehensive installation instructions.

## Using the CLI

Once you have installed the CLI, you can run it by typing spring and pressing Enter at the command line. If you run spring without any arguments, a help screen is displayed, as follows:

$ spring

usage: spring [--help] [--version]

<command> [<args>] Available commands are:

run [options] <files> [--] [args] Run a spring groovy script

... more command help is shown here

You can type spring help to get more details about any of the supported commands, as shown in the following example:

$ spring help run

spring run - Run a spring groovy script

usage: spring run [options] <files> [--] [args] Option Description

------ -----------

--autoconfigure [Boolean] Add autoconfigure compiler

transformations (default: true)

--classpath, -cp Additional classpath entries

--no-guess-dependencies Do not attempt to guess dependencies

--no-guess-imports Do not attempt to guess imports

-q, --quiet Quiet logging

-v, --verbose Verbose logging of dependency resolution

--watch Watch the specified file for changes

The version command provides a quick way to check which version of Spring Boot you are using, as follows:

$ spring version

Spring CLI v2.3.4.RELEASE

### Running Applications with the CLI

You can compile and run Groovy source code by using the run command. The Spring Boot CLI is completely self-contained, so you do not need any external Groovy installation.

The following example shows a “hello world” web application written in Groovy:

*hello.groovy*

@RestController

class WebApplication {

@RequestMapping("/") String home() {

"Hello World!"

}

}

To compile and run the application, type the following command:

$ spring run hello.groovy

To pass command-line arguments to the application, use -- to separate the commands from the “spring” command arguments, as shown in the following example:

$ spring run hello.groovy server.port=9000

To set JVM command line arguments, you can use the JAVA\_OPTS environment variable, as shown in the following example:

$ JAVA\_OPTS=-Xmx1024m spring run hello.groovy

When setting JAVA\_OPTS on Microsoft Windows, make sure to quote the entire instruction, such as set "JAVA\_OPTS=-Xms256m -Xmx2048m". Doing so ensures the values are properly passed to the process.

#### Deduced “grab” Dependencies

Standard Groovy includes a @Grab annotation, which lets you declare dependencies on third-party libraries. This useful technique lets Groovy download jars in the same way as Maven or Gradle would but without requiring you to use a build tool.

Spring Boot extends this technique further and tries to deduce which libraries to “grab” based on your code. For example, since the WebApplication code shown previously uses @RestController annotations, Spring Boot grabs "Tomcat" and "Spring MVC".

The following items are used as “grab hints”:

|  |  |
| --- | --- |
| **Items** | **Grabs** |
| JdbcTemplate, NamedParameterJdbcTemplate, DataSource | JDBC Application. |
| @EnableJms | JMS Application. |
| @EnableCaching | Caching abstraction. |
| @Test | JUnit. |
| @EnableRabbit | RabbitMQ. |
| extends Specification | Spock test. |
| @EnableBatchProcessing | Spring Batch. |
| @MessageEndpoint @EnableIntegration | Spring Integration. |
| @Controller @RestController @EnableWebMvc | Spring MVC + Embedded Tomcat. |
| @EnableWebSecurity | Spring Security. |
| @EnableTransactionManagement | Spring Transaction Management. |

See subclasses of [CompilerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-cli/src/main/java/org/springframework/boot/cli/compiler/CompilerAutoConfiguration.java) in the Spring Boot CLI source code to understand exactly how customizations are applied.

#### Deduced “grab” Coordinates

Spring Boot extends Groovy’s standard @Grab support by letting you specify a dependency without a group or version (for example, @Grab('freemarker')). Doing so consults Spring Boot’s default dependency metadata to deduce the artifact’s group and version.

The default metadata is tied to the version of the CLI that you use. It changes only when you move to a new version of the CLI, putting you in control of when the versions of your dependencies may change. A table showing the dependencies and their versions that are included in the default metadata can be found in the [appendix](#_bookmark705).

#### Default Import Statements

To help reduce the size of your Groovy code, several import statements are automatically included. Notice how the preceding example refers to @Component, @RestController, and @RequestMapping without needing to use fully-qualified names or import statements.

Many Spring annotations work without using import statements. Try running your application to see what fails before adding imports.

#### Automatic Main Method

Unlike the equivalent Java application, you do not need to include a public static void main(String[] args) method with your Groovy scripts. A SpringApplication is automatically created, with your compiled code acting as the source.

#### Custom Dependency Management

By default, the CLI uses the dependency management declared in spring-boot-dependencies when resolving @Grab dependencies. Additional dependency management, which overrides the default dependency management, can be configured by using the @DependencyManagementBom annotation. The annotation’s value should specify the coordinates (groupId:artifactId:version) of one or more Maven BOMs.

For example, consider the following declaration:

@DependencyManagementBom("com.example.custom-bom:1.0.0")

The preceding declaration picks up custom-bom-1.0.0.pom in a Maven repository under

com/example/custom-versions/1.0.0/.

When you specify multiple BOMs, they are applied in the order in which you declare them, as shown in the following example:

@DependencyManagementBom(["com.example.custom-bom:1.0.0", "com.example.another-bom:1.0.0"])

The preceding example indicates that the dependency management in another-bom overrides the dependency management in custom-bom.

You can use @DependencyManagementBom anywhere that you can use @Grab. However, to ensure consistent ordering of the dependency management, you can use @DependencyManagementBom at most once in your application.

### Applications with Multiple Source Files

You can use “shell globbing” with all commands that accept file input. Doing so lets you use multiple files from a single directory, as shown in the following example:

$ spring run \*.groovy

### Packaging Your Application

You can use the jar command to package your application into a self-contained executable jar file, as shown in the following example:

$ spring jar my-app.jar \*.groovy

The resulting jar contains the classes produced by compiling the application and all of the application’s dependencies so that it can then be run by using java -jar. The jar file also contains entries from the application’s classpath. You can add and remove explicit paths to the jar by using

--include and --exclude. Both are comma-separated, and both accept prefixes, in the form of “+” and “-”, to signify that they should be removed from the defaults. The default includes are as follows:

public/\*\*, resources/\*\*, static/\*\*, templates/\*\*, META-INF/\*\*, \*

The default excludes are as follows:

.\*, repository/\*\*, build/\*\*, target/\*\*, \*\*/\*.jar, \*\*/\*.groovy

Type spring help jar on the command line for more information.

### Initialize a New Project

The init command lets you create a new project by using [start.spring.io](https://start.spring.io/) without leaving the shell, as shown in the following example:

$ spring init --dependencies=web,data-jpa my-project Using service at https://start.spring.io

Project extracted to '/Users/developer/example/my-project'

The preceding example creates a my-project directory with a Maven-based project that uses spring- boot-starter-web and spring-boot-starter-data-jpa. You can list the capabilities of the service by using the --list flag, as shown in the following example:

$ spring init --list

=======================================

Capabilities of https://start.spring.io

=======================================

Available dependencies:

actuator - Actuator: Production ready features to help you monitor and manage your application

...

web - Web: Support for full-stack web development, including Tomcat and spring-webmvc websocket - Websocket: Support for WebSocket development

ws - WS: Support for Spring Web Services Available project types:

gradle-build - Gradle Config [format:build, build:gradle] gradle-project - Gradle Project [format:project, build:gradle] maven-build - Maven POM [format:build, build:maven]

maven-project - Maven Project [format:project, build:maven] (default)

...

The init command supports many options. See the help output for more details. For instance, the following command creates a Gradle project that uses Java 8 and war packaging:

$ spring init --build=gradle --java-version=1.8 --dependencies=websocket

--packaging=war sample-app.zip

Using service at https://start.spring.io Content saved to 'sample-app.zip'

### Using the Embedded Shell

Spring Boot includes command-line completion scripts for the BASH and zsh shells. If you do not use either of these shells (perhaps you are a Windows user), you can use the shell command to launch an integrated shell, as shown in the following example:

$ spring shell

Spring Boot (v2.3.4.RELEASE)

Hit TAB to complete. Type \'help' and hit RETURN for help, and \'exit' to quit.

From inside the embedded shell, you can run other commands directly:

$ version

Spring CLI v2.3.4.RELEASE

The embedded shell supports ANSI color output as well as tab completion. If you need to run a native command, you can use the ! prefix. To exit the embedded shell, press ctrl-c.

### Adding Extensions to the CLI

You can add extensions to the CLI by using the install command. The command takes one or more sets of artifact coordinates in the format group:artifact:version, as shown in the following example:

$ spring install com.example:spring-boot-cli-extension:1.0.0.RELEASE

In addition to installing the artifacts identified by the coordinates you supply, all of the artifacts' dependencies are also installed.

To uninstall a dependency, use the uninstall command. As with the install command, it takes one or more sets of artifact coordinates in the format of group:artifact:version, as shown in the following example:

$ spring uninstall com.example:spring-boot-cli-extension:1.0.0.RELEASE

It uninstalls the artifacts identified by the coordinates you supply and their dependencies.

To uninstall all additional dependencies, you can use the --all option, as shown in the following example:

$ spring uninstall --all

## Developing Applications with the Groovy Beans DSL

Spring Framework 4.0 has native support for a beans{} “DSL” (borrowed from [Grails](https://grails.org/)), and you can embed bean definitions in your Groovy application scripts by using the same format. This is sometimes a good way to include external features like middleware declarations, as shown in the following example:

@Configuration(proxyBeanMethods = false)

class Application implements CommandLineRunner {

@Autowired SharedService service

@Override

void run(String... args) { println service.message

}

}

import my.company.SharedService beans {

service(SharedService) { message = "Hello World"

}

}

You can mix class declarations with beans{} in the same file as long as they stay at the top level, or, if you prefer, you can put the beans DSL in a separate file.

## Configuring the CLI with settings.xml

The Spring Boot CLI uses Aether, Maven’s dependency resolution engine, to resolve dependencies. The CLI makes use of the Maven configuration found in ~/.m2/settings.xml to configure Aether. The following configuration settings are honored by the CLI:

* Offline
* Mirrors
* Servers
* Proxies
* Profiles
  + Activation
  + Repositories
* Active profiles

See [Maven’s settings documentation](https://maven.apache.org/settings.html) for further information.

## What to Read Next

There are some [sample groovy scripts](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-cli/samples) available from the GitHub repository that you can use to try out the Spring Boot CLI. There is also extensive Javadoc throughout the [source code](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-cli/src/main/java/org/springframework/boot/cli).

If you find that you reach the limit of the CLI tool, you probably want to look at converting your application to a full Gradle or Maven built “Groovy project”. The next section covers Spring Boot’s "[Build tool plugins](#_bookmark510)", which you can use with Gradle or Maven.

**Chapter 8. Build Tool Plugins**

Spring Boot provides build tool plugins for Maven and Gradle. The plugins offer a variety of features, including the packaging of executable jars. This section provides more details on both plugins as well as some help should you need to extend an unsupported build system. If you are just getting started, you might want to read “[Build Systems](#_bookmark37)” from the “[Using Spring Boot](#_bookmark36)” section first.

## Spring Boot Maven Plugin

The Spring Boot Maven Plugin provides Spring Boot support in Maven, letting you package executable jar or war archives and run an application “in-place”. To use it, you must use Maven 3.2 (or later).

Please refer to the plugin’s documentation to learn more:

* + - Reference ([HTML](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/maven-plugin/reference/html/) and [PDF](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/maven-plugin/reference/pdf/spring-boot-maven-plugin-reference.pdf))
    - [API](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/maven-plugin/api/)

## Spring Boot Gradle Plugin

The Spring Boot Gradle Plugin provides Spring Boot support in Gradle, letting you package executable jar or war archives, run Spring Boot applications, and use the dependency management provided by spring-boot-dependencies. It requires Gradle 6 (6.3 or later). Gradle 5.6.x is also supported but this support is deprecated and will be removed in a future release. Please refer to the plugin’s documentation to learn more:

* + - Reference ([HTML](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/gradle-plugin/reference/html/) and [PDF](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/gradle-plugin/reference/pdf/spring-boot-gradle-plugin-reference.pdf))
    - [API](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/gradle-plugin/api/)

## Spring Boot AntLib Module

The Spring Boot AntLib module provides basic Spring Boot support for Apache Ant. You can use the module to create executable jars. To use the module, you need to declare an additional spring-boot namespace in your build.xml, as shown in the following example:

<project xmlns:ivy="antlib:org.apache.ivy.ant"

xmlns:spring-boot="antlib:org.springframework.boot.ant" name="myapp" default="build">

...

</project>

You need to remember to start Ant using the -lib option, as shown in the following example:

$ ant -lib <directory containing spring-boot-antlib-2.3.4.RELEASE.jar>

The “Using Spring Boot” section includes a more complete example of [using](#_bookmark41) [Apache Ant with spring-boot-antlib](#_bookmark41).

### Spring Boot Ant Tasks

Once the spring-boot-antlib namespace has been declared, the following additional tasks are available:

* + - * [Using the “exejar” Task](#_bookmark515)
      * [Using the “findmainclass” Task](#_bookmark517)

#### Using the “exejar” Task

You can use the exejar task to create a Spring Boot executable jar. The following attributes are supported by the task:

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Required** |
| destfile | The destination jar file to create | Yes |
| classes | The root directory of Java class files | Yes |
| start-class | The main application class to run | No *(the default is the first class found that declares a* main *method)* |

The following nested elements can be used with the task:

|  |  |
| --- | --- |
| **Element** | **Description** |
| resources | One or more [Resource Collections](https://ant.apache.org/manual/Types/resources.html#collection) describing a set of [Resources](https://ant.apache.org/manual/Types/resources.html) that should be added to the content of the created jar file. |
| lib | One or more [Resource Collections](https://ant.apache.org/manual/Types/resources.html#collection) that should be added to the set of jar libraries that make up the runtime dependency classpath of the application. |

#### Examples

This section shows two examples of Ant tasks.

*Specify start-class*

<spring-boot:exejar destfile="target/my-application.jar" classes="target/classes" start-class="com.example.MyApplication">

<resources>

<fileset dir="src/main/resources" />

</resources>

<lib>

<fileset dir="lib" />

</lib>

</spring-boot:exejar>

*Detect start-class*

<exejar destfile="target/my-application.jar" classes="target/classes">

<lib>

<fileset dir="lib" />

</lib>

</exejar>

### Using the “findmainclass” Task

The findmainclass task is used internally by exejar to locate a class declaring a main. If necessary, you can also use this task directly in your build. The following attributes are supported:

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Required** |
| classesroot | The root directory of Java class files | Yes *(unless* mainclass *is specified)* |
| mainclass | Can be used to short-circuit the main  class search | No |
| property | The Ant property that should be set with the result | No *(result will be logged if unspecified)* |

#### Examples

This section contains three examples of using findmainclass.

*Find and log*

<findmainclass classesroot="target/classes" />

*Find and set*

<findmainclass classesroot="target/classes" property="main-class" />

*Override and set*

<findmainclass mainclass="com.example.MainClass" property="main-class" />

## Supporting Other Build Systems

If you want to use a build tool other than Maven, Gradle, or Ant, you likely need to develop your own plugin. Executable jars need to follow a specific format and certain entries need to be written in an uncompressed form (see the “[executable jar format](#_bookmark691)” section in the appendix for details).

The Spring Boot Maven and Gradle plugins both make use of spring-boot-loader-tools to actually generate jars. If you need to, you may use this library directly.

### Repackaging Archives

To repackage an existing archive so that it becomes a self-contained executable archive, use org.springframework.boot.loader.tools.Repackager. The Repackager class takes a single constructor argument that refers to an existing jar or war archive. Use one of the two available repackage() methods to either replace the original file or write to a new destination. Various settings can also be configured on the repackager before it is run.

### Nested Libraries

When repackaging an archive, you can include references to dependency files by using the org.springframework.boot.loader.tools.Libraries interface. We do not provide any concrete implementations of Libraries here as they are usually build-system-specific.

If your archive already includes libraries, you can use Libraries.NONE.

### Finding a Main Class

If you do not use Repackager.setMainClass() to specify a main class, the repackager uses [ASM](https://asm.ow2.io/) to read class files and tries to find a suitable class with a public static void main(String[] args) method. An exception is thrown if more than one candidate is found.

### Example Repackage Implementation

The following example shows a typical repackage implementation:

Repackager repackager = new Repackager(sourceJarFile); repackager.setBackupSource(false); repackager.repackage(new Libraries() {

@Override

public void doWithLibraries(LibraryCallback callback) throws IOException {

// Build system specific implementation, callback for each dependency

// callback.library(new Library(nestedFile, LibraryScope.COMPILE));

}

});

## What to Read Next

If you are interested in how the build tool plugins work, you can look at the [spring-boot-tools](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-tools)

module on GitHub. More technical details of the executable jar format are covered in [the appendix](#_bookmark691). If you have specific build-related questions, you can check out the “[how-to](#_bookmark525)” guides.

**Chapter 9. “How-to” Guides**

This section provides answers to some common ‘how do I do that…’ questions that often arise when using Spring Boot. Its coverage is not exhaustive, but it does cover quite a lot.

If you have a specific problem that we do not cover here, you might want to check out [stackoverflow.com](https://stackoverflow.com/tags/spring-boot) to see if someone has already provided an answer. This is also a great place to ask new questions (please use the spring-boot tag).

We are also more than happy to extend this section. If you want to add a ‘how-to’, send us a [pull](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE) [request](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE).

## Spring Boot Application

This section includes topics relating directly to Spring Boot applications.

### Create Your Own FailureAnalyzer

[FailureAnalyzer](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/diagnostics/FailureAnalyzer.html) is a great way to intercept an exception on startup and turn it into a human- readable message, wrapped in a [FailureAnalysis](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/diagnostics/FailureAnalysis.html). Spring Boot provides such an analyzer for application-context-related exceptions, JSR-303 validations, and more. You can also create your own.

AbstractFailureAnalyzer is a convenient extension of FailureAnalyzer that checks the presence of a specified exception type in the exception to handle. You can extend from that so that your implementation gets a chance to handle the exception only when it is actually present. If, for whatever reason, you cannot handle the exception, return null to give another implementation a chance to handle the exception.

FailureAnalyzer implementations must be registered in META-INF/spring.factories. The following example registers ProjectConstraintViolationFailureAnalyzer:

org.springframework.boot.diagnostics.FailureAnalyzer=\ com.example.ProjectConstraintViolationFailureAnalyzer

If you need access to the BeanFactory or the Environment, your FailureAnalyzer can implement BeanFactoryAware or EnvironmentAware respectively.

### Troubleshoot Auto-configuration

The Spring Boot auto-configuration tries its best to “do the right thing”, but sometimes things fail, and it can be hard to tell why.

There is a really useful ConditionEvaluationReport available in any Spring Boot ApplicationContext. You can see it if you enable DEBUG logging output. If you use the spring-boot-actuator (see [the](#_bookmark378) [Actuator chapter](#_bookmark378)), there is also a conditions endpoint that renders the report in JSON. Use that endpoint to debug the application and see what features have been added (and which have not

been added) by Spring Boot at runtime.

Many more questions can be answered by looking at the source code and the Javadoc. When reading the code, remember the following rules of thumb:

* + - * Look for classes called \*AutoConfiguration and read their sources. Pay special attention to the @Conditional\* annotations to find out what features they enable and when. Add --debug to the command line or a System property -Ddebug to get a log on the console of all the auto- configuration decisions that were made in your app. In a running application with actuator enabled, look at the conditions endpoint (/actuator/conditions or the JMX equivalent) for the same information.
      * Look for classes that are @ConfigurationProperties (such as [ServerProperties](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/ServerProperties.java)) and read from there the available external configuration options. The @ConfigurationProperties annotation has a name attribute that acts as a prefix to external properties. Thus, ServerProperties has prefix="server" and its configuration properties are server.port, server.address, and others. In a running application with actuator enabled, look at the configprops endpoint.
      * Look for uses of the bind method on the Binder to pull configuration values explicitly out of the

Environment in a relaxed manner. It is often used with a prefix.

* + - * Look for @Value annotations that bind directly to the Environment.
      * Look for @ConditionalOnExpression annotations that switch features on and off in response to SpEL expressions, normally evaluated with placeholders resolved from the Environment.

### Customize the Environment or ApplicationContext Before It Starts

A SpringApplication has ApplicationListeners and ApplicationContextInitializers that are used to apply customizations to the context or environment. Spring Boot loads a number of such customizations for use internally from META-INF/spring.factories. There is more than one way to register additional customizations:

* + - * Programmatically, per application, by calling the addListeners and addInitializers methods on

SpringApplication before you run it.

* + - * Declaratively, per application, by setting the context.initializer.classes or

context.listener.classes properties.

* + - * Declaratively, for all applications, by adding a META-INF/spring.factories and packaging a jar file that the applications all use as a library.

The SpringApplication sends some special ApplicationEvents to the listeners (some even before the context is created) and then registers the listeners for events published by the ApplicationContext as well. See “[Application Events and Listeners](#_bookmark95)” in the ‘Spring Boot features’ section for a complete list.

It is also possible to customize the Environment before the application context is refreshed by using EnvironmentPostProcessor. Each implementation should be registered in META-INF/spring.factories, as shown in the following example:

org.springframework.boot.env.EnvironmentPostProcessor=com.example.YourEnvironmentPostP rocessor

The implementation can load arbitrary files and add them to the Environment. For instance, the following example loads a YAML configuration file from the classpath:

public class EnvironmentPostProcessorExample implements EnvironmentPostProcessor { private final YamlPropertySourceLoader loader = new YamlPropertySourceLoader();

@Override

public void postProcessEnvironment(ConfigurableEnvironment environment, SpringApplication application) {

Resource path = new ClassPathResource("com/example/myapp/config.yml"); PropertySource<?> propertySource = loadYaml(path); environment.getPropertySources().addLast(propertySource);

}

private PropertySource<?> loadYaml(Resource path) { if (!path.exists()) {

throw new IllegalArgumentException("Resource " + path + " does not

exist");

}

try {

return this.loader.load("custom-resource", path).get(0);

}

catch (IOException ex) {

throw new IllegalStateException("Failed to load yaml configuration from "

+ path, ex);

}

}

}

The Environment has already been prepared with all the usual property sources that Spring Boot loads by default. It is therefore possible to get the location of the file from the environment. The preceding example adds the custom-resource property source at the end of the list so that a key defined in any of the usual other locations takes precedence. A custom implementation may define another order.

While using @PropertySource on your @SpringBootApplication may seem to be a convenient way to load a custom resource in the Environment, we do not recommend it. Such property sources are not added to the Environment until the application context is being refreshed. This is too late to configure certain properties such as logging.\* and spring.main.\* which are read before refresh begins.

### Build an ApplicationContext Hierarchy (Adding a Parent or Root Context)

You can use the ApplicationBuilder class to create parent/child ApplicationContext hierarchies. See “[Fluent Builder API](#_bookmark90)” in the ‘Spring Boot features’ section for more information.

### Create a Non-web Application

Not all Spring applications have to be web applications (or web services). If you want to execute some code in a main method but also bootstrap a Spring application to set up the infrastructure to use, you can use the SpringApplication features of Spring Boot. A SpringApplication changes its ApplicationContext class, depending on whether it thinks it needs a web application or not. The first thing you can do to help it is to leave server-related dependencies (e.g. servlet API) off the classpath. If you cannot do that (for example, you run two applications from the same code base) then you can explicitly call setWebApplicationType(WebApplicationType.NONE) on your SpringApplication instance or set the applicationContextClass property (through the Java API or with external properties). Application code that you want to run as your business logic can be implemented as a CommandLineRunner and dropped into the context as a @Bean definition.

## Properties and Configuration

This section includes topics about setting and reading properties and configuration settings and their interaction with Spring Boot applications.

### Automatically Expand Properties at Build Time

Rather than hardcoding some properties that are also specified in your project’s build configuration, you can automatically expand them by instead using the existing build configuration. This is possible in both Maven and Gradle.

#### Automatic Property Expansion Using Maven

You can automatically expand properties from the Maven project by using resource filtering. If you use the spring-boot-starter-parent, you can then refer to your Maven ‘project properties’ with @..@ placeholders, as shown in the following example:

[app.encoding=@project.build.sourceEncoding@](mailto:app.encoding%3D@project.build.sourceEncoding@) [app.java.version=@java.version@](mailto:app.java.version%3D@java.version@)

Only production configuration is filtered that way (in other words, no filtering is applied on src/test/resources).

If you enable the addResources flag, the spring-boot:run goal can add src/main/resources directly to the classpath (for hot reloading purposes). Doing so circumvents the resource filtering and this feature. Instead, you can use the exec:java goal or customize the plugin’s configuration. See the [plugin usage page](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/maven-plugin/reference/html/#getting-started) for more details.

If you do not use the starter parent, you need to include the following element inside the <build/>

element of your pom.xml:

<resources>

<resource>

<directory>src/main/resources</directory>

<filtering>true</filtering>

</resource>

</resources>

You also need to include the following element inside <plugins/>:

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-resources-plugin</artifactId>

<version>2.7</version>

<configuration>

<delimiters>

<delimiter>@</delimiter>

</delimiters>

<useDefaultDelimiters>false</useDefaultDelimiters>

</configuration>

</plugin>

The useDefaultDelimiters property is important if you use standard Spring placeholders (such as ${placeholder}) in your configuration. If that property is not set to false, these may be expanded by the build.

#### Automatic Property Expansion Using Gradle

You can automatically expand properties from the Gradle project by configuring the Java plugin’s

processResources task to do so, as shown in the following example:

processResources { expand(project.properties)

}

You can then refer to your Gradle project’s properties by using placeholders, as shown in the following example:

app.name=${name} app.description=${description}

Gradle’s expand method uses Groovy’s SimpleTemplateEngine, which transforms

${..} tokens. The ${..} style conflicts with Spring’s own property placeholder mechanism. To use Spring property placeholders together with automatic expansion, escape the Spring property placeholders as follows: \${..}.

### Externalize the Configuration of SpringApplication

A SpringApplication has bean properties (mainly setters), so you can use its Java API as you create the application to modify its behavior. Alternatively, you can externalize the configuration by setting properties in spring.main.\*. For example, in application.properties, you might have the following settings:

spring.main.web-application-type=none spring.main.banner-mode=off

Then the Spring Boot banner is not printed on startup, and the application is not starting an embedded web server.

Properties defined in external configuration override the values specified with the Java API, with the notable exception of the sources used to create the ApplicationContext. Consider the following application:

new SpringApplicationBuilder()

.bannerMode(Banner.Mode.OFF)

.sources(demo.MyApp.class)

.run(args);

Now consider the following configuration:

spring.main.sources=com.acme.Config,com.acme.ExtraConfig spring.main.banner-mode=console

The actual application *now* shows the banner (as overridden by configuration) and uses three sources for the ApplicationContext (in the following order): demo.MyApp, com.acme.Config, and com.acme.ExtraConfig.

### Change the Location of External Properties of an Application

By default, properties from different sources are added to the Spring Environment in a defined order (see “[Externalized Configuration](#_bookmark101)” in the ‘Spring Boot features’ section for the exact order).

You can also provide the following System properties (or environment variables) to change the

behavior:

* + - * spring.config.name (SPRING\_CONFIG\_NAME): Defaults to application as the root of the file name.
      * spring.config.location (SPRING\_CONFIG\_LOCATION): The file to load (such as a classpath resource or a URL). A separate Environment property source is set up for this document and it can be overridden by system properties, environment variables, or the command line.

No matter what you set in the environment, Spring Boot always loads application.properties as described above. By default, if YAML is used, then files with the ‘.yml’ extension are also added to the list.

Spring Boot logs the configuration files that are loaded at the DEBUG level and the candidates it has not found at TRACE level.

See [ConfigFileApplicationListener](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/context/config/ConfigFileApplicationListener.java) for more detail.

### Use ‘Short’ Command Line Arguments

Some people like to use (for example) --port=9000 instead of --server.port=9000 to set configuration properties on the command line. You can enable this behavior by using placeholders in application.properties, as shown in the following example:

server.port=${port:8080}

If you inherit from the spring-boot-starter-parent POM, the default filter token of the maven-resources-plugins has been changed from ${\*} to @ (that is, @maven.token@ instead of ${maven.token}) to prevent conflicts with Spring-style placeholders. If you have enabled Maven filtering for the application.properties directly, you may want to also change the default filter token to use [other delimiters](https://maven.apache.org/plugins/maven-resources-plugin/resources-mojo.html#delimiters).

In this specific case, the port binding works in a PaaS environment such as Heroku or Cloud Foundry. In those two platforms, the PORT environment variable is set automatically and Spring can bind to capitalized synonyms for Environment properties.

### Use YAML for External Properties

YAML is a superset of JSON and, as such, is a convenient syntax for storing external properties in a hierarchical format, as shown in the following example:

spring:

application:

name: cruncher datasource:

driverClassName: com.mysql.jdbc.Driver url: jdbc:mysql://localhost/test

server:

port: 9000

Create a file called application.yml and put it in the root of your classpath. Then add snakeyaml to your dependencies (Maven coordinates org.yaml:snakeyaml, already included if you use the spring- boot-starter). A YAML file is parsed to a Java Map<String,Object> (like a JSON object), and Spring Boot flattens the map so that it is one level deep and has period-separated keys, as many people are used to with Properties files in Java.

The preceding example YAML corresponds to the following application.properties file:

spring.application.name=cruncher

spring.datasource.driver-class-name=com.mysql.jdbc.Driver spring.datasource.url=jdbc:mysql://localhost/test server.port=9000

See “[Using YAML Instead of Properties](#_bookmark109)” in the ‘Spring Boot features’ section for more information about YAML.

### Set the Active Spring Profiles

The Spring Environment has an API for this, but you would normally set a System property (spring.profiles.active) or an OS environment variable (SPRING\_PROFILES\_ACTIVE). Also, you can launch your application with a -D argument (remember to put it before the main class or jar archive), as follows:

$ java -jar -Dspring.profiles.active=production demo-0.0.1-SNAPSHOT.jar

In Spring Boot, you can also set the active profile in application.properties, as shown in the following example:

spring.profiles.active=production

A value set this way is replaced by the System property or environment variable setting but not by the SpringApplicationBuilder.profiles() method. Thus, the latter Java API can be used to augment the profiles without changing the defaults.

See “[Profiles](#_bookmark126)” in the “Spring Boot features” section for more information.

### Change Configuration Depending on the Environment

A YAML file is actually a sequence of documents separated by --- lines, and each document is parsed separately to a flattened map.

If a YAML document contains a spring.profiles key, then the profiles value (a comma-separated list of profiles) is fed into the Spring Environment.acceptsProfiles() method. If any of those profiles is active, that document is included in the final merge (otherwise, it is not), as shown in the following example:

server:

port: 9000

---

spring:

profiles: development server:

port: 9001

---

spring:

profiles: production server:

port: 0

In the preceding example, the default port is 9000. However, if the Spring profile called ‘development’ is active, then the port is 9001. If ‘production’ is active, then the port is 0.

The YAML documents are merged in the order in which they are encountered. Later values override earlier values.

To do the same thing with properties files, you can use application-${profile}.properties to specify profile-specific values.

### Discover Built-in Options for External Properties

Spring Boot binds external properties from application.properties (or .yml files and other places) into an application at runtime. There is not (and technically cannot be) an exhaustive list of all supported properties in a single location, because contributions can come from additional jar files on your classpath.

A running application with the Actuator features has a configprops endpoint that shows all the bound and bindable properties available through @ConfigurationProperties.

The appendix includes an [application.properties](#_bookmark656) example with a list of the most common properties supported by Spring Boot. The definitive list comes from searching the source code for @ConfigurationProperties and @Value annotations as well as the occasional use of Binder. For more about the exact ordering of loading properties, see "[Externalized Configuration](#_bookmark101)".

## Embedded Web Servers

Each Spring Boot web application includes an embedded web server. This feature leads to a number of how-to questions, including how to change the embedded server and how to configure the embedded server. This section answers those questions.

### Use Another Web Server

Many Spring Boot starters include default embedded containers.

* + - * For servlet stack applications, the spring-boot-starter-web includes Tomcat by including spring- boot-starter-tomcat, but you can use spring-boot-starter-jetty or spring-boot-starter- undertow instead.
      * For reactive stack applications, the spring-boot-starter-webflux includes Reactor Netty by including spring-boot-starter-reactor-netty, but you can use spring-boot-starter-tomcat, spring-boot-starter-jetty, or spring-boot-starter-undertow instead.

When switching to a different HTTP server, you need to swap the default dependencies for those that you need instead. To help with this process, Spring Boot provides a separate starter for each of the supported HTTP servers.

The following Maven example shows how to exclude Tomcat and include Jetty for Spring MVC:

<properties>

<servlet-api.version>3.1.0</servlet-api.version>

</properties>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<exclusions>

<!-- Exclude the Tomcat dependency -->

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

</exclusion>

</exclusions>

</dependency>

<!-- Use Jetty instead -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jetty</artifactId>

</dependency>

The version of the Servlet API has been overridden as, unlike Tomcat 9 and Undertow 2.0, Jetty 9.4 does not support Servlet 4.0.

The following Gradle example shows how to use Undertow in place of Reactor Netty for Spring WebFlux:

configurations.all { resolutionStrategy.dependencySubstitution.all { dependency ->

if (dependency.requested instanceof ModuleComponentSelector && dependency.requested.module == 'spring-boot-starter-reactor-netty') {

dependency.useTarget("org.springframework.boot:spring-boot-starter- undertow:$dependency.requested.version", 'Use Undertow instead of Reactor Netty')

}

}

}

dependencies {

compile 'org.springframework.boot:spring-boot-starter-webflux'

// ...

}

spring-boot-starter-reactor-netty is required to use the WebClient class, so you may need to keep a dependency on Netty even when you need to include a different HTTP server.

### Disabling the Web Server

If your classpath contains the necessary bits to start a web server, Spring Boot will automatically start it. To disable this behavior configure the WebApplicationType in your application.properties, as shown in the following example:

spring.main.web-application-type=none

### Change the HTTP Port

In a standalone application, the main HTTP port defaults to 8080 but can be set with server.port (for example, in application.properties or as a System property). Thanks to relaxed binding of Environment values, you can also use SERVER\_PORT (for example, as an OS environment variable).

To switch off the HTTP endpoints completely but still create a WebApplicationContext, use

server.port=-1 (doing so is sometimes useful for testing).

For more details, see “[Customizing Embedded Servlet Containers](#_bookmark177)” in the ‘Spring Boot Features’ section, or the [ServerProperties](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/ServerProperties.java) source code.

### Use a Random Unassigned HTTP Port

To scan for a free port (using OS natives to prevent clashes) use server.port=0.

### Discover the HTTP Port at Runtime

You can access the port the server is running on from log output or from the

WebServerApplicationContext through its WebServer. The best way to get that and be sure it has been

initialized is to add a @Bean of type ApplicationListener<WebServerInitializedEvent> and pull the container out of the event when it is published.

Tests that use @SpringBootTest(webEnvironment=WebEnvironment.RANDOM\_PORT) can also inject the actual port into a field by using the @LocalServerPort annotation, as shown in the following example:

@SpringBootTest(webEnvironment=WebEnvironment.RANDOM\_PORT) public class MyWebIntegrationTests {

@LocalServerPort int port;

// ...

}

@LocalServerPort is a meta-annotation for @Value("${local.server.port}"). Do not try to inject the port in a regular application. As we just saw, the value is set only after the container has been initialized. Contrary to a test, application code callbacks are processed early (before the value is actually available).

### Enable HTTP Response Compression

HTTP response compression is supported by Jetty, Tomcat, and Undertow. It can be enabled in

application.properties, as follows:

server.compression.enabled=true

By default, responses must be at least 2048 bytes in length for compression to be performed. You can configure this behavior by setting the server.compression.min-response-size property.

By default, responses are compressed only if their content type is one of the following:

* + - * text/html
      * text/xml
      * text/plain
      * text/css
      * text/javascript
      * application/javascript
      * application/json
      * application/xml

You can configure this behavior by setting the server.compression.mime-types property.

### Configure SSL

SSL can be configured declaratively by setting the various server.ssl.\* properties, typically in application.properties or application.yml. The following example shows setting SSL properties in application.properties:

server.port=8443

server.ssl.key-store=classpath:keystore.jks server.ssl.key-store-password=secret server.ssl.key-password=another-secret

See [Ssl](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/web/server/Ssl.java) for details of all of the supported properties.

Using configuration such as the preceding example means the application no longer supports a plain HTTP connector at port 8080. Spring Boot does not support the configuration of both an HTTP connector and an HTTPS connector through application.properties. If you want to have both, you need to configure one of them programmatically. We recommend using application.properties to configure HTTPS, as the HTTP connector is the easier of the two to configure programmatically.

### Configure HTTP/2

You can enable HTTP/2 support in your Spring Boot application with the server.http2.enabled configuration property. This support depends on the chosen web server and the application environment, since that protocol is not supported out-of-the-box by all JDK8 releases.

Spring Boot does not support h2c, the cleartext version of the HTTP/2 protocol. So you must [configure SSL first](#_bookmark550).

#### HTTP/2 with Undertow

As of Undertow 1.4.0+, HTTP/2 is supported without any additional requirement on JDK8.

#### HTTP/2 with Jetty

For HTTP/2 support, Jetty requires the additional org.eclipse.jetty.http2:http2-server

dependency. Now depending on your deployment, you also need to choose other dependencies:

* org.eclipse.jetty:jetty-alpn-java-server for applications running on JDK9+
* org.eclipse.jetty:jetty-alpn-openjdk8-server for applications running on JDK8u252+
* org.eclipse.jetty:jetty-alpn-conscrypt-server and the [Conscrypt library](https://www.conscrypt.org/) with no JDK requirement

#### HTTP/2 with Tomcat

Spring Boot ships by default with Tomcat 9.0.x which supports HTTP/2 out of the box when using JDK 9 or later. Alternatively, HTTP/2 can be used on JDK 8 if the libtcnative library and its dependencies are installed on the host operating system.

The library directory must be made available, if not already, to the JVM library path. You can do so with a JVM argument such as -Djava.library.path=/usr/local/opt/tomcat-native/lib. More on this in the [official Tomcat documentation](https://tomcat.apache.org/tomcat-9.0-doc/apr.html).

Starting Tomcat 9.0.x on JDK 8 without that native support logs the following error:

ERROR 8787 --- [

main] o.a.coyote.http11.Http11NioProtocol

: The

upgrade handler [org.apache.coyote.http2.Http2Protocol] for [h2] only supports upgrade via ALPN but has been configured for the ["https-jsse-nio-8443"] connector that does not support ALPN.

This error is not fatal, and the application still starts with HTTP/1.1 SSL support.

#### HTTP/2 with Reactor Netty

The spring-boot-webflux-starter is using by default Reactor Netty as a server. Reactor Netty can be configured for HTTP/2 using the JDK support with JDK 9 or later. For JDK 8 environments, or for optimal runtime performance, this server also supports HTTP/2 with native libraries. To enable that, your application needs to have an additional dependency.

Spring Boot manages the version for the io.netty:netty-tcnative-boringssl-static "uber jar", containing native libraries for all platforms. Developers can choose to import only the required dependencies using a classifier (see [the Netty official documentation](https://netty.io/wiki/forked-tomcat-native.html)).

### Configure the Web Server

Generally, you should first consider using one of the many available configuration keys and customize your web server by adding new entries in your application.properties (or application.yml, or environment, etc. see “[Discover Built-in Options for External Properties](#_bookmark542)”). The server.\* namespace is quite useful here, and it includes namespaces like server.tomcat.\*, server.jetty.\* and others, for server-specific features. See the list of [Common Application](#_bookmark656) [properties](#_bookmark656).

The previous sections covered already many common use cases, such as compression, SSL or HTTP/2. However, if a configuration key doesn’t exist for your use case, you should then look at [WebServerFactoryCustomizer](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/web/server/WebServerFactoryCustomizer.html). You can declare such a component and get access to the server factory relevant to your choice: you should select the variant for the chosen Server (Tomcat, Jetty, Reactor Netty, Undertow) and the chosen web stack (Servlet or Reactive).

The example below is for Tomcat with the spring-boot-starter-web (Servlet stack):

@Component

public class MyTomcatWebServerCustomizer

implements WebServerFactoryCustomizer<TomcatServletWebServerFactory> {

@Override

public void customize(TomcatServletWebServerFactory factory) {

// customize the factory here

}

}

In addition Spring Boot provides:

|  |  |  |
| --- | --- | --- |
| **Server** | **Servlet stack** | **Reactive stack** |
| Tomcat | TomcatServletWebServerFactory | TomcatReactiveWebServerFactory |
| Jetty | JettyServletWebServerFactory | JettyReactiveWebServerFactory |
| Undertow | UndertowServletWebServerFactory | UndertowReactiveWebServerFactory |
| Reactor | N/A | NettyReactiveWebServerFactory |

Once you’ve got access to a WebServerFactory, you can often add customizers to it to configure specific parts, like connectors, server resources, or the server itself - all using server-specific APIs.

As a last resort, you can also declare your own WebServerFactory component, which will override the one provided by Spring Boot. In this case, you can’t rely on configuration properties in the server namespace anymore.

### Add a Servlet, Filter, or Listener to an Application

In a servlet stack application, i.e. with the spring-boot-starter-web, there are two ways to add Servlet, Filter, ServletContextListener, and the other listeners supported by the Servlet API to your application:

* + - * [Add a Servlet, Filter, or Listener by Using a Spring Bean](#_bookmark558)
      * [Add Servlets, Filters, and Listeners by Using Classpath Scanning](#_bookmark559)

#### Add a Servlet, Filter, or Listener by Using a Spring Bean

To add a Servlet, Filter, or Servlet \*Listener by using a Spring bean, you must provide a @Bean definition for it. Doing so can be very useful when you want to inject configuration or dependencies. However, you must be very careful that they do not cause eager initialization of too many other beans, because they have to be installed in the container very early in the application lifecycle. (For example, it is not a good idea to have them depend on your DataSource or JPA configuration.) You can work around such restrictions by initializing the beans lazily when first used instead of on initialization.

In the case of Filters and Servlets, you can also add mappings and init parameters by adding a

FilterRegistrationBean or a ServletRegistrationBean instead of or in addition to the underlying

component.



If no dispatcherType is specified on a filter registration, REQUEST is used. This aligns with the Servlet Specification’s default dispatcher type.

Like any other Spring bean, you can define the order of Servlet filter beans; please make sure to check the “[Registering Servlets, Filters, and Listeners as Spring Beans](#_bookmark174)” section.

**Disable Registration of a Servlet or Filter**

As [described earlier](#_bookmark558), any Servlet or Filter beans are registered with the servlet container automatically. To disable registration of a particular Filter or Servlet bean, create a registration bean for it and mark it as disabled, as shown in the following example:

@Bean

public FilterRegistrationBean registration(MyFilter filter) { FilterRegistrationBean registration = new FilterRegistrationBean(filter); registration.setEnabled(false);

return registration;

}

#### Add Servlets, Filters, and Listeners by Using Classpath Scanning

@WebServlet, @WebFilter, and @WebListener annotated classes can be automatically registered with an embedded servlet container by annotating a @Configuration class with @ServletComponentScan and specifying the package(s) containing the components that you want to register. By default, @ServletComponentScan scans from the package of the annotated class.

### Configure Access Logging

Access logs can be configured for Tomcat, Undertow, and Jetty through their respective namespaces.

For instance, the following settings log access on Tomcat with a [custom pattern](https://tomcat.apache.org/tomcat-9.0-doc/config/valve.html#Access_Logging).

server.tomcat.basedir=my-tomcat server.tomcat.accesslog.enabled=true server.tomcat.accesslog.pattern=%t %a "%r" %s (%D ms)

The default location for logs is a logs directory relative to the Tomcat base directory. By default, the logs directory is a temporary directory, so you may want to fix Tomcat’s base directory or use an absolute path for the logs. In the preceding example, the logs are available in my-tomcat/logs relative to the working directory of the application.

Access logging for Undertow can be configured in a similar fashion, as shown in the following example:

server.undertow.accesslog.enabled=true server.undertow.accesslog.pattern=%t %a "%r" %s (%D ms)

Logs are stored in a logs directory relative to the working directory of the application. You can customize this location by setting the server.undertow.accesslog.dir property.

Finally, access logging for Jetty can also be configured as follows:

server.jetty.accesslog.enabled=true server.jetty.accesslog.filename=/var/log/jetty-access.log

By default, logs are redirected to System.err. For more details, see [the Jetty documentation](https://www.eclipse.org/jetty/documentation/9.4.31.v20200723/configuring-jetty-request-logs.html).

### Running Behind a Front-end Proxy Server

If your application is running behind a proxy, a load-balancer or in the cloud, the request information (like the host, port, scheme…) might change along the way. Your application may be running on 10.10.10.10:8080, but HTTP clients should only see example.org.

[RFC7239 "Forwarded Headers"](https://tools.ietf.org/html/rfc7239) defines the Forwarded HTTP header; proxies can use this header to provide information about the original request. You can configure your application to read those headers and automatically use that information when creating links and sending them to clients in HTTP 302 responses, JSON documents or HTML pages. There are also non-standard headers, like X- Forwarded-Host, X-Forwarded-Port, X-Forwarded-Proto, X-Forwarded-Ssl, and X-Forwarded-Prefix.

If the proxy adds the commonly used X-Forwarded-For and X-Forwarded-Proto headers, setting server.forward-headers-strategy to NATIVE is enough to support those. With this option, the Web servers themselves natively support this feature; you can check their specific documentation to learn about specific behavior.

If this is not enough, Spring Framework provides a [ForwardedHeaderFilter](https://docs.spring.io/spring/docs/5.2.9.RELEASE/spring-framework-reference/web.html#filters-forwarded-headers). You can register it as a Servlet Filter in your application by setting server.forward-headers-strategy is set to FRAMEWORK.

If your application runs in Cloud Foundry or Heroku, the server.forward-headers- strategy property defaults to NATIVE. In all other instances, it defaults to NONE.

#### Customize Tomcat’s Proxy Configuration

If you use Tomcat, you can additionally configure the names of the headers used to carry “forwarded” information, as shown in the following example:

server.tomcat.remoteip.remote-ip-header=x-your-remote-ip-header server.tomcat.remoteip.protocol-header=x-your-protocol-header

Tomcat is also configured with a default regular expression that matches internal proxies that are to be trusted. By default, IP addresses in 10/8, 192.168/16, 169.254/16 and 127/8 are trusted. You can

customize the valve’s configuration by adding an entry to application.properties, as shown in the following example:

server.tomcat.remoteip.internal-proxies=192\\.168\\.\\d{1,3}\\.\\d{1,3}

The double backslashes are required only when you use a properties file for configuration. If you use YAML, single backslashes are sufficient, and a value equivalent to that shown in the preceding example would be 192\.168\.\d{1,3}\.\d{1,3}.

You can trust all proxies by setting the internal-proxies to empty (but do not do so in production).

You can take complete control of the configuration of Tomcat’s RemoteIpValve by switching the automatic one off (to do so, set server.forward-headers-strategy=NONE) and adding a new valve instance in a TomcatServletWebServerFactory bean.

### Enable Multiple Connectors with Tomcat

You can add an org.apache.catalina.connector.Connector to the TomcatServletWebServerFactory, which can allow multiple connectors, including HTTP and HTTPS connectors, as shown in the following example:

@Bean

public ServletWebServerFactory servletContainer() { TomcatServletWebServerFactory tomcat = new TomcatServletWebServerFactory(); tomcat.addAdditionalTomcatConnectors(createSslConnector());

return tomcat;

}

private Connector createSslConnector() {

Connector connector = new Connector("org.apache.coyote.http11.Http11NioProtocol"); Http11NioProtocol protocol = (Http11NioProtocol) connector.getProtocolHandler(); try {

File keystore = new ClassPathResource("keystore").getFile(); File truststore = new ClassPathResource("keystore").getFile(); connector.setScheme("https");

connector.setSecure(true); connector.setPort(8443); protocol.setSSLEnabled(true);

protocol.setKeystoreFile(keystore.getAbsolutePath()); protocol.setKeystorePass("changeit"); protocol.setTruststoreFile(truststore.getAbsolutePath()); protocol.setTruststorePass("changeit"); protocol.setKeyAlias("apitester");

return connector;

}

catch (IOException ex) {

throw new IllegalStateException("can't access keystore: [" + keystore

+ "] or truststore: [" + truststore + "]", ex);

}

}

### Use Tomcat’s LegacyCookieProcessor

By default, the embedded Tomcat used by Spring Boot does not support "Version 0" of the Cookie format, so you may see the following error:

java.lang.IllegalArgumentException: An invalid character [32] was present in the Cookie value

If at all possible, you should consider updating your code to only store values compliant with later Cookie specifications. If, however, you cannot change the way that cookies are written, you can instead configure Tomcat to use a LegacyCookieProcessor. To switch to the LegacyCookieProcessor, use an WebServerFactoryCustomizer bean that adds a TomcatContextCustomizer, as shown in the following example:

@Bean

public WebServerFactoryCustomizer<TomcatServletWebServerFactory> cookieProcessorCustomizer() {

return (factory) -> factory

.addContextCustomizers((context) -> context.setCookieProcessor(new LegacyCookieProcessor()));

}

### Enable Tomcat’s MBean Registry

Embedded Tomcat’s MBean registry is disabled by default. This minimizes Tomcat’s memory footprint. If you want to use Tomcat’s MBeans, for example so that they can be used to expose metrics via Micrometer, you must use the server.tomcat.mbeanregistry.enabled property to do so, as shown in the following example:

server.tomcat.mbeanregistry.enabled=true

### Enable Multiple Listeners with Undertow

Add an UndertowBuilderCustomizer to the UndertowServletWebServerFactory and add a listener to the

Builder, as shown in the following example:

@Bean

public UndertowServletWebServerFactory servletWebServerFactory() { UndertowServletWebServerFactory factory = new UndertowServletWebServerFactory(); factory.addBuilderCustomizers(new UndertowBuilderCustomizer() {

@Override

public void customize(Builder builder) { builder.addHttpListener(8080, "0.0.0.0");

}

});

return factory;

}

### Create WebSocket Endpoints Using @ServerEndpoint

If you want to use @ServerEndpoint in a Spring Boot application that used an embedded container, you must declare a single ServerEndpointExporter @Bean, as shown in the following example:

@Bean

public ServerEndpointExporter serverEndpointExporter() { return new ServerEndpointExporter();

}

The bean shown in the preceding example registers any @ServerEndpoint annotated beans with the underlying WebSocket container. When deployed to a standalone servlet container, this role is performed by a servlet container initializer, and the ServerEndpointExporter bean is not required.

## Spring MVC

Spring Boot has a number of starters that include Spring MVC. Note that some starters include a dependency on Spring MVC rather than include it directly. This section answers common questions about Spring MVC and Spring Boot.

### Write a JSON REST Service

Any Spring @RestController in a Spring Boot application should render JSON response by default as long as Jackson2 is on the classpath, as shown in the following example:

@RestController

public class MyController {

@RequestMapping("/thing") public MyThing thing() {

return new MyThing();

}

}

As long as MyThing can be serialized by Jackson2 (true for a normal POJO or Groovy object), then localhost:8080/thing serves a JSON representation of it by default. Note that, in a browser, you might sometimes see XML responses, because browsers tend to send accept headers that prefer XML.

### Write an XML REST Service

If you have the Jackson XML extension (jackson-dataformat-xml) on the classpath, you can use it to render XML responses. The previous example that we used for JSON would work. To use the Jackson XML renderer, add the following dependency to your project:

<dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>

<artifactId>jackson-dataformat-xml</artifactId>

</dependency>

If Jackson’s XML extension is not available and JAXB is available, XML can be rendered with the additional requirement of having MyThing annotated as @XmlRootElement, as shown in the following example:

@XmlRootElement

public class MyThing { private String name;

// .. getters and setters

}

JAXB is only available out of the box with Java 8. If you’re using a more recent Java generation, add the following dependency to your project:

<dependency>

<groupId>org.glassfish.jaxb</groupId>

<artifactId>jaxb-runtime</artifactId>

</dependency>

To get the server to render XML instead of JSON, you might have to send an

Accept: text/xml header (or use a browser).

### Customize the Jackson ObjectMapper

Spring MVC (client and server side) uses HttpMessageConverters to negotiate content conversion in an HTTP exchange. If Jackson is on the classpath, you already get the default converter(s) provided by Jackson2ObjectMapperBuilder, an instance of which is auto-configured for you.

The ObjectMapper (or XmlMapper for Jackson XML converter) instance (created by default) has the following customized properties:

* + - * MapperFeature.DEFAULT\_VIEW\_INCLUSION is disabled
      * DeserializationFeature.FAIL\_ON\_UNKNOWN\_PROPERTIES is disabled
      * SerializationFeature.WRITE\_DATES\_AS\_TIMESTAMPS is disabled

Spring Boot also has some features to make it easier to customize this behavior.

You can configure the ObjectMapper and XmlMapper instances by using the environment. Jackson provides an extensive suite of on/off features that can be used to configure various aspects of its processing. These features are described in six enums (in Jackson) that map onto properties in the environment:

|  |  |  |
| --- | --- | --- |
| **Enum** | **Property** | **Values** |
| com.fasterxml.jackson.databind  .DeserializationFeature | spring.jackson.deserialization  .<feature\_name> | true, false |
| com.fasterxml.jackson.core.Jso nGenerator.Feature | spring.jackson.generator.<feat ure\_name> | true, false |
| com.fasterxml.jackson.databind  .MapperFeature | spring.jackson.mapper.<feature  \_name> | true, false |
| com.fasterxml.jackson.core.Jso nParser.Feature | spring.jackson.parser.<feature  \_name> | true, false |

|  |  |  |
| --- | --- | --- |
| **Enum** | **Property** | **Values** |
| com.fasterxml.jackson.databind  .SerializationFeature | spring.jackson.serialization.< feature\_name> | true, false |
| com.fasterxml.jackson.annotati on.JsonInclude.Include | spring.jackson.default- property-inclusion | always, non\_null, non\_absent, non\_default, non\_empty |

For example, to enable pretty print, set spring.jackson.serialization.indent\_output=true. Note that, thanks to the use of [relaxed binding](#_bookmark120), the case of indent\_output does not have to match the case of the corresponding enum constant, which is INDENT\_OUTPUT.

This environment-based configuration is applied to the auto-configured Jackson2ObjectMapperBuilder bean and applies to any mappers created by using the builder, including the auto-configured ObjectMapper bean.

The context’s Jackson2ObjectMapperBuilder can be customized by one or more Jackson2ObjectMapperBuilderCustomizer beans. Such customizer beans can be ordered (Boot’s own customizer has an order of 0), letting additional customization be applied both before and after Boot’s customization.

Any beans of type com.fasterxml.jackson.databind.Module are automatically registered with the auto-configured Jackson2ObjectMapperBuilder and are applied to any ObjectMapper instances that it creates. This provides a global mechanism for contributing custom modules when you add new features to your application.

If you want to replace the default ObjectMapper completely, either define a @Bean of that type and mark it as @Primary or, if you prefer the builder-based approach, define a Jackson2ObjectMapperBuilder @Bean. Note that, in either case, doing so disables all auto-configuration of the ObjectMapper.

If you provide any @Beans of type MappingJackson2HttpMessageConverter, they replace the default value in the MVC configuration. Also, a convenience bean of type HttpMessageConverters is provided (and is always available if you use the default MVC configuration). It has some useful methods to access the default and user-enhanced message converters.

See the “[Customize the @ResponseBody Rendering](#_bookmark572)” section and the [WebMvcAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/servlet/WebMvcAutoConfiguration.java) source code for more details.

### Customize the @ResponseBody Rendering

Spring uses HttpMessageConverters to render @ResponseBody (or responses from @RestController). You can contribute additional converters by adding beans of the appropriate type in a Spring Boot context. If a bean you add is of a type that would have been included by default anyway (such as MappingJackson2HttpMessageConverter for JSON conversions), it replaces the default value. A convenience bean of type HttpMessageConverters is provided and is always available if you use the default MVC configuration. It has some useful methods to access the default and user-enhanced message converters (For example, it can be useful if you want to manually inject them into a custom RestTemplate).

As in normal MVC usage, any WebMvcConfigurer beans that you provide can also contribute

converters by overriding the configureMessageConverters method. However, unlike with normal MVC, you can supply only additional converters that you need (because Spring Boot uses the same mechanism to contribute its defaults). Finally, if you opt out of the Spring Boot default MVC configuration by providing your own @EnableWebMvc configuration, you can take control completely and do everything manually by using getMessageConverters from WebMvcConfigurationSupport.

See the [WebMvcAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/servlet/WebMvcAutoConfiguration.java) source code for more details.

### Handling Multipart File Uploads

Spring Boot embraces the Servlet 3 javax.servlet.http.Part API to support uploading files. By default, Spring Boot configures Spring MVC with a maximum size of 1MB per file and a maximum of 10MB of file data in a single request. You may override these values, the location to which intermediate data is stored (for example, to the /tmp directory), and the threshold past which data is flushed to disk by using the properties exposed in the MultipartProperties class. For example, if you want to specify that files be unlimited, set the spring.servlet.multipart.max-file-size property to

-1.

The multipart support is helpful when you want to receive multipart encoded file data as a @RequestParam-annotated parameter of type MultipartFile in a Spring MVC controller handler method.

See the [MultipartAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/servlet/MultipartAutoConfiguration.java) source for more details.

It is recommended to use the container’s built-in support for multipart uploads rather than introducing an additional dependency such as Apache Commons File Upload.

### Switch Off the Spring MVC DispatcherServlet

By default, all content is served from the root of your application (/). If you would rather map to a different path, you can configure one as follows:

spring.mvc.servlet.path=/acme

If you have additional servlets you can declare a @Bean of type Servlet or ServletRegistrationBean for each and Spring Boot will register them transparently to the container. Because servlets are registered that way, they can be mapped to a sub-context of the DispatcherServlet without invoking it.

Configuring the DispatcherServlet yourself is unusual but if you really need to do it, a @Bean of type DispatcherServletPath must be provided as well to provide the path of your custom DispatcherServlet.

### Switch off the Default MVC Configuration

The easiest way to take complete control over MVC configuration is to provide your own

@Configuration with the @EnableWebMvc annotation. Doing so leaves all MVC configuration in your

hands.

### Customize ViewResolvers

A ViewResolver is a core component of Spring MVC, translating view names in @Controller to actual View implementations. Note that ViewResolvers are mainly used in UI applications, rather than REST-style services (a View is not used to render a @ResponseBody). There are many implementations of ViewResolver to choose from, and Spring on its own is not opinionated about which ones you should use. Spring Boot, on the other hand, installs one or two for you, depending on what it finds on the classpath and in the application context. The DispatcherServlet uses all the resolvers it finds in the application context, trying each one in turn until it gets a result. If you add your own, you have to be aware of the order and in which position your resolver is added.

WebMvcAutoConfiguration adds the following ViewResolvers to your context:

* + - * An InternalResourceViewResolver named ‘defaultViewResolver’. This one locates physical resources that can be rendered by using the DefaultServlet (including static resources and JSP pages, if you use those). It applies a prefix and a suffix to the view name and then looks for a physical resource with that path in the servlet context (the defaults are both empty but are accessible for external configuration through spring.mvc.view.prefix and spring.mvc.view.suffix). You can override it by providing a bean of the same type.
      * A BeanNameViewResolver named ‘beanNameViewResolver’. This is a useful member of the view resolver chain and picks up any beans with the same name as the View being resolved. It should not be necessary to override or replace it.
      * A ContentNegotiatingViewResolver named ‘viewResolver’ is added only if there **are** actually beans of type View present. This is a ‘master’ resolver, delegating to all the others and attempting to find a match to the ‘Accept’ HTTP header sent by the client. There is a useful [blog](https://spring.io/blog/2013/06/03/content-negotiation-using-views) [about ContentNegotiatingViewResolver](https://spring.io/blog/2013/06/03/content-negotiation-using-views) that you might like to study to learn more, and you might also look at the source code for detail. You can switch off the auto-configured ContentNegotiatingViewResolver by defining a bean named ‘viewResolver’.
      * If you use Thymeleaf, you also have a ThymeleafViewResolver named ‘thymeleafViewResolver’. It looks for resources by surrounding the view name with a prefix and suffix. The prefix is spring.thymeleaf.prefix, and the suffix is spring.thymeleaf.suffix. The values of the prefix and suffix default to ‘classpath:/templates/’ and ‘.html’, respectively. You can override ThymeleafViewResolver by providing a bean of the same name.
      * If you use FreeMarker, you also have a FreeMarkerViewResolver named ‘freeMarkerViewResolver’. It looks for resources in a loader path (which is externalized to spring.freemarker.templateLoaderPath and has a default value of ‘classpath:/templates/’) by surrounding the view name with a prefix and a suffix. The prefix is externalized to spring.freemarker.prefix, and the suffix is externalized to spring.freemarker.suffix. The default values of the prefix and suffix are empty and ‘.ftlh’, respectively. You can override FreeMarkerViewResolver by providing a bean of the same name.
      * If you use Groovy templates (actually, if groovy-templates is on your classpath), you also have a GroovyMarkupViewResolver named ‘groovyMarkupViewResolver’. It looks for resources in a loader path by surrounding the view name with a prefix and suffix (externalized to spring.groovy.template.prefix and spring.groovy.template.suffix). The prefix and suffix have

default values of ‘classpath:/templates/’ and ‘.tpl’, respectively. You can override

GroovyMarkupViewResolver by providing a bean of the same name.

* + - * If you use Mustache, you also have a MustacheViewResolver named ‘mustacheViewResolver’. It looks for resources by surrounding the view name with a prefix and suffix. The prefix is spring.mustache.prefix, and the suffix is spring.mustache.suffix. The values of the prefix and suffix default to ‘classpath:/templates/’ and ‘.mustache’, respectively. You can override MustacheViewResolver by providing a bean of the same name.

For more detail, see the following sections:

* + - * [WebMvcAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/servlet/WebMvcAutoConfiguration.java)
      * [ThymeleafAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/thymeleaf/ThymeleafAutoConfiguration.java)
      * [FreeMarkerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/freemarker/FreeMarkerAutoConfiguration.java)
      * [GroovyTemplateAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/groovy/template/GroovyTemplateAutoConfiguration.java)

## Testing With Spring Security

Spring Security provides support for running tests as a specific user. For example, the test in the snippet below will run with an authenticated user that has the ADMIN role.

@Test @WithMockUser(roles="ADMIN")

public void requestProtectedUrlWithUser() throws Exception { mvc

.perform(get("/"))

...

}

Spring Security provides comprehensive integration with Spring MVC Test and this can also be used when testing controllers using the @WebMvcTest slice and MockMvc.

For additional details on Spring Security’s testing support, refer to Spring Security’s [reference](https://docs.spring.io/spring-security/site/docs/5.3.4.RELEASE/reference/html5/#test) [documentation](https://docs.spring.io/spring-security/site/docs/5.3.4.RELEASE/reference/html5/#test)).

## Jersey

### Secure Jersey endpoints with Spring Security

Spring Security can be used to secure a Jersey-based web application in much the same way as it can be used to secure a Spring MVC-based web application. However, if you want to use Spring Security’s method-level security with Jersey, you must configure Jersey to use setStatus(int) rather sendError(int). This prevents Jersey from committing the response before Spring Security has had an opportunity to report an authentication or authorization failure to the client.

The jersey.config.server.response.setStatusOverSendError property must be set to true on the application’s ResourceConfig bean, as shown in the following example:

@Component

public class JerseyConfig extends ResourceConfig {

public JerseyConfig() { register(Endpoint.class);

setProperties(Collections.singletonMap("jersey.config.server.response.setStatusOverSen dError", true));

}

}

### Use Jersey Alongside Another Web Framework

To use Jersey alongside another web framework, such as Spring MVC, it should be configured so that it will allow the other framework to handle requests that it cannot handle. First, configure Jersey to use a Filter rather than a Servlet by configuring the spring.jersey.type application property with a value of filter. Second, configure your ResourceConfig to forward requests that would have resulted in a 404, as shown in the following example.

@Component

public class JerseyConfig extends ResourceConfig {

public JerseyConfig() { register(Endpoint.class);

property(ServletProperties.FILTER\_FORWARD\_ON\_404, true);

}

}

## HTTP Clients

Spring Boot offers a number of starters that work with HTTP clients. This section answers questions related to using them.

### Configure RestTemplate to Use a Proxy

As described in [RestTemplate Customization](#_bookmark285), you can use a RestTemplateCustomizer with RestTemplateBuilder to build a customized RestTemplate. This is the recommended approach for creating a RestTemplate configured to use a proxy.

The exact details of the proxy configuration depend on the underlying client request factory that is being used. The following example configures HttpComponentsClientRequestFactory with an HttpClient that uses a proxy for all hosts except 192.168.0.5:

static class ProxyCustomizer implements RestTemplateCustomizer {

@Override

public void customize(RestTemplate restTemplate) { HttpHost proxy = new HttpHost("proxy.example.com");

HttpClient httpClient = HttpClientBuilder.create().setRoutePlanner(new DefaultProxyRoutePlanner(proxy) {

@Override

public HttpHost determineProxy(HttpHost target, HttpRequest request, HttpContext context)

throws HttpException {

if (target.getHostName().equals("192.168.0.5")) { return null;

}

return super.determineProxy(target, request, context);

}

}).build(); restTemplate.setRequestFactory(new

HttpComponentsClientHttpRequestFactory(httpClient));

}

}

### Configure the TcpClient used by a Reactor Netty-based WebClient

When Reactor Netty is on the classpath a Reactor Netty-based WebClient is auto-configured. To customize the client’s handling of network connections, provide a ClientHttpConnector bean. The following example configures a 60 second connect timeout and adds a ReadTimeoutHandler:

@Bean

ClientHttpConnector clientHttpConnector(ReactorResourceFactory resourceFactory) { TcpClient tcpClient = TcpClient.create(resourceFactory.getConnectionProvider())

.runOn(resourceFactory.getLoopResources()).option(ChannelOption.CONNECT\_TIMEOUT\_MILLIS

, 60000)

.doOnConnected((connection) -> connection.addHandlerLast(new ReadTimeoutHandler(60)));

return new ReactorClientHttpConnector(HttpClient.from(tcpClient));

}

Note the use of ReactorResourceFactory for the connection provider and event loop resources. This ensures efficient sharing of resources for the server receiving requests and the client making requests.

## Logging

Spring Boot has no mandatory logging dependency, except for the Commons Logging API, which is typically provided by Spring Framework’s spring-jcl module. To use [Logback](https://logback.qos.ch/), you need to include it and spring-jcl on the classpath. The recommended way to do that is through the starters, which all depend on spring-boot-starter-logging. For a web application, you need only spring-boot- starter-web, since it depends transitively on the logging starter. If you use Maven, the following dependency adds logging for you:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

Spring Boot has a LoggingSystem abstraction that attempts to configure logging based on the content of the classpath. If Logback is available, it is the first choice.

If the only change you need to make to logging is to set the levels of various loggers, you can do so in application.properties by using the "logging.level" prefix, as shown in the following example:

logging.level.org.springframework.web=DEBUG logging.level.org.hibernate=ERROR

You can also set the location of a file to which to write the log (in addition to the console) by using "logging.file.name".

To configure the more fine-grained settings of a logging system, you need to use the native configuration format supported by the LoggingSystem in question. By default, Spring Boot picks up the native configuration from its default location for the system (such as classpath:logback.xml for Logback), but you can set the location of the config file by using the logging.config property.

### Configure Logback for Logging

If you need to apply customizations to logback beyond those that can be achieved with application.properties, you’ll need to add a standard logback configuration file. You can add a logback.xml file to the root of your classpath for logback to find. You can also use logback-spring.xml if you want to use the [Spring Boot Logback extensions](#_bookmark138).

The Logback documentation has a [dedicated section that covers configuration](https://logback.qos.ch/manual/configuration.html) in some detail.

Spring Boot provides a number of logback configurations that be included from your own configuration. These includes are designed to allow certain common Spring Boot conventions to be re-applied.

The following files are provided under org/springframework/boot/logging/logback/:

* + - * defaults.xml - Provides conversion rules, pattern properties and common logger configurations.
      * console-appender.xml - Adds a ConsoleAppender using the CONSOLE\_LOG\_PATTERN.
      * file-appender.xml - Adds a RollingFileAppender using the FILE\_LOG\_PATTERN and

ROLLING\_FILE\_NAME\_PATTERN with appropriate settings.

In addition, a legacy base.xml file is provided for compatibility with earlier versions of Spring Boot. A typical custom logback.xml file would look something like this:

<?xml version="1.0" encoding="UTF-8"?>

<configuration>

<include resource="org/springframework/boot/logging/logback/defaults.xml"/>

<include resource="org/springframework/boot/logging/logback/console-appender.xml"

/>

<root level="INFO">

<appender-ref ref="CONSOLE" />

</root>

<logger name="org.springframework.web" level="DEBUG"/>

</configuration>

Your logback configuration file can also make use of System properties that the LoggingSystem takes care of creating for you:

* + - * ${PID}: The current process ID.
      * ${LOG\_FILE}: Whether logging.file.name was set in Boot’s external configuration.
      * ${LOG\_PATH}: Whether logging.file.path (representing a directory for log files to live in) was set in Boot’s external configuration.
      * ${LOG\_EXCEPTION\_CONVERSION\_WORD}: Whether logging.exception-conversion-word was set in Boot’s external configuration.
      * ${ROLLING\_FILE\_NAME\_PATTERN}: Whether logging.pattern.rolling-file-name was set in Boot’s external configuration.

Spring Boot also provides some nice ANSI color terminal output on a console (but not in a log file) by using a custom Logback converter. See the CONSOLE\_LOG\_PATTERN in the defaults.xml configuration for an example.

If Groovy is on the classpath, you should be able to configure Logback with logback.groovy as well. If present, this setting is given preference.

Spring extensions are not supported with Groovy configuration. Any logback- spring.groovy files will not be detected.

#### Configure Logback for File-only Output

If you want to disable console logging and write output only to a file, you need a custom logback- spring.xml that imports file-appender.xml but not console-appender.xml, as shown in the following

example:

<?xml version="1.0" encoding="UTF-8"?>

<configuration>

<include resource="org/springframework/boot/logging/logback/defaults.xml" />

<property name="LOG\_FILE" value="${LOG\_FILE:-${LOG\_PATH:-${LOG\_TEMP:-

${java.io.tmpdir:-/tmp}}/}spring.log}"/>

<include resource="org/springframework/boot/logging/logback/file-appender.xml" />

<root level="INFO">

<appender-ref ref="FILE" />

</root>

</configuration>

You also need to add logging.file.name to your application.properties, as shown in the following example:

logging.file.name=myapplication.log

### Configure Log4j for Logging

Spring Boot supports [Log4j 2](https://logging.apache.org/log4j/2.x/) for logging configuration if it is on the classpath. If you use the starters for assembling dependencies, you have to exclude Logback and then include log4j 2 instead. If you do not use the starters, you need to provide (at least) spring-jcl in addition to Log4j 2.

The recommended path is through the starters, even though it requires some jiggling. The following example shows how to set up the starters in Maven:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

<exclusions>

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-logging</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-log4j2</artifactId>

</dependency>

And the following example shows one way to set up the starters in Gradle:

dependencies {

compile 'org.springframework.boot:spring-boot-starter-web'

}

configurations.all { resolutionStrategy.dependencySubstitution.all { dependency ->

if (dependency.requested instanceof ModuleComponentSelector && dependency.requested.module == 'spring-boot-starter-logging') {

dependency.useTarget("org.springframework.boot:spring-boot-starter- log4j2:$dependency.requested.version", 'Use Log4j2 instead of Logback')

}

}

}

}

The Log4j starters gather together the dependencies for common logging requirements (such as having Tomcat use java.util.logging but configuring the output using Log4j 2).

To ensure that debug logging performed using java.util.logging is routed into Log4j 2, configure its [JDK logging adapter](https://logging.apache.org/log4j/2.0/log4j-jul/index.html) by setting the java.util.logging.manager system property to org.apache.logging.log4j.jul.LogManager.

#### Use YAML or JSON to Configure Log4j 2

In addition to its default XML configuration format, Log4j 2 also supports YAML and JSON configuration files. To configure Log4j 2 to use an alternative configuration file format, add the appropriate dependencies to the classpath and name your configuration files to match your chosen file format, as shown in the following example:

|  |  |  |
| --- | --- | --- |
| **Format** | **Dependencies** | **File names** |
| YAML | com.fasterxml.jackson.core:jackson-databind +  com.fasterxml.jackson.dataformat:jackson-dataformat-yaml | log4j2.yaml +  log4j2.yml |
| JSON | com.fasterxml.jackson.core:jackson-databind | log4j2.json +  log4j2.jsn |

## Data Access

Spring Boot includes a number of starters for working with data sources. This section answers questions related to doing so.

### Configure a Custom DataSource

To configure your own DataSource, define a @Bean of that type in your configuration. Spring Boot

reuses your DataSource anywhere one is required, including database initialization. If you need to externalize some settings, you can bind your DataSource to the environment (see “[Third-party](#_bookmark119) [Configuration](#_bookmark119)”).

The following example shows how to define a data source in a bean:

@Bean @ConfigurationProperties(prefix="app.datasource") public DataSource dataSource() {

return new FancyDataSource();

}

The following example shows how to define a data source by setting properties:

app.datasource.url=jdbc:h2:mem:mydb app.datasource.username=sa app.datasource.pool-size=30

Assuming that your FancyDataSource has regular JavaBean properties for the URL, the username, and the pool size, these settings are bound automatically before the DataSource is made available to other components. The regular [database initialization](#_bookmark609) also happens (so the relevant sub-set of spring.datasource.\* can still be used with your custom configuration).

Spring Boot also provides a utility builder class, called DataSourceBuilder, that can be used to create one of the standard data sources (if it is on the classpath). The builder can detect the one to use based on what’s available on the classpath. It also auto-detects the driver based on the JDBC URL.

The following example shows how to create a data source by using a DataSourceBuilder:

@Bean @ConfigurationProperties("app.datasource") public DataSource dataSource() {

return DataSourceBuilder.create().build();

}

To run an app with that DataSource, all you need is the connection information. Pool-specific settings can also be provided. Check the implementation that is going to be used at runtime for more details.

The following example shows how to define a JDBC data source by setting properties:

app.datasource.url=jdbc:mysql://localhost/test app.datasource.username=dbuser app.datasource.password=dbpass app.datasource.pool-size=30

However, there is a catch. Because the actual type of the connection pool is not exposed, no keys are generated in the metadata for your custom DataSource and no completion is available in your IDE (because the DataSource interface exposes no properties). Also, if you happen to have Hikari on the classpath, this basic setup does not work, because Hikari has no url property (but does have a jdbcUrl property). In that case, you must rewrite your configuration as follows:

app.datasource.jdbc-url=jdbc:mysql://localhost/test app.datasource.username=dbuser app.datasource.password=dbpass app.datasource.maximum-pool-size=30

You can fix that by forcing the connection pool to use and return a dedicated implementation rather than DataSource. You cannot change the implementation at runtime, but the list of options will be explicit.

The following example shows how create a HikariDataSource with DataSourceBuilder:

@Bean @ConfigurationProperties("app.datasource") public HikariDataSource dataSource() {

return DataSourceBuilder.create().type(HikariDataSource.class).build();

}

You can even go further by leveraging what DataSourceProperties does for you — that is, by providing a default embedded database with a sensible username and password if no URL is provided. You can easily initialize a DataSourceBuilder from the state of any DataSourceProperties object, so you could also inject the DataSource that Spring Boot creates automatically. However, that would split your configuration into two namespaces: url, username, password, type, and driver on spring.datasource and the rest on your custom namespace (app.datasource). To avoid that, you can redefine a custom DataSourceProperties on your custom namespace, as shown in the following example:

@Bean @Primary

@ConfigurationProperties("app.datasource")

public DataSourceProperties dataSourceProperties() { return new DataSourceProperties();

}

@Bean @ConfigurationProperties("app.datasource.configuration")

public HikariDataSource dataSource(DataSourceProperties properties) { return

properties.initializeDataSourceBuilder().type(HikariDataSource.class).build();

}

This setup puts you *in sync* with what Spring Boot does for you by default, except that a dedicated

connection pool is chosen (in code) and its settings are exposed in the app.datasource.configuration sub namespace. Because DataSourceProperties is taking care of the url/jdbcUrl translation for you, you can configure it as follows:

app.datasource.url=jdbc:mysql://localhost/test app.datasource.username=dbuser app.datasource.password=dbpass app.datasource.configuration.maximum-pool-size=30

Spring Boot will expose Hikari-specific settings to spring.datasource.hikari. This example uses a more generic configuration sub namespace as the example does not support multiple datasource implementations.

Because your custom configuration chooses to go with Hikari, app.datasource.type has no effect. In practice, the builder is initialized with whatever value you might set there and then overridden by the call to .type().

See “[Configure a DataSource](#_bookmark199)” in the “Spring Boot features” section and the

[DataSourceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/DataSourceAutoConfiguration.java) class for more details.

### Configure Two DataSources

If you need to configure multiple data sources, you can apply the same tricks that are described in the previous section. You must, however, mark one of the DataSource instances as @Primary, because various auto-configurations down the road expect to be able to get one by type.

If you create your own DataSource, the auto-configuration backs off. In the following example, we provide the *exact* same feature set as the auto-configuration provides on the primary data source:

@Bean @Primary

@ConfigurationProperties("app.datasource.first")

public DataSourceProperties firstDataSourceProperties() { return new DataSourceProperties();

}

@Bean @Primary

@ConfigurationProperties("app.datasource.first.configuration") public HikariDataSource firstDataSource() {

return firstDataSourceProperties().initializeDataSourceBuilder().type(HikariDataSource.class)

.build();

}

@Bean @ConfigurationProperties("app.datasource.second") public BasicDataSource secondDataSource() {

return DataSourceBuilder.create().type(BasicDataSource.class).build();

}

firstDataSourceProperties has to be flagged as @Primary so that the database initializer feature uses your copy (if you use the initializer).

Both data sources are also bound for advanced customizations. For instance, you could configure them as follows:

app.datasource.first.url=jdbc:mysql://localhost/first app.datasource.first.username=dbuser app.datasource.first.password=dbpass app.datasource.first.configuration.maximum-pool-size=30

app.datasource.second.url=jdbc:mysql://localhost/second app.datasource.second.username=dbuser app.datasource.second.password=dbpass app.datasource.second.max-total=30

You can apply the same concept to the secondary DataSource as well, as shown in the following example:

@Bean @Primary

@ConfigurationProperties("app.datasource.first")

public DataSourceProperties firstDataSourceProperties() { return new DataSourceProperties();

}

@Bean @Primary

@ConfigurationProperties("app.datasource.first.configuration") public HikariDataSource firstDataSource() {

return firstDataSourceProperties().initializeDataSourceBuilder().type(HikariDataSource.class)

.build();

}

@Bean @ConfigurationProperties("app.datasource.second")

public DataSourceProperties secondDataSourceProperties() { return new DataSourceProperties();

}

@Bean @ConfigurationProperties("app.datasource.second.configuration") public BasicDataSource secondDataSource() {

return secondDataSourceProperties().initializeDataSourceBuilder().type(BasicDataSource.class)

.build();

}

The preceding example configures two data sources on custom namespaces with the same logic as Spring Boot would use in auto-configuration. Note that each configuration sub namespace provides advanced settings based on the chosen implementation.

### Use Spring Data Repositories

Spring Data can create implementations of @Repository interfaces of various flavors. Spring Boot handles all of that for you, as long as those @Repositories are included in the same package (or a sub-package) of your @EnableAutoConfiguration class.

For many applications, all you need is to put the right Spring Data dependencies on your classpath. There is a spring-boot-starter-data-jpa for JPA, spring-boot-starter-data-mongodb for Mongodb, etc. To get started, create some repository interfaces to handle your @Entity objects.

Spring Boot tries to guess the location of your @Repository definitions, based on the @EnableAutoConfiguration it finds. To get more control, use the @EnableJpaRepositories annotation (from Spring Data JPA).

For more about Spring Data, see the [Spring Data project page](https://spring.io/projects/spring-data).

### Separate @Entity Definitions from Spring Configuration

Spring Boot tries to guess the location of your @Entity definitions, based on the @EnableAutoConfiguration it finds. To get more control, you can use the @EntityScan annotation, as shown in the following example:

@Configuration(proxyBeanMethods = false) @EnableAutoConfiguration @EntityScan(basePackageClasses=City.class) public class Application {

//...

}

### Configure JPA Properties

Spring Data JPA already provides some vendor-independent configuration options (such as those for SQL logging), and Spring Boot exposes those options and a few more for Hibernate as external configuration properties. Some of them are automatically detected according to the context so you should not have to set them.

The spring.jpa.hibernate.ddl-auto is a special case, because, depending on runtime conditions, it has different defaults. If an embedded database is used and no schema manager (such as Liquibase or Flyway) is handling the DataSource, it defaults to create-drop. In all other cases, it defaults to none.

The dialect to use is detected by the JPA provider. If you prefer to set the dialect yourself, set the

spring.jpa.database-platform property.

The most common options to set are shown in the following example:

spring.jpa.hibernate.naming.physical-strategy=com.example.MyPhysicalNamingStrategy spring.jpa.show-sql=true

In addition, all properties in spring.jpa.properties.\* are passed through as normal JPA properties (with the prefix stripped) when the local EntityManagerFactory is created.

You need to ensure that names defined under spring.jpa.properties.\* exactly match those expected by your JPA provider. Spring Boot will not attempt any kind of relaxed binding for these entries.

For example, if you want to configure Hibernate’s batch size you must use spring.jpa.properties.hibernate.jdbc.batch\_size. If you use other forms, such as batchSize or batch-size, Hibernate will not apply the setting.

If you need to apply advanced customization to Hibernate properties, consider registering a HibernatePropertiesCustomizer bean that will be invoked prior to creating the EntityManagerFactory. This takes precedence to anything that is applied by the auto-configuration.

### Configure Hibernate Naming Strategy

Hibernate uses [two different naming strategies](https://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html#naming) to map names from the object model to the corresponding database names. The fully qualified class name of the physical and the implicit strategy implementations can be configured by setting the spring.jpa.hibernate.naming.physical- strategy and spring.jpa.hibernate.naming.implicit-strategy properties, respectively. Alternatively, if ImplicitNamingStrategy or PhysicalNamingStrategy beans are available in the application context, Hibernate will be automatically configured to use them.

By default, Spring Boot configures the physical naming strategy with SpringPhysicalNamingStrategy. This implementation provides the same table structure as Hibernate 4: all dots are replaced by underscores and camel casing is replaced by underscores as well. By default, all table names are generated in lower case, but it is possible to override that flag if your schema requires it.

For example, a TelephoneNumber entity is mapped to the telephone\_number table. If you prefer to use Hibernate 5’s default instead, set the following property:

spring.jpa.hibernate.naming.physical- strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl

Alternatively, you can configure the following bean:

@Bean

public PhysicalNamingStrategy physicalNamingStrategy() { return new PhysicalNamingStrategyStandardImpl();

}

See [HibernateJpaAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/orm/jpa/HibernateJpaAutoConfiguration.java) and [JpaBaseConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/orm/jpa/JpaBaseConfiguration.java) for more details.

### Configure Hibernate Second-Level Caching

Hibernate [second-level cache](https://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html#caching) can be configured for a range of cache providers. Rather than configuring Hibernate to lookup the cache provider again, it is better to provide the one that is available in the context whenever possible.

To do this with JCache, first make sure that org.hibernate:hibernate-jcache is available on the classpath. Then, add a HibernatePropertiesCustomizer bean as shown in the following example:

@Configuration(proxyBeanMethods = false)

public class HibernateSecondLevelCacheExample {

@Bean

public HibernatePropertiesCustomizer hibernateSecondLevelCacheCustomizer(JCacheCacheManager cacheManager) {

return (properties) -> properties.put(ConfigSettings.CACHE\_MANAGER, cacheManager.getCacheManager());

}

}

This customizer will configure Hibernate to use the same CacheManager as the one that the application uses. It is also possible to use separate CacheManager instances. For details, refer to [the](https://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html#caching-provider-jcache) [Hibernate user guide](https://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html#caching-provider-jcache).

### Use Dependency Injection in Hibernate Components

By default, Spring Boot registers a BeanContainer implementation that uses the BeanFactory so that converters and entity listeners can use regular dependency injection.

You can disable or tune this behaviour by registering a HibernatePropertiesCustomizer that removes or changes the hibernate.resource.beans.container property.

### Use a Custom EntityManagerFactory

To take full control of the configuration of the EntityManagerFactory, you need to add a @Bean named ‘entityManagerFactory’. Spring Boot auto-configuration switches off its entity manager in the presence of a bean of that type.

### Use Two EntityManagers

Even if the default EntityManagerFactory works fine, you need to define a new one, otherwise the presence of the second bean of that type switches off the default. You can use the EntityManagerBuilder provided by Spring Boot to help you to create one. Alternatively, you can use the LocalContainerEntityManagerFactoryBean directly from Spring ORM, as shown in the following example:

// add two data sources configured as above

@Bean

public LocalContainerEntityManagerFactoryBean customerEntityManagerFactory( EntityManagerFactoryBuilder builder) {

return builder

.dataSource(customerDataSource())

.packages(Customer.class)

.persistenceUnit("customers")

.build();

}

@Bean

public LocalContainerEntityManagerFactoryBean orderEntityManagerFactory( EntityManagerFactoryBuilder builder) {

return builder

.dataSource(orderDataSource())

.packages(Order.class)

.persistenceUnit("orders")

.build();

}

When you create a bean for LocalContainerEntityManagerFactoryBean yourself, any customization that was applied during the creation of the auto-configured LocalContainerEntityManagerFactoryBean is lost. For example, in case of Hibernate, any properties under the spring.jpa.hibernate prefix will not be automatically applied to your LocalContainerEntityManagerFactoryBean. If you were relying on these properties for configuring things like the naming strategy or the DDL mode, you will need to explicitly configure that when creating the LocalContainerEntityManagerFactoryBean bean. On the other hand, properties that get applied to the auto-configured EntityManagerFactoryBuilder, which are specified via spring.jpa.properties, will automatically be applied, provided you use the auto-configured EntityManagerFactoryBuilder to build the LocalContainerEntityManagerFactoryBean bean.

The configuration above almost works on its own. To complete the picture, you need to configure TransactionManagers for the two EntityManagers as well. If you mark one of them as @Primary, it could be picked up by the default JpaTransactionManager in Spring Boot. The other would have to be explicitly injected into a new instance. Alternatively, you might be able to use a JTA transaction manager that spans both.

If you use Spring Data, you need to configure @EnableJpaRepositories accordingly, as shown in the following example:

@Configuration(proxyBeanMethods = false) @EnableJpaRepositories(basePackageClasses = Customer.class,

entityManagerFactoryRef = "customerEntityManagerFactory") public class CustomerConfiguration {

...

}

@Configuration(proxyBeanMethods = false) @EnableJpaRepositories(basePackageClasses = Order.class,

entityManagerFactoryRef = "orderEntityManagerFactory") public class OrderConfiguration {

...

}

### Use a Traditional persistence.xml File

Spring Boot will not search for or use a META-INF/persistence.xml by default. If you prefer to use a traditional persistence.xml, you need to define your own @Bean of type LocalEntityManagerFactoryBean (with an ID of ‘entityManagerFactory’) and set the persistence unit name there.

See [JpaBaseConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/orm/jpa/JpaBaseConfiguration.java) for the default settings.

### Use Spring Data JPA and Mongo Repositories

Spring Data JPA and Spring Data Mongo can both automatically create Repository implementations for you. If they are both present on the classpath, you might have to do some extra configuration to tell Spring Boot which repositories to create. The most explicit way to do that is to use the standard Spring Data @EnableJpaRepositories and @EnableMongoRepositories annotations and provide the location of your Repository interfaces.

There are also flags (spring.data.\*.repositories.enabled and spring.data.\*.repositories.type) that you can use to switch the auto-configured repositories on and off in external configuration. Doing so is useful, for instance, in case you want to switch off the Mongo repositories and still use the auto-configured MongoTemplate.

The same obstacle and the same features exist for other auto-configured Spring Data repository types (Elasticsearch, Solr, and others). To work with them, change the names of the annotations and flags accordingly.

### Customize Spring Data’s Web Support

Spring Data provides web support that simplifies the use of Spring Data repositories in a web application. Spring Boot provides properties in the spring.data.web namespace for customizing its configuration. Note that if you are using Spring Data REST, you must use the properties in the spring.data.rest namespace instead.

### Expose Spring Data Repositories as REST Endpoint

Spring Data REST can expose the Repository implementations as REST endpoints for you, provided Spring MVC has been enabled for the application.

Spring Boot exposes a set of useful properties (from the spring.data.rest namespace) that customize the [RepositoryRestConfiguration](https://docs.spring.io/spring-data/rest/docs/3.3.4.RELEASE/api/org/springframework/data/rest/core/config/RepositoryRestConfiguration.html). If you need to provide additional customization, you should use a [RepositoryRestConfigurer](https://docs.spring.io/spring-data/rest/docs/3.3.4.RELEASE/api/org/springframework/data/rest/webmvc/config/RepositoryRestConfigurer.html) bean.

If you do not specify any order on your custom RepositoryRestConfigurer, it runs after the one Spring Boot uses internally. If you need to specify an order, make sure it is higher than 0.

### Configure a Component that is Used by JPA

If you want to configure a component that JPA uses, then you need to ensure that the component is initialized before JPA. When the component is auto-configured, Spring Boot takes care of this for you. For example, when Flyway is auto-configured, Hibernate is configured to depend upon Flyway so that Flyway has a chance to initialize the database before Hibernate tries to use it.

If you are configuring a component yourself, you can use an EntityManagerFactoryDependsOnPostProcessor subclass as a convenient way of setting up the necessary dependencies. For example, if you use Hibernate Search with Elasticsearch as its index manager, any EntityManagerFactory beans must be configured to depend on the elasticsearchClient bean, as shown in the following example:

/\*\*

* {@link EntityManagerFactoryDependsOnPostProcessor} that ensures that
* {@link EntityManagerFactory} beans depend on the {@code elasticsearchClient} bean.

\*/ @Component

static class ElasticsearchEntityManagerFactoryDependsOnPostProcessor extends EntityManagerFactoryDependsOnPostProcessor {

ElasticsearchEntityManagerFactoryDependsOnPostProcessor() { super("elasticsearchClient");

}

}

### Configure jOOQ with Two DataSources

If you need to use jOOQ with multiple data sources, you should create your own DSLContext for each one. Refer to [JooqAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jooq/JooqAutoConfiguration.java) for more details.

In particular, JooqExceptionTranslator and SpringTransactionProvider can be reused to provide similar features to what the auto-configuration does with a single DataSource.

## Database Initialization

An SQL database can be initialized in different ways depending on what your stack is. Of course, you can also do it manually, provided the database is a separate process. It is recommended to use a single mechanism for schema generation.

### Initialize a Database Using JPA

JPA has features for DDL generation, and these can be set up to run on startup against the database. This is controlled through two external properties:

* + - * spring.jpa.generate-ddl (boolean) switches the feature on and off and is vendor independent.
      * spring.jpa.hibernate.ddl-auto (enum) is a Hibernate feature that controls the behavior in a more fine-grained way. This feature is described in more detail later in this guide.

### Initialize a Database Using Hibernate

You can set spring.jpa.hibernate.ddl-auto explicitly and the standard Hibernate property values are none, validate, update, create, and create-drop. Spring Boot chooses a default value for you based on whether it thinks your database is embedded. It defaults to create-drop if no schema manager has been detected or none in all other cases. An embedded database is detected by looking at the Connection type. hsqldb, h2, and derby are embedded, and others are not. Be careful when switching from in-memory to a ‘real’ database that you do not make assumptions about the existence of the tables and data in the new platform. You either have to set ddl-auto explicitly or use one of the other mechanisms to initialize the database.

You can output the schema creation by enabling the org.hibernate.SQL logger. This is done for you automatically if you enable the [debug mode](#_bookmark132).

In addition, a file named import.sql in the root of the classpath is executed on startup if Hibernate creates the schema from scratch (that is, if the ddl-auto property is set to create or create-drop). This can be useful for demos and for testing if you are careful but is probably not something you want to be on the classpath in production. It is a Hibernate feature (and has nothing to do with Spring).

### Initialize a Database using basic SQL scripts

Spring Boot can automatically create the schema (DDL scripts) of your DataSource and initialize it (DML scripts). It loads SQL from the standard root classpath locations: schema.sql and data.sql, respectively. In addition, Spring Boot processes the schema-${platform}.sql and data-

${platform}.sql files (if present), where platform is the value of spring.datasource.platform. This allows you to switch to database-specific scripts if necessary. For example, you might choose to set it to the vendor name of the database (hsqldb, h2, oracle, mysql, postgresql, and so on).

When only basic SQL scripts are used, Spring Boot automatically creates the schema of an embedded DataSource. This behavior can be customized by using the spring.datasource.initialization-mode property. For instance, if you want to always initialize the DataSource regardless of its type:

spring.datasource.initialization-mode=always

In a JPA-based app, you can choose to let Hibernate create the schema or use schema.sql, but you cannot do both. Make sure to disable spring.jpa.hibernate.ddl-auto if you use schema.sql.

spring.jpa.hibernate.ddl-auto=none

If you are using a [Higher-level Database Migration Tool](#_bookmark612), like Flyway or Liquibase, you should use them alone to create and initialize the schema. Using the basic schema.sql and data.sql scripts alongside Flyway or Liquibase is not recommended and support will be removed in a future release.

By default, Spring Boot enables the fail-fast feature of the Spring JDBC initializer. This means that, if the scripts cause exceptions, the application fails to start. You can tune that behavior by setting spring.datasource.continue-on-error.

### Initialize a Database Using R2DBC

If you are using R2DBC, the regular DataSource auto-configuration backs off so none of the options described above can be used.

If you are using Spring Data R2DBC, you can initialize the database on startup using SQL scripts as shown in the following example:

@Configuration(proxyBeanMethods = false)

static class DatabaseInitializationConfiguration {

@Autowired

void initializeDatabase(ConnectionFactory connectionFactory) { ResourceLoader resourceLoader = new DefaultResourceLoader(); Resource[] scripts = new Resource[] {

resourceLoader.getResource("classpath:schema.sql"),

resourceLoader.getResource("classpath:data.sql") };

new ResourceDatabasePopulator(scripts).execute(connectionFactory).block();

}

}

Alternatively, you can configure either [Flyway](#_bookmark613) or [Liquibase](#_bookmark614) to configure a DataSource for you for the duration of the migration. Both these libraries offer properties to set the url, username and

password of the database to migrate.

When choosing this option, org.springframework:spring-jdbc is still a required dependency.

### Initialize a Spring Batch Database

If you use Spring Batch, it comes pre-packaged with SQL initialization scripts for most popular database platforms. Spring Boot can detect your database type and execute those scripts on startup. If you use an embedded database, this happens by default. You can also enable it for any database type, as shown in the following example:

spring.batch.initialize-schema=always

You can also switch off the initialization explicitly by setting spring.batch.initialize-schema=never.

### Use a Higher-level Database Migration Tool

Spring Boot supports two higher-level migration tools: [Flyway](https://flywaydb.org/) and [Liquibase](https://www.liquibase.org/).

#### Execute Flyway Database Migrations on Startup

To automatically run Flyway database migrations on startup, add the org.flywaydb:flyway-core to your classpath.

Typically, migrations are scripts in the form V<VERSION> <NAME>.sql (with <VERSION> an underscore- separated version, such as ‘1’ or ‘2\_1’). By default, they are in a directory called classpath:db/migration, but you can modify that location by setting spring.flyway.locations. This is a comma-separated list of one or more classpath: or filesystem: locations. For example, the following configuration would search for scripts in both the default classpath location and the

/opt/migration directory:

spring.flyway.locations=classpath:db/migration,filesystem:/opt/migration

You can also add a special {vendor} placeholder to use vendor-specific scripts. Assume the following:

spring.flyway.locations=classpath:db/migration/{vendor}

Rather than using db/migration, the preceding configuration sets the directory to use according to the type of the database (such as db/migration/mysql for MySQL). The list of supported databases is available in [DatabaseDriver](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/jdbc/DatabaseDriver.java).

Migrations can also be written in Java. Flyway will be auto-configured with any beans that implement JavaMigration.

[FlywayProperties](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/flyway/FlywayProperties.java) provides most of Flyway’s settings and a small set of additional properties that can be used to disable the migrations or switch off the location checking. If you need more control over the configuration, consider registering a FlywayConfigurationCustomizer bean.

Spring Boot calls Flyway.migrate() to perform the database migration. If you would like more control, provide a @Bean that implements [FlywayMigrationStrategy](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/flyway/FlywayMigrationStrategy.java).

Flyway supports SQL and Java [callbacks](https://flywaydb.org/documentation/callbacks.html). To use SQL-based callbacks, place the callback scripts in the classpath:db/migration directory. To use Java-based callbacks, create one or more beans that implement Callback. Any such beans are automatically registered with Flyway. They can be ordered by using @Order or by implementing Ordered. Beans that implement the deprecated FlywayCallback interface can also be detected, however they cannot be used alongside Callback beans.

By default, Flyway autowires the (@Primary) DataSource in your context and uses that for migrations. If you like to use a different DataSource, you can create one and mark its @Bean as @FlywayDataSource. If you do so and want two data sources, remember to create another one and mark it as @Primary. Alternatively, you can use Flyway’s native DataSource by setting spring.flyway.[url,user,password] in external properties. Setting either spring.flyway.url or spring.flyway.user is sufficient to cause Flyway to use its own DataSource. If any of the three properties has not been set, the value of its equivalent spring.datasource property will be used.

You can also use Flyway to provide data for specific scenarios. For example, you can place test- specific migrations in src/test/resources and they are run only when your application starts for testing. Also, you can use profile-specific configuration to customize spring.flyway.locations so that certain migrations run only when a particular profile is active. For example, in application- dev.properties, you might specify the following setting:

spring.flyway.locations=classpath:/db/migration,classpath:/dev/db/migration

With that setup, migrations in dev/db/migration run only when the dev profile is active.

#### Execute Liquibase Database Migrations on Startup

To automatically run Liquibase database migrations on startup, add the org.liquibase:liquibase- core to your classpath.

When you add the org.liquibase:liquibase-core to your classpath, database migrations run by default for both during application startup and before your tests run. This behavior can be customized by using the spring.liquibase.enabled property, setting different values in the main and test configurations. It is not possible to use two different ways to initialize the database (e.g. Liquibase for application startup, JPA for test runs).

By default, the master change log is read from db/changelog/db.changelog-master.yaml, but you can change the location by setting spring.liquibase.change-log. In addition to YAML, Liquibase also supports JSON, XML, and SQL change log formats.

By default, Liquibase autowires the (@Primary) DataSource in your context and uses that for

migrations. If you need to use a different DataSource, you can create one and mark its @Bean as @LiquibaseDataSource. If you do so and you want two data sources, remember to create another one and mark it as @Primary. Alternatively, you can use Liquibase’s native DataSource by setting spring.liquibase.[url,user,password] in external properties. Setting either spring.liquibase.url or spring.liquibase.user is sufficient to cause Liquibase to use its own DataSource. If any of the three properties has not been set, the value of its equivalent spring.datasource property will be used.

See [LiquibaseProperties](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/liquibase/LiquibaseProperties.java) for details about available settings such as contexts, the default schema, and others.

## Messaging

Spring Boot offers a number of starters that include messaging. This section answers questions that arise from using messaging with Spring Boot.

### Disable Transacted JMS Session

If your JMS broker does not support transacted sessions, you have to disable the support of transactions altogether. If you create your own JmsListenerContainerFactory, there is nothing to do, since, by default it cannot be transacted. If you want to use the DefaultJmsListenerContainerFactoryConfigurer to reuse Spring Boot’s default, you can disable transacted sessions, as follows:

@Bean

public DefaultJmsListenerContainerFactory jmsListenerContainerFactory( ConnectionFactory connectionFactory, DefaultJmsListenerContainerFactoryConfigurer configurer) {

DefaultJmsListenerContainerFactory listenerFactory = new DefaultJmsListenerContainerFactory();

configurer.configure(listenerFactory, connectionFactory); listenerFactory.setTransactionManager(null); listenerFactory.setSessionTransacted(false);

return listenerFactory;

}

The preceding example overrides the default factory, and it should be applied to any other factory that your application defines, if any.

## Batch Applications

A number of questions often arise when people use Spring Batch from within a Spring Boot application. This section addresses those questions.

### Specifying a Batch Data Source

By default, batch applications require a DataSource to store job details. Spring Batch expects a single

DataSource by default. To have it use a DataSource other than the application’s main DataSource,

declare a DataSource bean, annotating its @Bean method with @BatchDataSource. If you do so and want two data sources, remember to mark the other one @Primary. To take greater control, implement BatchConfigurer. See [The Javadoc of @EnableBatchProcessing](https://docs.spring.io/spring-batch/docs/4.2.4.RELEASE/api/org/springframework/batch/core/configuration/annotation/EnableBatchProcessing.html) for more details.

For more info about Spring Batch, see the [Spring Batch project page](https://spring.io/projects/spring-batch).

### Running Spring Batch Jobs on Startup

Spring Batch auto-configuration is enabled by adding @EnableBatchProcessing to one of your

@Configuration classes.

By default, it executes **all** Jobs in the application context on startup (see [JobLauncherApplicationRunner](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/batch/JobLauncherApplicationRunner.java) for details). You can narrow down to a specific job or jobs by specifying spring.batch.job.names (which takes a comma-separated list of job name patterns).

See [BatchAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/batch/BatchAutoConfiguration.java) and [@EnableBatchProcessing](https://docs.spring.io/spring-batch/docs/4.2.4.RELEASE/api/org/springframework/batch/core/configuration/annotation/EnableBatchProcessing.html) for more details.

### Running from the Command Line

Spring Boot converts any command line argument starting with -- to a property to add to the Environment, see [accessing command line properties](#_bookmark104). This should not be used to pass arguments to batch jobs. To specify batch arguments on the command line, use the regular format (i.e. without --

), as shown in the following example:

$ java -jar myapp.jar someParameter=someValue anotherParameter=anotherValue

If you specify a property of the Environment on the command line, it is ignored by the job. Consider the following command:

$ java -jar myapp.jar --server.port=7070 someParameter=someValue

This provides only one argument to the batch job: someParameter=someValue.

### Storing the Job Repository

Spring Batch requires a data store for the Job repository. If you use Spring Boot, you must use an actual database. Note that it can be an in-memory database, see [Configuring a Job Repository](https://docs.spring.io/spring-batch/docs/4.2.4.RELEASE/reference/html/job.html#configuringJobRepository).

## Actuator

Spring Boot includes the Spring Boot Actuator. This section answers questions that often arise from its use.

### Change the HTTP Port or Address of the Actuator Endpoints

In a standalone application, the Actuator HTTP port defaults to the same as the main HTTP port. To make the application listen on a different port, set the external property: management.server.port. To

listen on a completely different network address (such as when you have an internal network for management and an external one for user applications), you can also set management.server.address to a valid IP address to which the server is able to bind.

For more detail, see the [ManagementServerProperties](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/web/server/ManagementServerProperties.java) source code and “[Customizing the Management](#_bookmark409) [Server Port](#_bookmark409)” in the “Production-ready features” section.

### Customize the ‘whitelabel’ Error Page

Spring Boot installs a ‘whitelabel’ error page that you see in a browser client if you encounter a server error (machine clients consuming JSON and other media types should see a sensible response with the right error code).

Set server.error.whitelabel.enabled=false to switch the default error page off. Doing so restores the default of the servlet container that you are using. Note that Spring Boot still tries to resolve the error view, so you should probably add your own error page rather than disabling it completely.

Overriding the error page with your own depends on the templating technology that you use. For example, if you use Thymeleaf, you can add an error.html template. If you use FreeMarker, you can add an error.ftlh template. In general, you need a View that resolves with a name of error or a @Controller that handles the /error path. Unless you replaced some of the default configuration, you should find a BeanNameViewResolver in your ApplicationContext, so a @Bean named error would be one way of doing that. See [ErrorMvcAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/servlet/error/ErrorMvcAutoConfiguration.java) for more options.

See also the section on “[Error Handling](#_bookmark158)” for details of how to register handlers in the servlet container.

### Sanitize Sensitive Values

Information returned by the env and configprops endpoints can be somewhat sensitive so keys matching a certain pattern are sanitized by default (i.e. their values are replaced by \*\*\*\*\*\*).

The patterns to use can be customized using the management.endpoint.env.keys-to-sanitize and

management.endpoint.configprops.keys-to-sanitize respectively.

Spring Boot uses sensible defaults for such keys: any key ending with the word "password", "secret", "key", "token", "vcap\_services", "sun.java.command" is entirely sanitized. Additionally, any key that holds the word credentials as part of the key is sanitized (configured as a regular expression, i.e.

\*credentials.\*).

Furthermore, Spring Boot only sanitizes the sensitive portion of URIs for keys which end with "uri", "uris", "address", or "addresses". The sensitive portion of the URI is identified using the format

<scheme>://<username>:<password>@<host>:<port>/. For example, for the property myclient.uri=http://user1:password1@localhost:8081, the resulting sanitized value is [http://user1:\*\*\*\*\*\*@localhost:8081](http://user1/).

### Map Health Indicators to Micrometer Metrics

Spring Boot health indicators return a Status type to indicate the overall system health. If you want to monitor or alert on levels of health for a particular application, you can export these statuses as metrics via Micrometer. By default, the status codes “UP”, “DOWN”, “OUT\_OF\_SERVICE” and “UNKNOWN” are used by Spring Boot. To export these, you’ll need to convert these states to some set of numbers so that they can be used with a Micrometer Gauge.

The following example shows one way to write such an exporter:

@Configuration

public class HealthMetricsConfiguration {

public HealthMetricsConfiguration(MeterRegistry registry, HealthEndpoint healthEndpoint) {

// This example presumes common tags (such as the app) are applied elsewhere Gauge.builder("health", healthEndpoint,

this::getStatusCode).strongReference(true).register(registry);

}

private int getStatusCode(HealthEndpoint health) { Status status = health.health().getStatus(); if (Status.UP.equals(status)) {

return 3;

}

if (Status.OUT\_OF\_SERVICE.equals(status)) { return 2;

}

if (Status.DOWN.equals(status)) { return 1;

}

return 0;

}

}

## Security

This section addresses questions about security when working with Spring Boot, including questions that arise from using Spring Security with Spring Boot.

For more about Spring Security, see the [Spring Security project page](https://spring.io/projects/spring-security).

### Switch off the Spring Boot Security Configuration

If you define a @Configuration with a WebSecurityConfigurerAdapter in your application, it switches off the default webapp security settings in Spring Boot.

### Change the UserDetailsService and Add User Accounts

If you provide a @Bean of type AuthenticationManager, AuthenticationProvider, or UserDetailsService, the default @Bean for InMemoryUserDetailsManager is not created. This means you have the full feature set of Spring Security available (such as [various authentication options](https://docs.spring.io/spring-security/site/docs/5.3.4.RELEASE/reference/html5/#servlet-authentication)).

The easiest way to add user accounts is to provide your own UserDetailsService bean.

### Enable HTTPS When Running behind a Proxy Server

Ensuring that all your main endpoints are only available over HTTPS is an important chore for any application. If you use Tomcat as a servlet container, then Spring Boot adds Tomcat’s own RemoteIpValve automatically if it detects some environment settings, and you should be able to rely on the HttpServletRequest to report whether it is secure or not (even downstream of a proxy server that handles the real SSL termination). The standard behavior is determined by the presence or absence of certain request headers (x-forwarded-for and x-forwarded-proto), whose names are conventional, so it should work with most front-end proxies. You can switch on the valve by adding some entries to application.properties, as shown in the following example:

server.tomcat.remoteip.remote-ip-header=x-forwarded-for server.tomcat.remoteip.protocol-header=x-forwarded-proto

(The presence of either of those properties switches on the valve. Alternatively, you can add the

RemoteIpValve by adding a TomcatServletWebServerFactory bean.)

To configure Spring Security to require a secure channel for all (or some) requests, consider adding your own WebSecurityConfigurerAdapter that adds the following HttpSecurity configuration:

@Configuration(proxyBeanMethods = false)

public class SslWebSecurityConfigurerAdapter extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

// Customize the application security http.requiresChannel().anyRequest().requiresSecure();

}

}

## Hot Swapping

Spring Boot supports hot swapping. This section answers questions about how it works.

### Reload Static Content

There are several options for hot reloading. The recommended approach is to use [spring-boot-](#_bookmark65) [devtools](#_bookmark65), as it provides additional development-time features, such as support for fast application

restarts and LiveReload as well as sensible development-time configuration (such as template caching). Devtools works by monitoring the classpath for changes. This means that static resource changes must be "built" for the change to take effect. By default, this happens automatically in Eclipse when you save your changes. In IntelliJ IDEA, the Make Project command triggers the necessary build. Due to the [default restart exclusions](#_bookmark70), changes to static resources do not trigger a restart of your application. They do, however, trigger a live reload.

Alternatively, running in an IDE (especially with debugging on) is a good way to do development (all modern IDEs allow reloading of static resources and usually also allow hot-swapping of Java class changes).

Finally, the [Maven and Gradle plugins](#_bookmark510) can be configured (see the addResources property) to support running from the command line with reloading of static files directly from source. You can use that with an external css/js compiler process if you are writing that code with higher-level tools.

### Reload Templates without Restarting the Container

Most of the templating technologies supported by Spring Boot include a configuration option to disable caching (described later in this document). If you use the spring-boot-devtools module, these properties are [automatically configured](#_bookmark66) for you at development time.

#### Thymeleaf Templates

If you use Thymeleaf, set spring.thymeleaf.cache to false. See [ThymeleafAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/thymeleaf/ThymeleafAutoConfiguration.java) for other Thymeleaf customization options.

#### FreeMarker Templates

If you use FreeMarker, set spring.freemarker.cache to false. See [FreeMarkerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/freemarker/FreeMarkerAutoConfiguration.java) for other FreeMarker customization options.

#### Groovy Templates

If you use Groovy templates, set spring.groovy.template.cache to false. See

[GroovyTemplateAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/groovy/template/GroovyTemplateAutoConfiguration.java) for other Groovy customization options.

### Fast Application Restarts

The spring-boot-devtools module includes support for automatic application restarts. While not as fast as technologies such as [JRebel](https://www.jrebel.com/products/jrebel) it is usually significantly faster than a “cold start”. You should probably give it a try before investigating some of the more complex reload options discussed later in this document.

For more details, see the [Developer Tools](#_bookmark65) section.

### Reload Java Classes without Restarting the Container

Many modern IDEs (Eclipse, IDEA, and others) support hot swapping of bytecode. Consequently, if you make a change that does not affect class or method signatures, it should reload cleanly with no side effects.

## Build

Spring Boot includes build plugins for Maven and Gradle. This section answers common questions about these plugins.

### Generate Build Information

Both the Maven plugin and the Gradle plugin allow generating build information containing the coordinates, name, and version of the project. The plugins can also be configured to add additional properties through configuration. When such a file is present, Spring Boot auto-configures a BuildProperties bean.

To generate build information with Maven, add an execution for the build-info goal, as shown in the following example:

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<version>2.3.4.RELEASE</version>

<executions>

<execution>

<goals>

<goal>build-info</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

 See the [Spring Boot Maven Plugin documentation](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/maven-plugin/reference/html/#goals-build-info) for more details.

The following example does the same with Gradle:

springBoot {

buildInfo()

}

 See the [Spring Boot Gradle Plugin documentation](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/gradle-plugin/reference/html/#integrating-with-actuator-build-info) for more details.

### Generate Git Information

Both Maven and Gradle allow generating a git.properties file containing information about the state of your git source code repository when the project was built.

For Maven users, the spring-boot-starter-parent POM includes a pre-configured plugin to generate a git.properties file. To use it, add the following declaration to your POM:

<build>

<plugins>

<plugin>

<groupId>pl.project13.maven</groupId>

<artifactId>git-commit-id-plugin</artifactId>

</plugin>

</plugins>

</build>

Gradle users can achieve the same result by using the [gradle-git-properties](https://plugins.gradle.org/plugin/com.gorylenko.gradle-git-properties) plugin, as shown in the following example:

plugins {

id "com.gorylenko.gradle-git-properties" version "2.2.2"

}

The commit time in git.properties is expected to match the following format: yyyy-MM-dd’T’HH:mm:ssZ. This is the default format for both plugins listed above. Using this format lets the time be parsed into a Date and its format, when serialized to JSON, to be controlled by Jackson’s date serialization configuration settings.

### Customize Dependency Versions

The spring-boot-dependencies POM manages the versions of common dependencies. The Spring Boot plugins for Maven and Gradle allow these managed dependency versions to be customized using build properties.

Each Spring Boot release is designed and tested against this specific set of third- party dependencies. Overriding versions may cause compatibility issues.

To override dependency versions with Maven, see [this section](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/maven-plugin/reference/html/#using) of the Maven plugin’s documentation.

To override dependency versions in Gradle, see [this section](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/gradle-plugin/reference/html/#managing-dependencies-customizing) of the Gradle plugin’s documentation.

### Create an Executable JAR with Maven

The spring-boot-maven-plugin can be used to create an executable “fat” JAR. If you use the spring- boot-starter-parent POM, you can declare the plugin and your jars are repackaged as follows:

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

If you do not use the parent POM, you can still use the plugin. However, you must additionally add an <executions> section, as follows:

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<version>2.3.4.RELEASE</version>

<executions>

<execution>

<goals>

<goal>repackage</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

See the [plugin documentation](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/maven-plugin/reference/html/#repackage) for full usage details.

### Use a Spring Boot Application as a Dependency

Like a war file, a Spring Boot application is not intended to be used as a dependency. If your application contains classes that you want to share with other projects, the recommended approach is to move that code into a separate module. The separate module can then be depended upon by your application and other projects.

If you cannot rearrange your code as recommended above, Spring Boot’s Maven and Gradle plugins must be configured to produce a separate artifact that is suitable for use as a dependency. The executable archive cannot be used as a dependency as the [executable jar format](#_bookmark693) packages application classes in BOOT-INF/classes. This means that they cannot be found when the executable jar is used as a dependency.

To produce the two artifacts, one that can be used as a dependency and one that is executable, a classifier must be specified. This classifier is applied to the name of the executable archive, leaving the default archive for use as a dependency.

To configure a classifier of exec in Maven, you can use the following configuration:

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<classifier>exec</classifier>

</configuration>

</plugin>

</plugins>

</build>

### Extract Specific Libraries When an Executable Jar Runs

Most nested libraries in an executable jar do not need to be unpacked in order to run. However, certain libraries can have problems. For example, JRuby includes its own nested jar support, which assumes that the jruby-complete.jar is always directly available as a file in its own right.

To deal with any problematic libraries, you can flag that specific nested jars should be automatically unpacked when the executable jar first runs. Such nested jars are written beneath the temporary directory identified by the java.io.tmpdir system property.

Care should be taken to ensure that your operating system is configured so that it will not delete the jars that have been unpacked to the temporary directory while the application is still running.

For example, to indicate that JRuby should be flagged for unpacking by using the Maven Plugin, you would add the following configuration:

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<requiresUnpack>

<dependency>

<groupId>org.jruby</groupId>

<artifactId>jruby-complete</artifactId>

</dependency>

</requiresUnpack>

</configuration>

</plugin>

</plugins>

</build>

### Create a Non-executable JAR with Exclusions

Often, if you have an executable and a non-executable jar as two separate build products, the executable version has additional configuration files that are not needed in a library jar. For example, the application.yml configuration file might be excluded from the non-executable JAR.

In Maven, the executable jar must be the main artifact and you can add a classified jar for the library, as follows:

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

<plugin>

<artifactId>maven-jar-plugin</artifactId>

<executions>

<execution>

<id>lib</id>

<phase>package</phase>

<goals>

<goal>jar</goal>

</goals>

<configuration>

<classifier>lib</classifier>

<excludes>

<exclude>application.yml</exclude>

</excludes>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

### Remote Debug a Spring Boot Application Started with Maven

To attach a remote debugger to a Spring Boot application that was started with Maven, you can use the jvmArguments property of the [maven plugin](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/maven-plugin/reference/html/).

See [this example](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/maven-plugin/reference/html/#run-example-debug) for more details.

### Build an Executable Archive from Ant without Using spring-boot- antlib

To build with Ant, you need to grab dependencies, compile, and then create a jar or war archive. To make it executable, you can either use the spring-boot-antlib module or you can follow these instructions:

1. If you are building a jar, package the application’s classes and resources in a nested BOOT- INF/classes directory. If you are building a war, package the application’s classes in a nested WEB-INF/classes directory as usual.
2. Add the runtime dependencies in a nested BOOT-INF/lib directory for a jar or WEB-INF/lib for a war. Remember **not** to compress the entries in the archive.
3. Add the provided (embedded container) dependencies in a nested BOOT-INF/lib directory for a jar or WEB-INF/lib-provided for a war. Remember **not** to compress the entries in the archive.
4. Add the spring-boot-loader classes at the root of the archive (so that the Main-Class is available).
5. Use the appropriate launcher (such as JarLauncher for a jar file) as a Main-Class attribute in the manifest and specify the other properties it needs as manifest entries — principally, by setting a Start-Class property.

The following example shows how to build an executable archive with Ant:

<target name="build" depends="compile">

<jar destfile="target/${ant.project.name}-${spring-boot.version}.jar" compress="false">

<mappedresources>

<fileset dir="target/classes" />

<globmapper from="\*" to="BOOT-INF/classes/\*"/>

</mappedresources>

<mappedresources>

<fileset dir="src/main/resources" erroronmissingdir="false"/>

<globmapper from="\*" to="BOOT-INF/classes/\*"/>

</mappedresources>

<mappedresources>

<fileset dir="${lib.dir}/runtime" />

<globmapper from="\*" to="BOOT-INF/lib/\*"/>

</mappedresources>

<zipfileset src="${lib.dir}/loader/spring-boot-loader-jar-${spring- boot.version}.jar" />

<manifest>

<attribute name="Main-Class" value="org.springframework.boot.loader.JarLauncher" />

<attribute name="Start-Class" value="${start-class}" />

</manifest>

</jar>

</target>

## Traditional Deployment

Spring Boot supports traditional deployment as well as more modern forms of deployment. This section answers common questions about traditional deployment.

### Create a Deployable War File

Because Spring WebFlux does not strictly depend on the Servlet API and applications are deployed by default on an embedded Reactor Netty server, War deployment is not supported for WebFlux applications.

The first step in producing a deployable war file is to provide a SpringBootServletInitializer subclass and override its configure method. Doing so makes use of Spring Framework’s Servlet 3.0 support and lets you configure your application when it is launched by the servlet container. Typically, you should update your application’s main class to extend SpringBootServletInitializer, as shown in the following example:

@SpringBootApplication

public class Application extends SpringBootServletInitializer {

@Override

protected SpringApplicationBuilder configure(SpringApplicationBuilder application)

{

return application.sources(Application.class);

}

public static void main(String[] args) { SpringApplication.run(Application.class, args);

}

}

The next step is to update your build configuration such that your project produces a war file rather than a jar file. If you use Maven and spring-boot-starter-parent (which configures Maven’s war plugin for you), all you need to do is to modify pom.xml to change the packaging to war, as follows:

<packaging>war</packaging>

If you use Gradle, you need to modify build.gradle to apply the war plugin to the project, as follows:

apply plugin: 'war'

The final step in the process is to ensure that the embedded servlet container does not interfere with the servlet container to which the war file is deployed. To do so, you need to mark the embedded servlet container dependency as being provided.

If you use Maven, the following example marks the servlet container (Tomcat, in this case) as being provided:

<dependencies>

<!-- … -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

<scope>provided</scope>

</dependency>

<!-- … -->

</dependencies>

If you use Gradle, the following example marks the servlet container (Tomcat, in this case) as being provided:

dependencies {

// …

providedRuntime 'org.springframework.boot:spring-boot-starter-tomcat'

// …

}

providedRuntime is preferred to Gradle’s compileOnly configuration. Among other limitations, compileOnly dependencies are not on the test classpath, so any web- based integration tests fail.

If you use the [Spring Boot build tools](#_bookmark510), marking the embedded servlet container dependency as provided produces an executable war file with the provided dependencies packaged in a lib- provided directory. This means that, in addition to being deployable to a servlet container, you can also run your application by using java -jar on the command line.

### Convert an Existing Application to Spring Boot

To convert an existing non-web Spring application to a Spring Boot application, replace the code that creates your ApplicationContext and replace it with calls to SpringApplication or SpringApplicationBuilder. Spring MVC web applications are generally amenable to first creating a deployable war application and then migrating it later to an executable war or jar. See the [Getting](https://spring.io/guides/gs/convert-jar-to-war/) [Started Guide on Converting a jar to a war](https://spring.io/guides/gs/convert-jar-to-war/).

To create a deployable war by extending SpringBootServletInitializer (for example, in a class called Application) and adding the Spring Boot @SpringBootApplication annotation, use code similar to that shown in the following example:

@SpringBootApplication

public class Application extends SpringBootServletInitializer {

@Override

protected SpringApplicationBuilder configure(SpringApplicationBuilder application)

{

// Customize the application or call application.sources(...) to add sources

// Since our example is itself a @Configuration class (via @SpringBootApplication)

// we actually don't need to override this method. return application;

}

}

Remember that, whatever you put in the sources is merely a Spring ApplicationContext. Normally, anything that already works should work here. There might be some beans you can remove later and let Spring Boot provide its own defaults for them, but it should be possible to get something working before you need to do that.

Static resources can be moved to /public (or /static or /resources or /META-INF/resources) in the classpath root. The same applies to messages.properties (which Spring Boot automatically detects in the root of the classpath).

Vanilla usage of Spring DispatcherServlet and Spring Security should require no further changes. If you have other features in your application (for instance, using other servlets or filters), you may need to add some configuration to your Application context, by replacing those elements from the web.xml, as follows:

* + - * A @Bean of type Servlet or ServletRegistrationBean installs that bean in the container as if it were a <servlet/> and <servlet-mapping/> in web.xml.
      * A @Bean of type Filter or FilterRegistrationBean behaves similarly (as a <filter/> and <filter- mapping/>).
      * An ApplicationContext in an XML file can be added through an @ImportResource in your Application. Alternatively, cases where annotation configuration is heavily used already can be recreated in a few lines as @Bean definitions.

Once the war file is working, you can make it executable by adding a main method to your

Application, as shown in the following example:

public static void main(String[] args) { SpringApplication.run(Application.class, args);

}

If you intend to start your application as a war or as an executable application, you need to share the customizations of the builder in a method that is both available to the SpringBootServletInitializer callback and in the main method in a class similar to the following:



@SpringBootApplication

public class Application extends SpringBootServletInitializer {

@Override

protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {

return configureApplication(builder);

}

public static void main(String[] args) {

configureApplication(new SpringApplicationBuilder()).run(args);

}

private static SpringApplicationBuilder configureApplication(SpringApplicationBuilder builder) {

return builder.sources(Application.class).bannerMode(Banner.Mode.OFF);

}

}

Applications can fall into more than one category:

* + - * Servlet 3.0+ applications with no web.xml.
      * Applications with a web.xml.
      * Applications with a context hierarchy.
      * Applications without a context hierarchy.

All of these should be amenable to translation, but each might require slightly different techniques.

Servlet 3.0+ applications might translate pretty easily if they already use the Spring Servlet 3.0+ initializer support classes. Normally, all the code from an existing WebApplicationInitializer can be moved into a SpringBootServletInitializer. If your existing application has more than one ApplicationContext (for example, if it uses AbstractDispatcherServletInitializer) then you might be able to combine all your context sources into a single SpringApplication. The main complication you might encounter is if combining does not work and you need to maintain the context hierarchy. See the [entry on building a hierarchy](#_bookmark530) for examples. An existing parent context that contains web- specific features usually needs to be broken up so that all the ServletContextAware components are in the child context.

Applications that are not already Spring applications might be convertible to Spring Boot applications, and the previously mentioned guidance may help. However, you may yet encounter

problems. In that case, we suggest [asking questions on Stack Overflow with a tag of spring-boot](https://stackoverflow.com/questions/tagged/spring-boot).

### Deploying a WAR to WebLogic

To deploy a Spring Boot application to WebLogic, you must ensure that your servlet initializer **directly** implements WebApplicationInitializer (even if you extend from a base class that already implements it).

A typical initializer for WebLogic should resemble the following example:

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.web.servlet.support.SpringBootServletInitializer; import org.springframework.web.WebApplicationInitializer;

@SpringBootApplication

public class MyApplication extends SpringBootServletInitializer implements WebApplicationInitializer {

}

If you use Logback, you also need to tell WebLogic to prefer the packaged version rather than the version that was pre-installed with the server. You can do so by adding a WEB-INF/weblogic.xml file with the following contents:

<?xml version="1.0" encoding="UTF-8"?>

<wls:weblogic-web-app [xmlns:wls="http://xmlns.oracle.com/weblogic/weblogic-web-app"](http://xmlns.oracle.com/weblogic/weblogic-web-app) [xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance) [xsi:schemaLocation="http://java.sun.com/xml/ns/javaee](http://java.sun.com/xml/ns/javaee)

https://java.sun.com/xml/ns/javaee/ejb-jar\_3\_0.xsd <http://xmlns.oracle.com/weblogic/weblogic-web-app> https://xmlns.oracle.com/weblogic/weblogic-web-app/1.4/weblogic-web-app.xsd">

<wls:container-descriptor>

<wls:prefer-application-packages>

<wls:package-name>org.slf4j</wls:package-name>

</wls:prefer-application-packages>

</wls:container-descriptor>

</wls:weblogic-web-app>

### Use Jedis Instead of Lettuce

By default, the Spring Boot starter (spring-boot-starter-data-redis) uses [Lettuce](https://github.com/lettuce-io/lettuce-core/). You need to exclude that dependency and include the [Jedis](https://github.com/xetorthio/jedis/) one instead. Spring Boot manages both of these dependencies so you can switch to Jedis without specifying a version.

The following example shows how to do so in Maven:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-redis</artifactId>

<exclusions>

<exclusion>

<groupId>io.lettuce</groupId>

<artifactId>lettuce-core</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

</dependency>

The following example shows how to do so in Gradle:

dependencies {

implementation('org.springframework.boot:spring-boot-starter-data-redis') { exclude group: 'io.lettuce', module: 'lettuce-core'

}

implementation 'redis.clients:jedis'

// ...

}

### Use Testcontainers for integration testing

The [Testcontainers](https://www.testcontainers.org/) library provides a way to manage services running inside Docker containers. It integrates with JUnit, allowing you to write a test class that can start up a container before any of the tests run. Testcontainers is especially useful for writing integration tests that talk to a real backend service such as MySQL, MongoDB, Cassandra etc. Testcontainers can be used in a Spring Boot test as follows:

@SpringBootTest @Testcontainers

class ExampleIntegrationTests {

@Container

static Neo4jContainer<?> neo4j = new Neo4jContainer<>();

}

This will start up a docker container running Neo4j (if Docker is running locally) before any of the tests are run. In most cases, you will need to configure the application using details from the running container, such as container IP or port.

This can be done with a static @DynamicPropertySource method that allows adding dynamic property

values to the Spring Environment.

@SpringBootTest @Testcontainers

class ExampleIntegrationTests {

@Container

static Neo4jContainer<?> neo4j = new Neo4jContainer<>();

@DynamicPropertySource

static void neo4jProperties(DynamicPropertyRegistry registry) { registry.add("spring.data.neo4j.uri", neo4j::getBoltUrl);

}

}

The above configuration allows Neo4j-related beans in the application to communicate with Neo4j running inside the Testcontainers-managed Docker container.

**Chapter 10. Appendices**

## Appendix A: Common Application properties

Various properties can be specified inside your application.properties file, inside your application.yml file, or as command line switches. This appendix provides a list of common Spring Boot properties and references to the underlying classes that consume them.

Spring Boot provides various conversion mechanism with advanced value formatting, make sure to review [the properties conversion section](#_bookmark123).

Property contributions can come from additional jar files on your classpath, so you should not consider this an exhaustive list. Also, you can define your own properties.

### Core properties

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| debug | false | Enable debug logs. |
| info.\* |  | Arbitrary properties to add to the info endpoint. |
| logging.config |  | Location of the logging configuration file. For instance, `classpath:logback.xml` for Logback. |
| logging.exception- conversion-word | %wEx | Conversion word used when logging exceptions. |
| logging.file.clean- history-on-start | false | Whether to clean the archive log files on startup. Only supported with the default logback setup. |
| logging.file.max- history | 7.0 | Maximum number of days archive log files are kept. Only supported with the default logback setup. |
| logging.file.max-size | 10MB | Maximum log file size. Only supported with the default logback setup. |
| logging.file.name |  | Log file name (for instance, `myapp.log`). Names can be an exact location or relative to the current directory. |
| logging.file.path |  | Location of the log file. For instance, `/var/log`. |
| logging.file.total- size-cap | 0B | Total size of log backups to be kept. Only supported with the default logback setup. |
| logging.group.\* |  | Log groups to quickly change multiple loggers at the same time. For instance,  `logging.group.db=org.hibernate,org.springfram  ework.jdbc`. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| logging.level.\* |  | Log levels severity mapping. For instance,  `logging.level.org.springframework=DEBUG`. |
| logging.pattern.consol e | %clr(%d{${LOG\_DATEFORM  AT\_PATTERN:-yyyy-MM-dd HH:mm:ss.SSS}}){faint}  %clr(${LOG\_LEVEL\_PATTE RN:-%5p}) %clr(${PID:-  }){magenta} %clr(---  ){faint}  %clr([%15.15t]){faint}  %clr(%- 40.40logger{39}){cyan}  %clr(:){faint}  %m%n${LOG\_EXCEPTION\_CO NVERSION\_WORD:-%wEx} | Appender pattern for output to the console. Supported only with the default Logback setup. |
| logging.pattern.datefo rmat | yyyy-MM-dd HH:mm:ss.SSS | Appender pattern for log date format. Supported only with the default Logback setup. |
| logging.pattern.file | %d{${LOG\_DATEFORMAT\_PA  TTERN:-yyyy-MM-dd HH:mm:ss.SSS}}  ${LOG\_LEVEL\_PATTERN:-  %5p} ${PID:- } ---  [%t] %-40.40logger{39}  :  %m%n${LOG\_EXCEPTION\_CO NVERSION\_WORD:-%wEx} | Appender pattern for output to a file. Supported only with the default Logback setup. |
| logging.pattern.level | %5p | Appender pattern for log level. Supported only with the default Logback setup. |
| logging.pattern.rollin g-file-name | ${LOG\_FILE}.%d{yyyy- MM-dd}.%i.gz | Pattern for rolled-over log file names. Supported only with the default Logback setup. |
| logging.register- shutdown-hook | false | Register a shutdown hook for the logging system when it is initialized. |
| spring.aop.auto | true | Add @EnableAspectJAutoProxy. |
| spring.aop.proxy- target-class | true | Whether subclass-based (CGLIB) proxies are to be created (true), as opposed to standard Java interface-based proxies (false). |
| spring.application.adm in.enabled | false | Whether to enable admin features for the application. |
| spring.application.adm in.jmx-name | org.springframework.bo ot:type=Admin,name=Spr ingApplication | JMX name of the application admin MBean. |
| spring.application.nam e |  | Application name. |
| spring.autoconfigure.e xclude |  | Auto-configuration classes to exclude. |
| spring.banner.charset | UTF-8 | Banner file encoding. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.banner.image.bi tdepth | 4.0 | The bit depth to use for ANSI colors. Supported values are 4 (16 color) or 8 (256 color). |
| spring.banner.image.he ight |  | Height of the banner image in chars (default based on image height). |
| spring.banner.image.in vert | false | Whether images should be inverted for dark terminal themes. |
| spring.banner.image.lo cation | classpath:banner.gif | Banner image file location (jpg or png can also be used). |
| spring.banner.image.ma rgin | 2.0 | Left hand image margin in chars. |
| spring.banner.image.pi xelmode | TEXT | The pixel mode to use when rendering the image. |
| spring.banner.image.wi dth | 76.0 | Width of the banner image in chars. |
| spring.banner.location | classpath:banner.txt | Banner text resource location. |
| spring.beaninfo.ignore | true | Whether to skip search of BeanInfo classes. |
| spring.codec.log- request-details | false | Whether to log form data at DEBUG level, and headers at TRACE level. |
| spring.codec.max-in- memory-size |  | Limit on the number of bytes that can be buffered whenever the input stream needs to be aggregated. This applies only to the auto- configured WebFlux server and WebClient instances. By default this is not set, in which case individual codec defaults apply. Most codecs are limited to 256K by default. |
| spring.config.addition al-location |  | Config file locations used in addition to the defaults. |
| spring.config.location |  | Config file locations that replace the defaults. |
| spring.config.name | application | Config file name. |
| spring.info.build.enco ding | UTF-8 | File encoding. |
| spring.info.build.loca tion | classpath:META- INF/build- info.properties | Location of the generated build-info.properties file. |
| spring.info.git.encodi ng | UTF-8 | File encoding. |
| spring.info.git.locati on | classpath:git.properti es | Location of the generated git.properties file. |
| spring.jmx.default- domain |  | JMX domain name. |
| spring.jmx.enabled | false | Expose management beans to the JMX domain. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.jmx.server | mbeanServer | MBeanServer bean name. |
| spring.jmx.unique- names | false | Whether unique runtime object names should be ensured. |
| spring.lifecycle.timeo ut-per-shutdown-phase | 30s | Timeout for the shutdown of any phase (group of SmartLifecycle beans with the same 'phase' value). |
| spring.main.allow- bean-definition- overriding | false | Whether bean definition overriding, by registering a definition with the same name as an existing definition, is allowed. |
| spring.main.banner- mode | console | Mode used to display the banner when the application runs. |
| spring.main.cloud- platform |  | Override the Cloud Platform auto-detection. |
| spring.main.lazy- initialization | false | Whether initialization should be performed lazily. |
| spring.main.log- startup-info | true | Whether to log information about the application when it starts. |
| spring.main.register- shutdown-hook | true | Whether the application should have a shutdown hook registered. |
| spring.main.sources |  | Sources (class names, package names, or XML resource locations) to include in the ApplicationContext. |
| spring.main.web- application-type |  | Flag to explicitly request a specific type of web application. If not set, auto-detected based on the classpath. |
| spring.mandatory-file- encoding |  | Expected character encoding the application must use. |
| spring.messages.always  -use-message-format | false | Whether to always apply the MessageFormat rules, parsing even messages without arguments. |
| spring.messages.basena me | messages | Comma-separated list of basenames (essentially a fully-qualified classpath location), each following the ResourceBundle convention with relaxed support for slash based locations. If it doesn't contain a package qualifier (such as "org.mypackage"), it will be resolved from the classpath root. |
| spring.messages.cache- duration |  | Loaded resource bundle files cache duration. When not set, bundles are cached forever. If a duration suffix is not specified, seconds will be used. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.messages.encodi ng | UTF-8 | Message bundles encoding. |
| spring.messages.fallba ck-to-system-locale | true | Whether to fall back to the system Locale if no files for a specific Locale have been found. if this is turned off, the only fallback will be the default file (e.g. "messages.properties" for basename "messages"). |
| spring.messages.use- code-as-default- message | false | Whether to use the message code as the default message instead of throwing a "NoSuchMessageException". Recommended during development only. |
| spring.output.ansi.ena bled | detect | Configures the ANSI output. |
| spring.pid.fail-on- write-error |  | Fails if ApplicationPidFileWriter is used but it cannot write the PID file. |
| spring.pid.file |  | Location of the PID file to write (if ApplicationPidFileWriter is used). |
| spring.profiles |  | Comma-separated list of profile expressions that at least one should match for the document to be included. |
| spring.profiles.active |  | Comma-separated list of active profiles. Can be overridden by a command line switch. |
| spring.profiles.includ e |  | Unconditionally activate the specified comma- separated list of profiles (or list of profiles if using YAML). |
| spring.quartz.auto- startup | true | Whether to automatically start the scheduler after initialization. |
| spring.quartz.jdbc.com ment-prefix | [#, --] | Prefixes for single-line comments in SQL initialization scripts. |
| spring.quartz.jdbc.ini tialize-schema | embedded | Database schema initialization mode. |
| spring.quartz.jdbc.sch ema | classpath:org/quartz/i mpl/jdbcjobstore/table [s\_@@platform@@.sql](mailto:s_@@platform@@.sql) | Path to the SQL file to use to initialize the database schema. |
| spring.quartz.job- store-type | memory | Quartz job store type. |
| spring.quartz.overwrit e-existing-jobs | false | Whether configured jobs should overwrite existing job definitions. |
| spring.quartz.properti es.\* |  | Additional Quartz Scheduler properties. |
| spring.quartz.schedule r-name | quartzScheduler | Name of the scheduler. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.quartz.startup- delay | 0s | Delay after which the scheduler is started once initialization completes. Setting this property makes sense if no jobs should be run before the entire application has started up. |
| spring.quartz.wait- for-jobs-to-complete- on-shutdown | false | Whether to wait for running jobs to complete on shutdown. |
| spring.reactor.debug- agent.enabled | true | Whether the Reactor Debug Agent should be enabled when reactor-tools is present. |
| spring.task.execution. pool.allow-core- thread-timeout | true | Whether core threads are allowed to time out. This enables dynamic growing and shrinking of the pool. |
| spring.task.execution. pool.core-size | 8.0 | Core number of threads. |
| spring.task.execution. pool.keep-alive | 60s | Time limit for which threads may remain idle before being terminated. |
| spring.task.execution. pool.max-size |  | Maximum allowed number of threads. If tasks are filling up the queue, the pool can expand up to that size to accommodate the load. Ignored if the queue is unbounded. |
| spring.task.execution. pool.queue-capacity |  | Queue capacity. An unbounded capacity does not increase the pool and therefore ignores the "max-size" property. |
| spring.task.execution. shutdown.await- termination | false | Whether the executor should wait for scheduled tasks to complete on shutdown. |
| spring.task.execution. shutdown.await- termination-period |  | Maximum time the executor should wait for remaining tasks to complete. |
| spring.task.execution. thread-name-prefix | task- | Prefix to use for the names of newly created threads. |
| spring.task.scheduling  .pool.size | 1.0 | Maximum allowed number of threads. |
| spring.task.scheduling  .shutdown.await- termination | false | Whether the executor should wait for scheduled tasks to complete on shutdown. |
| spring.task.scheduling  .shutdown.await- termination-period |  | Maximum time the executor should wait for remaining tasks to complete. |
| spring.task.scheduling  .thread-name-prefix | scheduling- | Prefix to use for the names of newly created threads. |
| trace | false | Enable trace logs. |

### Cache properties

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.cache.cache- names |  | Comma-separated list of cache names to create if supported by the underlying cache manager.  Usually, this disables the ability to create  additional caches on-the-fly. |
| spring.cache.caffeine. spec |  | The spec to use to create caches. See CaffeineSpec for more details on the spec format. |
| spring.cache.couchbase  .expiration |  | Entry expiration. By default the entries never expire. Note that this value is ultimately converted to seconds. |
| spring.cache.ehcache.c onfig |  | The location of the configuration file to use to initialize EhCache. |
| spring.cache.infinispa n.config |  | The location of the configuration file to use to initialize Infinispan. |
| spring.cache.jcache.co nfig |  | The location of the configuration file to use to initialize the cache manager. The configuration file is dependent of the underlying cache implementation. |
| spring.cache.jcache.pr ovider |  | Fully qualified name of the CachingProvider implementation to use to retrieve the JSR-107 compliant cache manager. Needed only if more than one JSR-107 implementation is available on the classpath. |
| spring.cache.redis.cac he-null-values | true | Allow caching null values. |
| spring.cache.redis.key  -prefix |  | Key prefix. |
| spring.cache.redis.tim e-to-live |  | Entry expiration. By default the entries never expire. |
| spring.cache.redis.use  -key-prefix | true | Whether to use the key prefix when writing to Redis. |
| spring.cache.type |  | Cache type. By default, auto-detected according to the environment. |

### Mail properties

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.mail.default- encoding | UTF-8 | Default MimeMessage encoding. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.mail.host |  | SMTP server host. For instance,  `smtp.example.com`. |
| spring.mail.jndi-name |  | Session JNDI name. When set, takes precedence over other Session settings. |
| spring.mail.password |  | Login password of the SMTP server. |
| spring.mail.port |  | SMTP server port. |
| spring.mail.properties  .\* |  | Additional JavaMail Session properties. |
| spring.mail.protocol | smtp | Protocol used by the SMTP server. |
| spring.mail.test- connection | false | Whether to test that the mail server is available on startup. |
| spring.mail.username |  | Login user of the SMTP server. |
| spring.sendgrid.api- key |  | SendGrid API key. |
| spring.sendgrid.proxy. host |  | SendGrid proxy host. |
| spring.sendgrid.proxy. port |  | SendGrid proxy port. |

### JSON properties

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.gson.date- format |  | Format to use when serializing Date objects. |
| spring.gson.disable- html-escaping |  | Whether to disable the escaping of HTML characters such as '<', '>', etc. |
| spring.gson.disable- inner-class- serialization |  | Whether to exclude inner classes during serialization. |
| spring.gson.enable- complex-map-key- serialization |  | Whether to enable serialization of complex map keys (i.e. non-primitives). |
| spring.gson.exclude- fields-without-expose- annotation |  | Whether to exclude all fields from consideration for serialization or deserialization that do not have the "Expose" annotation. |
| spring.gson.field- naming-policy |  | Naming policy that should be applied to an object's field during serialization and deserialization. |
| spring.gson.generate- non-executable-json |  | Whether to generate non executable JSON by prefixing the output with some special text. |
| spring.gson.lenient |  | Whether to be lenient about parsing JSON that doesn't conform to RFC 4627. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.gson.long- serialization-policy |  | Serialization policy for Long and long types. |
| spring.gson.pretty- printing |  | Whether to output serialized JSON that fits in a page for pretty printing. |
| spring.gson.serialize- nulls |  | Whether to serialize null fields. |
| spring.jackson.date- format |  | Date format string or a fully-qualified date format class name. For instance, `yyyy-MM-dd HH:mm:ss`. |
| spring.jackson.default  -property-inclusion |  | Controls the inclusion of properties during serialization. Configured with one of the values in Jackson's JsonInclude.Include enumeration. |
| spring.jackson.deseria lization.\* |  | Jackson on/off features that affect the way Java objects are deserialized. |
| spring.jackson.generat or.\* |  | Jackson on/off features for generators. |
| spring.jackson.locale |  | Locale used for formatting. |
| spring.jackson.mapper.  \* |  | Jackson general purpose on/off features. |
| spring.jackson.parser.  \* |  | Jackson on/off features for parsers. |
| spring.jackson.propert y-naming-strategy |  | One of the constants on Jackson's PropertyNamingStrategy. Can also be a fully- qualified class name of a PropertyNamingStrategy subclass. |
| spring.jackson.seriali zation.\* |  | Jackson on/off features that affect the way Java objects are serialized. |
| spring.jackson.time- zone |  | Time zone used when formatting dates. For instance, "America/Los\_Angeles" or "GMT+10". |
| spring.jackson.visibil ity.\* |  | Jackson visibility thresholds that can be used to limit which methods (and fields) are auto- detected. |

### Data properties

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.couchbase.conne ction-string |  | Connection string used to locate the Couchbase cluster. |
| spring.couchbase.env.i o.idle-http- connection-timeout | 4500ms | Length of time an HTTP connection may remain idle before it is closed and removed from the pool. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.couchbase.env.i o.max-endpoints | 12.0 | Maximum number of sockets per node. |
| spring.couchbase.env.i o.min-endpoints | 1.0 | Minimum number of sockets per node. |
| spring.couchbase.env.s sl.enabled |  | Whether to enable SSL support. Enabled automatically if a "keyStore" is provided unless specified otherwise. |
| spring.couchbase.env.s sl.key-store |  | Path to the JVM key store that holds the certificates. |
| spring.couchbase.env.s sl.key-store-password |  | Password used to access the key store. |
| spring.couchbase.env.t imeouts.analytics | 75s | Timeout for the analytics service. |
| spring.couchbase.env.t imeouts.connect | 10s | Bucket connect timeout. |
| spring.couchbase.env.t imeouts.disconnect | 10s | Bucket disconnect timeout. |
| spring.couchbase.env.t imeouts.key-value | 2500ms | Timeout for operations on a specific key-value. |
| spring.couchbase.env.t imeouts.key-value- durable | 10s | Timeout for operations on a specific key-value with a durability level. |
| spring.couchbase.env.t imeouts.management | 75s | Timeout for the management operations. |
| spring.couchbase.env.t imeouts.query | 75s | N1QL query operations timeout. |
| spring.couchbase.env.t imeouts.search | 75s | Timeout for the search service. |
| spring.couchbase.env.t imeouts.view | 75s | Regular and geospatial view operations timeout. |
| spring.couchbase.passw ord |  | Cluster password. |
| spring.couchbase.usern ame |  | Cluster username. |
| spring.dao.exceptiontr anslation.enabled | true | Whether to enable the PersistenceExceptionTranslationPostProcessor. |
| spring.data.cassandra. compression | none | Compression supported by the Cassandra binary protocol. |
| spring.data.cassandra. connection.connect- timeout | 5s | Timeout to use when establishing driver connections. |
| spring.data.cassandra. connection.init-query- timeout | 500ms | Timeout to use for internal queries that run as part of the initialization process, just after a connection is opened. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.data.cassandra. contact-points | [127.0.0.1:9042] | Cluster node addresses in the form 'host:port', or a simple 'host' to use the configured port. |
| spring.data.cassandra. keyspace-name |  | Keyspace name to use. |
| spring.data.cassandra. local-datacenter |  | Datacenter that is considered "local". Contact points should be from this datacenter. |
| spring.data.cassandra. password |  | Login password of the server. |
| spring.data.cassandra. pool.heartbeat- interval | 30s | Heartbeat interval after which a message is sent on an idle connection to make sure it's still alive. |
| spring.data.cassandra. pool.idle-timeout | 120s | Idle timeout before an idle connection is removed. |
| spring.data.cassandra. port | 9042.0 | Port to use if a contact point does not specify one. |
| spring.data.cassandra. repositories.type | auto | Type of Cassandra repositories to enable. |
| spring.data.cassandra. request.consistency |  | Queries consistency level. |
| spring.data.cassandra. request.page-size | 5000.0 | How many rows will be retrieved simultaneously in a single network roundtrip. |
| spring.data.cassandra. request.serial- consistency |  | Queries serial consistency level. |
| spring.data.cassandra. request.throttler.drai n-interval | 10ms | How often the throttler attempts to dequeue requests. Set this high enough that each attempt will process multiple entries in the queue, but not delay requests too much. |
| spring.data.cassandra. request.throttler.max- concurrent-requests | 10000.0 | Maximum number of requests that are allowed to execute in parallel. |
| spring.data.cassandra. request.throttler.max- queue-size | 10000.0 | Maximum number of requests that can be enqueued when the throttling threshold is exceeded. |
| spring.data.cassandra. request.throttler.max- requests-per-second | 10000.0 | Maximum allowed request rate. |
| spring.data.cassandra. request.throttler.type | none | Request throttling type. |
| spring.data.cassandra. request.timeout | 2s | How long the driver waits for a request to complete. |
| spring.data.cassandra. schema-action | none | Schema action to take at startup. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.data.cassandra. session-name |  | Name of the Cassandra session. |
| spring.data.cassandra. ssl | false | Enable SSL support. |
| spring.data.cassandra. username |  | Login user of the server. |
| spring.data.couchbase. auto-index | false | Automatically create views and indexes. Use the meta-data provided by "@ViewIndexed", "@N1qlPrimaryIndexed" and "@N1qlSecondaryIndexed". |
| spring.data.couchbase. bucket-name |  | Name of the bucket to connect to. |
| spring.data.couchbase. field-naming-strategy |  | Fully qualified name of the FieldNamingStrategy to use. |
| spring.data.couchbase. repositories.type | auto | Type of Couchbase repositories to enable. |
| spring.data.couchbase. scope-name |  | Name of the scope used for all collection access. |
| spring.data.couchbase. type-key | \_class | Name of the field that stores the type information for complex types when using "MappingCouchbaseConverter". |
| spring.data.elasticsea rch.client.reactive.co nnection-timeout |  | Connection timeout. |
| spring.data.elasticsea rch.client.reactive.en dpoints |  | Comma-separated list of the Elasticsearch endpoints to connect to. |
| spring.data.elasticsea rch.client.reactive.ma x-in-memory-size |  | Limit on the number of bytes that can be buffered whenever the input stream needs to be aggregated. |
| spring.data.elasticsea rch.client.reactive.pa ssword |  | Credentials password. |
| spring.data.elasticsea rch.client.reactive.so cket-timeout |  | Read and Write Socket timeout. |
| spring.data.elasticsea rch.client.reactive.us e-ssl | false | Whether the client should use SSL to connect to the endpoints. |
| spring.data.elasticsea rch.client.reactive.us ername |  | Credentials username. |
| spring.data.elasticsea rch.repositories.enabl ed | true | Whether to enable Elasticsearch repositories. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.data.jdbc.repos itories.enabled | true | Whether to enable JDBC repositories. |
| spring.data.jpa.reposi tories.bootstrap-mode | deferred | Bootstrap mode for JPA repositories. |
| spring.data.jpa.reposi tories.enabled | true | Whether to enable JPA repositories. |
| spring.data.ldap.repos itories.enabled | true | Whether to enable LDAP repositories. |
| spring.data.mongodb.au thentication-database |  | Authentication database name. |
| spring.data.mongodb.au to-index-creation |  | Whether to enable auto-index creation. |
| spring.data.mongodb.da tabase |  | Database name. |
| spring.data.mongodb.fi eld-naming-strategy |  | Fully qualified name of the FieldNamingStrategy to use. |
| spring.data.mongodb.gr id-fs-database |  | GridFS database name. |
| spring.data.mongodb.ho st |  | Mongo server host. Cannot be set with URI. |
| spring.data.mongodb.pa ssword |  | Login password of the mongo server. Cannot be set with URI. |
| spring.data.mongodb.po rt |  | Mongo server port. Cannot be set with URI. |
| spring.data.mongodb.re plica-set-name |  | Required replica set name for the cluster. Cannot be set with URI. |
| spring.data.mongodb.re positories.type | auto | Type of Mongo repositories to enable. |
| spring.data.mongodb.ur i | mongodb://localhost/te st | Mongo database URI. Cannot be set with host, port, credentials and replica set name. |
| spring.data.mongodb.us ername |  | Login user of the mongo server. Cannot be set with URI. |
| spring.data.mongodb.uu id-representation | java-legacy | Representation to use when converting a UUID to a BSON binary value. |
| spring.data.neo4j.auto  -index | none | Auto index mode. |
| spring.data.neo4j.embe dded.enabled | true | Whether to enable embedded mode if the embedded driver is available. |
| spring.data.neo4j.open  -in-view | false | Register OpenSessionInViewInterceptor that binds a Neo4j Session to the thread for the entire processing of the request. |
| spring.data.neo4j.pass word |  | Login password of the server. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.data.neo4j.repo sitories.enabled | true | Whether to enable Neo4j repositories. |
| spring.data.neo4j.uri |  | URI used by the driver. Auto-detected by default. |
| spring.data.neo4j.use- native-types | false | Whether to use Neo4j native types wherever possible. |
| spring.data.neo4j.user name |  | Login user of the server. |
| spring.data.r2dbc.repo sitories.enabled | true | Whether to enable R2DBC repositories. |
| spring.data.redis.repo sitories.enabled | true | Whether to enable Redis repositories. |
| spring.data.rest.base- path |  | Base path to be used by Spring Data REST to expose repository resources. |
| spring.data.rest.defau lt-media-type |  | Content type to use as a default when none is specified. |
| spring.data.rest.defau lt-page-size |  | Default size of pages. |
| spring.data.rest.detec tion-strategy | default | Strategy to use to determine which repositories get exposed. |
| spring.data.rest.enabl e-enum-translation |  | Whether to enable enum value translation through the Spring Data REST default resource bundle. |
| spring.data.rest.limit  -param-name |  | Name of the URL query string parameter that indicates how many results to return at once. |
| spring.data.rest.max- page-size |  | Maximum size of pages. |
| spring.data.rest.page- param-name |  | Name of the URL query string parameter that indicates what page to return. |
| spring.data.rest.retur n-body-on-create |  | Whether to return a response body after creating an entity. |
| spring.data.rest.retur n-body-on-update |  | Whether to return a response body after updating an entity. |
| spring.data.rest.sort- param-name |  | Name of the URL query string parameter that indicates what direction to sort results. |
| spring.data.solr.host | http://127.0.0.1:8983/ solr | Solr host. Ignored if "zk-host" is set. |
| spring.data.solr.repos itories.enabled | true | Whether to enable Solr repositories. |
| spring.data.solr.zk- host |  | ZooKeeper host address in the form HOST:PORT. |
| spring.data.web.pageab le.default-page-size | 20.0 | Default page size. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.data.web.pageab le.max-page-size | 2000.0 | Maximum page size to be accepted. |
| spring.data.web.pageab le.one-indexed- parameters | false | Whether to expose and assume 1-based page number indexes. Defaults to "false", meaning a page number of 0 in the request equals the first page. |
| spring.data.web.pageab le.page-parameter | page | Page index parameter name. |
| spring.data.web.pageab le.prefix |  | General prefix to be prepended to the page number and page size parameters. |
| spring.data.web.pageab le.qualifier-delimiter | \_ | Delimiter to be used between the qualifier and the actual page number and size properties. |
| spring.data.web.pageab le.size-parameter | size | Page size parameter name. |
| spring.data.web.sort.s ort-parameter | sort | Sort parameter name. |
| spring.datasource.cont inue-on-error | false | Whether to stop if an error occurs while initializing the database. |
| spring.datasource.data |  | Data (DML) script resource references. |
| spring.datasource.data  -password |  | Password of the database to execute DML scripts (if different). |
| spring.datasource.data  -username |  | Username of the database to execute DML scripts (if different). |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.datasource.dbcp 2.abandoned-usage-  tracking  spring.datasource.dbcp 2.access-to- underlying-connection-  allowed  spring.datasource.dbcp 2.auto-commit-on-  return spring.datasource.dbcp 2.cache-state  spring.datasource.dbcp 2.connection-factory-  class-name spring.datasource.dbcp 2.connection-init-sqls spring.datasource.dbcp 2.default-auto-commit spring.datasource.dbcp 2.default-catalog  spring.datasource.dbcp 2.default-query-  timeout spring.datasource.dbcp 2.default-read-only spring.datasource.dbcp 2.default-schema  spring.datasource.dbcp 2.default-transaction-  isolation  spring.datasource.dbcp 2.disconnection-sql-  codes spring.datasource.dbcp 2.driver spring.datasource.dbcp 2.driver-class-name  spring.datasource.dbcp 2.eviction-policy-  class-name spring.datasource.dbcp 2.fast-fail-validation spring.datasource.dbcp 2.initial-size spring.datasource.dbcp 2.jmx-name spring.datasource.dbcp 2.lifo |  | Commons DBCP2 specific settings bound to an instance of DBCP2's BasicDataSource |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.datasource.driv er-class-name |  | Fully qualified name of the JDBC driver. Auto- detected based on the URL by default. |
| spring.datasource.gene rate-unique-name | true | Whether to generate a random datasource name. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.datasource.hika ri.allow-pool-  suspension spring.datasource.hika ri.auto-commit spring.datasource.hika ri.catalog spring.datasource.hika ri.connection-init-sql  spring.datasource.hika ri.connection-test-  query spring.datasource.hika ri.connection-timeout  spring.datasource.hika ri.data-source-class-  name spring.datasource.hika ri.data-source-j-n-d-i  spring.datasource.hika ri.data-source-  properties spring.datasource.hika ri.driver-class-name  spring.datasource.hika ri.exception-override-  class-name  spring.datasource.hika ri.health-check-  properties  spring.datasource.hika ri.health-check-  registry spring.datasource.hika ri.idle-timeout  spring.datasource.hika ri.initialization-  fail-timeout  spring.datasource.hika ri.isolate-internal-  queries spring.datasource.hika ri.jdbc-url  spring.datasource.hika ri.leak-detection-  threshold spring.datasource.hika ri.login-timeout spring.datasource.hika ri.max-lifetime |  | Hikari specific settings bound to an instance of Hikari's HikariDataSource |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.datasource.init ialization-mode | embedded | Initialize the datasource with available DDL and DML scripts. |
| spring.datasource.jndi  -name |  | JNDI location of the datasource. Class, url, username and password are ignored when set. |
| spring.datasource.name |  | Name of the datasource. Default to "testdb" when using an embedded database. |
| spring.datasource.pass word |  | Login password of the database. |
| spring.datasource.plat form | all | Platform to use in the DDL or DML scripts (such as schema-${platform}.sql or data-  ${platform}.sql). |
| spring.datasource.sche ma |  | Schema (DDL) script resource references. |
| spring.datasource.sche ma-password |  | Password of the database to execute DDL scripts (if different). |
| spring.datasource.sche ma-username |  | Username of the database to execute DDL scripts (if different). |
| spring.datasource.sepa rator | ; | Statement separator in SQL initialization scripts. |
| spring.datasource.sql- script-encoding |  | SQL scripts encoding. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.datasource.tomc at.abandon-when-  percentage-full  spring.datasource.tomc at.access-to- underlying-connection-  allowed  spring.datasource.tomc at.alternate-username-  allowed spring.datasource.tomc at.commit-on-return  spring.datasource.tomc at.connection-  properties spring.datasource.tomc at.data-source spring.datasource.tomc at.data-source-j-n-d-i spring.datasource.tomc at.db-properties spring.datasource.tomc at.default-auto-commit spring.datasource.tomc at.default-catalog spring.datasource.tomc at.default-read-only  spring.datasource.tomc at.default-  transaction-isolation spring.datasource.tomc at.driver-class-name spring.datasource.tomc at.fair-queue  spring.datasource.tomc at.ignore-exception-  on-pre-load spring.datasource.tomc at.init-s-q-l spring.datasource.tomc at.initial-size spring.datasource.tomc at.jdbc-interceptors spring.datasource.tomc at.jmx-enabled spring.datasource.tomc at.log-abandoned  spring.datasource.tomc at.log-validation- |  | Tomcat datasource specific settings bound to an instance of Tomcat JDBC's DataSource |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.datasource.type |  | Fully qualified name of the connection pool implementation to use. By default, it is auto- detected from the classpath. |
| spring.datasource.url |  | JDBC URL of the database. |
| spring.datasource.user name |  | Login username of the database. |
| spring.datasource.xa.d ata-source-class-name |  | XA datasource fully qualified name. |
| spring.datasource.xa.p roperties.\* |  | Properties to pass to the XA data source. |
| spring.elasticsearch.r est.connection-timeout | 1s | Connection timeout. |
| spring.elasticsearch.r est.password |  | Credentials password. |
| spring.elasticsearch.r est.read-timeout | 30s | Read timeout. |
| spring.elasticsearch.r est.uris | [http://localhost:9200  ] | Comma-separated list of the Elasticsearch instances to use. |
| spring.elasticsearch.r est.username |  | Credentials username. |
| spring.h2.console.enab led | false | Whether to enable the console. |
| spring.h2.console.path | /h2-console | Path at which the console is available. |
| spring.h2.console.sett ings.trace | false | Whether to enable trace output. |
| spring.h2.console.sett ings.web-allow-others | false | Whether to enable remote access. |
| spring.influx.password |  | Login password. |
| spring.influx.url |  | URL of the InfluxDB instance to which to connect. |
| spring.influx.user |  | Login user. |
| spring.jdbc.template.f etch-size | -1.0 | Number of rows that should be fetched from the database when more rows are needed. Use -1 to use the JDBC driver's default configuration. |
| spring.jdbc.template.m ax-rows | -1.0 | Maximum number of rows. Use -1 to use the JDBC driver's default configuration. |
| spring.jdbc.template.q uery-timeout |  | Query timeout. Default is to use the JDBC driver's default configuration. If a duration suffix is not specified, seconds will be used. |
| spring.jooq.sql- dialect |  | SQL dialect to use. Auto-detected by default. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.jpa.database |  | Target database to operate on, auto-detected by default. Can be alternatively set using the "databasePlatform" property. |
| spring.jpa.database- platform |  | Name of the target database to operate on, auto- detected by default. Can be alternatively set using the "Database" enum. |
| spring.jpa.generate- ddl | false | Whether to initialize the schema on startup. |
| spring.jpa.hibernate.d dl-auto |  | DDL mode. This is actually a shortcut for the "hibernate.hbm2ddl.auto" property. Defaults to "create-drop" when using an embedded database and no schema manager was detected. Otherwise, defaults to "none". |
| spring.jpa.hibernate.n aming.implicit- strategy |  | Fully qualified name of the implicit naming strategy. |
| spring.jpa.hibernate.n aming.physical- strategy |  | Fully qualified name of the physical naming strategy. |
| spring.jpa.hibernate.u se-new-id-generator- mappings |  | Whether to use Hibernate's newer IdentifierGenerator for AUTO, TABLE and SEQUENCE. This is actually a shortcut for the "hibernate.id.new\_generator\_mappings" property. When not specified will default to "true". |
| spring.jpa.mapping- resources |  | Mapping resources (equivalent to "mapping-file" entries in persistence.xml). |
| spring.jpa.open-in- view | true | Register OpenEntityManagerInViewInterceptor. Binds a JPA EntityManager to the thread for the entire processing of the request. |
| spring.jpa.properties.  \* |  | Additional native properties to set on the JPA provider. |
| spring.jpa.show-sql | false | Whether to enable logging of SQL statements. |
| spring.mongodb.embedde d.features | [sync\_delay] | Comma-separated list of features to enable. Uses the defaults of the configured version by default. |
| spring.mongodb.embedde d.storage.database-dir |  | Directory used for data storage. |
| spring.mongodb.embedde d.storage.oplog-size |  | Maximum size of the oplog. |
| spring.mongodb.embedde d.storage.repl-set- name |  | Name of the replica set. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.mongodb.embedde d.version | 3.5.5 | Version of Mongo to use. |
| spring.r2dbc.generate- unique-name | false | Whether to generate a random database name. Ignore any configured name when enabled. |
| spring.r2dbc.name |  | Database name. Set if no name is specified in the url. Default to "testdb" when using an embedded database. |
| spring.r2dbc.password |  | Login password of the database. Set if no password is specified in the url. |
| spring.r2dbc.pool.enab led |  | Whether pooling is enabled. Enabled automatically if "r2dbc-pool" is on the classpath. |
| spring.r2dbc.pool.init ial-size | 10.0 | Initial connection pool size. |
| spring.r2dbc.pool.max- idle-time | 30m | Maximum amount of time that a connection is allowed to sit idle in the pool. |
| spring.r2dbc.pool.max- size | 10.0 | Maximal connection pool size. |
| spring.r2dbc.pool.vali dation-query |  | Validation query. |
| spring.r2dbc.propertie s.\* |  | Additional R2DBC options. |
| spring.r2dbc.url |  | R2DBC URL of the database. database name, username, password and pooling options specified in the url take precedence over individual options. |
| spring.r2dbc.username |  | Login username of the database. Set if no username is specified in the url. |
| spring.redis.client- name |  | Client name to be set on connections with CLIENT SETNAME. |
| spring.redis.cluster.m ax-redirects |  | Maximum number of redirects to follow when executing commands across the cluster. |
| spring.redis.cluster.n odes |  | Comma-separated list of "host:port" pairs to bootstrap from. This represents an "initial" list of cluster nodes and is required to have at least one entry. |
| spring.redis.database | 0.0 | Database index used by the connection factory. |
| spring.redis.host | localhost | Redis server host. |
| spring.redis.jedis.poo l.max-active | 8.0 | Maximum number of connections that can be allocated by the pool at a given time. Use a negative value for no limit. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.redis.jedis.poo l.max-idle | 8.0 | Maximum number of "idle" connections in the pool. Use a negative value to indicate an unlimited number of idle connections. |
| spring.redis.jedis.poo l.max-wait | -1ms | Maximum amount of time a connection allocation should block before throwing an exception when the pool is exhausted. Use a negative value to block indefinitely. |
| spring.redis.jedis.poo l.min-idle | 0.0 | Target for the minimum number of idle connections to maintain in the pool. This setting only has an effect if both it and time between eviction runs are positive. |
| spring.redis.jedis.poo l.time-between- eviction-runs |  | Time between runs of the idle object evictor thread. When positive, the idle object evictor thread starts, otherwise no idle object eviction is performed. |
| spring.redis.lettuce.c luster.refresh.adaptiv e | false | Whether adaptive topology refreshing using all available refresh triggers should be used. |
| spring.redis.lettuce.c luster.refresh.period |  | Cluster topology refresh period. |
| spring.redis.lettuce.p ool.max-active | 8.0 | Maximum number of connections that can be allocated by the pool at a given time. Use a negative value for no limit. |
| spring.redis.lettuce.p ool.max-idle | 8.0 | Maximum number of "idle" connections in the pool. Use a negative value to indicate an unlimited number of idle connections. |
| spring.redis.lettuce.p ool.max-wait | -1ms | Maximum amount of time a connection allocation should block before throwing an exception when the pool is exhausted. Use a negative value to block indefinitely. |
| spring.redis.lettuce.p ool.min-idle | 0.0 | Target for the minimum number of idle connections to maintain in the pool. This setting only has an effect if both it and time between eviction runs are positive. |
| spring.redis.lettuce.p ool.time-between- eviction-runs |  | Time between runs of the idle object evictor thread. When positive, the idle object evictor thread starts, otherwise no idle object eviction is performed. |
| spring.redis.lettuce.s hutdown-timeout | 100ms | Shutdown timeout. |
| spring.redis.password |  | Login password of the redis server. |
| spring.redis.port | 6379.0 | Redis server port. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.redis.sentinel. master |  | Name of the Redis server. |
| spring.redis.sentinel. nodes |  | Comma-separated list of "host:port" pairs. |
| spring.redis.sentinel. password |  | Password for authenticating with sentinel(s). |
| spring.redis.ssl | false | Whether to enable SSL support. |
| spring.redis.timeout |  | Connection timeout. |
| spring.redis.url |  | Connection URL. Overrides host, port, and password. User is ignored. Example: redis://user:password@example.com:6379 |

### Transaction properties

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.jta.atomikos.co nnectionfactory.borrow  -connection-timeout | 30.0 | Timeout, in seconds, for borrowing connections from the pool. |
| spring.jta.atomikos.co nnectionfactory.ignore  -session-transacted  -flag | true | Whether to ignore the transacted flag when creating session. |
| spring.jta.atomikos.co nnectionfactory.local- transaction-mode | false | Whether local transactions are desired. |
| spring.jta.atomikos.co nnectionfactory.mainte nance-interval | 60.0 | Time, in seconds, between runs of the pool's maintenance thread. |
| spring.jta.atomikos.co nnectionfactory.max- idle-time | 60.0 | Time, in seconds, after which connections are cleaned up from the pool. |
| spring.jta.atomikos.co nnectionfactory.max- lifetime | 0.0 | Time, in seconds, that a connection can be pooled for before being destroyed. 0 denotes no limit. |
| spring.jta.atomikos.co nnectionfactory.max- pool-size | 1.0 | Maximum size of the pool. |
| spring.jta.atomikos.co nnectionfactory.min- pool-size | 1.0 | Minimum size of the pool. |
| spring.jta.atomikos.co nnectionfactory.reap- timeout | 0.0 | Reap timeout, in seconds, for borrowed connections. 0 denotes no limit. |
| spring.jta.atomikos.co nnectionfactory.unique  -resource-name | jmsConnectionFactory | Unique name used to identify the resource during recovery. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.jta.atomikos.co nnectionfactory.xa- connection-factory- class-name |  | Vendor-specific implementation of XAConnectionFactory. |
| spring.jta.atomikos.co nnectionfactory.xa- properties |  | Vendor-specific XA properties. |
| spring.jta.atomikos.da tasource.borrow- connection-timeout | 30.0 | Timeout, in seconds, for borrowing connections from the pool. |
| spring.jta.atomikos.da tasource.concurrent- connection-validation | true | Whether to use concurrent connection validation. |
| spring.jta.atomikos.da tasource.default- isolation-level |  | Default isolation level of connections provided by the pool. |
| spring.jta.atomikos.da tasource.login-timeout | 0.0 | Timeout, in seconds, for establishing a database connection. |
| spring.jta.atomikos.da tasource.maintenance- interval | 60.0 | Time, in seconds, between runs of the pool's maintenance thread. |
| spring.jta.atomikos.da tasource.max-idle-time | 60.0 | Time, in seconds, after which connections are cleaned up from the pool. |
| spring.jta.atomikos.da tasource.max-lifetime | 0.0 | Time, in seconds, that a connection can be pooled for before being destroyed. 0 denotes no limit. |
| spring.jta.atomikos.da tasource.max-pool-size | 1.0 | Maximum size of the pool. |
| spring.jta.atomikos.da tasource.min-pool-size | 1.0 | Minimum size of the pool. |
| spring.jta.atomikos.da tasource.reap-timeout | 0.0 | Reap timeout, in seconds, for borrowed connections. 0 denotes no limit. |
| spring.jta.atomikos.da tasource.test-query |  | SQL query or statement used to validate a connection before returning it. |
| spring.jta.atomikos.da tasource.unique- resource-name | dataSource | Unique name used to identify the resource during recovery. |
| spring.jta.atomikos.da tasource.xa-data- source-class-name |  | Vendor-specific implementation of XAConnectionFactory. |
| spring.jta.atomikos.da tasource.xa-properties |  | Vendor-specific XA properties. |
| spring.jta.atomikos.pr operties.allow-sub- transactions | true | Specify whether sub-transactions are allowed. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.jta.atomikos.pr operties.checkpoint- interval | 500.0 | Interval between checkpoints, expressed as the number of log writes between two checkpoints. A checkpoint reduces the log file size at the expense of adding some overhead in the runtime. |
| spring.jta.atomikos.pr operties.default-jta- timeout | 10000ms | Default timeout for JTA transactions. |
| spring.jta.atomikos.pr operties.default-max- wait-time-on-shutdown |  | How long should normal shutdown (no-force) wait for transactions to complete. |
| spring.jta.atomikos.pr operties.enable- logging | true | Whether to enable disk logging. |
| spring.jta.atomikos.pr operties.force- shutdown-on-vm-exit | false | Whether a VM shutdown should trigger forced shutdown of the transaction core. |
| spring.jta.atomikos.pr operties.log-base-dir |  | Directory in which the log files should be stored. Defaults to the current working directory. |
| spring.jta.atomikos.pr operties.log-base-name | tmlog | Transactions log file base name. |
| spring.jta.atomikos.pr operties.max-actives | 50.0 | Maximum number of active transactions. |
| spring.jta.atomikos.pr operties.max-timeout | 300000ms | Maximum timeout that can be allowed for transactions. |
| spring.jta.atomikos.pr operties.recovery.dela y | 10000ms | Delay between two recovery scans. |
| spring.jta.atomikos.pr operties.recovery.forg et-orphaned-log- entries-delay | 86400000ms | Delay after which recovery can cleanup pending ('orphaned') log entries. |
| spring.jta.atomikos.pr operties.recovery.max- retries | 5.0 | Number of retry attempts to commit the transaction before throwing an exception. |
| spring.jta.atomikos.pr operties.recovery.retr y-interval | 10000ms | Delay between retry attempts. |
| spring.jta.atomikos.pr operties.serial-jta- transactions | true | Whether sub-transactions should be joined when possible. |
| spring.jta.atomikos.pr operties.service |  | Transaction manager implementation that should be started. |
| spring.jta.atomikos.pr operties.threaded-two- phase-commit | false | Whether to use different (and concurrent) threads for two-phase commit on the participating resources. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.jta.atomikos.pr operties.transaction- manager-unique-name |  | The transaction manager's unique name. Defaults to the machine's IP address. If you plan to run more than one transaction manager against one database you must set this property to a unique value. |
| spring.jta.bitronix.co nnectionfactory.acquir e-increment | 1.0 | Number of connections to create when growing the pool. |
| spring.jta.bitronix.co nnectionfactory.acquis ition-interval | 1.0 | Time, in seconds, to wait before trying to acquire a connection again after an invalid connection was acquired. |
| spring.jta.bitronix.co nnectionfactory.acquis ition-timeout | 30.0 | Timeout, in seconds, for acquiring connections from the pool. |
| spring.jta.bitronix.co nnectionfactory.allow- local-transactions | false | Whether the transaction manager should allow mixing XA and non-XA transactions. |
| spring.jta.bitronix.co nnectionfactory.apply- transaction-timeout | false | Whether the transaction timeout should be set on the XAResource when it is enlisted. |
| spring.jta.bitronix.co nnectionfactory.automa tic-enlisting-enabled | true | Whether resources should be enlisted and delisted automatically. |
| spring.jta.bitronix.co nnectionfactory.cache- producers-consumers | true | Whether producers and consumers should be cached. |
| spring.jta.bitronix.co nnectionfactory.class- name |  | Underlying implementation class name of the XA resource. |
| spring.jta.bitronix.co nnectionfactory.defer- connection-release | true | Whether the provider can run many transactions on the same connection and supports transaction interleaving. |
| spring.jta.bitronix.co nnectionfactory.disabl ed | false | Whether this resource is disabled, meaning it's temporarily forbidden to acquire a connection from its pool. |
| spring.jta.bitronix.co nnectionfactory.driver  -properties |  | Properties that should be set on the underlying implementation. |
| spring.jta.bitronix.co nnectionfactory.failed |  |  |
| spring.jta.bitronix.co nnectionfactory.ignore  -recovery-failures | false | Whether recovery failures should be ignored. |
| spring.jta.bitronix.co nnectionfactory.max- idle-time | 60.0 | Time, in seconds, after which connections are cleaned up from the pool. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.jta.bitronix.co nnectionfactory.max- pool-size | 0.0 | Maximum size of the pool. 0 denotes no limit. |
| spring.jta.bitronix.co nnectionfactory.min- pool-size | 0.0 | Minimum size of the pool. |
| spring.jta.bitronix.co nnectionfactory.passwo rd |  | Password to use to connect to the JMS provider. |
| spring.jta.bitronix.co nnectionfactory.share- transaction- connections | false | Whether connections in the ACCESSIBLE state can be shared within the context of a transaction. |
| spring.jta.bitronix.co nnectionfactory.test- connections | false | Whether connections should be tested when acquired from the pool. |
| spring.jta.bitronix.co nnectionfactory.two- pc-ordering-position | 1.0 | Position that this resource should take during two-phase commit (always first is Integer.MIN\_VALUE, always last is Integer.MAX\_VALUE). |
| spring.jta.bitronix.co nnectionfactory.unique  -name | jmsConnectionFactory | Unique name used to identify the resource during recovery. |
| spring.jta.bitronix.co nnectionfactory.use- tm-join | true | Whether TMJOIN should be used when starting XAResources. |
| spring.jta.bitronix.co nnectionfactory.user |  | User to use to connect to the JMS provider. |
| spring.jta.bitronix.da tasource.acquire- increment | 1.0 | Number of connections to create when growing the pool. |
| spring.jta.bitronix.da tasource.acquisition- interval | 1.0 | Time, in seconds, to wait before trying to acquire a connection again after an invalid connection was acquired. |
| spring.jta.bitronix.da tasource.acquisition- timeout | 30.0 | Timeout, in seconds, for acquiring connections from the pool. |
| spring.jta.bitronix.da tasource.allow-local- transactions | false | Whether the transaction manager should allow mixing XA and non-XA transactions. |
| spring.jta.bitronix.da tasource.apply- transaction-timeout | false | Whether the transaction timeout should be set on the XAResource when it is enlisted. |
| spring.jta.bitronix.da tasource.automatic- enlisting-enabled | true | Whether resources should be enlisted and delisted automatically. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.jta.bitronix.da tasource.class-name |  | Underlying implementation class name of the XA resource. |
| spring.jta.bitronix.da tasource.cursor- holdability |  | Default cursor holdability for connections. |
| spring.jta.bitronix.da tasource.defer- connection-release | true | Whether the database can run many transactions on the same connection and supports transaction interleaving. |
| spring.jta.bitronix.da tasource.disabled | false | Whether this resource is disabled, meaning it's temporarily forbidden to acquire a connection from its pool. |
| spring.jta.bitronix.da tasource.driver- properties |  | Properties that should be set on the underlying implementation. |
| spring.jta.bitronix.da tasource.enable-jdbc4- connection-test | false | Whether Connection.isValid() is called when acquiring a connection from the pool. |
| spring.jta.bitronix.da tasource.failed |  |  |
| spring.jta.bitronix.da tasource.ignore- recovery-failures | false | Whether recovery failures should be ignored. |
| spring.jta.bitronix.da tasource.isolation- level |  | Default isolation level for connections. |
| spring.jta.bitronix.da tasource.local-auto- commit |  | Default auto-commit mode for local transactions. |
| spring.jta.bitronix.da tasource.login-timeout |  | Timeout, in seconds, for establishing a database connection. |
| spring.jta.bitronix.da tasource.max-idle-time | 60.0 | Time, in seconds, after which connections are cleaned up from the pool. |
| spring.jta.bitronix.da tasource.max-pool-size | 0.0 | Maximum size of the pool. 0 denotes no limit. |
| spring.jta.bitronix.da tasource.min-pool-size | 0.0 | Minimum size of the pool. |
| spring.jta.bitronix.da tasource.prepared- statement-cache-size | 0.0 | Target size of the prepared statement cache. 0 disables the cache. |
| spring.jta.bitronix.da tasource.share- transaction- connections | false | Whether connections in the ACCESSIBLE state can be shared within the context of a transaction. |
| spring.jta.bitronix.da tasource.test-query |  | SQL query or statement used to validate a connection before returning it. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.jta.bitronix.da tasource.two-pc- ordering-position | 1.0 | Position that this resource should take during two-phase commit (always first is Integer.MIN\_VALUE, and always last is Integer.MAX\_VALUE). |
| spring.jta.bitronix.da tasource.unique-name | dataSource | Unique name used to identify the resource during recovery. |
| spring.jta.bitronix.da tasource.use-tm-join | true | Whether TMJOIN should be used when starting XAResources. |
| spring.jta.enabled | true | Whether to enable JTA support. |
| spring.jta.log-dir |  | Transaction logs directory. |
| spring.jta.transaction  -manager-id |  | Transaction manager unique identifier. |
| spring.transaction.def ault-timeout |  | Default transaction timeout. If a duration suffix is not specified, seconds will be used. |
| spring.transaction.rol lback-on-commit- failure |  | Whether to roll back on commit failures. |

### Data migration properties

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.flyway.baseline  -description | << Flyway Baseline >> | Description to tag an existing schema with when applying a baseline. |
| spring.flyway.baseline  -on-migrate | false | Whether to automatically call baseline when migrating a non-empty schema. |
| spring.flyway.baseline  -version | 1 | Version to tag an existing schema with when executing baseline. |
| spring.flyway.batch |  | Whether to batch SQL statements when executing them. Requires Flyway Pro or Flyway Enterprise. |
| spring.flyway.check- location | true | Whether to check that migration scripts location exists. |
| spring.flyway.clean- disabled | false | Whether to disable cleaning of the database. |
| spring.flyway.clean- on-validation-error | false | Whether to automatically call clean when a validation error occurs. |
| spring.flyway.connect- retries | 0.0 | Maximum number of retries when attempting to connect to the database. |
| spring.flyway.default- schema |  | Default schema name managed by Flyway (case- sensitive). |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.flyway.enabled | true | Whether to enable flyway. |
| spring.flyway.encoding | UTF-8 | Encoding of SQL migrations. |
| spring.flyway.error- overrides |  | Rules for the built-in error handling to override specific SQL states and error codes. Requires Flyway Pro or Flyway Enterprise. |
| spring.flyway.group | false | Whether to group all pending migrations together in the same transaction when applying them. |
| spring.flyway.ignore- future-migrations | true | Whether to ignore future migrations when reading the schema history table. |
| spring.flyway.ignore- ignored-migrations | false | Whether to ignore ignored migrations when reading the schema history table. |
| spring.flyway.ignore- missing-migrations | false | Whether to ignore missing migrations when reading the schema history table. |
| spring.flyway.ignore- pending-migrations | false | Whether to ignore pending migrations when reading the schema history table. |
| spring.flyway.init- sqls |  | SQL statements to execute to initialize a connection immediately after obtaining it. |
| spring.flyway.installe d-by |  | Username recorded in the schema history table as having applied the migration. |
| spring.flyway.license- key |  | Licence key for Flyway Pro or Flyway Enterprise. |
| spring.flyway.location s | [classpath:db/migratio n] | Locations of migrations scripts. Can contain the special "{vendor}" placeholder to use vendor- specific locations. |
| spring.flyway.mixed | false | Whether to allow mixing transactional and non- transactional statements within the same migration. |
| spring.flyway.oracle- sqlplus |  | Whether to enable support for Oracle SQL\*Plus commands. Requires Flyway Pro or Flyway Enterprise. |
| spring.flyway.oracle- sqlplus-warn |  | Whether to issue a warning rather than an error when a not-yet-supported Oracle SQL\*Plus statement is encountered. Requires Flyway Pro or Flyway Enterprise. |
| spring.flyway.out-of- order | false | Whether to allow migrations to be run out of order. |
| spring.flyway.password |  | Login password of the database to migrate. |
| spring.flyway.placehol der-prefix | ${ | Prefix of placeholders in migration scripts. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.flyway.placehol der-replacement | true | Perform placeholder replacement in migration scripts. |
| spring.flyway.placehol der-suffix | } | Suffix of placeholders in migration scripts. |
| spring.flyway.placehol ders.\* |  | Placeholders and their replacements to apply to sql migration scripts. |
| spring.flyway.repeatab le-sql-migration- prefix | R | File name prefix for repeatable SQL migrations. |
| spring.flyway.schemas |  | Scheme names managed by Flyway (case- sensitive). |
| spring.flyway.skip- default-callbacks | false | Whether to skip default callbacks. If true, only custom callbacks are used. |
| spring.flyway.skip- default-resolvers | false | Whether to skip default resolvers. If true, only custom resolvers are used. |
| spring.flyway.sql- migration-prefix | V | File name prefix for SQL migrations. |
| spring.flyway.sql- migration-separator |  | File name separator for SQL migrations. |
| spring.flyway.sql- migration-suffixes | [.sql] | File name suffix for SQL migrations. |
| spring.flyway.stream |  | Whether to stream SQL migrations when executing them. Requires Flyway Pro or Flyway Enterprise. |
| spring.flyway.table | flyway\_schema\_history | Name of the schema history table that will be used by Flyway. |
| spring.flyway.tablespa ce |  | Tablespace in which the schema history table is created. Ignored when using a database that does not support tablespaces. Defaults to the default tablespace of the connection used by Flyway. |
| spring.flyway.target |  | Target version up to which migrations should be considered. |
| spring.flyway.url |  | JDBC url of the database to migrate. If not set, the primary configured data source is used. |
| spring.flyway.user |  | Login user of the database to migrate. |
| spring.flyway.validate  -migration-naming | false | Whether to validate migrations and callbacks whose scripts do not obey the correct naming convention. |
| spring.flyway.validate  -on-migrate | true | Whether to automatically call validate when performing a migration. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.liquibase.chang e-log | classpath:/db/changelo g/db.changelog- master.yaml | Change log configuration path. |
| spring.liquibase.clear  -checksums | false | Whether to clear all checksums in the current changelog, so they will be recalculated upon the next update. |
| spring.liquibase.conte xts |  | Comma-separated list of runtime contexts to use. |
| spring.liquibase.datab ase-change-log-lock- table | DATABASECHANGELOGLOCK | Name of table to use for tracking concurrent Liquibase usage. |
| spring.liquibase.datab ase-change-log-table | DATABASECHANGELOG | Name of table to use for tracking change history. |
| spring.liquibase.defau lt-schema |  | Default database schema. |
| spring.liquibase.drop- first | false | Whether to first drop the database schema. |
| spring.liquibase.enabl ed | true | Whether to enable Liquibase support. |
| spring.liquibase.label s |  | Comma-separated list of runtime labels to use. |
| spring.liquibase.liqui base-schema |  | Schema to use for Liquibase objects. |
| spring.liquibase.liqui base-tablespace |  | Tablespace to use for Liquibase objects. |
| spring.liquibase.param eters.\* |  | Change log parameters. |
| spring.liquibase.passw ord |  | Login password of the database to migrate. |
| spring.liquibase.rollb ack-file |  | File to which rollback SQL is written when an update is performed. |
| spring.liquibase.tag |  | Tag name to use when applying database changes. Can also be used with "rollbackFile" to generate a rollback script for all existing changes associated with that tag. |
| spring.liquibase.test- rollback-on-update | false | Whether rollback should be tested before update is performed. |
| spring.liquibase.url |  | JDBC URL of the database to migrate. If not set, the primary configured data source is used. |
| spring.liquibase.user |  | Login user of the database to migrate. |

### Integration properties

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.activemq.broker  -url |  | URL of the ActiveMQ broker. Auto-generated by default. |
| spring.activemq.close- timeout | 15s | Time to wait before considering a close complete. |
| spring.activemq.in- memory | true | Whether the default broker URL should be in memory. Ignored if an explicit broker has been specified. |
| spring.activemq.non- blocking-redelivery | false | Whether to stop message delivery before re- delivering messages from a rolled back transaction. This implies that message order is not preserved when this is enabled. |
| spring.activemq.packag es.trust-all |  | Whether to trust all packages. |
| spring.activemq.packag es.trusted |  | Comma-separated list of specific packages to trust (when not trusting all packages). |
| spring.activemq.passwo rd |  | Login password of the broker. |
| spring.activemq.pool.b lock-if-full | true | Whether to block when a connection is requested and the pool is full. Set it to false to throw a "JMSException" instead. |
| spring.activemq.pool.b lock-if-full-timeout | -1ms | Blocking period before throwing an exception if the pool is still full. |
| spring.activemq.pool.e nabled | false | Whether a JmsPoolConnectionFactory should be created, instead of a regular ConnectionFactory. |
| spring.activemq.pool.i dle-timeout | 30s | Connection idle timeout. |
| spring.activemq.pool.m ax-connections | 1.0 | Maximum number of pooled connections. |
| spring.activemq.pool.m ax-sessions-per- connection | 500.0 | Maximum number of pooled sessions per connection in the pool. |
| spring.activemq.pool.t ime-between- expiration-check | -1ms | Time to sleep between runs of the idle connection eviction thread. When negative, no idle connection eviction thread runs. |
| spring.activemq.pool.u se-anonymous-producers | true | Whether to use only one anonymous "MessageProducer" instance. Set it to false to create one "MessageProducer" every time one is required. |
| spring.activemq.send- timeout | 0ms | Time to wait on message sends for a response. Set it to 0 to wait forever. |
| spring.activemq.user |  | Login user of the broker. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.artemis.embedde d.cluster-password |  | Cluster password. Randomly generated on startup by default. |
| spring.artemis.embedde d.data-directory |  | Journal file directory. Not necessary if persistence is turned off. |
| spring.artemis.embedde d.enabled | true | Whether to enable embedded mode if the Artemis server APIs are available. |
| spring.artemis.embedde d.persistent | false | Whether to enable persistent store. |
| spring.artemis.embedde d.queues | [] | Comma-separated list of queues to create on startup. |
| spring.artemis.embedde d.server-id | 0.0 | Server ID. By default, an auto-incremented counter is used. |
| spring.artemis.embedde d.topics | [] | Comma-separated list of topics to create on startup. |
| spring.artemis.host | localhost | Artemis broker host. |
| spring.artemis.mode |  | Artemis deployment mode, auto-detected by default. |
| spring.artemis.passwor d |  | Login password of the broker. |
| spring.artemis.pool.bl ock-if-full | true | Whether to block when a connection is requested and the pool is full. Set it to false to throw a "JMSException" instead. |
| spring.artemis.pool.bl ock-if-full-timeout | -1ms | Blocking period before throwing an exception if the pool is still full. |
| spring.artemis.pool.en abled | false | Whether a JmsPoolConnectionFactory should be created, instead of a regular ConnectionFactory. |
| spring.artemis.pool.id le-timeout | 30s | Connection idle timeout. |
| spring.artemis.pool.ma x-connections | 1.0 | Maximum number of pooled connections. |
| spring.artemis.pool.ma x-sessions-per- connection | 500.0 | Maximum number of pooled sessions per connection in the pool. |
| spring.artemis.pool.ti me-between-expiration- check | -1ms | Time to sleep between runs of the idle connection eviction thread. When negative, no idle connection eviction thread runs. |
| spring.artemis.pool.us e-anonymous-producers | true | Whether to use only one anonymous "MessageProducer" instance. Set it to false to create one "MessageProducer" every time one is required. |
| spring.artemis.port | 61616.0 | Artemis broker port. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.artemis.user |  | Login user of the broker. |
| spring.batch.initializ e-schema | embedded | Database schema initialization mode. |
| spring.batch.job.enabl ed | true | Execute all Spring Batch jobs in the context on startup. |
| spring.batch.job.names |  | Comma-separated list of job names to execute on startup (for instance, `job1,job2`). By default, all Jobs found in the context are executed. |
| spring.batch.schema | classpath:org/springfr amework/batch/core/sch [ema-@@platform@@.sql](mailto:ema-@@platform@@.sql) | Path to the SQL file to use to initialize the database schema. |
| spring.batch.table- prefix |  | Table prefix for all the batch meta-data tables. |
| spring.hazelcast.confi g |  | The location of the configuration file to use to initialize Hazelcast. |
| spring.integration.jdb c.initialize-schema | embedded | Database schema initialization mode. |
| spring.integration.jdb c.schema | classpath:org/springfr amework/integration/jd bc/schema- @@platform@@.sql | Path to the SQL file to use to initialize the database schema. |
| spring.integration.rso cket.client.host |  | TCP RSocket server host to connect to. |
| spring.integration.rso cket.client.port |  | TCP RSocket server port to connect to. |
| spring.integration.rso cket.client.uri |  | WebSocket RSocket server uri to connect to. |
| spring.integration.rso cket.server.message- mapping-enabled | false | Whether to handle message mapping for RSocket via Spring Integration. |
| spring.jms.cache.consu mers | false | Whether to cache message consumers. |
| spring.jms.cache.enabl ed | true | Whether to cache sessions. |
| spring.jms.cache.produ cers | true | Whether to cache message producers. |
| spring.jms.cache.sessi on-cache-size | 1.0 | Size of the session cache (per JMS Session type). |
| spring.jms.jndi-name |  | Connection factory JNDI name. When set, takes precedence to others connection factory auto- configurations. |
| spring.jms.listener.ac knowledge-mode |  | Acknowledge mode of the container. By default, the listener is transacted with automatic acknowledgment. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.jms.listener.au to-startup | true | Start the container automatically on startup. |
| spring.jms.listener.co ncurrency |  | Minimum number of concurrent consumers. |
| spring.jms.listener.ma x-concurrency |  | Maximum number of concurrent consumers. |
| spring.jms.listener.re ceive-timeout | 1s | Timeout to use for receive calls. Use -1 for a no- wait receive or 0 for no timeout at all. The latter is only feasible if not running within a transaction manager and is generally discouraged since it prevents clean shutdown. |
| spring.jms.pub-sub- domain | false | Whether the default destination type is topic. |
| spring.jms.template.de fault-destination |  | Default destination to use on send and receive operations that do not have a destination parameter. |
| spring.jms.template.de livery-delay |  | Delivery delay to use for send calls. |
| spring.jms.template.de livery-mode |  | Delivery mode. Enables QoS (Quality of Service) when set. |
| spring.jms.template.pr iority |  | Priority of a message when sending. Enables QoS (Quality of Service) when set. |
| spring.jms.template.qo s-enabled |  | Whether to enable explicit QoS (Quality of Service) when sending a message. When enabled, the delivery mode, priority and time-to- live properties will be used when sending a message. QoS is automatically enabled when at least one of those settings is customized. |
| spring.jms.template.re ceive-timeout |  | Timeout to use for receive calls. |
| spring.jms.template.ti me-to-live |  | Time-to-live of a message when sending. Enables QoS (Quality of Service) when set. |
| spring.kafka.admin.cli ent-id |  | ID to pass to the server when making requests. Used for server-side logging. |
| spring.kafka.admin.fai l-fast | false | Whether to fail fast if the broker is not available on startup. |
| spring.kafka.admin.pro perties.\* |  | Additional admin-specific properties used to configure the client. |
| spring.kafka.admin.sec urity.protocol |  | Security protocol used to communicate with brokers. |
| spring.kafka.admin.ssl  .key-password |  | Password of the private key in the key store file. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.kafka.admin.ssl  .key-store-location |  | Location of the key store file. |
| spring.kafka.admin.ssl  .key-store-password |  | Store password for the key store file. |
| spring.kafka.admin.ssl  .key-store-type |  | Type of the key store. |
| spring.kafka.admin.ssl  .protocol |  | SSL protocol to use. |
| spring.kafka.admin.ssl  .trust-store-location |  | Location of the trust store file. |
| spring.kafka.admin.ssl  .trust-store-password |  | Store password for the trust store file. |
| spring.kafka.admin.ssl  .trust-store-type |  | Type of the trust store. |
| spring.kafka.bootstrap  -servers |  | Comma-delimited list of host:port pairs to use for establishing the initial connections to the Kafka cluster. Applies to all components unless overridden. |
| spring.kafka.client-id |  | ID to pass to the server when making requests. Used for server-side logging. |
| spring.kafka.consumer. auto-commit-interval |  | Frequency with which the consumer offsets are auto-committed to Kafka if 'enable.auto.commit' is set to true. |
| spring.kafka.consumer. auto-offset-reset |  | What to do when there is no initial offset in Kafka or if the current offset no longer exists on the server. |
| spring.kafka.consumer. bootstrap-servers |  | Comma-delimited list of host:port pairs to use for establishing the initial connections to the Kafka cluster. Overrides the global property, for consumers. |
| spring.kafka.consumer. client-id |  | ID to pass to the server when making requests. Used for server-side logging. |
| spring.kafka.consumer. enable-auto-commit |  | Whether the consumer's offset is periodically committed in the background. |
| spring.kafka.consumer. fetch-max-wait |  | Maximum amount of time the server blocks before answering the fetch request if there isn't sufficient data to immediately satisfy the requirement given by "fetch-min-size". |
| spring.kafka.consumer. fetch-min-size |  | Minimum amount of data the server should return for a fetch request. |
| spring.kafka.consumer. group-id |  | Unique string that identifies the consumer group to which this consumer belongs. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.kafka.consumer. heartbeat-interval |  | Expected time between heartbeats to the consumer coordinator. |
| spring.kafka.consumer. isolation-level |  | Isolation level for reading messages that have been written transactionally. |
| spring.kafka.consumer. key-deserializer |  | Deserializer class for keys. |
| spring.kafka.consumer. max-poll-records |  | Maximum number of records returned in a single call to poll(). |
| spring.kafka.consumer. properties.\* |  | Additional consumer-specific properties used to configure the client. |
| spring.kafka.consumer. security.protocol |  | Security protocol used to communicate with brokers. |
| spring.kafka.consumer. ssl.key-password |  | Password of the private key in the key store file. |
| spring.kafka.consumer. ssl.key-store-location |  | Location of the key store file. |
| spring.kafka.consumer. ssl.key-store-password |  | Store password for the key store file. |
| spring.kafka.consumer. ssl.key-store-type |  | Type of the key store. |
| spring.kafka.consumer. ssl.protocol |  | SSL protocol to use. |
| spring.kafka.consumer. ssl.trust-store- location |  | Location of the trust store file. |
| spring.kafka.consumer. ssl.trust-store- password |  | Store password for the trust store file. |
| spring.kafka.consumer. ssl.trust-store-type |  | Type of the trust store. |
| spring.kafka.consumer. value-deserializer |  | Deserializer class for values. |
| spring.kafka.jaas.cont rol-flag | required | Control flag for login configuration. |
| spring.kafka.jaas.enab led | false | Whether to enable JAAS configuration. |
| spring.kafka.jaas.logi n-module | com.sun.security.auth. module.Krb5LoginModule | Login module. |
| spring.kafka.jaas.opti ons.\* |  | Additional JAAS options. |
| spring.kafka.listener. ack-count |  | Number of records between offset commits when ackMode is "COUNT" or "COUNT\_TIME". |
| spring.kafka.listener. ack-mode |  | Listener AckMode. See the spring-kafka documentation. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.kafka.listener. ack-time |  | Time between offset commits when ackMode is "TIME" or "COUNT\_TIME". |
| spring.kafka.listener. client-id |  | Prefix for the listener's consumer client.id property. |
| spring.kafka.listener. concurrency |  | Number of threads to run in the listener containers. |
| spring.kafka.listener. idle-event-interval |  | Time between publishing idle consumer events (no data received). |
| spring.kafka.listener. log-container-config |  | Whether to log the container configuration during initialization (INFO level). |
| spring.kafka.listener. missing-topics-fatal | false | Whether the container should fail to start if at least one of the configured topics are not present on the broker. |
| spring.kafka.listener. monitor-interval |  | Time between checks for non-responsive consumers. If a duration suffix is not specified, seconds will be used. |
| spring.kafka.listener. no-poll-threshold |  | Multiplier applied to "pollTimeout" to determine if a consumer is non-responsive. |
| spring.kafka.listener. poll-timeout |  | Timeout to use when polling the consumer. |
| spring.kafka.listener. type | single | Listener type. |
| spring.kafka.producer. acks |  | Number of acknowledgments the producer requires the leader to have received before considering a request complete. |
| spring.kafka.producer. batch-size |  | Default batch size. A small batch size will make batching less common and may reduce throughput (a batch size of zero disables batching entirely). |
| spring.kafka.producer. bootstrap-servers |  | Comma-delimited list of host:port pairs to use for establishing the initial connections to the Kafka cluster. Overrides the global property, for producers. |
| spring.kafka.producer. buffer-memory |  | Total memory size the producer can use to buffer records waiting to be sent to the server. |
| spring.kafka.producer. client-id |  | ID to pass to the server when making requests. Used for server-side logging. |
| spring.kafka.producer. compression-type |  | Compression type for all data generated by the producer. |
| spring.kafka.producer. key-serializer |  | Serializer class for keys. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.kafka.producer. properties.\* |  | Additional producer-specific properties used to configure the client. |
| spring.kafka.producer. retries |  | When greater than zero, enables retrying of failed sends. |
| spring.kafka.producer. security.protocol |  | Security protocol used to communicate with brokers. |
| spring.kafka.producer. ssl.key-password |  | Password of the private key in the key store file. |
| spring.kafka.producer. ssl.key-store-location |  | Location of the key store file. |
| spring.kafka.producer. ssl.key-store-password |  | Store password for the key store file. |
| spring.kafka.producer. ssl.key-store-type |  | Type of the key store. |
| spring.kafka.producer. ssl.protocol |  | SSL protocol to use. |
| spring.kafka.producer. ssl.trust-store- location |  | Location of the trust store file. |
| spring.kafka.producer. ssl.trust-store- password |  | Store password for the trust store file. |
| spring.kafka.producer. ssl.trust-store-type |  | Type of the trust store. |
| spring.kafka.producer. transaction-id-prefix |  | When non empty, enables transaction support for producer. |
| spring.kafka.producer. value-serializer |  | Serializer class for values. |
| spring.kafka.propertie s.\* |  | Additional properties, common to producers and consumers, used to configure the client. |
| spring.kafka.security. protocol |  | Security protocol used to communicate with brokers. |
| spring.kafka.ssl.key- password |  | Password of the private key in the key store file. |
| spring.kafka.ssl.key- store-location |  | Location of the key store file. |
| spring.kafka.ssl.key- store-password |  | Store password for the key store file. |
| spring.kafka.ssl.key- store-type |  | Type of the key store. |
| spring.kafka.ssl.proto col |  | SSL protocol to use. |
| spring.kafka.ssl.trust  -store-location |  | Location of the trust store file. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.kafka.ssl.trust  -store-password |  | Store password for the trust store file. |
| spring.kafka.ssl.trust  -store-type |  | Type of the trust store. |
| spring.kafka.streams.a pplication-id |  | Kafka streams application.id property; default spring.application.name. |
| spring.kafka.streams.a uto-startup | true | Whether or not to auto-start the streams factory bean. |
| spring.kafka.streams.b ootstrap-servers |  | Comma-delimited list of host:port pairs to use for establishing the initial connections to the Kafka cluster. Overrides the global property, for streams. |
| spring.kafka.streams.c ache-max-size- buffering |  | Maximum memory size to be used for buffering across all threads. |
| spring.kafka.streams.c lient-id |  | ID to pass to the server when making requests. Used for server-side logging. |
| spring.kafka.streams.p roperties.\* |  | Additional Kafka properties used to configure the streams. |
| spring.kafka.streams.r eplication-factor |  | The replication factor for change log topics and repartition topics created by the stream processing application. |
| spring.kafka.streams.s ecurity.protocol |  | Security protocol used to communicate with brokers. |
| spring.kafka.streams.s sl.key-password |  | Password of the private key in the key store file. |
| spring.kafka.streams.s sl.key-store-location |  | Location of the key store file. |
| spring.kafka.streams.s sl.key-store-password |  | Store password for the key store file. |
| spring.kafka.streams.s sl.key-store-type |  | Type of the key store. |
| spring.kafka.streams.s sl.protocol |  | SSL protocol to use. |
| spring.kafka.streams.s sl.trust-store- location |  | Location of the trust store file. |
| spring.kafka.streams.s sl.trust-store- password |  | Store password for the trust store file. |
| spring.kafka.streams.s sl.trust-store-type |  | Type of the trust store. |
| spring.kafka.streams.s tate-dir |  | Directory location for the state store. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.kafka.template. default-topic |  | Default topic to which messages are sent. |
| spring.rabbitmq.addres ses |  | Comma-separated list of addresses to which the client should connect. When set, the host and port are ignored. |
| spring.rabbitmq.cache. channel.checkout- timeout |  | Duration to wait to obtain a channel if the cache size has been reached. If 0, always create a new channel. |
| spring.rabbitmq.cache. channel.size |  | Number of channels to retain in the cache. When "check-timeout" > 0, max channels per connection. |
| spring.rabbitmq.cache. connection.mode | channel | Connection factory cache mode. |
| spring.rabbitmq.cache. connection.size |  | Number of connections to cache. Only applies when mode is CONNECTION. |
| spring.rabbitmq.connec tion-timeout |  | Connection timeout. Set it to zero to wait forever. |
| spring.rabbitmq.dynami c | true | Whether to create an AmqpAdmin bean. |
| spring.rabbitmq.host | localhost | RabbitMQ host. Ignored if an address is set. |
| spring.rabbitmq.listen er.direct.acknowledge- mode |  | Acknowledge mode of container. |
| spring.rabbitmq.listen er.direct.auto-startup | true | Whether to start the container automatically on startup. |
| spring.rabbitmq.listen er.direct.consumers- per-queue |  | Number of consumers per queue. |
| spring.rabbitmq.listen er.direct.default- requeue-rejected |  | Whether rejected deliveries are re-queued by default. |
| spring.rabbitmq.listen er.direct.idle-event- interval |  | How often idle container events should be published. |
| spring.rabbitmq.listen er.direct.missing- queues-fatal | false | Whether to fail if the queues declared by the container are not available on the broker. |
| spring.rabbitmq.listen er.direct.prefetch |  | Maximum number of unacknowledged messages that can be outstanding at each consumer. |
| spring.rabbitmq.listen er.direct.retry.enable d | false | Whether publishing retries are enabled. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.rabbitmq.listen er.direct.retry.initia l-interval | 1000ms | Duration between the first and second attempt to deliver a message. |
| spring.rabbitmq.listen er.direct.retry.max- attempts | 3.0 | Maximum number of attempts to deliver a message. |
| spring.rabbitmq.listen er.direct.retry.max- interval | 10000ms | Maximum duration between attempts. |
| spring.rabbitmq.listen er.direct.retry.multip lier | 1.0 | Multiplier to apply to the previous retry interval. |
| spring.rabbitmq.listen er.direct.retry.statel ess | true | Whether retries are stateless or stateful. |
| spring.rabbitmq.listen er.simple.acknowledge- mode |  | Acknowledge mode of container. |
| spring.rabbitmq.listen er.simple.auto-startup | true | Whether to start the container automatically on startup. |
| spring.rabbitmq.listen er.simple.batch-size |  | Batch size, expressed as the number of physical messages, to be used by the container. |
| spring.rabbitmq.listen er.simple.concurrency |  | Minimum number of listener invoker threads. |
| spring.rabbitmq.listen er.simple.default- requeue-rejected |  | Whether rejected deliveries are re-queued by default. |
| spring.rabbitmq.listen er.simple.idle-event- interval |  | How often idle container events should be published. |
| spring.rabbitmq.listen er.simple.max- concurrency |  | Maximum number of listener invoker threads. |
| spring.rabbitmq.listen er.simple.missing- queues-fatal | true | Whether to fail if the queues declared by the container are not available on the broker and/or whether to stop the container if one or more queues are deleted at runtime. |
| spring.rabbitmq.listen er.simple.prefetch |  | Maximum number of unacknowledged messages that can be outstanding at each consumer. |
| spring.rabbitmq.listen er.simple.retry.enable d | false | Whether publishing retries are enabled. |
| spring.rabbitmq.listen er.simple.retry.initia l-interval | 1000ms | Duration between the first and second attempt to deliver a message. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.rabbitmq.listen er.simple.retry.max- attempts | 3.0 | Maximum number of attempts to deliver a message. |
| spring.rabbitmq.listen er.simple.retry.max- interval | 10000ms | Maximum duration between attempts. |
| spring.rabbitmq.listen er.simple.retry.multip lier | 1.0 | Multiplier to apply to the previous retry interval. |
| spring.rabbitmq.listen er.simple.retry.statel ess | true | Whether retries are stateless or stateful. |
| spring.rabbitmq.listen er.type | simple | Listener container type. |
| spring.rabbitmq.passwo rd | guest | Login to authenticate against the broker. |
| spring.rabbitmq.port |  | RabbitMQ port. Ignored if an address is set. Default to 5672, or 5671 if SSL is enabled. |
| spring.rabbitmq.publis her-confirm-type |  | Type of publisher confirms to use. |
| spring.rabbitmq.publis her-returns | false | Whether to enable publisher returns. |
| spring.rabbitmq.reques ted-channel-max | 2047.0 | Number of channels per connection requested by the client. Use 0 for unlimited. |
| spring.rabbitmq.reques ted-heartbeat |  | Requested heartbeat timeout; zero for none. If a duration suffix is not specified, seconds will be used. |
| spring.rabbitmq.ssl.al gorithm |  | SSL algorithm to use. By default, configured by the Rabbit client library. |
| spring.rabbitmq.ssl.en abled |  | Whether to enable SSL support. Determined automatically if an address is provided with the protocol (amqp:// vs. amqps://). |
| spring.rabbitmq.ssl.ke y-store |  | Path to the key store that holds the SSL certificate. |
| spring.rabbitmq.ssl.ke y-store-password |  | Password used to access the key store. |
| spring.rabbitmq.ssl.ke y-store-type | PKCS12 | Key store type. |
| spring.rabbitmq.ssl.tr ust-store |  | Trust store that holds SSL certificates. |
| spring.rabbitmq.ssl.tr ust-store-password |  | Password used to access the trust store. |
| spring.rabbitmq.ssl.tr ust-store-type | JKS | Trust store type. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.rabbitmq.ssl.va lidate-server- certificate | true | Whether to enable server side certificate validation. |
| spring.rabbitmq.ssl.ve rify-hostname | true | Whether to enable hostname verification. |
| spring.rabbitmq.templa te.default-receive- queue |  | Name of the default queue to receive messages from when none is specified explicitly. |
| spring.rabbitmq.templa te.exchange |  | Name of the default exchange to use for send operations. |
| spring.rabbitmq.templa te.mandatory |  | Whether to enable mandatory messages. |
| spring.rabbitmq.templa te.receive-timeout |  | Timeout for `receive()` operations. |
| spring.rabbitmq.templa te.reply-timeout |  | Timeout for `sendAndReceive()` operations. |
| spring.rabbitmq.templa te.retry.enabled | false | Whether publishing retries are enabled. |
| spring.rabbitmq.templa te.retry.initial- interval | 1000ms | Duration between the first and second attempt to deliver a message. |
| spring.rabbitmq.templa te.retry.max-attempts | 3.0 | Maximum number of attempts to deliver a message. |
| spring.rabbitmq.templa te.retry.max-interval | 10000ms | Maximum duration between attempts. |
| spring.rabbitmq.templa te.retry.multiplier | 1.0 | Multiplier to apply to the previous retry interval. |
| spring.rabbitmq.templa te.routing-key |  | Value of a default routing key to use for send operations. |
| spring.rabbitmq.userna me | guest | Login user to authenticate to the broker. |
| spring.rabbitmq.virtua l-host |  | Virtual host to use when connecting to the broker. |
| spring.webservices.pat h | /services | Path that serves as the base URI for the services. |
| spring.webservices.ser vlet.init.\* |  | Servlet init parameters to pass to Spring Web Services. |
| spring.webservices.ser vlet.load-on-startup | -1.0 | Load on startup priority of the Spring Web Services servlet. |
| spring.webservices.wsd l-locations |  | Comma-separated list of locations of WSDLs and accompanying XSDs to be exposed as beans. |

### Web properties

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.hateoas.use- hal-as-default-json- media-type | true | Whether application/hal+json responses should be sent to requests that accept application/json. |
| spring.jersey.applicat ion-path |  | Path that serves as the base URI for the application. If specified, overrides the value of "@ApplicationPath". |
| spring.jersey.filter.o rder | 0.0 | Jersey filter chain order. |
| spring.jersey.init.\* |  | Init parameters to pass to Jersey through the servlet or filter. |
| spring.jersey.servlet. load-on-startup | -1.0 | Load on startup priority of the Jersey servlet. |
| spring.jersey.type | servlet | Jersey integration type. |
| spring.mvc.async.reque st-timeout |  | Amount of time before asynchronous request handling times out. If this value is not set, the default timeout of the underlying implementation is used. |
| spring.mvc.contentnego tiation.favor- parameter | false | Whether a request parameter ("format" by default) should be used to determine the requested media type. |
| spring.mvc.contentnego tiation.media-types.\* |  | Map file extensions to media types for content negotiation. For instance, yml to text/yaml. |
| spring.mvc.contentnego tiation.parameter-name |  | Query parameter name to use when "favor- parameter" is enabled. |
| spring.mvc.converters. preferred-json-mapper |  | Preferred JSON mapper to use for HTTP message conversion. By default, auto-detected according to the environment. |
| spring.mvc.dispatch- options-request | true | Whether to dispatch OPTIONS requests to the FrameworkServlet doService method. |
| spring.mvc.dispatch- trace-request | false | Whether to dispatch TRACE requests to the FrameworkServlet doService method. |
| spring.mvc.format.date |  | Date format to use, for example `dd/MM/yyyy`. |
| spring.mvc.format.date  -time |  | Date-time format to use, for example `yyyy-MM- dd HH:mm:ss`. |
| spring.mvc.format.time |  | Time format to use, for example `HH:mm:ss`. |
| spring.mvc.formcontent  .filter.enabled | true | Whether to enable Spring's FormContentFilter. |
| spring.mvc.hiddenmetho d.filter.enabled | false | Whether to enable Spring's HiddenHttpMethodFilter. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.mvc.ignore- default-model-on- redirect | true | Whether the content of the "default" model should be ignored during redirect scenarios. |
| spring.mvc.locale |  | Locale to use. By default, this locale is overridden by the "Accept-Language" header. |
| spring.mvc.locale- resolver | accept-header | Define how the locale should be resolved. |
| spring.mvc.log- request-details | false | Whether logging of (potentially sensitive) request details at DEBUG and TRACE level is allowed. |
| spring.mvc.log- resolved-exception | false | Whether to enable warn logging of exceptions resolved by a "HandlerExceptionResolver", except for "DefaultHandlerExceptionResolver". |
| spring.mvc.message- codes-resolver-format |  | Formatting strategy for message codes. For instance, `PREFIX\_ERROR\_CODE`. |
| spring.mvc.publish- request-handled-events | true | Whether to publish a ServletRequestHandledEvent at the end of each request. |
| spring.mvc.servlet.loa d-on-startup | -1.0 | Load on startup priority of the dispatcher servlet. |
| spring.mvc.servlet.pat h | / | Path of the dispatcher servlet. |
| spring.mvc.static- path-pattern | /\*\* | Path pattern used for static resources. |
| spring.mvc.throw- exception-if-no- handler-found | false | Whether a "NoHandlerFoundException" should be thrown if no Handler was found to process a request. |
| spring.mvc.view.prefix |  | Spring MVC view prefix. |
| spring.mvc.view.suffix |  | Spring MVC view suffix. |
| spring.resources.add- mappings | true | Whether to enable default resource handling. |
| spring.resources.cache  .cachecontrol.cache- private |  | Indicate that the response message is intended for a single user and must not be stored by a shared cache. |
| spring.resources.cache  .cachecontrol.cache- public |  | Indicate that any cache may store the response. |
| spring.resources.cache  .cachecontrol.max-age |  | Maximum time the response should be cached, in seconds if no duration suffix is not specified. |
| spring.resources.cache  .cachecontrol.must- revalidate |  | Indicate that once it has become stale, a cache must not use the response without re-validating it with the server. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.resources.cache  .cachecontrol.no-cache |  | Indicate that the cached response can be reused only if re-validated with the server. |
| spring.resources.cache  .cachecontrol.no-store |  | Indicate to not cache the response in any case. |
| spring.resources.cache  .cachecontrol.no- transform |  | Indicate intermediaries (caches and others) that they should not transform the response content. |
| spring.resources.cache  .cachecontrol.proxy- revalidate |  | Same meaning as the "must-revalidate" directive, except that it does not apply to private caches. |
| spring.resources.cache  .cachecontrol.s-max- age |  | Maximum time the response should be cached by shared caches, in seconds if no duration suffix is not specified. |
| spring.resources.cache  .cachecontrol.stale- if-error |  | Maximum time the response may be used when errors are encountered, in seconds if no duration suffix is not specified. |
| spring.resources.cache  .cachecontrol.stale- while-revalidate |  | Maximum time the response can be served after it becomes stale, in seconds if no duration suffix is not specified. |
| spring.resources.cache  .period |  | Cache period for the resources served by the resource handler. If a duration suffix is not specified, seconds will be used. Can be overridden by the 'spring.resources.cache.cachecontrol' properties. |
| spring.resources.chain  .cache | true | Whether to enable caching in the Resource chain. |
| spring.resources.chain  .compressed | false | Whether to enable resolution of already compressed resources (gzip, brotli). Checks for a resource name with the '.gz' or '.br' file extensions. |
| spring.resources.chain  .enabled |  | Whether to enable the Spring Resource Handling chain. By default, disabled unless at least one strategy has been enabled. |
| spring.resources.chain  .html-application- cache | false | Whether to enable HTML5 application cache manifest rewriting. |
| spring.resources.chain  .strategy.content.enab led | false | Whether to enable the content Version Strategy. |
| spring.resources.chain  .strategy.content.path s | [/\*\*] | Comma-separated list of patterns to apply to the content Version Strategy. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.resources.chain  .strategy.fixed.enable d | false | Whether to enable the fixed Version Strategy. |
| spring.resources.chain  .strategy.fixed.paths | [/\*\*] | Comma-separated list of patterns to apply to the fixed Version Strategy. |
| spring.resources.chain  .strategy.fixed.versio n |  | Version string to use for the fixed Version Strategy. |
| spring.resources.stati c-locations | [classpath:/META- INF/resources/, classpath:/resources/, classpath:/static/, classpath:/public/] | Locations of static resources. Defaults to classpath:[/META-INF/resources/, /resources/,  /static/, /public/]. |
| spring.servlet.multipa rt.enabled | true | Whether to enable support of multipart uploads. |
| spring.servlet.multipa rt.file-size-threshold | 0B | Threshold after which files are written to disk. |
| spring.servlet.multipa rt.location |  | Intermediate location of uploaded files. |
| spring.servlet.multipa rt.max-file-size | 1MB | Max file size. |
| spring.servlet.multipa rt.max-request-size | 10MB | Max request size. |
| spring.servlet.multipa rt.resolve-lazily | false | Whether to resolve the multipart request lazily at the time of file or parameter access. |
| spring.webflux.base- path |  | Base path for all web handlers. |
| spring.webflux.format. date |  | Date format to use, for example `dd/MM/yyyy`. |
| spring.webflux.format. date-time |  | Date-time format to use, for example `yyyy-MM- dd HH:mm:ss`. |
| spring.webflux.format. time |  | Time format to use, for example `HH:mm:ss`. |
| spring.webflux.hiddenm ethod.filter.enabled | false | Whether to enable Spring's HiddenHttpMethodFilter. |
| spring.webflux.static- path-pattern | /\*\* | Path pattern used for static resources. |

### Templating properties

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.freemarker.allo w-request-override | false | Whether HttpServletRequest attributes are allowed to override (hide) controller generated model attributes of the same name. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.freemarker.allo w-session-override | false | Whether HttpSession attributes are allowed to override (hide) controller generated model attributes of the same name. |
| spring.freemarker.cach e | false | Whether to enable template caching. |
| spring.freemarker.char set | UTF-8 | Template encoding. |
| spring.freemarker.chec k-template-location | true | Whether to check that the templates location exists. |
| spring.freemarker.cont ent-type | text/html | Content-Type value. |
| spring.freemarker.enab led | true | Whether to enable MVC view resolution for this technology. |
| spring.freemarker.expo se-request-attributes | false | Whether all request attributes should be added to the model prior to merging with the template. |
| spring.freemarker.expo se-session-attributes | false | Whether all HttpSession attributes should be added to the model prior to merging with the template. |
| spring.freemarker.expo se-spring-macro- helpers | true | Whether to expose a RequestContext for use by Spring's macro library, under the name "springMacroRequestContext". |
| spring.freemarker.pref er-file-system-access | true | Whether to prefer file system access for template loading. File system access enables hot detection of template changes. |
| spring.freemarker.pref ix |  | Prefix that gets prepended to view names when building a URL. |
| spring.freemarker.requ est-context-attribute |  | Name of the RequestContext attribute for all views. |
| spring.freemarker.sett ings.\* |  | Well-known FreeMarker keys which are passed to FreeMarker's Configuration. |
| spring.freemarker.suff ix | .ftlh | Suffix that gets appended to view names when building a URL. |
| spring.freemarker.temp late-loader-path | [classpath:/templates/  ] | Comma-separated list of template paths. |
| spring.freemarker.view  -names |  | View names that can be resolved. |
| spring.groovy.template  .allow-request- override | false | Whether HttpServletRequest attributes are allowed to override (hide) controller generated model attributes of the same name. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.groovy.template  .allow-session- override | false | Whether HttpSession attributes are allowed to override (hide) controller generated model attributes of the same name. |
| spring.groovy.template  .cache | false | Whether to enable template caching. |
| spring.groovy.template  .charset | UTF-8 | Template encoding. |
| spring.groovy.template  .check-template- location | true | Whether to check that the templates location exists. |
| spring.groovy.template  .configuration.auto- escape  spring.groovy.template  .configuration.auto- indent  spring.groovy.template  .configuration.auto- indent-string  spring.groovy.template  .configuration.auto- new-line  spring.groovy.template  .configuration.base- template-class  spring.groovy.template  .configuration.cache- templates  spring.groovy.template  .configuration.declara tion-encoding  spring.groovy.template  .configuration.expand- empty-elements spring.groovy.template  .configuration.locale  spring.groovy.template  .configuration.new- line-string  spring.groovy.template  .configuration.resourc e-loader-path  spring.groovy.template  .configuration.use- double-quotes |  | See GroovyMarkupConfigurer |
| spring.groovy.template  .content-type | text/html | Content-Type value. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.groovy.template  .enabled | true | Whether to enable MVC view resolution for this technology. |
| spring.groovy.template  .expose-request- attributes | false | Whether all request attributes should be added to the model prior to merging with the template. |
| spring.groovy.template  .expose-session- attributes | false | Whether all HttpSession attributes should be added to the model prior to merging with the template. |
| spring.groovy.template  .expose-spring-macro- helpers | true | Whether to expose a RequestContext for use by Spring's macro library, under the name "springMacroRequestContext". |
| spring.groovy.template  .prefix |  | Prefix that gets prepended to view names when building a URL. |
| spring.groovy.template  .request-context- attribute |  | Name of the RequestContext attribute for all views. |
| spring.groovy.template  .resource-loader-path | classpath:/templates/ | Template path. |
| spring.groovy.template  .suffix | .tpl | Suffix that gets appended to view names when building a URL. |
| spring.groovy.template  .view-names |  | View names that can be resolved. |
| spring.mustache.allow- request-override | false | Whether HttpServletRequest attributes are allowed to override (hide) controller generated model attributes of the same name. |
| spring.mustache.allow- session-override | false | Whether HttpSession attributes are allowed to override (hide) controller generated model attributes of the same name. |
| spring.mustache.cache | false | Whether to enable template caching. |
| spring.mustache.charse t | UTF-8 | Template encoding. |
| spring.mustache.check- template-location | true | Whether to check that the templates location exists. |
| spring.mustache.conten t-type | text/html | Content-Type value. |
| spring.mustache.enable d | true | Whether to enable MVC view resolution for this technology. |
| spring.mustache.expose  -request-attributes | false | Whether all request attributes should be added to the model prior to merging with the template. |
| spring.mustache.expose  -session-attributes | false | Whether all HttpSession attributes should be added to the model prior to merging with the template. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.mustache.expose  -spring-macro-helpers | true | Whether to expose a RequestContext for use by Spring's macro library, under the name "springMacroRequestContext". |
| spring.mustache.prefix | classpath:/templates/ | Prefix to apply to template names. |
| spring.mustache.reques t-context-attribute |  | Name of the RequestContext attribute for all views. |
| spring.mustache.suffix | .mustache | Suffix to apply to template names. |
| spring.mustache.view- names |  | View names that can be resolved. |
| spring.thymeleaf.cache | true | Whether to enable template caching. |
| spring.thymeleaf.check  -template | true | Whether to check that the template exists before rendering it. |
| spring.thymeleaf.check  -template-location | true | Whether to check that the templates location exists. |
| spring.thymeleaf.enabl e-spring-el-compiler | false | Enable the SpringEL compiler in SpringEL expressions. |
| spring.thymeleaf.enabl ed | true | Whether to enable Thymeleaf view resolution for Web frameworks. |
| spring.thymeleaf.encod ing | UTF-8 | Template files encoding. |
| spring.thymeleaf.exclu ded-view-names |  | Comma-separated list of view names (patterns allowed) that should be excluded from resolution. |
| spring.thymeleaf.mode | HTML | Template mode to be applied to templates. See also Thymeleaf's TemplateMode enum. |
| spring.thymeleaf.prefi x | classpath:/templates/ | Prefix that gets prepended to view names when building a URL. |
| spring.thymeleaf.react ive.chunked-mode-view- names |  | Comma-separated list of view names (patterns allowed) that should be the only ones executed in CHUNKED mode when a max chunk size is set. |
| spring.thymeleaf.react ive.full-mode-view- names |  | Comma-separated list of view names (patterns allowed) that should be executed in FULL mode even if a max chunk size is set. |
| spring.thymeleaf.react ive.max-chunk-size | 0B | Maximum size of data buffers used for writing to the response. Templates will execute in CHUNKED mode by default if this is set. |
| spring.thymeleaf.react ive.media-types |  | Media types supported by the view technology. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.thymeleaf.rende r-hidden-markers- before-checkboxes | false | Whether hidden form inputs acting as markers for checkboxes should be rendered before the checkbox element itself. |
| spring.thymeleaf.servl et.content-type | text/html | Content-Type value written to HTTP responses. |
| spring.thymeleaf.servl et.produce-partial- output-while- processing | true | Whether Thymeleaf should start writing partial output as soon as possible or buffer until template processing is finished. |
| spring.thymeleaf.suffi x | .html | Suffix that gets appended to view names when building a URL. |
| spring.thymeleaf.templ ate-resolver-order |  | Order of the template resolver in the chain. By default, the template resolver is first in the chain. Order start at 1 and should only be set if you have defined additional "TemplateResolver" beans. |
| spring.thymeleaf.view- names |  | Comma-separated list of view names (patterns allowed) that can be resolved. |

### Server properties

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| server.address |  | Network address to which the server should bind. |
| server.compression.ena bled | false | Whether response compression is enabled. |
| server.compression.exc luded-user-agents |  | Comma-separated list of user agents for which responses should not be compressed. |
| server.compression.mim e-types | [text/html, text/xml, text/plain, text/css, text/javascript, application/javascript  , application/json, application/xml] | Comma-separated list of MIME types that should be compressed. |
| server.compression.min  -response-size | 2KB | Minimum "Content-Length" value that is required for compression to be performed. |
| server.error.include- binding-errors | never | When to include "errors" attribute. |
| server.error.include- exception | false | Include the "exception" attribute. |
| server.error.include- message | never | When to include "message" attribute. |
| server.error.include- stacktrace | never | When to include the "trace" attribute. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| server.error.path | /error | Path of the error controller. |
| server.error.whitelabe l.enabled | true | Whether to enable the default error page displayed in browsers in case of a server error. |
| server.forward- headers-strategy |  | Strategy for handling X-Forwarded-\* headers. |
| server.http2.enabled | false | Whether to enable HTTP/2 support, if the current environment supports it. |
| server.jetty.accesslog  .append | false | Append to log. |
| server.jetty.accesslog  .custom-format |  | Custom log format, see org.eclipse.jetty.server.CustomRequestLog. If defined, overrides the "format" configuration key. |
| server.jetty.accesslog  .enabled | false | Enable access log. |
| server.jetty.accesslog  .file-date-format |  | Date format to place in log file name. |
| server.jetty.accesslog  .filename |  | Log filename. If not specified, logs redirect to "System.err". |
| server.jetty.accesslog  .format | ncsa | Log format. |
| server.jetty.accesslog  .ignore-paths |  | Request paths that should not be logged. |
| server.jetty.accesslog  .retention-period | 31.0 | Number of days before rotated log files are deleted. |
| server.jetty.connectio n-idle-timeout |  | Time that the connection can be idle before it is closed. |
| server.jetty.max-http- form-post-size | 200000B | Maximum size of the form content in any HTTP post request. |
| server.jetty.threads.a cceptors | -1.0 | Number of acceptor threads to use. When the value is -1, the default, the number of acceptors is derived from the operating environment. |
| server.jetty.threads.i dle-timeout | 60000ms | Maximum thread idle time. |
| server.jetty.threads.m ax | 200.0 | Maximum number of threads. |
| server.jetty.threads.m ax-queue-capacity |  | Maximum capacity of the thread pool's backing queue. A default is computed based on the threading configuration. |
| server.jetty.threads.m in | 8.0 | Minimum number of threads. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| server.jetty.threads.s electors | -1.0 | Number of selector threads to use. When the value is -1, the default, the number of selectors is derived from the operating environment. |
| server.max-http- header-size | 8KB | Maximum size of the HTTP message header. |
| server.netty.connectio n-timeout |  | Connection timeout of the Netty channel. |
| server.port | 8080.0 | Server HTTP port. |
| server.server-header |  | Value to use for the Server response header (if empty, no header is sent). |
| server.servlet.applica tion-display-name | application | Display name of the application. |
| server.servlet.context  -parameters.\* |  | Servlet context init parameters. |
| server.servlet.context  -path |  | Context path of the application. |
| server.servlet.encodin g.charset |  |  |
| server.servlet.encodin g.enabled | true | Whether to enable http encoding support. |
| server.servlet.encodin g.force |  |  |
| server.servlet.encodin g.force-request |  |  |
| server.servlet.encodin g.force-response |  |  |
| server.servlet.encodin g.mapping.\* |  |  |
| server.servlet.jsp.cla ss-name | org.apache.jasper.serv let.JspServlet | Class name of the servlet to use for JSPs. If registered is true and this class \* is on the classpath then it will be registered. |
| server.servlet.jsp.ini t-parameters.\* |  | Init parameters used to configure the JSP servlet. |
| server.servlet.jsp.reg istered | true | Whether the JSP servlet is registered. |
| server.servlet.registe r-default-servlet | true | Whether to register the default Servlet with the container. |
| server.servlet.session  .cookie.comment |  | Comment for the session cookie. |
| server.servlet.session  .cookie.domain |  | Domain for the session cookie. |
| server.servlet.session  .cookie.http-only |  | Whether to use "HttpOnly" cookies for session cookies. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| server.servlet.session  .cookie.max-age |  | Maximum age of the session cookie. If a duration suffix is not specified, seconds will be used. |
| server.servlet.session  .cookie.name |  | Session cookie name. |
| server.servlet.session  .cookie.path |  | Path of the session cookie. |
| server.servlet.session  .cookie.secure |  | Whether to always mark the session cookie as secure. |
| server.servlet.session  .persistent | false | Whether to persist session data between restarts. |
| server.servlet.session  .store-dir |  | Directory used to store session data. |
| server.servlet.session  .timeout | 30m | Session timeout. If a duration suffix is not specified, seconds will be used. |
| server.servlet.session  .tracking-modes |  | Session tracking modes. |
| server.shutdown |  | Type of shutdown that the server will support. |
| server.ssl.ciphers |  | Supported SSL ciphers. |
| server.ssl.client-auth |  | Client authentication mode. Requires a trust store. |
| server.ssl.enabled | true | Whether to enable SSL support. |
| server.ssl.enabled- protocols |  | Enabled SSL protocols. |
| server.ssl.key-alias |  | Alias that identifies the key in the key store. |
| server.ssl.key- password |  | Password used to access the key in the key store. |
| server.ssl.key-store |  | Path to the key store that holds the SSL certificate (typically a jks file). |
| server.ssl.key-store- password |  | Password used to access the key store. |
| server.ssl.key-store- provider |  | Provider for the key store. |
| server.ssl.key-store- type |  | Type of the key store. |
| server.ssl.protocol | TLS | SSL protocol to use. |
| server.ssl.trust-store |  | Trust store that holds SSL certificates. |
| server.ssl.trust- store-password |  | Password used to access the trust store. |
| server.ssl.trust- store-provider |  | Provider for the trust store. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| server.ssl.trust- store-type |  | Type of the trust store. |
| server.tomcat.accept- count | 100.0 | Maximum queue length for incoming connection requests when all possible request processing threads are in use. |
| server.tomcat.accesslo g.buffered | true | Whether to buffer output such that it is flushed only periodically. |
| server.tomcat.accesslo g.check-exists | false | Whether to check for log file existence so it can be recreated it if an external process has renamed it. |
| server.tomcat.accesslo g.condition-if |  | Whether logging of the request will only be enabled if "ServletRequest.getAttribute(conditionIf)" does not yield null. |
| server.tomcat.accesslo g.condition-unless |  | Whether logging of the request will only be enabled if "ServletRequest.getAttribute(conditionUnless)" yield null. |
| server.tomcat.accesslo g.directory | logs | Directory in which log files are created. Can be absolute or relative to the Tomcat base dir. |
| server.tomcat.accesslo g.enabled | false | Enable access log. |
| server.tomcat.accesslo g.encoding |  | Character set used by the log file. Default to the system default character set. |
| server.tomcat.accesslo g.file-date-format | .yyyy-MM-dd | Date format to place in the log file name. |
| server.tomcat.accesslo g.ipv6-canonical | false | Whether to use IPv6 canonical representation format as defined by RFC 5952. |
| server.tomcat.accesslo g.locale |  | Locale used to format timestamps in log entries and in log file name suffix. Default to the default locale of the Java process. |
| server.tomcat.accesslo g.max-days | -1.0 | Number of days to retain the access log files before they are removed. |
| server.tomcat.accesslo g.pattern | common | Format pattern for access logs. |
| server.tomcat.accesslo g.prefix | access\_log | Log file name prefix. |
| server.tomcat.accesslo g.rename-on-rotate | false | Whether to defer inclusion of the date stamp in the file name until rotate time. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| server.tomcat.accesslo g.request-attributes- enabled | false | Set request attributes for the IP address, Hostname, protocol, and port used for the request. |
| server.tomcat.accesslo g.rotate | true | Whether to enable access log rotation. |
| server.tomcat.accesslo g.suffix | .log | Log file name suffix. |
| server.tomcat.addition al-tld-skip-patterns |  | Comma-separated list of additional patterns that match jars to ignore for TLD scanning. The special '?' and '\*' characters can be used in the pattern to match one and only one character and zero or more characters respectively. |
| server.tomcat.backgrou nd-processor-delay | 10s | Delay between the invocation of backgroundProcess methods. If a duration suffix is not specified, seconds will be used. |
| server.tomcat.basedir |  | Tomcat base directory. If not specified, a temporary directory is used. |
| server.tomcat.connecti on-timeout |  | Amount of time the connector will wait, after accepting a connection, for the request URI line to be presented. |
| server.tomcat.max- connections | 8192.0 | Maximum number of connections that the server accepts and processes at any given time. Once the limit has been reached, the operating system may still accept connections based on the "acceptCount" property. |
| server.tomcat.max- http-form-post-size | 2MB | Maximum size of the form content in any HTTP post request. |
| server.tomcat.max- swallow-size | 2MB | Maximum amount of request body to swallow. |
| server.tomcat.mbeanreg istry.enabled | false | Whether Tomcat's MBean Registry should be enabled. |
| server.tomcat.processo r-cache | 200.0 | Maximum number of idle processors that will be retained in the cache and reused with a subsequent request. When set to -1 the cache will be unlimited with a theoretical maximum size equal to the maximum number of connections. |
| server.tomcat.redirect  -context-root | true | Whether requests to the context root should be redirected by appending a / to the path. |
| server.tomcat.relaxed- path-chars |  | Comma-separated list of additional unencoded characters that should be allowed in URI paths. Only "< > [ \ ] ^ ` { | }" are allowed. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| server.tomcat.relaxed- query-chars |  | Comma-separated list of additional unencoded characters that should be allowed in URI query strings. Only "< > [ \ ] ^ ` { | }" are allowed. |
| server.tomcat.remoteip  .host-header | X-Forwarded-Host | Name of the HTTP header from which the remote host is extracted. |
| server.tomcat.remoteip  .internal-proxies | 10\\.\\d{1,3}\\.\\d{1,  3}\\.\\d{1,3}|192\\.16  8\\.\\d{1,3}\\.\\d{1,3  }|169\\.254\\.\\d{1,3}  \\.\\d{1,3}|127\\.\\d{ 1,3}\\.\\d{1,3}\\.\\d{  1,3}|172\\.1[6-  9]{1}\\.\\d{1,3}\\.\\d  {1,3}|172\\.2[0-  9]{1}\\.\\d{1,3}\\.\\d  {1,3}|172\\.3[0-  1]{1}\\.\\d{1,3}\\.\\d  {1,3}|0:0:0:0:0:0:0:1|  ::1 | Regular expression that matches proxies that are to be trusted. |
| server.tomcat.remoteip  .port-header | X-Forwarded-Port | Name of the HTTP header used to override the original port value. |
| server.tomcat.remoteip  .protocol-header |  | Header that holds the incoming protocol, usually named "X-Forwarded-Proto". |
| server.tomcat.remoteip  .protocol-header- https-value | https | Value of the protocol header indicating whether the incoming request uses SSL. |
| server.tomcat.remoteip  .remote-ip-header |  | Name of the HTTP header from which the remote IP is extracted. For instance, `X- FORWARDED-FOR`. |
| server.tomcat.resource  .allow-caching | true | Whether static resource caching is permitted for this web application. |
| server.tomcat.resource  .cache-ttl |  | Time-to-live of the static resource cache. |
| server.tomcat.threads. max | 200.0 | Maximum amount of worker threads. |
| server.tomcat.threads. min-spare | 10.0 | Minimum amount of worker threads. |
| server.tomcat.uri- encoding | UTF-8 | Character encoding to use to decode the URI. |
| server.tomcat.use- relative-redirects | false | Whether HTTP 1.1 and later location headers generated by a call to sendRedirect will use relative or absolute redirects. |
| server.undertow.access log.dir |  | Undertow access log directory. |
| server.undertow.access log.enabled | false | Whether to enable the access log. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| server.undertow.access log.pattern | common | Format pattern for access logs. |
| server.undertow.access log.prefix | access\_log. | Log file name prefix. |
| server.undertow.access log.rotate | true | Whether to enable access log rotation. |
| server.undertow.access log.suffix | log | Log file name suffix. |
| server.undertow.allow- encoded-slash | false | Whether the server should decode percent encoded slash characters. Enabling encoded slashes can have security implications due to different servers interpreting the slash differently. Only enable this if you have a legacy application that requires it. |
| server.undertow.always  -set-keep-alive | true | Whether the 'Connection: keep-alive' header should be added to all responses, even if not required by the HTTP specification. |
| server.undertow.buffer  -size |  | Size of each buffer. The default is derived from the maximum amount of memory that is available to the JVM. |
| server.undertow.decode  -url | true | Whether the URL should be decoded. When disabled, percent-encoded characters in the URL will be left as-is. |
| server.undertow.direct  -buffers |  | Whether to allocate buffers outside the Java heap. The default is derived from the maximum amount of memory that is available to the JVM. |
| server.undertow.eager- filter-init | true | Whether servlet filters should be initialized on startup. |
| server.undertow.max- cookies | 200.0 | Maximum number of cookies that are allowed. This limit exists to prevent hash collision based DOS attacks. |
| server.undertow.max- headers |  | Maximum number of headers that are allowed. This limit exists to prevent hash collision based DOS attacks. |
| server.undertow.max- http-post-size | -1B | Maximum size of the HTTP post content. When the value is -1, the default, the size is unlimited. |
| server.undertow.max- parameters |  | Maximum number of query or path parameters that are allowed. This limit exists to prevent hash collision based DOS attacks. |
| server.undertow.no- request-timeout |  | Amount of time a connection can sit idle without processing a request, before it is closed by the server. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| server.undertow.option s.server.\* |  |  |
| server.undertow.option s.socket.\* |  |  |
| server.undertow.thread s.io |  | Number of I/O threads to create for the worker. The default is derived from the number of available processors. |
| server.undertow.thread s.worker |  | Number of worker threads. The default is 8 times the number of I/O threads. |
| server.undertow.url- charset | UTF-8 | Charset used to decode URLs. |

### Security properties

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.ldap.anonymous- read-only |  | Whether read-only operations should use an anonymous environment. Disabled by default unless a username is set. |
| spring.ldap.base |  | Base suffix from which all operations should originate. |
| spring.ldap.base- environment.\* |  | LDAP specification settings. |
| spring.ldap.embedded.b ase-dn |  | List of base DNs. |
| spring.ldap.embedded.c redential.password |  | Embedded LDAP password. |
| spring.ldap.embedded.c redential.username |  | Embedded LDAP username. |
| spring.ldap.embedded.l dif | classpath:schema.ldif | Schema (LDIF) script resource reference. |
| spring.ldap.embedded.p ort | 0.0 | Embedded LDAP port. |
| spring.ldap.embedded.v alidation.enabled | true | Whether to enable LDAP schema validation. |
| spring.ldap.embedded.v alidation.schema |  | Path to the custom schema. |
| spring.ldap.password |  | Login password of the server. |
| spring.ldap.urls |  | LDAP URLs of the server. |
| spring.ldap.username |  | Login username of the server. |
| spring.security.filter  .dispatcher-types | [async, error, request] | Security filter chain dispatcher types. |
| spring.security.filter  .order | -100.0 | Security filter chain order. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.security.oauth2  .client.provider.\* |  | OAuth provider details. |
| spring.security.oauth2  .client.registration.\* |  | OAuth client registrations. |
| spring.security.oauth2  .resourceserver.jwt.is suer-uri |  | URI that can either be an OpenID Connect discovery endpoint or an OAuth 2.0 Authorization Server Metadata endpoint defined by RFC 8414. |
| spring.security.oauth2  .resourceserver.jwt.jw k-set-uri |  | JSON Web Key URI to use to verify the JWT token. |
| spring.security.oauth2  .resourceserver.jwt.jw s-algorithm | RS256 | JSON Web Algorithm used for verifying the digital signatures. |
| spring.security.oauth2  .resourceserver.jwt.pu blic-key-location |  | Location of the file containing the public key used to verify a JWT. |
| spring.security.oauth2  .resourceserver.opaque token.client-id |  | Client id used to authenticate with the token introspection endpoint. |
| spring.security.oauth2  .resourceserver.opaque token.client-secret |  | Client secret used to authenticate with the token introspection endpoint. |
| spring.security.oauth2  .resourceserver.opaque token.introspection- uri |  | OAuth 2.0 endpoint through which token introspection is accomplished. |
| spring.security.saml2. relyingparty.registrat ion.\* |  | SAML2 relying party registrations. |
| spring.security.user.n ame | user | Default user name. |
| spring.security.user.p assword |  | Password for the default user name. |
| spring.security.user.r oles |  | Granted roles for the default user name. |
| spring.session.hazelca st.flush-mode | on-save | Sessions flush mode. Determines when session changes are written to the session store. |
| spring.session.hazelca st.map-name | spring:session:session s | Name of the map used to store sessions. |
| spring.session.hazelca st.save-mode | on-set-attribute | Sessions save mode. Determines how session changes are tracked and saved to the session store. |
| spring.session.jdbc.cl eanup-cron | 0 \* \* \* \* \* | Cron expression for expired session cleanup job. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.session.jdbc.fl ush-mode | on-save | Sessions flush mode. Determines when session changes are written to the session store. |
| spring.session.jdbc.in itialize-schema | embedded | Database schema initialization mode. |
| spring.session.jdbc.sa ve-mode | on-set-attribute | Sessions save mode. Determines how session changes are tracked and saved to the session store. |
| spring.session.jdbc.sc hema | classpath:org/springfr amework/session/jdbc/s [chema-@@platform@@.sql](mailto:chema-@@platform@@.sql) | Path to the SQL file to use to initialize the database schema. |
| spring.session.jdbc.ta ble-name | SPRING\_SESSION | Name of the database table used to store sessions. |
| spring.session.mongodb  .collection-name | sessions | Collection name used to store sessions. |
| spring.session.redis.c leanup-cron | 0 \* \* \* \* \* | Cron expression for expired session cleanup job. |
| spring.session.redis.c onfigure-action | notify-keyspace-events | The configure action to apply when no user defined ConfigureRedisAction bean is present. |
| spring.session.redis.f lush-mode | on-save | Sessions flush mode. Determines when session changes are written to the session store. |
| spring.session.redis.n amespace | spring:session | Namespace for keys used to store sessions. |
| spring.session.redis.s ave-mode | on-set-attribute | Sessions save mode. Determines how session changes are tracked and saved to the session store. |
| spring.session.servlet  .filter-dispatcher- types | [async, error, request] | Session repository filter dispatcher types. |
| spring.session.servlet  .filter-order |  | Session repository filter order. |
| spring.session.store- type |  | Session store type. |
| spring.session.timeout |  | Session timeout. If a duration suffix is not specified, seconds will be used. |

### RSocket properties

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.rsocket.server. address |  | Network address to which the server should bind. |
| spring.rsocket.server. mapping-path |  | Path under which RSocket handles requests (only works with websocket transport). |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.rsocket.server. port |  | Server port. |
| spring.rsocket.server. transport | tcp | RSocket transport protocol. |

### Actuator properties

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.auditevents  .enabled | true | Whether to enable storage of audit events. |
| management.cloudfoundr y.enabled | true | Whether to enable extended Cloud Foundry actuator endpoints. |
| management.cloudfoundr y.skip-ssl-validation | false | Whether to skip SSL verification for Cloud Foundry actuator endpoint security calls. |
| management.endpoint.au ditevents.cache.time- to-live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.au ditevents.enabled | true | Whether to enable the auditevents endpoint. |
| management.endpoint.be ans.cache.time-to-live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.be ans.enabled | true | Whether to enable the beans endpoint. |
| management.endpoint.ca ches.cache.time-to- live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.ca ches.enabled | true | Whether to enable the caches endpoint. |
| management.endpoint.co nditions.cache.time- to-live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.co nditions.enabled | true | Whether to enable the conditions endpoint. |
| management.endpoint.co nfigprops.cache.time- to-live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.co nfigprops.enabled | true | Whether to enable the configprops endpoint. |
| management.endpoint.co nfigprops.keys-to- sanitize | [password, secret, key, token,  .\*credentials.\*, vcap\_services, sun.java.command] | Keys that should be sanitized. Keys can be simple strings that the property ends with or regular expressions. |
| management.endpoint.en v.cache.time-to-live | 0ms | Maximum time that a response can be cached. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.endpoint.en v.enabled | true | Whether to enable the env endpoint. |
| management.endpoint.en v.keys-to-sanitize | [password, secret, key, token,  .\*credentials.\*, vcap\_services, sun.java.command] | Keys that should be sanitized. Keys can be simple strings that the property ends with or regular expressions. |
| management.endpoint.fl yway.cache.time-to- live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.fl yway.enabled | true | Whether to enable the flyway endpoint. |
| management.endpoint.he alth.cache.time-to- live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.he alth.enabled | true | Whether to enable the health endpoint. |
| management.endpoint.he alth.group.\* |  | Health endpoint groups. |
| management.endpoint.he alth.probes.enabled | false | Whether to enable liveness and readiness probes. |
| management.endpoint.he alth.roles |  | Roles used to determine whether or not a user is authorized to be shown details. When empty, all authenticated users are authorized. |
| management.endpoint.he alth.show-components |  | When to show components. If not specified the 'show-details' setting will be used. |
| management.endpoint.he alth.show-details | never | When to show full health details. |
| management.endpoint.he alth.status.http- mapping.\* |  | Mapping of health statuses to HTTP status codes. By default, registered health statuses map to sensible defaults (for example, UP maps to 200). |
| management.endpoint.he alth.status.order | [DOWN, OUT\_OF\_SERVICE, UP, UNKNOWN] | Comma-separated list of health statuses in order of severity. |
| management.endpoint.he apdump.cache.time-to- live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.he apdump.enabled | true | Whether to enable the heapdump endpoint. |
| management.endpoint.ht tptrace.cache.time-to- live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.ht tptrace.enabled | true | Whether to enable the httptrace endpoint. |
| management.endpoint.in fo.cache.time-to-live | 0ms | Maximum time that a response can be cached. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.endpoint.in fo.enabled | true | Whether to enable the info endpoint. |
| management.endpoint.in tegrationgraph.cache.t ime-to-live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.in tegrationgraph.enabled | true | Whether to enable the integrationgraph endpoint. |
| management.endpoint.jo lokia.config.\* |  | Jolokia settings. Refer to the documentation of Jolokia for more details. |
| management.endpoint.jo lokia.enabled | true | Whether to enable the jolokia endpoint. |
| management.endpoint.li quibase.cache.time-to- live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.li quibase.enabled | true | Whether to enable the liquibase endpoint. |
| management.endpoint.lo gfile.cache.time-to- live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.lo gfile.enabled | true | Whether to enable the logfile endpoint. |
| management.endpoint.lo gfile.external-file |  | External Logfile to be accessed. Can be used if the logfile is written by output redirect and not by the logging system itself. |
| management.endpoint.lo ggers.cache.time-to- live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.lo ggers.enabled | true | Whether to enable the loggers endpoint. |
| management.endpoint.ma ppings.cache.time-to- live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.ma ppings.enabled | true | Whether to enable the mappings endpoint. |
| management.endpoint.me trics.cache.time-to- live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.me trics.enabled | true | Whether to enable the metrics endpoint. |
| management.endpoint.pr ometheus.cache.time- to-live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.pr ometheus.enabled | true | Whether to enable the prometheus endpoint. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.endpoint.sc heduledtasks.cache.tim e-to-live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.sc heduledtasks.enabled | true | Whether to enable the scheduledtasks endpoint. |
| management.endpoint.se ssions.enabled | true | Whether to enable the sessions endpoint. |
| management.endpoint.sh utdown.enabled | false | Whether to enable the shutdown endpoint. |
| management.endpoint.th readdump.cache.time- to-live | 0ms | Maximum time that a response can be cached. |
| management.endpoint.th readdump.enabled | true | Whether to enable the threaddump endpoint. |
| management.endpoints.e nabled-by-default |  | Whether to enable or disable all endpoints by default. |
| management.endpoints.j mx.domain | org.springframework.bo ot | Endpoints JMX domain name. Fallback to 'spring.jmx.default-domain' if set. |
| management.endpoints.j mx.exposure.exclude |  | Endpoint IDs that should be excluded or '\*' for all. |
| management.endpoints.j mx.exposure.include | \* | Endpoint IDs that should be included or '\*' for all. |
| management.endpoints.j mx.static-names |  | Additional static properties to append to all ObjectNames of MBeans representing Endpoints. |
| management.endpoints.m igrate-legacy-ids | false | Whether to transparently migrate legacy endpoint IDs. |
| management.endpoints.w eb.base-path | /actuator | Base path for Web endpoints. Relative to server.servlet.context-path or management.server.servlet.context-path if management.server.port is configured. |
| management.endpoints.w eb.cors.allow- credentials |  | Whether credentials are supported. When not set, credentials are not supported. |
| management.endpoints.w eb.cors.allowed- headers |  | Comma-separated list of headers to allow in a request. '\*' allows all headers. |
| management.endpoints.w eb.cors.allowed- methods |  | Comma-separated list of methods to allow. '\*' allows all methods. When not set, defaults to GET. |
| management.endpoints.w eb.cors.allowed- origins |  | Comma-separated list of origins to allow. '\*' allows all origins. When not set, CORS support is disabled. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.endpoints.w eb.cors.exposed- headers |  | Comma-separated list of headers to include in a response. |
| management.endpoints.w eb.cors.max-age | 1800s | How long the response from a pre-flight request can be cached by clients. If a duration suffix is not specified, seconds will be used. |
| management.endpoints.w eb.exposure.exclude |  | Endpoint IDs that should be excluded or '\*' for all. |
| management.endpoints.w eb.exposure.include | [health, info] | Endpoint IDs that should be included or '\*' for all. |
| management.endpoints.w eb.path-mapping.\* |  | Mapping between endpoint IDs and the path that should expose them. |
| management.health.cass andra.enabled | true | Whether to enable Cassandra health check. |
| management.health.couc hbase.enabled | true | Whether to enable Couchbase health check. |
| management.health.db.e nabled | true | Whether to enable database health check. |
| management.health.defa ults.enabled | true | Whether to enable default health indicators. |
| management.health.disk space.enabled | true | Whether to enable disk space health check. |
| management.health.disk space.path |  | Path used to compute the available disk space. |
| management.health.disk space.threshold | 10MB | Minimum disk space that should be available. |
| management.health.elas ticsearch.enabled | true | Whether to enable Elasticsearch health check. |
| management.health.infl uxdb.enabled | true | Whether to enable InfluxDB health check. |
| management.health.jms. enabled | true | Whether to enable JMS health check. |
| management.health.ldap  .enabled | true | Whether to enable LDAP health check. |
| management.health.live nessstate.enabled | false | Whether to enable liveness state health check. |
| management.health.mail  .enabled | true | Whether to enable Mail health check. |
| management.health.mong o.enabled | true | Whether to enable MongoDB health check. |
| management.health.neo4 j.enabled | true | Whether to enable Neo4j health check. |
| management.health.ping  .enabled | true | Whether to enable ping health check. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.health.rabb it.enabled | true | Whether to enable RabbitMQ health check. |
| management.health.read inessstate.enabled | false | Whether to enable readiness state health check. |
| management.health.redi s.enabled | true | Whether to enable Redis health check. |
| management.health.solr  .enabled | true | Whether to enable Solr health check. |
| management.info.build. enabled | true | Whether to enable build info. |
| management.info.defaul ts.enabled | true | Whether to enable default info contributors. |
| management.info.env.en abled | true | Whether to enable environment info. |
| management.info.git.en abled | true | Whether to enable git info. |
| management.info.git.mo de | simple | Mode to use to expose git information. |
| management.metrics.dis tribution.maximum- expected-value.\* |  | Maximum value that meter IDs starting with the specified name are expected to observe. The longest match wins. Values can be specified as a long or as a Duration value (for timer meters, defaulting to ms if no unit specified). |
| management.metrics.dis tribution.minimum- expected-value.\* |  | Minimum value that meter IDs starting with the specified name are expected to observe. The longest match wins. Values can be specified as a long or as a Duration value (for timer meters, defaulting to ms if no unit specified). |
| management.metrics.dis tribution.percentiles- histogram.\* |  | Whether meter IDs starting with the specified name should publish percentile histograms. For monitoring systems that support aggregable percentile calculation based on a histogram, this can be set to true. For other systems, this has no effect. The longest match wins, the key `all` can also be used to configure all meters. |
| management.metrics.dis tribution.percentiles.  \* |  | Specific computed non-aggregable percentiles to ship to the backend for meter IDs starting-with the specified name. The longest match wins, the key `all` can also be used to configure all meters. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.metrics.dis tribution.slo.\* |  | Specific service-level objective boundaries for meter IDs starting with the specified name. The longest match wins. Counters will be published for each specified boundary. Values can be specified as a long or as a Duration value (for timer meters, defaulting to ms if no unit specified). |
| management.metrics.ena ble.\* |  | Whether meter IDs starting with the specified name should be enabled. The longest match wins, the key `all` can also be used to configure all meters. |
| management.metrics.exp ort.appoptics.api- token |  | AppOptics API token. |
| management.metrics.exp ort.appoptics.batch- size | 500.0 | Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made. |
| management.metrics.exp ort.appoptics.connect- timeout | 5s | Connection timeout for requests to this backend. |
| management.metrics.exp ort.appoptics.enabled | true | Whether exporting of metrics to this backend is enabled. |
| management.metrics.exp ort.appoptics.floor- times | false | Whether to ship a floored time, useful when sending measurements from multiple hosts to align them on a given time boundary. |
| management.metrics.exp ort.appoptics.host-tag | instance | Tag that will be mapped to "@host" when shipping metrics to AppOptics. |
| management.metrics.exp ort.appoptics.read- timeout | 10s | Read timeout for requests to this backend. |
| management.metrics.exp ort.appoptics.step | 1m | Step size (i.e. reporting frequency) to use. |
| management.metrics.exp ort.appoptics.uri | https://api.appoptics. com/v1/measurements | URI to ship metrics to. |
| management.metrics.exp ort.atlas.batch-size | 10000.0 | Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made. |
| management.metrics.exp ort.atlas.config- refresh-frequency | 10s | Frequency for refreshing config settings from the LWC service. |
| management.metrics.exp ort.atlas.config-time- to-live | 150s | Time to live for subscriptions from the LWC service. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.metrics.exp ort.atlas.config-uri | http://localhost:7101/ lwc/api/v1/expressions  /local-dev | URI for the Atlas LWC endpoint to retrieve current subscriptions. |
| management.metrics.exp ort.atlas.connect- timeout | 1s | Connection timeout for requests to this backend. |
| management.metrics.exp ort.atlas.enabled | true | Whether exporting of metrics to this backend is enabled. |
| management.metrics.exp ort.atlas.eval-uri | http://localhost:7101/ lwc/api/v1/evaluate | URI for the Atlas LWC endpoint to evaluate the data for a subscription. |
| management.metrics.exp ort.atlas.lwc-enabled | false | Whether to enable streaming to Atlas LWC. |
| management.metrics.exp ort.atlas.meter-time- to-live | 15m | Time to live for meters that do not have any activity. After this period the meter will be considered expired and will not get reported. |
| management.metrics.exp ort.atlas.num-threads | 4.0 | Number of threads to use with the metrics publishing scheduler. |
| management.metrics.exp ort.atlas.read-timeout | 10s | Read timeout for requests to this backend. |
| management.metrics.exp ort.atlas.step | 1m | Step size (i.e. reporting frequency) to use. |
| management.metrics.exp ort.atlas.uri | http://localhost:7101/ api/v1/publish | URI of the Atlas server. |
| management.metrics.exp ort.datadog.api-key |  | Datadog API key. |
| management.metrics.exp ort.datadog.applicatio n-key |  | Datadog application key. Not strictly required, but improves the Datadog experience by sending meter descriptions, types, and base units to Datadog. |
| management.metrics.exp ort.datadog.batch-size | 10000.0 | Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made. |
| management.metrics.exp ort.datadog.connect- timeout | 1s | Connection timeout for requests to this backend. |
| management.metrics.exp ort.datadog.descriptio ns | true | Whether to publish descriptions metadata to Datadog. Turn this off to minimize the amount of metadata sent. |
| management.metrics.exp ort.datadog.enabled | true | Whether exporting of metrics to this backend is enabled. |
| management.metrics.exp ort.datadog.host-tag | instance | Tag that will be mapped to "host" when shipping metrics to Datadog. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.metrics.exp ort.datadog.read- timeout | 10s | Read timeout for requests to this backend. |
| management.metrics.exp ort.datadog.step | 1m | Step size (i.e. reporting frequency) to use. |
| management.metrics.exp ort.datadog.uri | https://api.datadoghq. com | URI to ship metrics to. If you need to publish metrics to an internal proxy en-route to Datadog, you can define the location of the proxy with this. |
| management.metrics.exp ort.dynatrace.api- token |  | Dynatrace authentication token. |
| management.metrics.exp ort.dynatrace.batch- size | 10000.0 | Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made. |
| management.metrics.exp ort.dynatrace.connect- timeout | 1s | Connection timeout for requests to this backend. |
| management.metrics.exp ort.dynatrace.device- id |  | ID of the custom device that is exporting metrics to Dynatrace. |
| management.metrics.exp ort.dynatrace.enabled | true | Whether exporting of metrics to this backend is enabled. |
| management.metrics.exp ort.dynatrace.group |  | Group for exported metrics. Used to specify custom device group name in the Dynatrace UI. |
| management.metrics.exp ort.dynatrace.read- timeout | 10s | Read timeout for requests to this backend. |
| management.metrics.exp ort.dynatrace.step | 1m | Step size (i.e. reporting frequency) to use. |
| management.metrics.exp ort.dynatrace.technolo gy-type | java | Technology type for exported metrics. Used to group metrics under a logical technology name in the Dynatrace UI. |
| management.metrics.exp ort.dynatrace.uri |  | URI to ship metrics to. Should be used for SaaS, self managed instances or to en-route through an internal proxy. |
| management.metrics.exp ort.elastic.auto- create-index | true | Whether to create the index automatically if it does not exist. |
| management.metrics.exp ort.elastic.batch-size | 10000.0 | Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made. |
| management.metrics.exp ort.elastic.connect- timeout | 1s | Connection timeout for requests to this backend. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.metrics.exp ort.elastic.enabled | true | Whether exporting of metrics to this backend is enabled. |
| management.metrics.exp ort.elastic.host | http://localhost:9200 | Host to export metrics to. |
| management.metrics.exp ort.elastic.index | metrics | Index to export metrics to. |
| management.metrics.exp ort.elastic.index- date-format | yyyy-MM | Index date format used for rolling indices. Appended to the index name. |
| management.metrics.exp ort.elastic.index- date-separator | - | Prefix to separate the index name from the date format used for rolling indices. |
| management.metrics.exp ort.elastic.password |  | Login password of the Elastic server. |
| management.metrics.exp ort.elastic.pipeline |  | Ingest pipeline name. By default, events are not pre-processed. |
| management.metrics.exp ort.elastic.read- timeout | 10s | Read timeout for requests to this backend. |
| management.metrics.exp ort.elastic.step | 1m | Step size (i.e. reporting frequency) to use. |
| management.metrics.exp ort.elastic.timestamp- field-name | @timestamp | Name of the timestamp field. |
| management.metrics.exp ort.elastic.user-name |  | Login user of the Elastic server. |
| management.metrics.exp ort.ganglia.addressing  -mode | multicast | UDP addressing mode, either unicast or multicast. |
| management.metrics.exp ort.ganglia.duration- units | milliseconds | Base time unit used to report durations. |
| management.metrics.exp ort.ganglia.enabled | true | Whether exporting of metrics to Ganglia is enabled. |
| management.metrics.exp ort.ganglia.host | localhost | Host of the Ganglia server to receive exported metrics. |
| management.metrics.exp ort.ganglia.port | 8649.0 | Port of the Ganglia server to receive exported metrics. |
| management.metrics.exp ort.ganglia.step | 1m | Step size (i.e. reporting frequency) to use. |
| management.metrics.exp ort.ganglia.time-to- live | 1.0 | Time to live for metrics on Ganglia. Set the multi-cast Time-To-Live to be one greater than the number of hops (routers) between the hosts. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.metrics.exp ort.graphite.duration- units | milliseconds | Base time unit used to report durations. |
| management.metrics.exp ort.graphite.enabled | true | Whether exporting of metrics to Graphite is enabled. |
| management.metrics.exp ort.graphite.graphite- tags-enabled |  | Whether Graphite tags should be used, as opposed to a hierarchical naming convention. Enabled by default unless "tagsAsPrefix" is set. |
| management.metrics.exp ort.graphite.host | localhost | Host of the Graphite server to receive exported metrics. |
| management.metrics.exp ort.graphite.port | 2004.0 | Port of the Graphite server to receive exported metrics. |
| management.metrics.exp ort.graphite.protocol | pickled | Protocol to use while shipping data to Graphite. |
| management.metrics.exp ort.graphite.rate- units | seconds | Base time unit used to report rates. |
| management.metrics.exp ort.graphite.step | 1m | Step size (i.e. reporting frequency) to use. |
| management.metrics.exp ort.graphite.tags-as- prefix | [] | For the hierarchical naming convention, turn the specified tag keys into part of the metric prefix. Ignored if "graphiteTagsEnabled" is true. |
| management.metrics.exp ort.humio.api-token |  | Humio API token. |
| management.metrics.exp ort.humio.batch-size | 10000.0 | Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made. |
| management.metrics.exp ort.humio.connect- timeout | 5s | Connection timeout for requests to this backend. |
| management.metrics.exp ort.humio.enabled | true | Whether exporting of metrics to this backend is enabled. |
| management.metrics.exp ort.humio.read-timeout | 10s | Read timeout for requests to this backend. |
| management.metrics.exp ort.humio.step | 1m | Step size (i.e. reporting frequency) to use. |
| management.metrics.exp ort.humio.tags.\* |  | Humio tags describing the data source in which metrics will be stored. Humio tags are a distinct concept from Micrometer's tags. Micrometer's tags are used to divide metrics along dimensional boundaries. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.metrics.exp ort.humio.uri | https://cloud.humio.co m | URI to ship metrics to. If you need to publish metrics to an internal proxy en-route to Humio, you can define the location of the proxy with this. |
| management.metrics.exp ort.influx.auto- create-db | true | Whether to create the Influx database if it does not exist before attempting to publish metrics to it. |
| management.metrics.exp ort.influx.batch-size | 10000.0 | Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made. |
| management.metrics.exp ort.influx.compressed | true | Whether to enable GZIP compression of metrics batches published to Influx. |
| management.metrics.exp ort.influx.connect- timeout | 1s | Connection timeout for requests to this backend. |
| management.metrics.exp ort.influx.consistency | one | Write consistency for each point. |
| management.metrics.exp ort.influx.db | mydb | Tag that will be mapped to "host" when shipping metrics to Influx. |
| management.metrics.exp ort.influx.enabled | true | Whether exporting of metrics to this backend is enabled. |
| management.metrics.exp ort.influx.password |  | Login password of the Influx server. |
| management.metrics.exp ort.influx.read- timeout | 10s | Read timeout for requests to this backend. |
| management.metrics.exp ort.influx.retention- duration |  | Time period for which Influx should retain data in the current database. For instance 7d, check the influx documentation for more details on the duration format. |
| management.metrics.exp ort.influx.retention- policy |  | Retention policy to use (Influx writes to the DEFAULT retention policy if one is not specified). |
| management.metrics.exp ort.influx.retention- replication-factor |  | How many copies of the data are stored in the cluster. Must be 1 for a single node instance. |
| management.metrics.exp ort.influx.retention- shard-duration |  | Time range covered by a shard group. For instance 2w, check the influx documentation for more details on the duration format. |
| management.metrics.exp ort.influx.step | 1m | Step size (i.e. reporting frequency) to use. |
| management.metrics.exp ort.influx.uri | http://localhost:8086 | URI of the Influx server. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.metrics.exp ort.influx.user-name |  | Login user of the Influx server. |
| management.metrics.exp ort.jmx.domain | metrics | Metrics JMX domain name. |
| management.metrics.exp ort.jmx.enabled | true | Whether exporting of metrics to JMX is enabled. |
| management.metrics.exp ort.jmx.step | 1m | Step size (i.e. reporting frequency) to use. |
| management.metrics.exp ort.kairos.batch-size | 10000.0 | Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made. |
| management.metrics.exp ort.kairos.connect- timeout | 1s | Connection timeout for requests to this backend. |
| management.metrics.exp ort.kairos.enabled | true | Whether exporting of metrics to this backend is enabled. |
| management.metrics.exp ort.kairos.password |  | Login password of the KairosDB server. |
| management.metrics.exp ort.kairos.read- timeout | 10s | Read timeout for requests to this backend. |
| management.metrics.exp ort.kairos.step | 1m | Step size (i.e. reporting frequency) to use. |
| management.metrics.exp ort.kairos.uri | http://localhost:8080/ api/v1/datapoints | URI of the KairosDB server. |
| management.metrics.exp ort.kairos.user-name |  | Login user of the KairosDB server. |
| management.metrics.exp ort.newrelic.account- id |  | New Relic account ID. |
| management.metrics.exp ort.newrelic.api-key |  | New Relic API key. |
| management.metrics.exp ort.newrelic.batch- size | 10000.0 | Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made. |
| management.metrics.exp ort.newrelic.client- provider-type |  | Client provider type to use. |
| management.metrics.exp ort.newrelic.connect- timeout | 1s | Connection timeout for requests to this backend. |
| management.metrics.exp ort.newrelic.enabled | true | Whether exporting of metrics to this backend is enabled. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.metrics.exp ort.newrelic.event- type | SpringBootSample | The event type that should be published. This property will be ignored if 'meter-name-event- type-enabled' is set to 'true'. |
| management.metrics.exp ort.newrelic.meter- name-event-type- enabled | false | Whether to send the meter name as the event type instead of using the 'event-type' configuration property value. Can be set to 'true' if New Relic guidelines are not being followed or event types consistent with previous Spring Boot releases are required. |
| management.metrics.exp ort.newrelic.read- timeout | 10s | Read timeout for requests to this backend. |
| management.metrics.exp ort.newrelic.step | 1m | Step size (i.e. reporting frequency) to use. |
| management.metrics.exp ort.newrelic.uri | https://insights- collector.newrelic.com | URI to ship metrics to. |
| management.metrics.exp ort.prometheus.descrip tions | true | Whether to enable publishing descriptions as part of the scrape payload to Prometheus. Turn this off to minimize the amount of data sent on each scrape. |
| management.metrics.exp ort.prometheus.enabled | true | Whether exporting of metrics to Prometheus is enabled. |
| management.metrics.exp ort.prometheus.histogr am-flavor | prometheus | Histogram type for backing DistributionSummary and Timer. |
| management.metrics.exp ort.prometheus.pushgat eway.base-url | http://localhost:9091 | Base URL for the Pushgateway. |
| management.metrics.exp ort.prometheus.pushgat eway.enabled | false | Enable publishing via a Prometheus Pushgateway. |
| management.metrics.exp ort.prometheus.pushgat eway.grouping-key.\* |  | Grouping key for the pushed metrics. |
| management.metrics.exp ort.prometheus.pushgat eway.job |  | Job identifier for this application instance. |
| management.metrics.exp ort.prometheus.pushgat eway.push-rate | 1m | Frequency with which to push metrics. |
| management.metrics.exp ort.prometheus.pushgat eway.shutdown- operation | none | Operation that should be performed on shutdown. |
| management.metrics.exp ort.prometheus.step | 1m | Step size (i.e. reporting frequency) to use. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.metrics.exp ort.signalfx.access- token |  | SignalFX access token. |
| management.metrics.exp ort.signalfx.batch- size | 10000.0 | Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made. |
| management.metrics.exp ort.signalfx.connect- timeout | 1s | Connection timeout for requests to this backend. |
| management.metrics.exp ort.signalfx.enabled | true | Whether exporting of metrics to this backend is enabled. |
| management.metrics.exp ort.signalfx.read- timeout | 10s | Read timeout for requests to this backend. |
| management.metrics.exp ort.signalfx.source |  | Uniquely identifies the app instance that is publishing metrics to SignalFx. Defaults to the local host name. |
| management.metrics.exp ort.signalfx.step | 10s | Step size (i.e. reporting frequency) to use. |
| management.metrics.exp ort.signalfx.uri | https://ingest.signalf x.com | URI to ship metrics to. |
| management.metrics.exp ort.simple.enabled | true | Whether, in the absence of any other exporter, exporting of metrics to an in-memory backend is enabled. |
| management.metrics.exp ort.simple.mode | cumulative | Counting mode. |
| management.metrics.exp ort.simple.step | 1m | Step size (i.e. reporting frequency) to use. |
| management.metrics.exp ort.stackdriver.batch- size | 10000.0 | Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made. |
| management.metrics.exp ort.stackdriver.connec t-timeout | 1s | Connection timeout for requests to this backend. |
| management.metrics.exp ort.stackdriver.enable d | true | Whether exporting of metrics to this backend is enabled. |
| management.metrics.exp ort.stackdriver.projec t-id |  | Identifier of the Google Cloud project to monitor. |
| management.metrics.exp ort.stackdriver.read- timeout | 10s | Read timeout for requests to this backend. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.metrics.exp ort.stackdriver.resour ce-type | global | Monitored resource type. |
| management.metrics.exp ort.stackdriver.step | 1m | Step size (i.e. reporting frequency) to use. |
| management.metrics.exp ort.statsd.enabled | true | Whether exporting of metrics to StatsD is enabled. |
| management.metrics.exp ort.statsd.flavor | datadog | StatsD line protocol to use. |
| management.metrics.exp ort.statsd.host | localhost | Host of the StatsD server to receive exported metrics. |
| management.metrics.exp ort.statsd.max-packet- length | 1400.0 | Total length of a single payload should be kept within your network's MTU. |
| management.metrics.exp ort.statsd.polling- frequency | 10s | How often gauges will be polled. When a gauge is polled, its value is recalculated and if the value has changed (or publishUnchangedMeters is true), it is sent to the StatsD server. |
| management.metrics.exp ort.statsd.port | 8125.0 | Port of the StatsD server to receive exported metrics. |
| management.metrics.exp ort.statsd.publish- unchanged-meters | true | Whether to send unchanged meters to the StatsD server. |
| management.metrics.exp ort.wavefront.api- token |  | API token used when publishing metrics directly to the Wavefront API host. |
| management.metrics.exp ort.wavefront.batch- size | 10000.0 | Number of measurements per request to use for this backend. If more measurements are found, then multiple requests will be made. |
| management.metrics.exp ort.wavefront.enabled | true | Whether exporting of metrics to this backend is enabled. |
| management.metrics.exp ort.wavefront.global- prefix |  | Global prefix to separate metrics originating from this app's white box instrumentation from those originating from other Wavefront integrations when viewed in the Wavefront UI. |
| management.metrics.exp ort.wavefront.sender.f lush-interval | 1s |  |
| management.metrics.exp ort.wavefront.sender.m ax-queue-size | 50000.0 |  |
| management.metrics.exp ort.wavefront.sender.m essage-size |  |  |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.metrics.exp ort.wavefront.source |  | Unique identifier for the app instance that is the source of metrics being published to Wavefront. Defaults to the local host name. |
| management.metrics.exp ort.wavefront.step | 1m | Step size (i.e. reporting frequency) to use. |
| management.metrics.exp ort.wavefront.uri | https://longboard.wave front.com | URI to ship metrics to. |
| management.metrics.tag s.\* |  | Common tags that are applied to every meter. |
| management.metrics.use  -global-registry | true | Whether auto-configured MeterRegistry implementations should be bound to the global static registry on Metrics. For testing, set this to 'false' to maximize test independence. |
| management.metrics.web  .client.max-uri-tags | 100.0 | Maximum number of unique URI tag values allowed. After the max number of tag values is reached, metrics with additional tag values are denied by filter. |
| management.metrics.web  .client.request.autoti me.enabled | true | Whether to automatically time web client requests. |
| management.metrics.web  .client.request.autoti me.percentiles |  | Computed non-aggregable percentiles to publish. |
| management.metrics.web  .client.request.autoti me.percentiles- histogram | false | Whether percentile histograms should be published. |
| management.metrics.web  .client.request.metric  -name | http.client.requests | Name of the metric for sent requests. |
| management.metrics.web  .server.max-uri-tags | 100.0 | Maximum number of unique URI tag values allowed. After the max number of tag values is reached, metrics with additional tag values are denied by filter. |
| management.metrics.web  .server.request.autoti me.enabled | true | Whether to automatically time web server requests. |
| management.metrics.web  .server.request.autoti me.percentiles |  | Computed non-aggregable percentiles to publish. |
| management.metrics.web  .server.request.autoti me.percentiles- histogram | false | Whether percentile histograms should be published. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.metrics.web  .server.request.ignore  -trailing-slash | true | Whether the trailing slash should be ignored when recording metrics. |
| management.metrics.web  .server.request.metric  -name | http.server.requests | Name of the metric for received requests. |
| management.server.add- application-context- header | false | Add the "X-Application-Context" HTTP header in each response. |
| management.server.addr ess |  | Network address to which the management endpoints should bind. Requires a custom management.server.port. |
| management.server.port |  | Management endpoint HTTP port (uses the same port as the application by default). Configure a different port to use management-specific SSL. |
| management.server.serv let.context-path |  | Management endpoint context-path (for instance, `/management`). Requires a custom management.server.port. |
| management.server.ssl. ciphers |  | Supported SSL ciphers. |
| management.server.ssl. client-auth |  | Client authentication mode. Requires a trust store. |
| management.server.ssl. enabled | true | Whether to enable SSL support. |
| management.server.ssl. enabled-protocols |  | Enabled SSL protocols. |
| management.server.ssl. key-alias |  | Alias that identifies the key in the key store. |
| management.server.ssl. key-password |  | Password used to access the key in the key store. |
| management.server.ssl. key-store |  | Path to the key store that holds the SSL certificate (typically a jks file). |
| management.server.ssl. key-store-password |  | Password used to access the key store. |
| management.server.ssl. key-store-provider |  | Provider for the key store. |
| management.server.ssl. key-store-type |  | Type of the key store. |
| management.server.ssl. protocol | TLS | SSL protocol to use. |
| management.server.ssl. trust-store |  | Trust store that holds SSL certificates. |
| management.server.ssl. trust-store-password |  | Password used to access the trust store. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| management.server.ssl. trust-store-provider |  | Provider for the trust store. |
| management.server.ssl. trust-store-type |  | Type of the trust store. |
| management.trace.http. enabled | true | Whether to enable HTTP request-response tracing. |
| management.trace.http. include | [request-headers, response-headers, cookies, errors] | Items to be included in the trace. Defaults to request headers (excluding Authorization but including Cookie), response headers (including Set-Cookie), and time taken. |

### Devtools properties

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.devtools.add- properties | true | Whether to enable development property defaults. |
| spring.devtools.livere load.enabled | true | Whether to enable a livereload.com-compatible server. |
| spring.devtools.livere load.port | 35729.0 | Server port. |
| spring.devtools.remote  .context-path | /.~~spring-boot!~ | Context path used to handle the remote connection. |
| spring.devtools.remote  .proxy.host |  | The host of the proxy to use to connect to the remote application. |
| spring.devtools.remote  .proxy.port |  | The port of the proxy to use to connect to the remote application. |
| spring.devtools.remote  .restart.enabled | true | Whether to enable remote restart. |
| spring.devtools.remote  .secret |  | A shared secret required to establish a connection (required to enable remote support). |
| spring.devtools.remote  .secret-header-name | X-AUTH-TOKEN | HTTP header used to transfer the shared secret. |
| spring.devtools.restar t.additional-exclude |  | Additional patterns that should be excluded from triggering a full restart. |
| spring.devtools.restar t.additional-paths |  | Additional paths to watch for changes. |
| spring.devtools.restar t.enabled | true | Whether to enable automatic restart. |

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |
| spring.devtools.restar t.exclude | META-  INF/maven/\*\*,META- INF/resources/\*\*,resou rces/\*\*,static/\*\*,publ ic/\*\*,templates/\*\*,\*\*/  \*Test.class,\*\*/\*Tests. class,git.properties,M ETA-INF/build- info.properties | Patterns that should be excluded from triggering a full restart. |
| spring.devtools.restar t.log-condition- evaluation-delta | true | Whether to log the condition evaluation delta upon restart. |
| spring.devtools.restar t.poll-interval | 1s | Amount of time to wait between polling for classpath changes. |
| spring.devtools.restar t.quiet-period | 400ms | Amount of quiet time required without any classpath changes before a restart is triggered. |
| spring.devtools.restar t.trigger-file |  | Name of a specific file that, when changed, triggers the restart check. Must be a simple name (without any path) of a file that appears on your classpath. If not specified, any classpath file change triggers the restart. |

### Testing properties

|  |  |  |
| --- | --- | --- |
| **Key** | **Default Value** | **Description** |

## Appendix B: Configuration Metadata

Spring Boot jars include metadata files that provide details of all supported configuration properties. The files are designed to let IDE developers offer contextual help and “code completion” as users are working with application.properties or application.yml files.

The majority of the metadata file is generated automatically at compile time by processing all items annotated with @ConfigurationProperties. However, it is possible to [write part of the metadata](#_bookmark685) [manually](#_bookmark685) for corner cases or more advanced use cases.

### Metadata Format

Configuration metadata files are located inside jars under META-INF/spring-configuration- metadata.json. They use a JSON format with items categorized under either “groups” or “properties” and additional values hints categorized under "hints", as shown in the following example:

{"groups": [

{

"name": "server",

"type": "org.springframework.boot.autoconfigure.web.ServerProperties",

"sourceType": "org.springframework.boot.autoconfigure.web.ServerProperties"

},

{

"name": "spring.jpa.hibernate", "type":

"org.springframework.boot.autoconfigure.orm.jpa.JpaProperties$Hibernate", "sourceType": "org.springframework.boot.autoconfigure.orm.jpa.JpaProperties", "sourceMethod": "getHibernate()"

}

...

],"properties": [

{

"name": "server.port", "type": "java.lang.Integer",

"sourceType": "org.springframework.boot.autoconfigure.web.ServerProperties"

},

{

"name": "server.address", "type": "java.net.InetAddress",

"sourceType": "org.springframework.boot.autoconfigure.web.ServerProperties"

},

{

"name": "spring.jpa.hibernate.ddl-auto", "type": "java.lang.String",

"description": "DDL mode. This is actually a shortcut for the

\"hibernate.hbm2ddl.auto\" property.", "sourceType":

"org.springframework.boot.autoconfigure.orm.jpa.JpaProperties$Hibernate"

}

...

],"hints": [

{

"name": "spring.jpa.hibernate.ddl-auto", "values": [

{

"value": "none",

"description": "Disable DDL handling."

},

{

"value": "validate",

"description": "Validate the schema, make no changes to the database."

},

{

"value": "update",

"description": "Update the schema if necessary."

},

{

"value": "create",

"description": "Create the schema and destroy previous data."

},

{

"value": "create-drop",

"description": "Create and then destroy the schema at the end of the

session."

}

]

}

]}

Each “property” is a configuration item that the user specifies with a given value. For example,

server.port and server.address might be specified in application.properties, as follows:

server.port=9090 server.address=127.0.0.1

The “groups” are higher level items that do not themselves specify a value but instead provide a contextual grouping for properties. For example, the server.port and server.address properties are part of the server group.

It is not required that every “property” has a “group”. Some properties might exist in their own right.

Finally, “hints” are additional information used to assist the user in configuring a given property. For example, when a developer is configuring the spring.jpa.hibernate.ddl-auto property, a tool can use the hints to offer some auto-completion help for the none, validate, update, create, and create-drop values.

#### Group Attributes

The JSON object contained in the groups array can contain the attributes shown in the following table:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Purpose** |
| name | String | The full name of the group. This attribute is mandatory. |
| type | String | The class name of the data type of the group. For example, if the group were based on a class annotated with @ConfigurationProperties, the attribute would contain the fully qualified name of that class. If it were based on a @Bean method, it would be the return type of that method. If the type is not known, the attribute may be omitted. |
| description | String | A short description of the group that can be displayed to users. If no description is available, it may be omitted. It is recommended that descriptions be short paragraphs, with the first line providing a concise summary. The last line in the description should end with a period (.). |

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Purpose** |
| sourceType | String | The class name of the source that contributed this group. For example, if the group were based on a @Bean method annotated with @ConfigurationProperties, this attribute would contain the fully qualified name of the @Configuration class that contains the method. If the source type is not known, the attribute may be omitted. |
| sourceMethod | String | The full name of the method (include parenthesis and argument types) that contributed this group (for example, the name of a @ConfigurationProperties annotated @Bean method). If the source method is not known, it may be omitted. |

#### Property Attributes

The JSON object contained in the properties array can contain the attributes described in the following table:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Purpose** |
| name | String | The full name of the property. Names are in lower-case period- separated form (for example, server.address). This attribute is mandatory. |
| type | String | The full signature of the data type of the property (for example, java.lang.String) but also a full generic type (such as java.util.Map<java.lang.String,acme.MyEnum>). You can use this attribute to guide the user as to the types of values that they can enter. For consistency, the type of a primitive is specified by using its wrapper counterpart (for example, boolean becomes java.lang.Boolean). Note that this class may be a complex type that gets converted from a String as values are bound. If the type is not known, it may be omitted. |
| description | String | A short description of the property that can be displayed to users. If no description is available, it may be omitted. It is recommended that descriptions be short paragraphs, with the first line providing a concise summary. The last line in the description should end with a period (.). |
| sourceType | String | The class name of the source that contributed this property. For example, if the property were from a class annotated with @ConfigurationProperties, this attribute would contain the fully qualified name of that class. If the source type is unknown, it may be omitted. |
| defaultValue | Object | The default value, which is used if the property is not specified. If the type of the property is an array, it can be an array of value(s). If the default value is unknown, it may be omitted. |

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Purpose** |
| deprecation | Deprecation | Specify whether the property is deprecated. If the field is not deprecated or if that information is not known, it may be omitted. The next table offers more detail about the deprecation attribute. |

The JSON object contained in the deprecation attribute of each properties element can contain the following attributes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Purpose** |
| level | String | The level of deprecation, which can be either warning (the default) or error. When a property has a warning deprecation level, it should still be bound in the environment. However, when it has an error deprecation level, the property is no longer managed and is not bound. |
| reason | String | A short description of the reason why the property was deprecated. If no reason is available, it may be omitted. It is recommended that descriptions be short paragraphs, with the first line providing a concise summary. The last line in the description should end with a period (.). |
| replacement | String | The full name of the property that *replaces* this deprecated property. If there is no replacement for this property, it may be omitted. |

Prior to Spring Boot 1.3, a single deprecated boolean attribute can be used instead of the deprecation element. This is still supported in a deprecated fashion and should no longer be used. If no reason and replacement are available, an empty deprecation object should be set.

Deprecation can also be specified declaratively in code by adding the @DeprecatedConfigurationProperty annotation to the getter exposing the deprecated property. For instance, assume that the app.acme.target property was confusing and was renamed to app.acme.name. The following example shows how to handle that situation:

@ConfigurationProperties("app.acme") public class AcmeProperties {

private String name;

public String getName() { ... }

public void setName(String name) { ... }

@DeprecatedConfigurationProperty(replacement = "app.acme.name") @Deprecated

public String getTarget() { return getName();

}

@Deprecated

public void setTarget(String target) { setName(target);

}

}

There is no way to set a level. warning is always assumed, since code is still handling the property.

The preceding code makes sure that the deprecated property still works (delegating to the name property behind the scenes). Once the getTarget and setTarget methods can be removed from your public API, the automatic deprecation hint in the metadata goes away as well. If you want to keep a hint, adding manual metadata with an error deprecation level ensures that users are still informed about that property. Doing so is particularly useful when a replacement is provided.

#### Hint Attributes

The JSON object contained in the hints array can contain the attributes shown in the following table:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Purpose** |
| name | String | The full name of the property to which this hint refers. Names are in lower-case period-separated form (such as spring.mvc.servlet.path). If the property refers to a map (such as system.contexts), the hint either applies to the *keys* of the map (system.contexts.keys) or the *values* (system.contexts.values) of the map. This attribute is mandatory. |
| values | ValueHint[] | A list of valid values as defined by the ValueHint object (described in the next table). Each entry defines the value and may have a description. |

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Purpose** |
| providers | ValueProvider[  ] | A list of providers as defined by the ValueProvider object (described later in this document). Each entry defines the name of the provider and its parameters, if any. |

The JSON object contained in the values attribute of each hint element can contain the attributes described in the following table:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Purpose** |
| value | Object | A valid value for the element to which the hint refers. If the type of the property is an array, it can also be an array of value(s).  This attribute is mandatory. |
| description | String | A short description of the value that can be displayed to users. If no description is available, it may be omitted. It is recommended that descriptions be short paragraphs, with the first line providing a concise summary. The last line in the description should end with a period (.). |

The JSON object contained in the providers attribute of each hint element can contain the attributes described in the following table:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Purpose** |
| name | String | The name of the provider to use to offer additional content assistance for the element to which the hint refers. |
| parameters | JSON object | Any additional parameter that the provider supports (check the documentation of the provider for more details). |

#### Repeated Metadata Items

Objects with the same “property” and “group” name can appear multiple times within a metadata file. For example, you could bind two separate classes to the same prefix, with each having potentially overlapping property names. While the same names appearing in the metadata multiple times should not be common, consumers of metadata should take care to ensure that they support it.

### Providing Manual Hints

To improve the user experience and further assist the user in configuring a given property, you can provide additional metadata that:

* + - * Describes the list of potential values for a property.
      * Associates a provider, to attach a well defined semantic to a property, so that a tool can discover the list of potential values based on the project’s context.

#### Value Hint

The name attribute of each hint refers to the name of a property. In the [initial example shown earlier](#_bookmark674), we provide five values for the spring.jpa.hibernate.ddl-auto property: none, validate, update, create, and create-drop. Each value may have a description as well.

If your property is of type Map, you can provide hints for both the keys and the values (but not for the map itself). The special .keys and .values suffixes must refer to the keys and the values, respectively.

Assume a sample.contexts maps magic String values to an integer, as shown in the following example:

@ConfigurationProperties("sample") public class SampleProperties {

private Map<String,Integer> contexts;

// getters and setters

}

The magic values are (in this example) are sample1 and sample2. In order to offer additional content assistance for the keys, you could add the following JSON to [the manual metadata of the module](#_bookmark685):

{"hints": [

{

"name": "sample.contexts.keys", "values": [

{

"value": "sample1"

},

{

"value": "sample2"

}

]

}

]}

We recommend that you use an Enum for those two values instead. If your IDE supports it, this is by far the most effective approach to auto-completion.

#### Value Providers

Providers are a powerful way to attach semantics to a property. In this section, we define the official providers that you can use for your own hints. However, your favorite IDE may implement some of these or none of them. Also, it could eventually provide its own.

As this is a new feature, IDE vendors must catch up with how it works. Adoption times naturally vary.

The following table summarizes the list of supported providers:

|  |  |
| --- | --- |
| **Name** | **Description** |
| any | Permits any additional value to be provided. |
| class-reference | Auto-completes the classes available in the project. Usually constrained by a base class that is specified by the target parameter. |
| handle-as | Handles the property as if it were defined by the type defined by the mandatory target parameter. |
| logger-name | Auto-completes valid logger names and [logger groups](#_bookmark136). Typically, package and class names available in the current project can be auto-completed as well as defined groups. |
| spring-bean-reference | Auto-completes the available bean names in the current project. Usually constrained by a base class that is specified by the target parameter. |
| spring-profile-name | Auto-completes the available Spring profile names in the project. |

Only one provider can be active for a given property, but you can specify several providers if they can all manage the property *in some way*. Make sure to place the

 most powerful provider first, as the IDE must use the first one in the JSON section

that it can handle. If no provider for a given property is supported, no special

content assistance is provided, either.

**Any**

The special **any** provider value permits any additional values to be provided. Regular value validation based on the property type should be applied if this is supported.

This provider is typically used if you have a list of values and any extra values should still be considered as valid.

The following example offers on and off as auto-completion values for system.state:

{"hints": [

{

"name": "system.state", "values": [

{

"value": "on"

},

{

"value": "off"

}

],

"providers": [

{

"name": "any"

}

]

}

]}

Note that, in the preceding example, any other value is also allowed.

**Class Reference**

The **class-reference** provider auto-completes classes available in the project. This provider supports the following parameters:

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Type** | **Default value** | **Description** |
| target | String  (Class) | *none* | The fully qualified name of the class that should be assignable to the chosen value. Typically used to filter out-non candidate classes. Note that this information can be provided by the type itself by exposing a class with the appropriate upper bound. |
| concrete | boolean | true | Specify whether only concrete classes are to be considered as valid candidates. |

The following metadata snippet corresponds to the standard server.servlet.jsp.class-name

property that defines the JspServlet class name to use:

{"hints": [

{

"name": "server.servlet.jsp.class-name", "providers": [

{

"name": "class-reference", "parameters": {

"target": "javax.servlet.http.HttpServlet"

}

}

]

}

]}

**Handle As**

The **handle-as** provider lets you substitute the type of the property to a more high-level type. This typically happens when the property has a java.lang.String type, because you do not want your configuration classes to rely on classes that may not be on the classpath. This provider supports the following parameters:

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Type** | **Default value** | **Description** |
| target | String  (Class) | *none* | The fully qualified name of the type to consider for the property. This parameter is mandatory. |

The following types can be used:

* + - * Any java.lang.Enum: Lists the possible values for the property. (We recommend defining the property with the Enum type, as no further hint should be required for the IDE to auto-complete the values)
      * java.nio.charset.Charset: Supports auto-completion of charset/encoding values (such as UTF-8)
      * java.util.Locale: auto-completion of locales (such as en\_US)
      * org.springframework.util.MimeType: Supports auto-completion of content type values (such as

text/plain)

* + - * org.springframework.core.io.Resource: Supports auto-completion of Spring’s Resource abstraction to refer to a file on the filesystem or on the classpath (such as classpath:/sample.properties)

If multiple values can be provided, use a Collection or *Array* type to teach the IDE about it.

The following metadata snippet corresponds to the standard spring.liquibase.change-log property that defines the path to the changelog to use. It is actually used internally as a org.springframework.core.io.Resource but cannot be exposed as such, because we need to keep the original String value to pass it to the Liquibase API.

{"hints": [

{

"name": "spring.liquibase.change-log", "providers": [

{

"name": "handle-as", "parameters": {

"target": "org.springframework.core.io.Resource"

}

}

]

}

]}

**Logger Name**

The **logger-name** provider auto-completes valid logger names and [logger groups](#_bookmark136). Typically, package and class names available in the current project can be auto-completed. If groups are enabled (default) and if a custom logger group is identified in the configuration, auto-completion for it should be provided. Specific frameworks may have extra magic logger names that can be supported as well.

This provider supports the following parameters:

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Type** | **Default value** | **Description** |
| group | boolean | true | Specify whether known groups should be considered. |

Since a logger name can be any arbitrary name, this provider should allow any value but could highlight valid package and class names that are not available in the project’s classpath.

The following metadata snippet corresponds to the standard logging.level property. Keys are *logger names*, and values correspond to the standard log levels or any custom level. As Spring Boot defines a few logger groups out-of-the-box, dedicated value hints have been added for those.

{"hints": [

{

"name": "logging.level.keys", "values": [

{

"value": "root",

"description": "Root logger used to assign the default logging level."

},

{

"value": "sql",

"description": "SQL logging group including Hibernate SQL logger."

},

{

"value": "web",

"description": "Web logging group including codecs."

}

],

"providers": [

{

"name": "logger-name"

}

]

},

{

"name": "logging.level.values", "values": [

{

"value": "trace"

},

{

"value": "debug"

},

{

"value": "info"

},

{

"value": "warn"

},

{

"value": "error"

},

{

"value": "fatal"

},

{

"value": "off"

}

],

"providers": [

{

"name": "any"

}

]

}

]}

**Spring Bean Reference**

The **spring-bean-reference** provider auto-completes the beans that are defined in the configuration of the current project. This provider supports the following parameters:

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Type** | **Default value** | **Description** |
| target | String  (Class) | *none* | The fully qualified name of the bean class that should be assignable to the candidate. Typically used to filter out non-candidate beans. |

The following metadata snippet corresponds to the standard spring.jmx.server property that defines the name of the MBeanServer bean to use:

{"hints": [

{

"name": "spring.jmx.server", "providers": [

{

"name": "spring-bean-reference", "parameters": {

"target": "javax.management.MBeanServer"

}

}

]

}

]}

The binder is not aware of the metadata. If you provide that hint, you still need to transform the bean name into an actual Bean reference using by the ApplicationContext.

**Spring Profile Name**

The **spring-profile-name** provider auto-completes the Spring profiles that are defined in the configuration of the current project.

The following metadata snippet corresponds to the standard spring.profiles.active property that defines the name of the Spring profile(s) to enable:

{"hints": [

{

"name": "spring.profiles.active", "providers": [

{

"name": "spring-profile-name"

}

]

}

]}

### Generating Your Own Metadata by Using the Annotation Processor

You can easily generate your own configuration metadata file from items annotated with @ConfigurationProperties by using the spring-boot-configuration-processor jar. The jar includes a Java annotation processor which is invoked as your project is compiled.

#### Configuring the Annotation Processor

To use the processor, include a dependency on spring-boot-configuration-processor.

With Maven the dependency should be declared as optional, as shown in the following example:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

If you have defined @ConfigurationProperties in your application, make sure to configure the

spring-boot-maven-plugin to prevent the repackage goal from adding the dependency into the fat jar:

<project>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<excludes>

<exclude>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-configuration-

processor</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

</plugins>

</build>

</project>

With Gradle 4.5 and earlier, the dependency should be declared in the compileOnly configuration, as shown in the following example:

dependencies {

compileOnly "org.springframework.boot:spring-boot-configuration-processor"

}

With Gradle 4.6 and later, the dependency should be declared in the annotationProcessor

configuration, as shown in the following example:

dependencies {

annotationProcessor "org.springframework.boot:spring-boot-configuration-processor"

}

If you are using an additional-spring-configuration-metadata.json file, the compileJava task should be configured to depend on the processResources task, as shown in the following example:

compileJava.inputs.files(processResources)

This dependency ensures that the additional metadata is available when the annotation processor runs during compilation.

If you are using AspectJ in your project, you need to make sure that the annotation processor runs only once. There are several ways to do this. With Maven, you can configure the maven-apt-plugin explicitly and add the dependency to the annotation processor only there. You could also let the AspectJ plugin run all the processing and disable annotation processing in the maven-compiler-plugin configuration, as follows:



<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<proc>none</proc>

</configuration>

</plugin>

#### Automatic Metadata Generation

The processor picks up both classes and methods that are annotated with @ConfigurationProperties.

If the class is also annotated with @ConstructorBinding, a single constructor is expected and one property is created per constructor parameter. Otherwise, properties are discovered through the presence of standard getters and setters with special handling for collection and map types (that is detected even if only a getter is present). The annotation processor also supports the use of the @Data, @Getter, and @Setter lombok annotations.

Consider the following example:

@ConfigurationProperties(prefix="server") public class ServerProperties {

/\*\*

* Name of the server.

\*/

private String name;

/\*\*

\* IP address to listen to.

\*/

private String ip = "127.0.0.1";

/\*\*

* Port to listener to.

\*/

private int port = 9797;

// ... getter and setters

}

This exposes three properties where server.name has no default and server.ip and server.port defaults to "127.0.0.1" and 9797 respectively. The Javadoc on fields is used to populate the description attribute. For instance, the description of server.ip is "IP address to listen to.".

You should only use plain text with @ConfigurationProperties field Javadoc, since they are not processed before being added to the JSON.

The annotation processor applies a number of heuristics to extract the default value from the source model. Default values have to be provided statically. In particular, do not refer to a constant defined in another class. Also, the annotation processor cannot auto-detect default values for Enums and Collectionss.

For cases where the default value could not be detected, [manual metadata](#_bookmark685) should be provided. Consider the following example:

@ConfigurationProperties(prefix = "acme.messaging") public class MessagingProperties {

private List<String> addresses = new ArrayList<>(Arrays.asList("a", "b")); private ContainerType containerType = ContainerType.SIMPLE;

// ... getter and setters public enum ContainerType {

SIMPLE, DIRECT

}

}

In order to document default values for properties in the class above, you could add the following content to [the manual metadata of the module](#_bookmark685):

{"properties": [

{

"name": "acme.messaging.addresses", "defaultValue": ["a", "b"]

},

{

"name": "acme.messaging.container-type", "defaultValue": "simple"

}

]}

Only the name of the property is required to document additional metadata for existing properties.

**Nested Properties**

The annotation processor automatically considers inner classes as nested properties. Rather than documenting the ip and port at the root of the namespace, we could create a sub-namespace for it. Consider the updated example:

@ConfigurationProperties(prefix="server") public class ServerProperties {

private String name; private Host host;

// ... getter and setters public static class Host { private String ip;

private int port;

// ... getter and setters

}

}

The preceding example produces metadata information for server.name, server.host.ip, and server.host.port properties. You can use the @NestedConfigurationProperty annotation on a field to indicate that a regular (non-inner) class should be treated as if it were nested.

This has no effect on collections and maps, as those types are automatically identified, and a single metadata property is generated for each of them.

#### Adding Additional Metadata

Spring Boot’s configuration file handling is quite flexible, and it is often the case that properties may exist that are not bound to a @ConfigurationProperties bean. You may also need to tune some attributes of an existing key. To support such cases and let you provide custom "hints", the annotation processor automatically merges items from META-INF/additional-spring-configuration- metadata.json into the main metadata file.

If you refer to a property that has been detected automatically, the description, default value, and deprecation information are overridden, if specified. If the manual property declaration is not identified in the current module, it is added as a new property.

The format of the additional-spring-configuration-metadata.json file is exactly the same as the regular spring-configuration-metadata.json. The additional properties file is optional. If you do not have any additional properties, do not add the file.

## Appendix C: Auto-configuration Classes

This appendix contains details of all of the auto-configuration classes provided by Spring Boot, with links to documentation and source code. Remember to also look at the conditions report in your

application for more details of which features are switched on. (To do so, start the app with --debug

or -Ddebug or, in an Actuator application, use the conditions endpoint).

### spring-boot-autoconfigure

The following auto-configuration classes are from the spring-boot-autoconfigure module:

|  |  |
| --- | --- |
| **Configuration Class** | **Links** |
| [ActiveMQAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/activemq/ActiveMQAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/jms/activemq/ActiveMQAutoConfiguration.html) |
| [AopAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/aop/AopAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/aop/AopAutoConfiguration.html) |
| [ApplicationAvailabilityAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/availability/ApplicationAvailabilityAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/availability/ApplicationAvailabilityAutoConfiguration.html) |
| [ArtemisAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/artemis/ArtemisAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/jms/artemis/ArtemisAutoConfiguration.html) |
| [BatchAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/batch/BatchAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/batch/BatchAutoConfiguration.html) |
| [CacheAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/cache/CacheAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/cache/CacheAutoConfiguration.html) |
| [CassandraAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/cassandra/CassandraAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/cassandra/CassandraAutoConfiguration.html) |
| [CassandraDataAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/cassandra/CassandraDataAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/cassandra/CassandraDataAutoConfiguration.html) |
| [CassandraReactiveDataAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/cassandra/CassandraReactiveDataAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/cassandra/CassandraReactiveDataAutoConfiguration.html) |
| [CassandraReactiveRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/cassandra/CassandraReactiveRepositoriesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/cassandra/CassandraReactiveRepositoriesAutoConfiguration.html) |
| [CassandraRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/cassandra/CassandraRepositoriesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/cassandra/CassandraRepositoriesAutoConfiguration.html) |
| [ClientHttpConnectorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/reactive/function/client/ClientHttpConnectorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/web/reactive/function/client/ClientHttpConnectorAutoConfiguration.html) |
| [CodecsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/http/codec/CodecsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/http/codec/CodecsAutoConfiguration.html) |
| [ConfigurationPropertiesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/context/ConfigurationPropertiesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/context/ConfigurationPropertiesAutoConfiguration.html) |
| [CouchbaseAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/couchbase/CouchbaseAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/couchbase/CouchbaseAutoConfiguration.html) |
| [CouchbaseDataAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/couchbase/CouchbaseDataAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/couchbase/CouchbaseDataAutoConfiguration.html) |
| [CouchbaseReactiveDataAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/couchbase/CouchbaseReactiveDataAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/couchbase/CouchbaseReactiveDataAutoConfiguration.html) |
| [CouchbaseReactiveRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/couchbase/CouchbaseReactiveRepositoriesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/couchbase/CouchbaseReactiveRepositoriesAutoConfiguration.html) |
| [CouchbaseRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/couchbase/CouchbaseRepositoriesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/couchbase/CouchbaseRepositoriesAutoConfiguration.html) |
| [DataSourceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/DataSourceAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/jdbc/DataSourceAutoConfiguration.html) |
| [DataSourceTransactionManagerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/DataSourceTransactionManagerAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/jdbc/DataSourceTransactionManagerAutoConfiguration.html) |
| [DispatcherServletAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/servlet/DispatcherServletAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/web/servlet/DispatcherServletAutoConfiguration.html) |
| [ElasticsearchDataAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/elasticsearch/ElasticsearchDataAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/elasticsearch/ElasticsearchDataAutoConfiguration.html) |
| [ElasticsearchRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/elasticsearch/ElasticsearchRepositoriesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/elasticsearch/ElasticsearchRepositoriesAutoConfiguration.html) |
| [ElasticsearchRestClientAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/elasticsearch/ElasticsearchRestClientAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/elasticsearch/ElasticsearchRestClientAutoConfiguration.html) |
| [EmbeddedLdapAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/ldap/embedded/EmbeddedLdapAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/ldap/embedded/EmbeddedLdapAutoConfiguration.html) |
| [EmbeddedMongoAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mongo/embedded/EmbeddedMongoAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/mongo/embedded/EmbeddedMongoAutoConfiguration.html) |
| [EmbeddedWebServerFactoryCustomizerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/embedded/EmbeddedWebServerFactoryCustomizerAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/web/embedded/EmbeddedWebServerFactoryCustomizerAutoConfiguration.html) |
| [ErrorMvcAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/servlet/error/ErrorMvcAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/web/servlet/error/ErrorMvcAutoConfiguration.html) |

|  |  |
| --- | --- |
| **Configuration Class** | **Links** |
| [ErrorWebFluxAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/reactive/error/ErrorWebFluxAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/web/reactive/error/ErrorWebFluxAutoConfiguration.html) |
| [FlywayAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/flyway/FlywayAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/flyway/FlywayAutoConfiguration.html) |
| [FreeMarkerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/freemarker/FreeMarkerAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/freemarker/FreeMarkerAutoConfiguration.html) |
| [GroovyTemplateAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/groovy/template/GroovyTemplateAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/groovy/template/GroovyTemplateAutoConfiguration.html) |
| [GsonAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/gson/GsonAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/gson/GsonAutoConfiguration.html) |
| [H2ConsoleAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/h2/H2ConsoleAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/h2/H2ConsoleAutoConfiguration.html) |
| [HazelcastAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/hazelcast/HazelcastAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/hazelcast/HazelcastAutoConfiguration.html) |
| [HazelcastJpaDependencyAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/hazelcast/HazelcastJpaDependencyAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/hazelcast/HazelcastJpaDependencyAutoConfiguration.html) |
| [HibernateJpaAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/orm/jpa/HibernateJpaAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/orm/jpa/HibernateJpaAutoConfiguration.html) |
| [HttpEncodingAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/servlet/HttpEncodingAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/web/servlet/HttpEncodingAutoConfiguration.html) |
| [HttpHandlerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/reactive/HttpHandlerAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/web/reactive/HttpHandlerAutoConfiguration.html) |
| [HttpMessageConvertersAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/http/HttpMessageConvertersAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/http/HttpMessageConvertersAutoConfiguration.html) |
| [HypermediaAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/hateoas/HypermediaAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/hateoas/HypermediaAutoConfiguration.html) |
| [InfluxDbAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/influx/InfluxDbAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/influx/InfluxDbAutoConfiguration.html) |
| [IntegrationAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/integration/IntegrationAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/integration/IntegrationAutoConfiguration.html) |
| [JacksonAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jackson/JacksonAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/jackson/JacksonAutoConfiguration.html) |
| [JdbcRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/jdbc/JdbcRepositoriesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/jdbc/JdbcRepositoriesAutoConfiguration.html) |
| [JdbcTemplateAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/JdbcTemplateAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/jdbc/JdbcTemplateAutoConfiguration.html) |
| [JerseyAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jersey/JerseyAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/jersey/JerseyAutoConfiguration.html) |
| [JmsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/JmsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/jms/JmsAutoConfiguration.html) |
| [JmxAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jmx/JmxAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/jmx/JmxAutoConfiguration.html) |
| [JndiConnectionFactoryAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jms/JndiConnectionFactoryAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/jms/JndiConnectionFactoryAutoConfiguration.html) |
| [JndiDataSourceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/JndiDataSourceAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/jdbc/JndiDataSourceAutoConfiguration.html) |
| [JooqAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jooq/JooqAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/jooq/JooqAutoConfiguration.html) |
| [JpaRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/jpa/JpaRepositoriesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/jpa/JpaRepositoriesAutoConfiguration.html) |
| [JsonbAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jsonb/JsonbAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/jsonb/JsonbAutoConfiguration.html) |
| [JtaAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/transaction/jta/JtaAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/transaction/jta/JtaAutoConfiguration.html) |
| [KafkaAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/kafka/KafkaAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/kafka/KafkaAutoConfiguration.html) |
| [LdapAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/ldap/LdapAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/ldap/LdapAutoConfiguration.html) |
| [LdapRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/ldap/LdapRepositoriesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/ldap/LdapRepositoriesAutoConfiguration.html) |
| [LifecycleAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/context/LifecycleAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/context/LifecycleAutoConfiguration.html) |
| [LiquibaseAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/liquibase/LiquibaseAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/liquibase/LiquibaseAutoConfiguration.html) |
| [MailSenderAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mail/MailSenderAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/mail/MailSenderAutoConfiguration.html) |

|  |  |
| --- | --- |
| **Configuration Class** | **Links** |
| [MailSenderValidatorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mail/MailSenderValidatorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/mail/MailSenderValidatorAutoConfiguration.html) |
| [MessageSourceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/context/MessageSourceAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/context/MessageSourceAutoConfiguration.html) |
| [MongoAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mongo/MongoAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/mongo/MongoAutoConfiguration.html) |
| [MongoDataAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/mongo/MongoDataAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/mongo/MongoDataAutoConfiguration.html) |
| [MongoReactiveAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mongo/MongoReactiveAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/mongo/MongoReactiveAutoConfiguration.html) |
| [MongoReactiveDataAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/mongo/MongoReactiveDataAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/mongo/MongoReactiveDataAutoConfiguration.html) |
| [MongoReactiveRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/mongo/MongoReactiveRepositoriesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/mongo/MongoReactiveRepositoriesAutoConfiguration.html) |
| [MongoRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/mongo/MongoRepositoriesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/mongo/MongoRepositoriesAutoConfiguration.html) |
| [MultipartAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/servlet/MultipartAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/web/servlet/MultipartAutoConfiguration.html) |
| [MustacheAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/mustache/MustacheAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/mustache/MustacheAutoConfiguration.html) |
| [Neo4jDataAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/neo4j/Neo4jDataAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/neo4j/Neo4jDataAutoConfiguration.html) |
| [Neo4jRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/neo4j/Neo4jRepositoriesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/neo4j/Neo4jRepositoriesAutoConfiguration.html) |
| [OAuth2ClientAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/oauth2/client/servlet/OAuth2ClientAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/security/oauth2/client/servlet/OAuth2ClientAutoConfiguration.html) |
| [OAuth2ResourceServerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/oauth2/resource/servlet/OAuth2ResourceServerAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/security/oauth2/resource/servlet/OAuth2ResourceServerAutoConfiguration.html) |
| [PersistenceExceptionTranslationAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/dao/PersistenceExceptionTranslationAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/dao/PersistenceExceptionTranslationAutoConfiguration.html) |
| [ProjectInfoAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/info/ProjectInfoAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/info/ProjectInfoAutoConfiguration.html) |
| [PropertyPlaceholderAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/context/PropertyPlaceholderAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/context/PropertyPlaceholderAutoConfiguration.html) |
| [QuartzAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/quartz/QuartzAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/quartz/QuartzAutoConfiguration.html) |
| [R2dbcAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/r2dbc/R2dbcAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/r2dbc/R2dbcAutoConfiguration.html) |
| [R2dbcDataAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/r2dbc/R2dbcDataAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/r2dbc/R2dbcDataAutoConfiguration.html) |
| [R2dbcRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/r2dbc/R2dbcRepositoriesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/r2dbc/R2dbcRepositoriesAutoConfiguration.html) |
| [R2dbcTransactionManagerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/r2dbc/R2dbcTransactionManagerAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/r2dbc/R2dbcTransactionManagerAutoConfiguration.html) |
| [RSocketMessagingAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/rsocket/RSocketMessagingAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/rsocket/RSocketMessagingAutoConfiguration.html) |
| [RSocketRequesterAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/rsocket/RSocketRequesterAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/rsocket/RSocketRequesterAutoConfiguration.html) |
| [RSocketSecurityAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/rsocket/RSocketSecurityAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/security/rsocket/RSocketSecurityAutoConfiguration.html) |
| [RSocketServerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/rsocket/RSocketServerAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/rsocket/RSocketServerAutoConfiguration.html) |
| [RSocketStrategiesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/rsocket/RSocketStrategiesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/rsocket/RSocketStrategiesAutoConfiguration.html) |
| [RabbitAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/amqp/RabbitAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/amqp/RabbitAutoConfiguration.html) |
| [ReactiveElasticsearchRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/elasticsearch/ReactiveElasticsearchRepositoriesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/elasticsearch/ReactiveElasticsearchRepositoriesAutoConfiguration.html) |
| [ReactiveElasticsearchRestClientAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/elasticsearch/ReactiveElasticsearchRestClientAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/elasticsearch/ReactiveElasticsearchRestClientAutoConfiguration.html) |
| [ReactiveOAuth2ClientAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/oauth2/client/reactive/ReactiveOAuth2ClientAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/security/oauth2/client/reactive/ReactiveOAuth2ClientAutoConfiguration.html) |
| [ReactiveOAuth2ResourceServerAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/oauth2/resource/reactive/ReactiveOAuth2ResourceServerAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/security/oauth2/resource/reactive/ReactiveOAuth2ResourceServerAutoConfiguration.html) |
| [ReactiveSecurityAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/reactive/ReactiveSecurityAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/security/reactive/ReactiveSecurityAutoConfiguration.html) |

|  |  |
| --- | --- |
| **Configuration Class** | **Links** |
| [ReactiveUserDetailsServiceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/reactive/ReactiveUserDetailsServiceAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/security/reactive/ReactiveUserDetailsServiceAutoConfiguration.html) |
| [ReactiveWebServerFactoryAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/reactive/ReactiveWebServerFactoryAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/web/reactive/ReactiveWebServerFactoryAutoConfiguration.html) |
| [RedisAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/redis/RedisAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/redis/RedisAutoConfiguration.html) |
| [RedisReactiveAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/redis/RedisReactiveAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/redis/RedisReactiveAutoConfiguration.html) |
| [RedisRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/redis/RedisRepositoriesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/redis/RedisRepositoriesAutoConfiguration.html) |
| [RepositoryRestMvcAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/rest/RepositoryRestMvcAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/rest/RepositoryRestMvcAutoConfiguration.html) |
| [RestTemplateAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/client/RestTemplateAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/web/client/RestTemplateAutoConfiguration.html) |
| [Saml2RelyingPartyAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/saml2/Saml2RelyingPartyAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/security/saml2/Saml2RelyingPartyAutoConfiguration.html) |
| [SecurityAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/servlet/SecurityAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/security/servlet/SecurityAutoConfiguration.html) |
| [SecurityFilterAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/servlet/SecurityFilterAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/security/servlet/SecurityFilterAutoConfiguration.html) |
| [SendGridAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/sendgrid/SendGridAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/sendgrid/SendGridAutoConfiguration.html) |
| [ServletWebServerFactoryAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/servlet/ServletWebServerFactoryAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/web/servlet/ServletWebServerFactoryAutoConfiguration.html) |
| [SessionAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/session/SessionAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/session/SessionAutoConfiguration.html) |
| [SolrAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/solr/SolrAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/solr/SolrAutoConfiguration.html) |
| [SolrRepositoriesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/solr/SolrRepositoriesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/solr/SolrRepositoriesAutoConfiguration.html) |
| [SpringApplicationAdminJmxAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/admin/SpringApplicationAdminJmxAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/admin/SpringApplicationAdminJmxAutoConfiguration.html) |
| [SpringDataWebAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/data/web/SpringDataWebAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/data/web/SpringDataWebAutoConfiguration.html) |
| [TaskExecutionAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/task/TaskExecutionAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/task/TaskExecutionAutoConfiguration.html) |
| [TaskSchedulingAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/task/TaskSchedulingAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/task/TaskSchedulingAutoConfiguration.html) |
| [ThymeleafAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/thymeleaf/ThymeleafAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/thymeleaf/ThymeleafAutoConfiguration.html) |
| [TransactionAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/transaction/TransactionAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/transaction/TransactionAutoConfiguration.html) |
| [UserDetailsServiceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/security/servlet/UserDetailsServiceAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/security/servlet/UserDetailsServiceAutoConfiguration.html) |
| [ValidationAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/validation/ValidationAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/validation/ValidationAutoConfiguration.html) |
| [WebClientAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/reactive/function/client/WebClientAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/web/reactive/function/client/WebClientAutoConfiguration.html) |
| [WebFluxAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/reactive/WebFluxAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/web/reactive/WebFluxAutoConfiguration.html) |
| [WebMvcAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/web/servlet/WebMvcAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/web/servlet/WebMvcAutoConfiguration.html) |
| [WebServiceTemplateAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/webservices/client/WebServiceTemplateAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/webservices/client/WebServiceTemplateAutoConfiguration.html) |
| [WebServicesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/webservices/WebServicesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/webservices/WebServicesAutoConfiguration.html) |
| [WebSocketMessagingAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/websocket/servlet/WebSocketMessagingAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/websocket/servlet/WebSocketMessagingAutoConfiguration.html) |
| [WebSocketReactiveAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/websocket/reactive/WebSocketReactiveAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/websocket/reactive/WebSocketReactiveAutoConfiguration.html) |
| [WebSocketServletAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/websocket/servlet/WebSocketServletAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/websocket/servlet/WebSocketServletAutoConfiguration.html) |
| [XADataSourceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/XADataSourceAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/autoconfigure/jdbc/XADataSourceAutoConfiguration.html) |

### spring-boot-actuator-autoconfigure

The following auto-configuration classes are from the spring-boot-actuator-autoconfigure module:

|  |  |
| --- | --- |
| **Configuration Class** | **Links** |
| [AppOpticsMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/appoptics/AppOpticsMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/appoptics/AppOpticsMetricsExportAutoConfiguration.html) |
| [AtlasMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/atlas/AtlasMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/atlas/AtlasMetricsExportAutoConfiguration.html) |
| [AuditAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/audit/AuditAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/audit/AuditAutoConfiguration.html) |
| [AuditEventsEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/audit/AuditEventsEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/audit/AuditEventsEndpointAutoConfiguration.html) |
| [AvailabilityHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/availability/AvailabilityHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/availability/AvailabilityHealthContributorAutoConfiguration.html) |
| [AvailabilityProbesAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/availability/AvailabilityProbesAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/availability/AvailabilityProbesAutoConfiguration.html) |
| [BeansEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/beans/BeansEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/beans/BeansEndpointAutoConfiguration.html) |
| [CacheMetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/cache/CacheMetricsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/cache/CacheMetricsAutoConfiguration.html) |
| [CachesEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/cache/CachesEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/cache/CachesEndpointAutoConfiguration.html) |
| [CassandraHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/cassandra/CassandraHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/cassandra/CassandraHealthContributorAutoConfiguration.html) |
| [CassandraReactiveHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/cassandra/CassandraReactiveHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/cassandra/CassandraReactiveHealthContributorAutoConfiguration.html) |
| [CloudFoundryActuatorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/cloudfoundry/servlet/CloudFoundryActuatorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/cloudfoundry/servlet/CloudFoundryActuatorAutoConfiguration.html) |
| [CompositeMeterRegistryAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/CompositeMeterRegistryAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/CompositeMeterRegistryAutoConfiguration.html) |
| [ConditionsReportEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/condition/ConditionsReportEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/condition/ConditionsReportEndpointAutoConfiguration.html) |
| [ConfigurationPropertiesReportEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/context/properties/ConfigurationPropertiesReportEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/context/properties/ConfigurationPropertiesReportEndpointAutoConfiguration.html) |
| [ConnectionFactoryHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/r2dbc/ConnectionFactoryHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/r2dbc/ConnectionFactoryHealthContributorAutoConfiguration.html) |
| [ConnectionPoolMetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/r2dbc/ConnectionPoolMetricsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/r2dbc/ConnectionPoolMetricsAutoConfiguration.html) |
| [CouchbaseHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/couchbase/CouchbaseHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/couchbase/CouchbaseHealthContributorAutoConfiguration.html) |
| [CouchbaseReactiveHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/couchbase/CouchbaseReactiveHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/couchbase/CouchbaseReactiveHealthContributorAutoConfiguration.html) |
| [DataSourceHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/jdbc/DataSourceHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/jdbc/DataSourceHealthContributorAutoConfiguration.html) |
| [DataSourcePoolMetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/jdbc/DataSourcePoolMetricsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/jdbc/DataSourcePoolMetricsAutoConfiguration.html) |
| [DatadogMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/datadog/DatadogMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/datadog/DatadogMetricsExportAutoConfiguration.html) |
| [DiskSpaceHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/system/DiskSpaceHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/system/DiskSpaceHealthContributorAutoConfiguration.html) |
| [DynatraceMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/dynatrace/DynatraceMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/dynatrace/DynatraceMetricsExportAutoConfiguration.html) |
| [ElasticMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/elastic/ElasticMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/elastic/ElasticMetricsExportAutoConfiguration.html) |
| [ElasticSearchReactiveHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/elasticsearch/ElasticSearchReactiveHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/elasticsearch/ElasticSearchReactiveHealthContributorAutoConfiguration.html) |
| [ElasticSearchRestHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/elasticsearch/ElasticSearchRestHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/elasticsearch/ElasticSearchRestHealthContributorAutoConfiguration.html) |
| [EndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/endpoint/EndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/endpoint/EndpointAutoConfiguration.html) |
| [EnvironmentEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/env/EnvironmentEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/env/EnvironmentEndpointAutoConfiguration.html) |
| [FlywayEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/flyway/FlywayEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/flyway/FlywayEndpointAutoConfiguration.html) |
| [GangliaMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/ganglia/GangliaMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/ganglia/GangliaMetricsExportAutoConfiguration.html) |

|  |  |
| --- | --- |
| **Configuration Class** | **Links** |
| [GraphiteMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/graphite/GraphiteMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/graphite/GraphiteMetricsExportAutoConfiguration.html) |
| [HazelcastHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/hazelcast/HazelcastHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/hazelcast/HazelcastHealthContributorAutoConfiguration.html) |
| [HealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/health/HealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/health/HealthContributorAutoConfiguration.html) |
| [HealthEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/health/HealthEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/health/HealthEndpointAutoConfiguration.html) |
| [HeapDumpWebEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/management/HeapDumpWebEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/management/HeapDumpWebEndpointAutoConfiguration.html) |
| [HibernateMetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/orm/jpa/HibernateMetricsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/orm/jpa/HibernateMetricsAutoConfiguration.html) |
| [HttpClientMetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/web/client/HttpClientMetricsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/web/client/HttpClientMetricsAutoConfiguration.html) |
| [HttpTraceAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/trace/http/HttpTraceAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/trace/http/HttpTraceAutoConfiguration.html) |
| [HttpTraceEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/trace/http/HttpTraceEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/trace/http/HttpTraceEndpointAutoConfiguration.html) |
| [HumioMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/humio/HumioMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/humio/HumioMetricsExportAutoConfiguration.html) |
| [InfluxDbHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/influx/InfluxDbHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/influx/InfluxDbHealthContributorAutoConfiguration.html) |
| [InfluxMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/influx/InfluxMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/influx/InfluxMetricsExportAutoConfiguration.html) |
| [InfoContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/info/InfoContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/info/InfoContributorAutoConfiguration.html) |
| [InfoEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/info/InfoEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/info/InfoEndpointAutoConfiguration.html) |
| [IntegrationGraphEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/integration/IntegrationGraphEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/integration/IntegrationGraphEndpointAutoConfiguration.html) |
| [JerseyServerMetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/jersey/JerseyServerMetricsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/jersey/JerseyServerMetricsAutoConfiguration.html) |
| [JettyMetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/web/jetty/JettyMetricsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/web/jetty/JettyMetricsAutoConfiguration.html) |
| [JmsHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/jms/JmsHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/jms/JmsHealthContributorAutoConfiguration.html) |
| [JmxEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/endpoint/jmx/JmxEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/endpoint/jmx/JmxEndpointAutoConfiguration.html) |
| [JmxMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/jmx/JmxMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/jmx/JmxMetricsExportAutoConfiguration.html) |
| [JolokiaEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/jolokia/JolokiaEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/jolokia/JolokiaEndpointAutoConfiguration.html) |
| [JvmMetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/JvmMetricsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/JvmMetricsAutoConfiguration.html) |
| [KafkaMetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/KafkaMetricsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/KafkaMetricsAutoConfiguration.html) |
| [KairosMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/kairos/KairosMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/kairos/KairosMetricsExportAutoConfiguration.html) |
| [LdapHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/ldap/LdapHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/ldap/LdapHealthContributorAutoConfiguration.html) |
| [LiquibaseEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/liquibase/LiquibaseEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/liquibase/LiquibaseEndpointAutoConfiguration.html) |
| [Log4J2MetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/Log4J2MetricsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/Log4J2MetricsAutoConfiguration.html) |
| [LogFileWebEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/logging/LogFileWebEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/logging/LogFileWebEndpointAutoConfiguration.html) |
| [LogbackMetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/LogbackMetricsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/LogbackMetricsAutoConfiguration.html) |
| [LoggersEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/logging/LoggersEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/logging/LoggersEndpointAutoConfiguration.html) |
| [MailHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/mail/MailHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/mail/MailHealthContributorAutoConfiguration.html) |
| [ManagementContextAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/web/server/ManagementContextAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/web/server/ManagementContextAutoConfiguration.html) |
| [ManagementWebSecurityAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/security/servlet/ManagementWebSecurityAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/security/servlet/ManagementWebSecurityAutoConfiguration.html) |

|  |  |
| --- | --- |
| **Configuration Class** | **Links** |
| [MappingsEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/web/mappings/MappingsEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/web/mappings/MappingsEndpointAutoConfiguration.html) |
| [MetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/MetricsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/MetricsAutoConfiguration.html) |
| [MetricsEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/MetricsEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/MetricsEndpointAutoConfiguration.html) |
| [MongoHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/mongo/MongoHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/mongo/MongoHealthContributorAutoConfiguration.html) |
| [MongoReactiveHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/mongo/MongoReactiveHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/mongo/MongoReactiveHealthContributorAutoConfiguration.html) |
| [Neo4jHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/neo4j/Neo4jHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/neo4j/Neo4jHealthContributorAutoConfiguration.html) |
| [NewRelicMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/newrelic/NewRelicMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/newrelic/NewRelicMetricsExportAutoConfiguration.html) |
| [PrometheusMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/prometheus/PrometheusMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/prometheus/PrometheusMetricsExportAutoConfiguration.html) |
| [RabbitHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/amqp/RabbitHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/amqp/RabbitHealthContributorAutoConfiguration.html) |
| [RabbitMetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/amqp/RabbitMetricsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/amqp/RabbitMetricsAutoConfiguration.html) |
| [ReactiveCloudFoundryActuatorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/cloudfoundry/reactive/ReactiveCloudFoundryActuatorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/cloudfoundry/reactive/ReactiveCloudFoundryActuatorAutoConfiguration.html) |
| [ReactiveManagementContextAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/web/reactive/ReactiveManagementContextAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/web/reactive/ReactiveManagementContextAutoConfiguration.html) |
| [ReactiveManagementWebSecurityAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/security/reactive/ReactiveManagementWebSecurityAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/security/reactive/ReactiveManagementWebSecurityAutoConfiguration.html) |
| [RedisHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/redis/RedisHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/redis/RedisHealthContributorAutoConfiguration.html) |
| [RedisReactiveHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/redis/RedisReactiveHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/redis/RedisReactiveHealthContributorAutoConfiguration.html) |
| [ScheduledTasksEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/scheduling/ScheduledTasksEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/scheduling/ScheduledTasksEndpointAutoConfiguration.html) |
| [ServletManagementContextAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/web/servlet/ServletManagementContextAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/web/servlet/ServletManagementContextAutoConfiguration.html) |
| [SessionsEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/session/SessionsEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/session/SessionsEndpointAutoConfiguration.html) |
| [ShutdownEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/context/ShutdownEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/context/ShutdownEndpointAutoConfiguration.html) |
| [SignalFxMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/signalfx/SignalFxMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/signalfx/SignalFxMetricsExportAutoConfiguration.html) |
| [SimpleMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/simple/SimpleMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/simple/SimpleMetricsExportAutoConfiguration.html) |
| [SolrHealthContributorAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/solr/SolrHealthContributorAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/solr/SolrHealthContributorAutoConfiguration.html) |
| [StackdriverMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/stackdriver/StackdriverMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/stackdriver/StackdriverMetricsExportAutoConfiguration.html) |
| [StatsdMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/statsd/StatsdMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/statsd/StatsdMetricsExportAutoConfiguration.html) |
| [SystemMetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/SystemMetricsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/SystemMetricsAutoConfiguration.html) |
| [ThreadDumpEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/management/ThreadDumpEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/management/ThreadDumpEndpointAutoConfiguration.html) |
| [TomcatMetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/web/tomcat/TomcatMetricsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/web/tomcat/TomcatMetricsAutoConfiguration.html) |
| [WavefrontMetricsExportAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/export/wavefront/WavefrontMetricsExportAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/export/wavefront/WavefrontMetricsExportAutoConfiguration.html) |
| [WebEndpointAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/endpoint/web/WebEndpointAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/endpoint/web/WebEndpointAutoConfiguration.html) |
| [WebFluxMetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/web/reactive/WebFluxMetricsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/web/reactive/WebFluxMetricsAutoConfiguration.html) |
| [WebMvcMetricsAutoConfiguration](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-actuator-autoconfigure/src/main/java/org/springframework/boot/actuate/autoconfigure/metrics/web/servlet/WebMvcMetricsAutoConfiguration.java) | [javadoc](https://docs.spring.io/spring-boot/docs/2.3.4.RELEASE/api/org/springframework/boot/actuate/autoconfigure/metrics/web/servlet/WebMvcMetricsAutoConfiguration.html) |

## Appendix D: Test Auto-configuration Annotations

This appendix describes the @…Test auto-configuration annotations that Spring Boot provides to test slices of your application.

### 10.D.1. Test Slices

The following table lists the various @…Test annotations that can be used to test slices of your application and the auto-configuration that they import by default:

|  |  |
| --- | --- |
| **Test slice** | **Imported auto-configuration** |
| @DataJdbcTest | org.springframework.boot.autoconfigure.cache.C  acheAutoConfiguration org.springframework.boot.autoconfigure.data.jd  bc.JdbcRepositoriesAutoConfiguration org.springframework.boot.autoconfigure.flyway.  FlywayAutoConfiguration org.springframework.boot.autoconfigure.jdbc.Da  taSourceAutoConfiguration org.springframework.boot.autoconfigure.jdbc.Da  taSourceTransactionManagerAutoConfiguration org.springframework.boot.autoconfigure.jdbc.Jd  bcTemplateAutoConfiguration org.springframework.boot.autoconfigure.liquiba  se.LiquibaseAutoConfiguration org.springframework.boot.autoconfigure.transac  tion.TransactionAutoConfiguration org.springframework.boot.test.autoconfigure.jd bc.TestDatabaseAutoConfiguration |

|  |  |
| --- | --- |
| **Test slice** | **Imported auto-configuration** |
| @DataJpaTest | org.springframework.boot.autoconfigure.cache.C  acheAutoConfiguration org.springframework.boot.autoconfigure.data.jp  a.JpaRepositoriesAutoConfiguration org.springframework.boot.autoconfigure.flyway.  FlywayAutoConfiguration org.springframework.boot.autoconfigure.jdbc.Da  taSourceAutoConfiguration org.springframework.boot.autoconfigure.jdbc.Da  taSourceTransactionManagerAutoConfiguration org.springframework.boot.autoconfigure.jdbc.Jd  bcTemplateAutoConfiguration org.springframework.boot.autoconfigure.liquiba  se.LiquibaseAutoConfiguration org.springframework.boot.autoconfigure.orm.jpa  .HibernateJpaAutoConfiguration org.springframework.boot.autoconfigure.transac  tion.TransactionAutoConfiguration org.springframework.boot.test.autoconfigure.jd  bc.TestDatabaseAutoConfiguration org.springframework.boot.test.autoconfigure.or m.jpa.TestEntityManagerAutoConfiguration |
| @DataLdapTest | org.springframework.boot.autoconfigure.cache.C  acheAutoConfiguration org.springframework.boot.autoconfigure.data.ld  ap.LdapRepositoriesAutoConfiguration org.springframework.boot.autoconfigure.ldap.Ld  apAutoConfiguration org.springframework.boot.autoconfigure.ldap.em bedded.EmbeddedLdapAutoConfiguration |
| @DataMongoTest | org.springframework.boot.autoconfigure.cache.C  acheAutoConfiguration org.springframework.boot.autoconfigure.data.mo  ngo.MongoDataAutoConfiguration org.springframework.boot.autoconfigure.data.mo  ngo.MongoReactiveDataAutoConfiguration org.springframework.boot.autoconfigure.data.mo ngo.MongoReactiveRepositoriesAutoConfiguration  org.springframework.boot.autoconfigure.data.mo  ngo.MongoRepositoriesAutoConfiguration org.springframework.boot.autoconfigure.mongo.M  ongoAutoConfiguration org.springframework.boot.autoconfigure.mongo.M  ongoReactiveAutoConfiguration org.springframework.boot.autoconfigure.mongo.e  mbedded.EmbeddedMongoAutoConfiguration org.springframework.boot.autoconfigure.transac tion.TransactionAutoConfiguration |

|  |  |
| --- | --- |
| **Test slice** | **Imported auto-configuration** |
| @DataNeo4jTest | org.springframework.boot.autoconfigure.cache.C  acheAutoConfiguration org.springframework.boot.autoconfigure.data.ne  o4j.Neo4jDataAutoConfiguration org.springframework.boot.autoconfigure.data.ne  o4j.Neo4jRepositoriesAutoConfiguration org.springframework.boot.autoconfigure.transac tion.TransactionAutoConfiguration |
| @DataR2dbcTest | org.springframework.boot.autoconfigure.data.r2  dbc.R2dbcDataAutoConfiguration org.springframework.boot.autoconfigure.data.r2  dbc.R2dbcRepositoriesAutoConfiguration org.springframework.boot.autoconfigure.data.r2  dbc.R2dbcTransactionManagerAutoConfiguration org.springframework.boot.autoconfigure.flyway.  FlywayAutoConfiguration org.springframework.boot.autoconfigure.liquiba  se.LiquibaseAutoConfiguration org.springframework.boot.autoconfigure.r2dbc.R  2dbcAutoConfiguration org.springframework.boot.autoconfigure.transac tion.TransactionAutoConfiguration |
| @DataRedisTest | org.springframework.boot.autoconfigure.cache.C  acheAutoConfiguration org.springframework.boot.autoconfigure.data.re  dis.RedisAutoConfiguration org.springframework.boot.autoconfigure.data.re dis.RedisRepositoriesAutoConfiguration |
| @JdbcTest | org.springframework.boot.autoconfigure.cache.C  acheAutoConfiguration org.springframework.boot.autoconfigure.flyway.  FlywayAutoConfiguration org.springframework.boot.autoconfigure.jdbc.Da  taSourceAutoConfiguration org.springframework.boot.autoconfigure.jdbc.Da  taSourceTransactionManagerAutoConfiguration org.springframework.boot.autoconfigure.jdbc.Jd  bcTemplateAutoConfiguration org.springframework.boot.autoconfigure.liquiba  se.LiquibaseAutoConfiguration org.springframework.boot.autoconfigure.transac  tion.TransactionAutoConfiguration org.springframework.boot.test.autoconfigure.jd bc.TestDatabaseAutoConfiguration |

|  |  |
| --- | --- |
| **Test slice** | **Imported auto-configuration** |
| @JooqTest | org.springframework.boot.autoconfigure.cache.C  acheAutoConfiguration org.springframework.boot.autoconfigure.flyway.  FlywayAutoConfiguration org.springframework.boot.autoconfigure.jdbc.Da  taSourceAutoConfiguration org.springframework.boot.autoconfigure.jdbc.Da  taSourceTransactionManagerAutoConfiguration org.springframework.boot.autoconfigure.jooq.Jo  oqAutoConfiguration org.springframework.boot.autoconfigure.liquiba  se.LiquibaseAutoConfiguration org.springframework.boot.autoconfigure.transac tion.TransactionAutoConfiguration |
| @JsonTest | org.springframework.boot.autoconfigure.cache.C  acheAutoConfiguration org.springframework.boot.autoconfigure.gson.Gs  onAutoConfiguration org.springframework.boot.autoconfigure.jackson  .JacksonAutoConfiguration org.springframework.boot.autoconfigure.jsonb.J  sonbAutoConfiguration org.springframework.boot.test.autoconfigure.js on.JsonTestersAutoConfiguration |
| @RestClientTest | org.springframework.boot.autoconfigure.cache.C  acheAutoConfiguration org.springframework.boot.autoconfigure.gson.Gs  onAutoConfiguration org.springframework.boot.autoconfigure.http.Ht  tpMessageConvertersAutoConfiguration org.springframework.boot.autoconfigure.http.co  dec.CodecsAutoConfiguration org.springframework.boot.autoconfigure.jackson  .JacksonAutoConfiguration org.springframework.boot.autoconfigure.jsonb.J  sonbAutoConfiguration org.springframework.boot.autoconfigure.web.cli  ent.RestTemplateAutoConfiguration org.springframework.boot.autoconfigure.web.rea ctive.function.client.WebClientAutoConfigurati  on org.springframework.boot.test.autoconfigure.we b.client.MockRestServiceServerAutoConfiguratio  n org.springframework.boot.test.autoconfigure.we b.client.WebClientRestTemplateAutoConfiguratio n |

|  |  |
| --- | --- |
| **Test slice** | **Imported auto-configuration** |
| @WebFluxTest | org.springframework.boot.autoconfigure.cache.C  acheAutoConfiguration org.springframework.boot.autoconfigure.context  .MessageSourceAutoConfiguration org.springframework.boot.autoconfigure.freemar  ker.FreeMarkerAutoConfiguration org.springframework.boot.autoconfigure.gson.Gs  onAutoConfiguration org.springframework.boot.autoconfigure.http.co  dec.CodecsAutoConfiguration org.springframework.boot.autoconfigure.jackson  .JacksonAutoConfiguration org.springframework.boot.autoconfigure.jsonb.J  sonbAutoConfiguration org.springframework.boot.autoconfigure.mustach  e.MustacheAutoConfiguration org.springframework.boot.autoconfigure.securit y.oauth2.client.reactive.ReactiveOAuth2ClientA  utoConfiguration org.springframework.boot.autoconfigure.securit y.oauth2.resource.reactive.ReactiveOAuth2Resou  rceServerAutoConfiguration org.springframework.boot.autoconfigure.securit  y.reactive.ReactiveSecurityAutoConfiguration org.springframework.boot.autoconfigure.securit y.reactive.ReactiveUserDetailsServiceAutoConfi  guration org.springframework.boot.autoconfigure.thymele  af.ThymeleafAutoConfiguration org.springframework.boot.autoconfigure.validat  ion.ValidationAutoConfiguration org.springframework.boot.autoconfigure.web.rea  ctive.WebFluxAutoConfiguration org.springframework.boot.autoconfigure.web.rea  ctive.error.ErrorWebFluxAutoConfiguration org.springframework.boot.test.autoconfigure.we b.reactive.WebTestClientAutoConfiguration |

|  |  |
| --- | --- |
| **Test slice** | **Imported auto-configuration** |
| @WebMvcTest | org.springframework.boot.autoconfigure.cache.C  acheAutoConfiguration org.springframework.boot.autoconfigure.context  .MessageSourceAutoConfiguration org.springframework.boot.autoconfigure.data.we  b.SpringDataWebAutoConfiguration org.springframework.boot.autoconfigure.freemar  ker.FreeMarkerAutoConfiguration org.springframework.boot.autoconfigure.groovy.  template.GroovyTemplateAutoConfiguration org.springframework.boot.autoconfigure.gson.Gs  onAutoConfiguration org.springframework.boot.autoconfigure.hateoas  .HypermediaAutoConfiguration org.springframework.boot.autoconfigure.http.Ht  tpMessageConvertersAutoConfiguration org.springframework.boot.autoconfigure.jackson  .JacksonAutoConfiguration org.springframework.boot.autoconfigure.jsonb.J  sonbAutoConfiguration org.springframework.boot.autoconfigure.mustach  e.MustacheAutoConfiguration org.springframework.boot.autoconfigure.securit y.oauth2.client.servlet.OAuth2ClientAutoConfig  uration org.springframework.boot.autoconfigure.securit y.oauth2.resource.servlet.OAuth2ResourceServer  AutoConfiguration org.springframework.boot.autoconfigure.securit  y.servlet.SecurityAutoConfiguration org.springframework.boot.autoconfigure.securit  y.servlet.SecurityFilterAutoConfiguration org.springframework.boot.autoconfigure.securit  y.servlet.UserDetailsServiceAutoConfiguration org.springframework.boot.autoconfigure.task.Ta  skExecutionAutoConfiguration org.springframework.boot.autoconfigure.thymele  af.ThymeleafAutoConfiguration org.springframework.boot.autoconfigure.validat  ion.ValidationAutoConfiguration org.springframework.boot.autoconfigure.web.ser  vlet.WebMvcAutoConfiguration org.springframework.boot.autoconfigure.web.ser  vlet.error.ErrorMvcAutoConfiguration org.springframework.boot.test.autoconfigure.we  b.servlet.MockMvcAutoConfiguration org.springframework.boot.test.autoconfigure.we  b.servlet.MockMvcSecurityConfiguration org.springframework.boot.test.autoconfigure.we  b.servlet.MockMvcWebClientAutoConfiguration org.springframework.boot.test.autoconfigure.we b.servlet.MockMvcWebDriverAutoConfiguration |
|  |  |

|  |  |
| --- | --- |
| **Test slice** | **Imported auto-configuration** |
| @WebServiceClientTest | org.springframework.boot.autoconfigure.cache.C  acheAutoConfiguration org.springframework.boot.autoconfigure.webserv ices.client.WebServiceTemplateAutoConfiguratio  n org.springframework.boot.test.autoconfigure.we bservices.client.MockWebServiceServerAutoConfi  guration org.springframework.boot.test.autoconfigure.we bservices.client.WebServiceClientTemplateAutoC onfiguration |

## Appendix E: The Executable Jar Format

The spring-boot-loader modules lets Spring Boot support executable jar and war files. If you use the Maven plugin or the Gradle plugin, executable jars are automatically generated, and you generally do not need to know the details of how they work.

If you need to create executable jars from a different build system or if you are just curious about the underlying technology, this appendix provides some background.

### Nested JARs

Java does not provide any standard way to load nested jar files (that is, jar files that are themselves contained within a jar). This can be problematic if you need to distribute a self-contained application that can be run from the command line without unpacking.

To solve this problem, many developers use “shaded” jars. A shaded jar packages all classes, from all jars, into a single “uber jar”. The problem with shaded jars is that it becomes hard to see which libraries are actually in your application. It can also be problematic if the same filename is used (but with different content) in multiple jars. Spring Boot takes a different approach and lets you actually nest jars directly.

#### The Executable Jar File Structure

Spring Boot Loader-compatible jar files should be structured in the following way:

example.jar

|

+-META-INF

| +-MANIFEST.MF

+-org

| +-springframework

| +-boot

| +-loader

|

+-<spring boot loader classes>

+-BOOT-INF

+-classes

| +-mycompany

| +-project

| +-YourClasses.class

+-lib

+-dependency1.jar

+-dependency2.jar

Application classes should be placed in a nested BOOT-INF/classes directory. Dependencies should be placed in a nested BOOT-INF/lib directory.

#### The Executable War File Structure

Spring Boot Loader-compatible war files should be structured in the following way:

example.war

|

+-META-INF

| +-MANIFEST.MF

+-org

| +-springframework

| +-boot

| +-loader

|

+-<spring boot loader classes>

+-WEB-INF

+-classes

| +-com

|

|

|

+-mycompany

+-project

+-YourClasses.class

+-lib

| +-dependency1.jar

| +-dependency2.jar

+-lib-provided

+-servlet-api.jar

+-dependency3.jar

Dependencies should be placed in a nested WEB-INF/lib directory. Any dependencies that are

required when running embedded but are not required when deploying to a traditional web container should be placed in WEB-INF/lib-provided.

#### Index Files

Spring Boot Loader-compatible jar and war archives can include additional index files under the BOOT-INF/ directory. A classpath.idx file can be provided for both jars and wars, it provides the ordering that jars should be added to the classpath. The layers.idx file can be used only for jars, it allows a jar to be split into logical layers for Docker/OCI image creation.

Index files follow a YAML compatible syntax so that they can be easily parsed by third-party tools. These files, however, are *not* parsed internally as YAML and they must be written in exactly the formats described below in order to be used.

#### Classpath Index

The classpath index file can be provided in BOOT-INF/classpath.idx. It provides a list of jar names (not including the directory) in the order that they should be added to the classpath. Each line must start with dash space ("-·") and names must be in double quotes.

For example, given the following jar:

example.jar

|

+-META-INF

| +-...

+-BOOT-INF

+-classes

| +...

+-lib

+-dependency1.jar

+-dependency2.jar

The index file would look like this:

* "dependency2.jar"
* "dependency1.jar"

#### Layer Index

The layers index file can be provided in BOOT-INF/layers.idx. It provides a list of layers and the parts of the jar that should be contained within them. Layers are written in the order that they should be added to the Docker/OCI image. Layers names are written as quoted strings prefixed with dash space ("-·") and with a colon (":") suffix. Layer content is either a file or directory name written as a quoted string prefixed by space space dash space ("··-·"). A directory name ends with

/, a file name does not. When a directory name is used it means that all files inside that directory

are in the same layer.

A typical example of a layers index would be:

* "dependencies":
  + "BOOT-INF/lib/dependency1.jar"
  + "BOOT-INF/lib/dependency2.jar"
* "application":
  + "BOOT-INF/classes/"
  + "META-INF/"

### Spring Boot’s “JarFile” Class

The core class used to support loading nested jars is org.springframework.boot.loader.jar.JarFile. It lets you load jar content from a standard jar file or from nested child jar data. When first loaded, the location of each JarEntry is mapped to a physical file offset of the outer jar, as shown in the following example:

myapp.jar

+-------------------+-------------------------+

| /BOOT-INF/classes | /BOOT-INF/lib/mylib.jar |

|+-----------------+||+-----------+----------+|

|| A.class ||| B.class | C.class ||

|+-----------------+||+-----------+----------+|

+-------------------+-------------------------+

^ ^ ^

0063 3452 3980

The preceding example shows how A.class can be found in /BOOT-INF/classes in myapp.jar at position 0063. B.class from the nested jar can actually be found in myapp.jar at position 3452, and C.class is at position 3980.

Armed with this information, we can load specific nested entries by seeking to the appropriate part of the outer jar. We do not need to unpack the archive, and we do not need to read all entry data into memory.

#### Compatibility with the Standard Java “JarFile”

Spring Boot Loader strives to remain compatible with existing code and libraries. org.springframework.boot.loader.jar.JarFile extends from java.util.jar.JarFile and should work as a drop-in replacement. The getURL() method returns a URL that opens a connection compatible with java.net.JarURLConnection and can be used with Java’s URLClassLoader.

### Launching Executable Jars

The org.springframework.boot.loader.Launcher class is a special bootstrap class that is used as an executable jar’s main entry point. It is the actual Main-Class in your jar file, and it is used to setup an appropriate URLClassLoader and ultimately call your main() method.

There are three launcher subclasses (JarLauncher, WarLauncher, and PropertiesLauncher). Their

purpose is to load resources (.class files and so on) from nested jar files or war files in directories (as opposed to those explicitly on the classpath). In the case of JarLauncher and WarLauncher, the nested paths are fixed. JarLauncher looks in BOOT-INF/lib/, and WarLauncher looks in WEB-INF/lib/ and WEB-INF/lib-provided/. You can add extra jars in those locations if you want more. The PropertiesLauncher looks in BOOT-INF/lib/ in your application archive by default. You can add additional locations by setting an environment variable called LOADER\_PATH or loader.path in loader.properties (which is a comma-separated list of directories, archives, or directories within archives).

#### Launcher Manifest

You need to specify an appropriate Launcher as the Main-Class attribute of META-INF/MANIFEST.MF. The actual class that you want to launch (that is, the class that contains a main method) should be specified in the Start-Class attribute.

The following example shows a typical MANIFEST.MF for an executable jar file:

Main-Class: org.springframework.boot.loader.JarLauncher Start-Class: com.mycompany.project.MyApplication

For a war file, it would be as follows:

Main-Class: org.springframework.boot.loader.WarLauncher Start-Class: com.mycompany.project.MyApplication

You need not specify Class-Path entries in your manifest file. The classpath is deduced from the nested jars.

### PropertiesLauncher Features

PropertiesLauncher has a few special features that can be enabled with external properties (System properties, environment variables, manifest entries, or loader.properties). The following table describes these properties:

|  |  |
| --- | --- |
| **Key** | **Purpose** |
| loader.path | Comma-separated Classpath, such as lib,${HOME}/app/lib. Earlier entries take precedence, like a regular -classpath on the javac command line. |

|  |  |
| --- | --- |
| **Key** | **Purpose** |
| loader.home | Used to resolve relative paths in loader.path. For example, given loader.path=lib, then  ${loader.home}/lib is a classpath location (along with all jar files in that directory). This property is also used to locate a loader.properties file, as in the following example [/opt/app](file://localhost/opt/app) It defaults to  ${user.dir}. |
| loader.args | Default arguments for the main method (space separated). |
| loader.main | Name of main class to launch (for example,  com.app.Application). |
| loader.config.name | Name of properties file (for example, launcher). It defaults to loader. |
| loader.config.location | Path to properties file (for example, classpath:loader.properties). It defaults to loader.properties. |
| loader.system | Boolean flag to indicate that all properties should be added to System properties. It defaults to false. |

When specified as environment variables or manifest entries, the following names should be used:

|  |  |  |
| --- | --- | --- |
| **Key** | **Manifest entry** | **Environment variable** |
| loader.path | Loader-Path | LOADER\_PATH |
| loader.home | Loader-Home | LOADER\_HOME |
| loader.args | Loader-Args | LOADER\_ARGS |
| loader.main | Start-Class | LOADER\_MAIN |
| loader.config.location | Loader-Config-Location | LOADER\_CONFIG\_LOCATION |
| loader.system | Loader-System | LOADER\_SYSTEM |

Build plugins automatically move the Main-Class attribute to Start-Class when the  fat jar is built. If you use that, specify the name of the class to launch by using the

Main-Class attribute and leaving out Start-Class.

The following rules apply to working with PropertiesLauncher:

* + - * loader.properties is searched for in loader.home, then in the root of the classpath, and then in

classpath:/BOOT-INF/classes. The first location where a file with that name exists is used.

* + - * loader.home is the directory location of an additional properties file (overriding the default) only when loader.config.location is not specified.
      * loader.path can contain directories (which are scanned recursively for jar and zip files), archive paths, a directory within an archive that is scanned for jar files (for example,

dependencies.jar!/lib), or wildcard patterns (for the default JVM behavior). Archive paths can be relative to loader.home or anywhere in the file system with a jar:file: prefix.

* + - * loader.path (if empty) defaults to BOOT-INF/lib (meaning a local directory or a nested one if running from an archive). Because of this, PropertiesLauncher behaves the same as JarLauncher when no additional configuration is provided.
      * loader.path can not be used to configure the location of loader.properties (the classpath used to search for the latter is the JVM classpath when PropertiesLauncher is launched).
      * Placeholder replacement is done from System and environment variables plus the properties file itself on all values before use.
      * The search order for properties (where it makes sense to look in more than one place) is environment variables, system properties, loader.properties, the exploded archive manifest, and the archive manifest.

### Executable Jar Restrictions

You need to consider the following restrictions when working with a Spring Boot Loader packaged application:

* + - * Zip entry compression: The ZipEntry for a nested jar must be saved by using the ZipEntry.STORED method. This is required so that we can seek directly to individual content within the nested jar. The content of the nested jar file itself can still be compressed, as can any other entries in the outer jar.
      * System classLoader: Launched applications should use Thread.getContextClassLoader() when loading classes (most libraries and frameworks do so by default). Trying to load nested jar classes with ClassLoader.getSystemClassLoader() fails. java.util.Logging always uses the system classloader. For this reason, you should consider a different logging implementation.

### Alternative Single Jar Solutions

If the preceding restrictions mean that you cannot use Spring Boot Loader, consider the following alternatives:

* + - * [Maven Shade Plugin](https://maven.apache.org/plugins/maven-shade-plugin/)
      * [JarClassLoader](http://www.jdotsoft.com/JarClassLoader.php)
      * [OneJar](https://sourceforge.net/projects/one-jar/)
      * [Gradle Shadow Plugin](https://imperceptiblethoughts.com/shadow/)

## Appendix F: Dependency versions

This appendix provides details of the dependencies that are managed by Spring Boot.

### Managed Dependency Coordinates

The following table provides details of all of the dependency versions that are provided by Spring Boot in its CLI (Command Line Interface), Maven dependency management, and Gradle plugin.

When you declare a dependency on one of these artifacts without declaring a version, the version listed in the table is used.

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| antlr | antlr | 2.7.7 |
| ch.qos.logback | logback-access | 1.2.3 |
| ch.qos.logback | logback-classic | 1.2.3 |
| ch.qos.logback | logback-core | 1.2.3 |
| com.atomikos | transactions-jdbc | 4.0.6 |
| com.atomikos | transactions-jms | 4.0.6 |
| com.atomikos | transactions-jta | 4.0.6 |
| com.couchbase.client | java-client | 3.0.8 |
| com.datastax.oss | java-driver-core | 4.6.1 |
| com.datastax.oss | java-driver-core-shaded | 4.6.1 |
| com.datastax.oss | java-driver-mapper-processor | 4.6.1 |
| com.datastax.oss | java-driver-mapper-runtime | 4.6.1 |
| com.datastax.oss | java-driver-query-builder | 4.6.1 |
| com.datastax.oss | java-driver-shaded-guava | 25.1-jre |
| com.datastax.oss | java-driver-test-infra | 4.6.1 |
| com.datastax.oss | native-protocol | 1.4.10 |
| com.fasterxml | classmate | 1.5.1 |
| com.fasterxml.jackson.core | jackson-annotations | 2.11.2 |
| com.fasterxml.jackson.core | jackson-core | 2.11.2 |
| com.fasterxml.jackson.core | jackson-databind | 2.11.2 |
| com.fasterxml.jackson.dataform at | jackson-dataformat-avro | 2.11.2 |
| com.fasterxml.jackson.dataform at | jackson-dataformat-cbor | 2.11.2 |
| com.fasterxml.jackson.dataform at | jackson-dataformat-csv | 2.11.2 |
| com.fasterxml.jackson.dataform at | jackson-dataformat-ion | 2.11.2 |
| com.fasterxml.jackson.dataform at | jackson-dataformat-properties | 2.11.2 |
| com.fasterxml.jackson.dataform at | jackson-dataformat-protobuf | 2.11.2 |
| com.fasterxml.jackson.dataform at | jackson-dataformat-smile | 2.11.2 |
| com.fasterxml.jackson.dataform at | jackson-dataformat-xml | 2.11.2 |
| com.fasterxml.jackson.dataform at | jackson-dataformat-yaml | 2.11.2 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| com.fasterxml.jackson.datatype | jackson-datatype-eclipse- collections | 2.11.2 |
| com.fasterxml.jackson.datatype | jackson-datatype-guava | 2.11.2 |
| com.fasterxml.jackson.datatype | jackson-datatype-hibernate3 | 2.11.2 |
| com.fasterxml.jackson.datatype | jackson-datatype-hibernate4 | 2.11.2 |
| com.fasterxml.jackson.datatype | jackson-datatype-hibernate5 | 2.11.2 |
| com.fasterxml.jackson.datatype | jackson-datatype-hppc | 2.11.2 |
| com.fasterxml.jackson.datatype | jackson-datatype-jaxrs | 2.11.2 |
| com.fasterxml.jackson.datatype | jackson-datatype-jdk8 | 2.11.2 |
| com.fasterxml.jackson.datatype | jackson-datatype-joda | 2.11.2 |
| com.fasterxml.jackson.datatype | jackson-datatype-joda-money | 2.11.2 |
| com.fasterxml.jackson.datatype | jackson-datatype-json-org | 2.11.2 |
| com.fasterxml.jackson.datatype | jackson-datatype-jsr310 | 2.11.2 |
| com.fasterxml.jackson.datatype | jackson-datatype-jsr353 | 2.11.2 |
| com.fasterxml.jackson.datatype | jackson-datatype-pcollections | 2.11.2 |
| com.fasterxml.jackson.jaxrs | jackson-jaxrs-base | 2.11.2 |
| com.fasterxml.jackson.jaxrs | jackson-jaxrs-cbor-provider | 2.11.2 |
| com.fasterxml.jackson.jaxrs | jackson-jaxrs-json-provider | 2.11.2 |
| com.fasterxml.jackson.jaxrs | jackson-jaxrs-smile-provider | 2.11.2 |
| com.fasterxml.jackson.jaxrs | jackson-jaxrs-xml-provider | 2.11.2 |
| com.fasterxml.jackson.jaxrs | jackson-jaxrs-yaml-provider | 2.11.2 |
| com.fasterxml.jackson.jr | jackson-jr-all | 2.11.2 |
| com.fasterxml.jackson.jr | jackson-jr-objects | 2.11.2 |
| com.fasterxml.jackson.jr | jackson-jr-retrofit2 | 2.11.2 |
| com.fasterxml.jackson.jr | jackson-jr-stree | 2.11.2 |
| com.fasterxml.jackson.module | jackson-module-afterburner | 2.11.2 |
| com.fasterxml.jackson.module | jackson-module-guice | 2.11.2 |
| com.fasterxml.jackson.module | jackson-module-jaxb- annotations | 2.11.2 |
| com.fasterxml.jackson.module | jackson-module-jsonSchema | 2.11.2 |
| com.fasterxml.jackson.module | jackson-module-kotlin | 2.11.2 |
| com.fasterxml.jackson.module | jackson-module-mrbean | 2.11.2 |
| com.fasterxml.jackson.module | jackson-module-osgi | 2.11.2 |
| com.fasterxml.jackson.module | jackson-module-parameter-names | 2.11.2 |
| com.fasterxml.jackson.module | jackson-module-paranamer | 2.11.2 |
| com.fasterxml.jackson.module | jackson-module-scala\_2.10 | 2.11.2 |
| com.fasterxml.jackson.module | jackson-module-scala\_2.11 | 2.11.2 |
| com.fasterxml.jackson.module | jackson-module-scala\_2.12 | 2.11.2 |
| com.fasterxml.jackson.module | jackson-module-scala\_2.13 | 2.11.2 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| com.github.ben-manes.caffeine | caffeine | 2.8.5 |
| com.github.ben-manes.caffeine | guava | 2.8.5 |
| com.github.ben-manes.caffeine | jcache | 2.8.5 |
| com.github.ben-manes.caffeine | simulator | 2.8.5 |
| com.github.mxab.thymeleaf.extr as | thymeleaf-extras-data- attribute | 2.0.1 |
| com.google.appengine | appengine-api-1.0-sdk | 1.9.82 |
| com.google.cloud | cloud-spanner-r2dbc | 0.2.0 |
| com.google.code.gson | gson | 2.8.6 |
| com.h2database | h2 | 1.4.200 |
| com.hazelcast | hazelcast | 3.12.9 |
| com.hazelcast | hazelcast-client | 3.12.9 |
| com.hazelcast | hazelcast-hibernate52 | 1.3.2 |
| com.hazelcast | hazelcast-hibernate53 | 1.3.2 |
| com.hazelcast | hazelcast-spring | 3.12.9 |
| com.ibm.db2 | jcc | 11.5.4.0 |
| com.jayway.jsonpath | json-path | 2.4.0 |
| com.jayway.jsonpath | json-path-assert | 2.4.0 |
| com.microsoft.sqlserver | mssql-jdbc | 7.4.1.jre8 |
| com.nimbusds | nimbus-jose-jwt | 8.19 |
| com.nimbusds | oauth2-oidc-sdk | 7.1.1 |
| com.oracle.database.ha | ons | 19.3.0.0 |
| com.oracle.database.ha | simplefan | 19.3.0.0 |
| com.oracle.database.jdbc | ojdbc10 | 19.3.0.0 |
| com.oracle.database.jdbc | ojdbc10dms | 19.3.0.0 |
| com.oracle.database.jdbc | ojdbc8 | 19.3.0.0 |
| com.oracle.database.jdbc | ojdbc8dms | 19.3.0.0 |
| com.oracle.database.jdbc | ucp | 19.3.0.0 |
| com.oracle.database.jdbc.debug | ojdbc10\_g | 19.3.0.0 |
| com.oracle.database.jdbc.debug | ojdbc10dms\_g | 19.3.0.0 |
| com.oracle.database.jdbc.debug | ojdbc8\_g | 19.3.0.0 |
| com.oracle.database.jdbc.debug | ojdbc8dms\_g | 19.3.0.0 |
| com.oracle.database.nls | orai18n | 19.3.0.0 |
| com.oracle.database.observabil ity | dms | 19.3.0.0 |
| com.oracle.database.security | oraclepki | 19.3.0.0 |
| com.oracle.database.security | osdt\_cert | 19.3.0.0 |
| com.oracle.database.security | osdt\_core | 19.3.0.0 |
| com.oracle.database.xml | xdb | 19.3.0.0 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| com.oracle.database.xml | xmlparserv2 | 19.3.0.0 |
| com.oracle.ojdbc | dms | 19.3.0.0 |
| com.oracle.ojdbc | ojdbc10 | 19.3.0.0 |
| com.oracle.ojdbc | ojdbc10\_g | 19.3.0.0 |
| com.oracle.ojdbc | ojdbc10dms | 19.3.0.0 |
| com.oracle.ojdbc | ojdbc10dms\_g | 19.3.0.0 |
| com.oracle.ojdbc | ojdbc8 | 19.3.0.0 |
| com.oracle.ojdbc | ojdbc8\_g | 19.3.0.0 |
| com.oracle.ojdbc | ojdbc8dms | 19.3.0.0 |
| com.oracle.ojdbc | ojdbc8dms\_g | 19.3.0.0 |
| com.oracle.ojdbc | ons | 19.3.0.0 |
| com.oracle.ojdbc | oraclepki | 19.3.0.0 |
| com.oracle.ojdbc | orai18n | 19.3.0.0 |
| com.oracle.ojdbc | osdt\_cert | 19.3.0.0 |
| com.oracle.ojdbc | osdt\_core | 19.3.0.0 |
| com.oracle.ojdbc | simplefan | 19.3.0.0 |
| com.oracle.ojdbc | ucp | 19.3.0.0 |
| com.oracle.ojdbc | xdb | 19.3.0.0 |
| com.oracle.ojdbc | xmlparserv2 | 19.3.0.0 |
| com.querydsl | querydsl-apt | 4.3.1 |
| com.querydsl | querydsl-collections | 4.3.1 |
| com.querydsl | querydsl-core | 4.3.1 |
| com.querydsl | querydsl-jpa | 4.3.1 |
| com.querydsl | querydsl-mongodb | 4.3.1 |
| com.rabbitmq | amqp-client | 5.9.0 |
| com.samskivert | jmustache | 1.15 |
| com.sendgrid | sendgrid-java | 4.4.8 |
| com.squareup.okhttp3 | logging-interceptor | 3.14.9 |
| com.squareup.okhttp3 | mockwebserver | 3.14.9 |
| com.squareup.okhttp3 | okcurl | 3.14.9 |
| com.squareup.okhttp3 | okhttp | 3.14.9 |
| com.squareup.okhttp3 | okhttp-dnsoverhttps | 3.14.9 |
| com.squareup.okhttp3 | okhttp-sse | 3.14.9 |
| com.squareup.okhttp3 | okhttp-testing-support | 3.14.9 |
| com.squareup.okhttp3 | okhttp-tls | 3.14.9 |
| com.squareup.okhttp3 | okhttp-urlconnection | 3.14.9 |
| com.sun.activation | jakarta.activation | 1.2.2 |
| com.sun.mail | jakarta.mail | 1.6.5 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| com.sun.xml.messaging.saaj | saaj-impl | 1.5.2 |
| com.unboundid | unboundid-ldapsdk | 4.0.14 |
| com.zaxxer | HikariCP | 3.4.5 |
| commons-codec | commons-codec | 1.14 |
| commons-pool | commons-pool | 1.6 |
| de.flapdoodle.embed | de.flapdoodle.embed.mongo | 2.2.0 |
| dev.miku | r2dbc-mysql | 0.8.2.RELEASE |
| io.dropwizard.metrics | metrics-annotation | 4.1.12.1 |
| io.dropwizard.metrics | metrics-caffeine | 4.1.12.1 |
| io.dropwizard.metrics | metrics-collectd | 4.1.12.1 |
| io.dropwizard.metrics | metrics-core | 4.1.12.1 |
| io.dropwizard.metrics | metrics-ehcache | 4.1.12.1 |
| io.dropwizard.metrics | metrics-graphite | 4.1.12.1 |
| io.dropwizard.metrics | metrics-healthchecks | 4.1.12.1 |
| io.dropwizard.metrics | metrics-httpasyncclient | 4.1.12.1 |
| io.dropwizard.metrics | metrics-httpclient | 4.1.12.1 |
| io.dropwizard.metrics | metrics-jcache | 4.1.12.1 |
| io.dropwizard.metrics | metrics-jdbi | 4.1.12.1 |
| io.dropwizard.metrics | metrics-jdbi3 | 4.1.12.1 |
| io.dropwizard.metrics | metrics-jersey2 | 4.1.12.1 |
| io.dropwizard.metrics | metrics-jetty9 | 4.1.12.1 |
| io.dropwizard.metrics | metrics-jmx | 4.1.12.1 |
| io.dropwizard.metrics | metrics-json | 4.1.12.1 |
| io.dropwizard.metrics | metrics-jvm | 4.1.12.1 |
| io.dropwizard.metrics | metrics-log4j2 | 4.1.12.1 |
| io.dropwizard.metrics | metrics-logback | 4.1.12.1 |
| io.dropwizard.metrics | metrics-servlet | 4.1.12.1 |
| io.dropwizard.metrics | metrics-servlets | 4.1.12.1 |
| io.lettuce | lettuce-core | 5.3.4.RELEASE |
| io.micrometer | micrometer-core | 1.5.5 |
| io.micrometer | micrometer-jersey2 | 1.5.5 |
| io.micrometer | micrometer-registry-appoptics | 1.5.5 |
| io.micrometer | micrometer-registry-atlas | 1.5.5 |
| io.micrometer | micrometer-registry-azure- monitor | 1.5.5 |
| io.micrometer | micrometer-registry-cloudwatch | 1.5.5 |
| io.micrometer | micrometer-registry- cloudwatch2 | 1.5.5 |
| io.micrometer | micrometer-registry-datadog | 1.5.5 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| io.micrometer | micrometer-registry-dynatrace | 1.5.5 |
| io.micrometer | micrometer-registry-elastic | 1.5.5 |
| io.micrometer | micrometer-registry-ganglia | 1.5.5 |
| io.micrometer | micrometer-registry-graphite | 1.5.5 |
| io.micrometer | micrometer-registry-humio | 1.5.5 |
| io.micrometer | micrometer-registry-influx | 1.5.5 |
| io.micrometer | micrometer-registry-jmx | 1.5.5 |
| io.micrometer | micrometer-registry-kairos | 1.5.5 |
| io.micrometer | micrometer-registry-new-relic | 1.5.5 |
| io.micrometer | micrometer-registry-opentsdb | 1.5.5 |
| io.micrometer | micrometer-registry-prometheus | 1.5.5 |
| io.micrometer | micrometer-registry-signalfx | 1.5.5 |
| io.micrometer | micrometer-registry- stackdriver | 1.5.5 |
| io.micrometer | micrometer-registry-statsd | 1.5.5 |
| io.micrometer | micrometer-registry-wavefront | 1.5.5 |
| io.micrometer | micrometer-test | 1.5.5 |
| io.netty | netty-all | 4.1.52.Final |
| io.netty | netty-buffer | 4.1.52.Final |
| io.netty | netty-codec | 4.1.52.Final |
| io.netty | netty-codec-dns | 4.1.52.Final |
| io.netty | netty-codec-haproxy | 4.1.52.Final |
| io.netty | netty-codec-http | 4.1.52.Final |
| io.netty | netty-codec-http2 | 4.1.52.Final |
| io.netty | netty-codec-memcache | 4.1.52.Final |
| io.netty | netty-codec-mqtt | 4.1.52.Final |
| io.netty | netty-codec-redis | 4.1.52.Final |
| io.netty | netty-codec-smtp | 4.1.52.Final |
| io.netty | netty-codec-socks | 4.1.52.Final |
| io.netty | netty-codec-stomp | 4.1.52.Final |
| io.netty | netty-codec-xml | 4.1.52.Final |
| io.netty | netty-common | 4.1.52.Final |
| io.netty | netty-dev-tools | 4.1.52.Final |
| io.netty | netty-example | 4.1.52.Final |
| io.netty | netty-handler | 4.1.52.Final |
| io.netty | netty-handler-proxy | 4.1.52.Final |
| io.netty | netty-resolver | 4.1.52.Final |
| io.netty | netty-resolver-dns | 4.1.52.Final |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| io.netty | netty-resolver-dns-native- macos | 4.1.52.Final |
| io.netty | netty-tcnative-boringssl- static | 2.0.34.Final |
| io.netty | netty-transport | 4.1.52.Final |
| io.netty | netty-transport-native-epoll | 4.1.52.Final |
| io.netty | netty-transport-native-kqueue | 4.1.52.Final |
| io.netty | netty-transport-native-unix- common | 4.1.52.Final |
| io.netty | netty-transport-rxtx | 4.1.52.Final |
| io.netty | netty-transport-sctp | 4.1.52.Final |
| io.netty | netty-transport-udt | 4.1.52.Final |
| io.projectreactor | reactor-core | 3.3.10.RELEASE |
| io.projectreactor | reactor-test | 3.3.10.RELEASE |
| io.projectreactor | reactor-tools | 3.3.10.RELEASE |
| io.projectreactor.addons | reactor-adapter | 3.3.4.RELEASE |
| io.projectreactor.addons | reactor-extra | 3.3.4.RELEASE |
| io.projectreactor.addons | reactor-pool | 0.1.6.RELEASE |
| io.projectreactor.kafka | reactor-kafka | 1.2.2.RELEASE |
| io.projectreactor.kotlin | reactor-kotlin-extensions | 1.0.1.RELEASE |
| io.projectreactor.netty | reactor-netty | 0.9.12.RELEASE |
| io.projectreactor.rabbitmq | reactor-rabbitmq | 1.4.4.RELEASE |
| io.prometheus | simpleclient\_pushgateway | 0.9.0 |
| io.r2dbc | r2dbc-h2 | 0.8.4.RELEASE |
| io.r2dbc | r2dbc-mssql | 0.8.4.RELEASE |
| io.r2dbc | r2dbc-pool | 0.8.4.RELEASE |
| io.r2dbc | r2dbc-postgresql | 0.8.5.RELEASE |
| io.r2dbc | r2dbc-proxy | 0.8.2.RELEASE |
| io.r2dbc | r2dbc-spi | 0.8.2.RELEASE |
| io.reactivex | rxjava | 1.3.8 |
| io.reactivex | rxjava-reactive-streams | 1.2.1 |
| io.reactivex.rxjava2 | rxjava | 2.2.19 |
| io.rest-assured | json-path | 3.3.0 |
| io.rest-assured | json-schema-validator | 3.3.0 |
| io.rest-assured | rest-assured | 3.3.0 |
| io.rest-assured | scala-support | 3.3.0 |
| io.rest-assured | spring-mock-mvc | 3.3.0 |
| io.rest-assured | spring-web-test-client | 3.3.0 |
| io.rest-assured | xml-path | 3.3.0 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| io.rsocket | rsocket-core | 1.0.2 |
| io.rsocket | rsocket-load-balancer | 1.0.2 |
| io.rsocket | rsocket-micrometer | 1.0.2 |
| io.rsocket | rsocket-test | 1.0.2 |
| io.rsocket | rsocket-transport-local | 1.0.2 |
| io.rsocket | rsocket-transport-netty | 1.0.2 |
| io.spring.gradle | dependency-management-plugin | 1.0.10.RELEASE |
| io.undertow | undertow-core | 2.1.4.Final |
| io.undertow | undertow-servlet | 2.1.4.Final |
| io.undertow | undertow-websockets-jsr | 2.1.4.Final |
| jakarta.activation | jakarta.activation-api | 1.2.2 |
| jakarta.annotation | jakarta.annotation-api | 1.3.5 |
| jakarta.jms | jakarta.jms-api | 2.0.3 |
| jakarta.json | jakarta.json-api | 1.1.6 |
| jakarta.json.bind | jakarta.json.bind-api | 1.0.2 |
| jakarta.mail | jakarta.mail-api | 1.6.5 |
| jakarta.persistence | jakarta.persistence-api | 2.2.3 |
| jakarta.servlet | jakarta.servlet-api | 4.0.4 |
| jakarta.servlet.jsp.jstl | jakarta.servlet.jsp.jstl-api | 1.2.7 |
| jakarta.transaction | jakarta.transaction-api | 1.3.3 |
| jakarta.validation | jakarta.validation-api | 2.0.2 |
| jakarta.websocket | jakarta.websocket-api | 1.1.2 |
| jakarta.ws.rs | jakarta.ws.rs-api | 2.1.6 |
| jakarta.xml.bind | jakarta.xml.bind-api | 2.3.3 |
| jakarta.xml.soap | jakarta.xml.soap-api | 1.4.2 |
| jakarta.xml.ws | jakarta.xml.ws-api | 2.3.3 |
| javax.activation | javax.activation-api | 1.2.0 |
| javax.annotation | javax.annotation-api | 1.3.2 |
| javax.cache | cache-api | 1.1.1 |
| javax.jms | javax.jms-api | 2.0.1 |
| javax.json | javax.json-api | 1.1.4 |
| javax.json.bind | javax.json.bind-api | 1.0 |
| javax.mail | javax.mail-api | 1.6.2 |
| javax.money | money-api | 1.0.3 |
| javax.persistence | javax.persistence-api | 2.2 |
| javax.servlet | javax.servlet-api | 4.0.1 |
| javax.servlet | jstl | 1.2 |
| javax.transaction | javax.transaction-api | 1.3 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| javax.validation | validation-api | 2.0.1.Final |
| javax.websocket | javax.websocket-api | 1.1 |
| javax.xml.bind | jaxb-api | 2.3.1 |
| javax.xml.ws | jaxws-api | 2.3.1 |
| jaxen | jaxen | 1.2.0 |
| junit | junit | 4.12 |
| mysql | mysql-connector-java | 8.0.21 |
| net.bytebuddy | byte-buddy | 1.10.14 |
| net.bytebuddy | byte-buddy-agent | 1.10.14 |
| net.minidev | json-smart | 2.3 |
| net.sf.ehcache | ehcache | 2.10.6 |
| net.sourceforge.htmlunit | htmlunit | 2.40.0 |
| net.sourceforge.jtds | jtds | 1.3.1 |
| net.sourceforge.nekohtml | nekohtml | 1.9.22 |
| nz.net.ultraq.thymeleaf | thymeleaf-layout-dialect | 2.4.1 |
| org.apache.activemq | activemq-amqp | 5.15.13 |
| org.apache.activemq | activemq-blueprint | 5.15.13 |
| org.apache.activemq | activemq-broker | 5.15.13 |
| org.apache.activemq | activemq-camel | 5.15.13 |
| org.apache.activemq | activemq-client | 5.15.13 |
| org.apache.activemq | activemq-console | 5.15.13 |
| org.apache.activemq | activemq-http | 5.15.13 |
| org.apache.activemq | activemq-jaas | 5.15.13 |
| org.apache.activemq | activemq-jdbc-store | 5.15.13 |
| org.apache.activemq | activemq-jms-pool | 5.15.13 |
| org.apache.activemq | activemq-kahadb-store | 5.15.13 |
| org.apache.activemq | activemq-karaf | 5.15.13 |
| org.apache.activemq | activemq-leveldb-store | 5.15.13 |
| org.apache.activemq | activemq-log4j-appender | 5.15.13 |
| org.apache.activemq | activemq-mqtt | 5.15.13 |
| org.apache.activemq | activemq-openwire-generator | 5.15.13 |
| org.apache.activemq | activemq-openwire-legacy | 5.15.13 |
| org.apache.activemq | activemq-osgi | 5.15.13 |
| org.apache.activemq | activemq-partition | 5.15.13 |
| org.apache.activemq | activemq-pool | 5.15.13 |
| org.apache.activemq | activemq-ra | 5.15.13 |
| org.apache.activemq | activemq-run | 5.15.13 |
| org.apache.activemq | activemq-runtime-config | 5.15.13 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.apache.activemq | activemq-shiro | 5.15.13 |
| org.apache.activemq | activemq-spring | 5.15.13 |
| org.apache.activemq | activemq-stomp | 5.15.13 |
| org.apache.activemq | activemq-web | 5.15.13 |
| org.apache.activemq | artemis-amqp-protocol | 2.12.0 |
| org.apache.activemq | artemis-commons | 2.12.0 |
| org.apache.activemq | artemis-core-client | 2.12.0 |
| org.apache.activemq | artemis-jms-client | 2.12.0 |
| org.apache.activemq | artemis-jms-server | 2.12.0 |
| org.apache.activemq | artemis-journal | 2.12.0 |
| org.apache.activemq | artemis-selector | 2.12.0 |
| org.apache.activemq | artemis-server | 2.12.0 |
| org.apache.activemq | artemis-service-extensions | 2.12.0 |
| org.apache.commons | commons-dbcp2 | 2.7.0 |
| org.apache.commons | commons-lang3 | 3.10 |
| org.apache.commons | commons-pool2 | 2.8.1 |
| org.apache.derby | derby | 10.14.2.0 |
| org.apache.derby | derbyclient | 10.14.2.0 |
| org.apache.httpcomponents | fluent-hc | 4.5.12 |
| org.apache.httpcomponents | httpasyncclient | 4.1.4 |
| org.apache.httpcomponents | httpclient | 4.5.12 |
| org.apache.httpcomponents | httpclient-cache | 4.5.12 |
| org.apache.httpcomponents | httpclient-osgi | 4.5.12 |
| org.apache.httpcomponents | httpclient-win | 4.5.12 |
| org.apache.httpcomponents | httpcore | 4.4.13 |
| org.apache.httpcomponents | httpcore-nio | 4.4.13 |
| org.apache.httpcomponents | httpmime | 4.5.12 |
| org.apache.johnzon | johnzon-core | 1.2.8 |
| org.apache.johnzon | johnzon-jaxrs | 1.2.8 |
| org.apache.johnzon | johnzon-jsonb | 1.2.8 |
| org.apache.johnzon | johnzon-jsonb-extras | 1.2.8 |
| org.apache.johnzon | johnzon-jsonschema | 1.2.8 |
| org.apache.johnzon | johnzon-mapper | 1.2.8 |
| org.apache.johnzon | johnzon-websocket | 1.2.8 |
| org.apache.kafka | connect-api | 2.5.1 |
| org.apache.kafka | connect-basic-auth-extension | 2.5.1 |
| org.apache.kafka | connect-file | 2.5.1 |
| org.apache.kafka | connect-json | 2.5.1 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.apache.kafka | connect-runtime | 2.5.1 |
| org.apache.kafka | connect-transforms | 2.5.1 |
| org.apache.kafka | kafka-clients | 2.5.1 |
| org.apache.kafka | kafka-log4j-appender | 2.5.1 |
| org.apache.kafka | kafka-streams | 2.5.1 |
| org.apache.kafka | kafka-streams-scala\_2.12 | 2.5.1 |
| org.apache.kafka | kafka-streams-scala\_2.13 | 2.5.1 |
| org.apache.kafka | kafka-streams-test-utils | 2.5.1 |
| org.apache.kafka | kafka-tools | 2.5.1 |
| org.apache.kafka | kafka\_2.12 | 2.5.1 |
| org.apache.kafka | kafka\_2.13 | 2.5.1 |
| org.apache.logging.log4j | log4j-1.2-api | 2.13.3 |
| org.apache.logging.log4j | log4j-api | 2.13.3 |
| org.apache.logging.log4j | log4j-appserver | 2.13.3 |
| org.apache.logging.log4j | log4j-cassandra | 2.13.3 |
| org.apache.logging.log4j | log4j-core | 2.13.3 |
| org.apache.logging.log4j | log4j-couchdb | 2.13.3 |
| org.apache.logging.log4j | log4j-docker | 2.13.3 |
| org.apache.logging.log4j | log4j-flume-ng | 2.13.3 |
| org.apache.logging.log4j | log4j-iostreams | 2.13.3 |
| org.apache.logging.log4j | log4j-jcl | 2.13.3 |
| org.apache.logging.log4j | log4j-jmx-gui | 2.13.3 |
| org.apache.logging.log4j | log4j-jpa | 2.13.3 |
| org.apache.logging.log4j | log4j-jpl | 2.13.3 |
| org.apache.logging.log4j | log4j-jul | 2.13.3 |
| org.apache.logging.log4j | log4j-kubernetes | 2.13.3 |
| org.apache.logging.log4j | log4j-liquibase | 2.13.3 |
| org.apache.logging.log4j | log4j-mongodb2 | 2.13.3 |
| org.apache.logging.log4j | log4j-mongodb3 | 2.13.3 |
| org.apache.logging.log4j | log4j-slf4j-impl | 2.13.3 |
| org.apache.logging.log4j | log4j-slf4j18-impl | 2.13.3 |
| org.apache.logging.log4j | log4j-spring-cloud-config- client | 2.13.3 |
| org.apache.logging.log4j | log4j-taglib | 2.13.3 |
| org.apache.logging.log4j | log4j-to-slf4j | 2.13.3 |
| org.apache.logging.log4j | log4j-web | 2.13.3 |
| org.apache.solr | solr-analysis-extras | 8.5.2 |
| org.apache.solr | solr-analytics | 8.5.2 |
| org.apache.solr | solr-cell | 8.5.2 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.apache.solr | solr-clustering | 8.5.2 |
| org.apache.solr | solr-core | 8.5.2 |
| org.apache.solr | solr-dataimporthandler | 8.5.2 |
| org.apache.solr | solr-dataimporthandler-extras | 8.5.2 |
| org.apache.solr | solr-langid | 8.5.2 |
| org.apache.solr | solr-ltr | 8.5.2 |
| org.apache.solr | solr-solrj | 8.5.2 |
| org.apache.solr | solr-test-framework | 8.5.2 |
| org.apache.solr | solr-velocity | 8.5.2 |
| org.apache.tomcat | tomcat-annotations-api | 9.0.38 |
| org.apache.tomcat | tomcat-jdbc | 9.0.38 |
| org.apache.tomcat | tomcat-jsp-api | 9.0.38 |
| org.apache.tomcat.embed | tomcat-embed-core | 9.0.38 |
| org.apache.tomcat.embed | tomcat-embed-el | 9.0.38 |
| org.apache.tomcat.embed | tomcat-embed-jasper | 9.0.38 |
| org.apache.tomcat.embed | tomcat-embed-websocket | 9.0.38 |
| org.aspectj | aspectjrt | 1.9.6 |
| org.aspectj | aspectjtools | 1.9.6 |
| org.aspectj | aspectjweaver | 1.9.6 |
| org.assertj | assertj-core | 3.16.1 |
| org.awaitility | awaitility | 4.0.3 |
| org.awaitility | awaitility-groovy | 4.0.3 |
| org.awaitility | awaitility-kotlin | 4.0.3 |
| org.awaitility | awaitility-scala | 4.0.3 |
| org.codehaus.btm | btm | 2.1.4 |
| org.codehaus.groovy | groovy | 2.5.13 |
| org.codehaus.groovy | groovy-ant | 2.5.13 |
| org.codehaus.groovy | groovy-bsf | 2.5.13 |
| org.codehaus.groovy | groovy-cli-commons | 2.5.13 |
| org.codehaus.groovy | groovy-cli-picocli | 2.5.13 |
| org.codehaus.groovy | groovy-console | 2.5.13 |
| org.codehaus.groovy | groovy-datetime | 2.5.13 |
| org.codehaus.groovy | groovy-dateutil | 2.5.13 |
| org.codehaus.groovy | groovy-docgenerator | 2.5.13 |
| org.codehaus.groovy | groovy-groovydoc | 2.5.13 |
| org.codehaus.groovy | groovy-groovysh | 2.5.13 |
| org.codehaus.groovy | groovy-jaxb | 2.5.13 |
| org.codehaus.groovy | groovy-jmx | 2.5.13 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.codehaus.groovy | groovy-json | 2.5.13 |
| org.codehaus.groovy | groovy-json-direct | 2.5.13 |
| org.codehaus.groovy | groovy-jsr223 | 2.5.13 |
| org.codehaus.groovy | groovy-macro | 2.5.13 |
| org.codehaus.groovy | groovy-nio | 2.5.13 |
| org.codehaus.groovy | groovy-servlet | 2.5.13 |
| org.codehaus.groovy | groovy-sql | 2.5.13 |
| org.codehaus.groovy | groovy-swing | 2.5.13 |
| org.codehaus.groovy | groovy-templates | 2.5.13 |
| org.codehaus.groovy | groovy-test | 2.5.13 |
| org.codehaus.groovy | groovy-test-junit5 | 2.5.13 |
| org.codehaus.groovy | groovy-testng | 2.5.13 |
| org.codehaus.groovy | groovy-xml | 2.5.13 |
| org.codehaus.janino | commons-compiler | 3.1.2 |
| org.codehaus.janino | commons-compiler-jdk | 3.1.2 |
| org.codehaus.janino | janino | 3.1.2 |
| org.eclipse.jetty | apache-jsp | 9.4.31.v20200723 |
| org.eclipse.jetty | apache-jstl | 9.4.31.v20200723 |
| org.eclipse.jetty | infinispan-common | 9.4.31.v20200723 |
| org.eclipse.jetty | infinispan-embedded-query | 9.4.31.v20200723 |
| org.eclipse.jetty | infinispan-remote-query | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-alpn-client | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-alpn-conscrypt-client | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-alpn-conscrypt-server | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-alpn-java-client | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-alpn-java-server | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-alpn-openjdk8-client | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-alpn-openjdk8-server | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-alpn-server | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-annotations | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-ant | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-client | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-continuation | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-deploy | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-distribution | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-hazelcast | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-home | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-http | 9.4.31.v20200723 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.eclipse.jetty | jetty-http-spi | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-io | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-jaas | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-jaspi | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-jmx | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-jndi | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-nosql | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-openid | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-plus | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-proxy | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-quickstart | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-reactive-httpclient | 1.1.4 |
| org.eclipse.jetty | jetty-rewrite | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-security | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-server | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-servlet | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-servlets | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-spring | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-unixsocket | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-util | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-util-ajax | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-webapp | 9.4.31.v20200723 |
| org.eclipse.jetty | jetty-xml | 9.4.31.v20200723 |
| org.eclipse.jetty.fcgi | fcgi-client | 9.4.31.v20200723 |
| org.eclipse.jetty.fcgi | fcgi-server | 9.4.31.v20200723 |
| org.eclipse.jetty.gcloud | jetty-gcloud-session-manager | 9.4.31.v20200723 |
| org.eclipse.jetty.http2 | http2-client | 9.4.31.v20200723 |
| org.eclipse.jetty.http2 | http2-common | 9.4.31.v20200723 |
| org.eclipse.jetty.http2 | http2-hpack | 9.4.31.v20200723 |
| org.eclipse.jetty.http2 | http2-http-client-transport | 9.4.31.v20200723 |
| org.eclipse.jetty.http2 | http2-server | 9.4.31.v20200723 |
| org.eclipse.jetty.memcached | jetty-memcached-sessions | 9.4.31.v20200723 |
| org.eclipse.jetty.orbit | javax.servlet.jsp | 2.2.0.v201112011158 |
| org.eclipse.jetty.osgi | jetty-httpservice | 9.4.31.v20200723 |
| org.eclipse.jetty.osgi | jetty-osgi-boot | 9.4.31.v20200723 |
| org.eclipse.jetty.osgi | jetty-osgi-boot-jsp | 9.4.31.v20200723 |
| org.eclipse.jetty.osgi | jetty-osgi-boot-warurl | 9.4.31.v20200723 |
| org.eclipse.jetty.websocket | javax-websocket-client-impl | 9.4.31.v20200723 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.eclipse.jetty.websocket | javax-websocket-server-impl | 9.4.31.v20200723 |
| org.eclipse.jetty.websocket | websocket-api | 9.4.31.v20200723 |
| org.eclipse.jetty.websocket | websocket-client | 9.4.31.v20200723 |
| org.eclipse.jetty.websocket | websocket-common | 9.4.31.v20200723 |
| org.eclipse.jetty.websocket | websocket-server | 9.4.31.v20200723 |
| org.eclipse.jetty.websocket | websocket-servlet | 9.4.31.v20200723 |
| org.ehcache | ehcache | 3.8.1 |
| org.ehcache | ehcache-clustered | 3.8.1 |
| org.ehcache | ehcache-transactions | 3.8.1 |
| org.elasticsearch | elasticsearch | 7.6.2 |
| org.elasticsearch.client | elasticsearch-rest-client | 7.6.2 |
| org.elasticsearch.client | elasticsearch-rest-high-level- client | 7.6.2 |
| org.elasticsearch.client | transport | 7.6.2 |
| org.elasticsearch.distribution  .integ-test-zip | elasticsearch | 7.6.2 |
| org.elasticsearch.plugin | transport-netty4-client | 7.6.2 |
| org.firebirdsql.jdbc | jaybird-jdk17 | 3.0.9 |
| org.firebirdsql.jdbc | jaybird-jdk18 | 3.0.9 |
| org.flywaydb | flyway-core | 6.4.4 |
| org.freemarker | freemarker | 2.3.30 |
| org.glassfish | jakarta.el | 3.0.3 |
| org.glassfish.jaxb | codemodel | 2.3.3 |
| org.glassfish.jaxb | codemodel-annotation-compiler | 2.3.3 |
| org.glassfish.jaxb | jaxb-jxc | 2.3.3 |
| org.glassfish.jaxb | jaxb-runtime | 2.3.3 |
| org.glassfish.jaxb | jaxb-xjc | 2.3.3 |
| org.glassfish.jaxb | txw2 | 2.3.3 |
| org.glassfish.jaxb | txwc2 | 2.3.3 |
| org.glassfish.jaxb | xsom | 2.3.3 |
| org.glassfish.jersey.bundles | jaxrs-ri | 2.30.1 |
| org.glassfish.jersey.connector s | jersey-apache-connector | 2.30.1 |
| org.glassfish.jersey.connector s | jersey-grizzly-connector | 2.30.1 |
| org.glassfish.jersey.connector s | jersey-jdk-connector | 2.30.1 |
| org.glassfish.jersey.connector s | jersey-jetty-connector | 2.30.1 |
| org.glassfish.jersey.connector s | jersey-netty-connector | 2.30.1 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.glassfish.jersey.container s | jersey-container-grizzly2-http | 2.30.1 |
| org.glassfish.jersey.container s | jersey-container-grizzly2- servlet | 2.30.1 |
| org.glassfish.jersey.container s | jersey-container-jdk-http | 2.30.1 |
| org.glassfish.jersey.container s | jersey-container-jetty-http | 2.30.1 |
| org.glassfish.jersey.container s | jersey-container-jetty-servlet | 2.30.1 |
| org.glassfish.jersey.container s | jersey-container-netty-http | 2.30.1 |
| org.glassfish.jersey.container s | jersey-container-servlet | 2.30.1 |
| org.glassfish.jersey.container s | jersey-container-servlet-core | 2.30.1 |
| org.glassfish.jersey.container s | jersey-container-simple-http | 2.30.1 |
| org.glassfish.jersey.container s.glassfish | jersey-gf-ejb | 2.30.1 |
| org.glassfish.jersey.core | jersey-client | 2.30.1 |
| org.glassfish.jersey.core | jersey-common | 2.30.1 |
| org.glassfish.jersey.core | jersey-server | 2.30.1 |
| org.glassfish.jersey.ext | jersey-bean-validation | 2.30.1 |
| org.glassfish.jersey.ext | jersey-declarative-linking | 2.30.1 |
| org.glassfish.jersey.ext | jersey-entity-filtering | 2.30.1 |
| org.glassfish.jersey.ext | jersey-metainf-services | 2.30.1 |
| org.glassfish.jersey.ext | jersey-mvc | 2.30.1 |
| org.glassfish.jersey.ext | jersey-mvc-bean-validation | 2.30.1 |
| org.glassfish.jersey.ext | jersey-mvc-freemarker | 2.30.1 |
| org.glassfish.jersey.ext | jersey-mvc-jsp | 2.30.1 |
| org.glassfish.jersey.ext | jersey-mvc-mustache | 2.30.1 |
| org.glassfish.jersey.ext | jersey-proxy-client | 2.30.1 |
| org.glassfish.jersey.ext | jersey-servlet-portability | 2.30.1 |
| org.glassfish.jersey.ext | jersey-spring4 | 2.30.1 |
| org.glassfish.jersey.ext | jersey-spring5 | 2.30.1 |
| org.glassfish.jersey.ext | jersey-wadl-doclet | 2.30.1 |
| org.glassfish.jersey.ext.cdi | jersey-cdi1x | 2.30.1 |
| org.glassfish.jersey.ext.cdi | jersey-cdi1x-ban-custom-hk2- binding | 2.30.1 |
| org.glassfish.jersey.ext.cdi | jersey-cdi1x-servlet | 2.30.1 |
| org.glassfish.jersey.ext.cdi | jersey-cdi1x-transaction | 2.30.1 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.glassfish.jersey.ext.cdi | jersey-cdi1x-validation | 2.30.1 |
| org.glassfish.jersey.ext.cdi | jersey-weld2-se | 2.30.1 |
| org.glassfish.jersey.ext.micro profile | jersey-mp-config | 2.30.1 |
| org.glassfish.jersey.ext.micro profile | jersey-mp-rest-client | 2.30.1 |
| org.glassfish.jersey.ext.rx | jersey-rx-client-guava | 2.30.1 |
| org.glassfish.jersey.ext.rx | jersey-rx-client-rxjava | 2.30.1 |
| org.glassfish.jersey.ext.rx | jersey-rx-client-rxjava2 | 2.30.1 |
| org.glassfish.jersey.inject | jersey-cdi2-se | 2.30.1 |
| org.glassfish.jersey.inject | jersey-hk2 | 2.30.1 |
| org.glassfish.jersey.media | jersey-media-jaxb | 2.30.1 |
| org.glassfish.jersey.media | jersey-media-json-binding | 2.30.1 |
| org.glassfish.jersey.media | jersey-media-json-jackson | 2.30.1 |
| org.glassfish.jersey.media | jersey-media-json-jackson1 | 2.30.1 |
| org.glassfish.jersey.media | jersey-media-json-jettison | 2.30.1 |
| org.glassfish.jersey.media | jersey-media-json-processing | 2.30.1 |
| org.glassfish.jersey.media | jersey-media-kryo | 2.30.1 |
| org.glassfish.jersey.media | jersey-media-moxy | 2.30.1 |
| org.glassfish.jersey.media | jersey-media-multipart | 2.30.1 |
| org.glassfish.jersey.media | jersey-media-sse | 2.30.1 |
| org.glassfish.jersey.security | oauth1-client | 2.30.1 |
| org.glassfish.jersey.security | oauth1-server | 2.30.1 |
| org.glassfish.jersey.security | oauth1-signature | 2.30.1 |
| org.glassfish.jersey.security | oauth2-client | 2.30.1 |
| org.glassfish.jersey.test- framework | jersey-test-framework-core | 2.30.1 |
| org.glassfish.jersey.test- framework | jersey-test-framework-util | 2.30.1 |
| org.glassfish.jersey.test- framework.providers | jersey-test-framework- provider-bundle | 2.30.1 |
| org.glassfish.jersey.test- framework.providers | jersey-test-framework- provider-external | 2.30.1 |
| org.glassfish.jersey.test- framework.providers | jersey-test-framework- provider-grizzly2 | 2.30.1 |
| org.glassfish.jersey.test- framework.providers | jersey-test-framework- provider-inmemory | 2.30.1 |
| org.glassfish.jersey.test- framework.providers | jersey-test-framework- provider-jdk-http | 2.30.1 |
| org.glassfish.jersey.test- framework.providers | jersey-test-framework- provider-jetty | 2.30.1 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.glassfish.jersey.test- framework.providers | jersey-test-framework- provider-simple | 2.30.1 |
| org.hamcrest | hamcrest | 2.2 |
| org.hamcrest | hamcrest-core | 2.2 |
| org.hamcrest | hamcrest-library | 2.2 |
| org.hibernate | hibernate-c3p0 | 5.4.21.Final |
| org.hibernate | hibernate-core | 5.4.21.Final |
| org.hibernate | hibernate-ehcache | 5.4.21.Final |
| org.hibernate | hibernate-entitymanager | 5.4.21.Final |
| org.hibernate | hibernate-envers | 5.4.21.Final |
| org.hibernate | hibernate-hikaricp | 5.4.21.Final |
| org.hibernate | hibernate-java8 | 5.4.21.Final |
| org.hibernate | hibernate-jcache | 5.4.21.Final |
| org.hibernate | hibernate-jpamodelgen | 5.4.21.Final |
| org.hibernate | hibernate-proxool | 5.4.21.Final |
| org.hibernate | hibernate-spatial | 5.4.21.Final |
| org.hibernate | hibernate-testing | 5.4.21.Final |
| org.hibernate | hibernate-vibur | 5.4.21.Final |
| org.hibernate.validator | hibernate-validator | 6.1.5.Final |
| org.hibernate.validator | hibernate-validator- annotation-processor | 6.1.5.Final |
| org.hsqldb | hsqldb | 2.5.1 |
| org.infinispan | infinispan-api | 10.1.8.Final |
| org.infinispan | infinispan-cachestore-jdbc | 10.1.8.Final |
| org.infinispan | infinispan-cachestore-jpa | 10.1.8.Final |
| org.infinispan | infinispan-cachestore-remote | 10.1.8.Final |
| org.infinispan | infinispan-cachestore-rest | 10.1.8.Final |
| org.infinispan | infinispan-cachestore-rocksdb | 10.1.8.Final |
| org.infinispan | infinispan-cdi-common | 10.1.8.Final |
| org.infinispan | infinispan-cdi-embedded | 10.1.8.Final |
| org.infinispan | infinispan-cdi-remote | 10.1.8.Final |
| org.infinispan | infinispan-client-hotrod | 10.1.8.Final |
| org.infinispan | infinispan-client-rest | 10.1.8.Final |
| org.infinispan | infinispan-clustered-counter | 10.1.8.Final |
| org.infinispan | infinispan-clustered-lock | 10.1.8.Final |
| org.infinispan | infinispan-commons | 10.1.8.Final |
| org.infinispan | infinispan-component- annotations | 10.1.8.Final |
| org.infinispan | infinispan-core | 10.1.8.Final |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.infinispan | infinispan-directory-provider | 10.1.8.Final |
| org.infinispan | infinispan-hibernate-cache-v53 | 10.1.8.Final |
| org.infinispan | infinispan-jboss-marshalling | 10.1.8.Final |
| org.infinispan | infinispan-jcache | 10.1.8.Final |
| org.infinispan | infinispan-jcache-commons | 10.1.8.Final |
| org.infinispan | infinispan-jcache-remote | 10.1.8.Final |
| org.infinispan | infinispan-key-value-store- client | 10.1.8.Final |
| org.infinispan | infinispan-lucene-directory | 10.1.8.Final |
| org.infinispan | infinispan-objectfilter | 10.1.8.Final |
| org.infinispan | infinispan-osgi | 10.1.8.Final |
| org.infinispan | infinispan-persistence-soft- index | 10.1.8.Final |
| org.infinispan | infinispan-query | 10.1.8.Final |
| org.infinispan | infinispan-query-core | 10.1.8.Final |
| org.infinispan | infinispan-query-dsl | 10.1.8.Final |
| org.infinispan | infinispan-remote-query-client | 10.1.8.Final |
| org.infinispan | infinispan-remote-query-server | 10.1.8.Final |
| org.infinispan | infinispan-scripting | 10.1.8.Final |
| org.infinispan | infinispan-server-core | 10.1.8.Final |
| org.infinispan | infinispan-server-hotrod | 10.1.8.Final |
| org.infinispan | infinispan-server-memcached | 10.1.8.Final |
| org.infinispan | infinispan-server-rest | 10.1.8.Final |
| org.infinispan | infinispan-server-router | 10.1.8.Final |
| org.infinispan | infinispan-spring5-common | 10.1.8.Final |
| org.infinispan | infinispan-spring5-embedded | 10.1.8.Final |
| org.infinispan | infinispan-spring5-remote | 10.1.8.Final |
| org.infinispan | infinispan-tasks | 10.1.8.Final |
| org.infinispan | infinispan-tasks-api | 10.1.8.Final |
| org.infinispan | infinispan-tools | 10.1.8.Final |
| org.influxdb | influxdb-java | 2.18 |
| org.jboss | jboss-transaction-spi | 7.6.0.Final |
| org.jboss.logging | jboss-logging | 3.4.1.Final |
| org.jdom | jdom2 | 2.0.6 |
| org.jetbrains.kotlin | kotlin-compiler | 1.3.72 |
| org.jetbrains.kotlin | kotlin-compiler-embeddable | 1.3.72 |
| org.jetbrains.kotlin | kotlin-daemon-client | 1.3.72 |
| org.jetbrains.kotlin | kotlin-main-kts | 1.3.72 |
| org.jetbrains.kotlin | kotlin-osgi-bundle | 1.3.72 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.jetbrains.kotlin | kotlin-reflect | 1.3.72 |
| org.jetbrains.kotlin | kotlin-script-runtime | 1.3.72 |
| org.jetbrains.kotlin | kotlin-script-util | 1.3.72 |
| org.jetbrains.kotlin | kotlin-scripting-common | 1.3.72 |
| org.jetbrains.kotlin | kotlin-scripting-jvm | 1.3.72 |
| org.jetbrains.kotlin | kotlin-scripting-jvm-host | 1.3.72 |
| org.jetbrains.kotlin | kotlin-stdlib | 1.3.72 |
| org.jetbrains.kotlin | kotlin-stdlib-common | 1.3.72 |
| org.jetbrains.kotlin | kotlin-stdlib-jdk7 | 1.3.72 |
| org.jetbrains.kotlin | kotlin-stdlib-jdk8 | 1.3.72 |
| org.jetbrains.kotlin | kotlin-stdlib-js | 1.3.72 |
| org.jetbrains.kotlin | kotlin-test | 1.3.72 |
| org.jetbrains.kotlin | kotlin-test-annotations-common | 1.3.72 |
| org.jetbrains.kotlin | kotlin-test-common | 1.3.72 |
| org.jetbrains.kotlin | kotlin-test-js | 1.3.72 |
| org.jetbrains.kotlin | kotlin-test-junit | 1.3.72 |
| org.jetbrains.kotlin | kotlin-test-junit5 | 1.3.72 |
| org.jetbrains.kotlin | kotlin-test-testng | 1.3.72 |
| org.jetbrains.kotlinx | kotlinx-coroutines-android | 1.3.8 |
| org.jetbrains.kotlinx | kotlinx-coroutines-core | 1.3.8 |
| org.jetbrains.kotlinx | kotlinx-coroutines-core-common | 1.3.8 |
| org.jetbrains.kotlinx | kotlinx-coroutines-core-js | 1.3.8 |
| org.jetbrains.kotlinx | kotlinx-coroutines-core-native | 1.3.8 |
| org.jetbrains.kotlinx | kotlinx-coroutines-debug | 1.3.8 |
| org.jetbrains.kotlinx | kotlinx-coroutines-guava | 1.3.8 |
| org.jetbrains.kotlinx | kotlinx-coroutines-javafx | 1.3.8 |
| org.jetbrains.kotlinx | kotlinx-coroutines-jdk8 | 1.3.8 |
| org.jetbrains.kotlinx | kotlinx-coroutines-jdk9 | 1.3.8 |
| org.jetbrains.kotlinx | kotlinx-coroutines-play- services | 1.3.8 |
| org.jetbrains.kotlinx | kotlinx-coroutines-reactive | 1.3.8 |
| org.jetbrains.kotlinx | kotlinx-coroutines-reactor | 1.3.8 |
| org.jetbrains.kotlinx | kotlinx-coroutines-rx2 | 1.3.8 |
| org.jetbrains.kotlinx | kotlinx-coroutines-rx3 | 1.3.8 |
| org.jetbrains.kotlinx | kotlinx-coroutines-slf4j | 1.3.8 |
| org.jetbrains.kotlinx | kotlinx-coroutines-swing | 1.3.8 |
| org.jetbrains.kotlinx | kotlinx-coroutines-test | 1.3.8 |
| org.jolokia | jolokia-core | 1.6.2 |
| org.jooq | jooq | 3.13.4 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.jooq | jooq-codegen | 3.13.4 |
| org.jooq | jooq-meta | 3.13.4 |
| org.junit.jupiter | junit-jupiter | 5.6.2 |
| org.junit.jupiter | junit-jupiter-api | 5.6.2 |
| org.junit.jupiter | junit-jupiter-engine | 5.6.2 |
| org.junit.jupiter | junit-jupiter-migrationsupport | 5.6.2 |
| org.junit.jupiter | junit-jupiter-params | 5.6.2 |
| org.junit.platform | junit-platform-commons | 1.6.2 |
| org.junit.platform | junit-platform-console | 1.6.2 |
| org.junit.platform | junit-platform-engine | 1.6.2 |
| org.junit.platform | junit-platform-launcher | 1.6.2 |
| org.junit.platform | junit-platform-reporting | 1.6.2 |
| org.junit.platform | junit-platform-runner | 1.6.2 |
| org.junit.platform | junit-platform-suite-api | 1.6.2 |
| org.junit.platform | junit-platform-testkit | 1.6.2 |
| org.junit.vintage | junit-vintage-engine | 5.6.2 |
| org.jvnet.mimepull | mimepull | 1.9.13 |
| org.liquibase | liquibase-core | 3.8.9 |
| org.mariadb | r2dbc-mariadb | 0.8.3-beta1 |
| org.mariadb.jdbc | mariadb-java-client | 2.6.2 |
| org.messaginghub | pooled-jms | 1.1.2 |
| org.mockito | mockito-core | 3.3.3 |
| org.mockito | mockito-inline | 3.3.3 |
| org.mockito | mockito-junit-jupiter | 3.3.3 |
| org.mongodb | bson | 4.0.5 |
| org.mongodb | mongodb-driver-core | 4.0.5 |
| org.mongodb | mongodb-driver-legacy | 4.0.5 |
| org.mongodb | mongodb-driver-reactivestreams | 4.0.5 |
| org.mongodb | mongodb-driver-sync | 4.0.5 |
| org.mortbay.jasper | apache-el | 8.5.54 |
| org.neo4j | neo4j-ogm-api | 3.2.16 |
| org.neo4j | neo4j-ogm-bolt-driver | 3.2.16 |
| org.neo4j | neo4j-ogm-bolt-native-types | 3.2.16 |
| org.neo4j | neo4j-ogm-core | 3.2.16 |
| org.neo4j | neo4j-ogm-embedded-driver | 3.2.16 |
| org.neo4j | neo4j-ogm-embedded-native- types | 3.2.16 |
| org.neo4j | neo4j-ogm-http-driver | 3.2.16 |
| org.postgresql | postgresql | 42.2.16 |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.projectlombok | lombok | 1.18.12 |
| org.quartz-scheduler | quartz | 2.3.2 |
| org.quartz-scheduler | quartz-jobs | 2.3.2 |
| org.reactivestreams | reactive-streams | 1.0.3 |
| org.seleniumhq.selenium | htmlunit-driver | 2.40.0 |
| org.seleniumhq.selenium | selenium-api | 3.141.59 |
| org.seleniumhq.selenium | selenium-chrome-driver | 3.141.59 |
| org.seleniumhq.selenium | selenium-edge-driver | 3.141.59 |
| org.seleniumhq.selenium | selenium-firefox-driver | 3.141.59 |
| org.seleniumhq.selenium | selenium-ie-driver | 3.141.59 |
| org.seleniumhq.selenium | selenium-java | 3.141.59 |
| org.seleniumhq.selenium | selenium-opera-driver | 3.141.59 |
| org.seleniumhq.selenium | selenium-remote-driver | 3.141.59 |
| org.seleniumhq.selenium | selenium-safari-driver | 3.141.59 |
| org.seleniumhq.selenium | selenium-support | 3.141.59 |
| org.skyscreamer | jsonassert | 1.5.0 |
| org.slf4j | jcl-over-slf4j | 1.7.30 |
| org.slf4j | jul-to-slf4j | 1.7.30 |
| org.slf4j | log4j-over-slf4j | 1.7.30 |
| org.slf4j | slf4j-api | 1.7.30 |
| org.slf4j | slf4j-ext | 1.7.30 |
| org.slf4j | slf4j-jcl | 1.7.30 |
| org.slf4j | slf4j-jdk14 | 1.7.30 |
| org.slf4j | slf4j-log4j12 | 1.7.30 |
| org.slf4j | slf4j-nop | 1.7.30 |
| org.slf4j | slf4j-simple | 1.7.30 |
| org.springframework | spring-aop | 5.2.9.RELEASE |
| org.springframework | spring-aspects | 5.2.9.RELEASE |
| org.springframework | spring-beans | 5.2.9.RELEASE |
| org.springframework | spring-context | 5.2.9.RELEASE |
| org.springframework | spring-context-indexer | 5.2.9.RELEASE |
| org.springframework | spring-context-support | 5.2.9.RELEASE |
| org.springframework | spring-core | 5.2.9.RELEASE |
| org.springframework | spring-expression | 5.2.9.RELEASE |
| org.springframework | spring-instrument | 5.2.9.RELEASE |
| org.springframework | spring-jcl | 5.2.9.RELEASE |
| org.springframework | spring-jdbc | 5.2.9.RELEASE |
| org.springframework | spring-jms | 5.2.9.RELEASE |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.springframework | spring-messaging | 5.2.9.RELEASE |
| org.springframework | spring-orm | 5.2.9.RELEASE |
| org.springframework | spring-oxm | 5.2.9.RELEASE |
| org.springframework | spring-test | 5.2.9.RELEASE |
| org.springframework | spring-tx | 5.2.9.RELEASE |
| org.springframework | spring-web | 5.2.9.RELEASE |
| org.springframework | spring-webflux | 5.2.9.RELEASE |
| org.springframework | spring-webmvc | 5.2.9.RELEASE |
| org.springframework | spring-websocket | 5.2.9.RELEASE |
| org.springframework.amqp | spring-amqp | 2.2.11.RELEASE |
| org.springframework.amqp | spring-rabbit | 2.2.11.RELEASE |
| org.springframework.amqp | spring-rabbit-junit | 2.2.11.RELEASE |
| org.springframework.amqp | spring-rabbit-test | 2.2.11.RELEASE |
| org.springframework.batch | spring-batch-core | 4.2.4.RELEASE |
| org.springframework.batch | spring-batch-infrastructure | 4.2.4.RELEASE |
| org.springframework.batch | spring-batch-integration | 4.2.4.RELEASE |
| org.springframework.batch | spring-batch-test | 4.2.4.RELEASE |
| org.springframework.boot | spring-boot | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-actuator | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-actuator- autoconfigure | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-autoconfigure | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-autoconfigure- processor | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-buildpack-platform | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-configuration- metadata | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-configuration- processor | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-devtools | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-jarmode-layertools | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-loader | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-loader-tools | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-properties- migrator | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-activemq | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-actuator | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-amqp | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-aop | 2.3.4.RELEASE |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.springframework.boot | spring-boot-starter-artemis | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-batch | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-cache | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-data- cassandra | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-data- cassandra-reactive | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-data- couchbase | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-data- couchbase-reactive | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-data- elasticsearch | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-data-jdbc | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-data-jpa | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-data-ldap | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-data- mongodb | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-data- mongodb-reactive | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-data-neo4j | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-data-r2dbc | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-data-redis | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-data- redis-reactive | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-data-rest | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-data-solr | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-freemarker | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-groovy- templates | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-hateoas | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter- integration | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-jdbc | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-jersey | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-jetty | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-jooq | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-json | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-jta- atomikos | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-jta- bitronix | 2.3.4.RELEASE |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.springframework.boot | spring-boot-starter-log4j2 | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-logging | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-mail | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-mustache | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-oauth2- client | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-oauth2- resource-server | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-quartz | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-reactor- netty | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-rsocket | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-security | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-test | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-thymeleaf | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-tomcat | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-undertow | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-validation | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-web | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-web- services | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-webflux | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-starter-websocket | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-test | 2.3.4.RELEASE |
| org.springframework.boot | spring-boot-test-autoconfigure | 2.3.4.RELEASE |
| org.springframework.data | spring-data-cassandra | 3.0.4.RELEASE |
| org.springframework.data | spring-data-commons | 2.3.4.RELEASE |
| org.springframework.data | spring-data-couchbase | 4.0.4.RELEASE |
| org.springframework.data | spring-data-elasticsearch | 4.0.4.RELEASE |
| org.springframework.data | spring-data-envers | 2.3.4.RELEASE |
| org.springframework.data | spring-data-gemfire | 2.3.4.RELEASE |
| org.springframework.data | spring-data-geode | 2.3.4.RELEASE |
| org.springframework.data | spring-data-jdbc | 2.0.4.RELEASE |
| org.springframework.data | spring-data-jpa | 2.3.4.RELEASE |
| org.springframework.data | spring-data-keyvalue | 2.3.4.RELEASE |
| org.springframework.data | spring-data-ldap | 2.3.4.RELEASE |
| org.springframework.data | spring-data-mongodb | 3.0.4.RELEASE |
| org.springframework.data | spring-data-neo4j | 5.3.4.RELEASE |
| org.springframework.data | spring-data-r2dbc | 1.1.4.RELEASE |
| org.springframework.data | spring-data-redis | 2.3.4.RELEASE |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.springframework.data | spring-data-relational | 2.0.4.RELEASE |
| org.springframework.data | spring-data-rest-core | 3.3.4.RELEASE |
| org.springframework.data | spring-data-rest-hal-browser | 3.3.4.RELEASE |
| org.springframework.data | spring-data-rest-hal-explorer | 3.3.4.RELEASE |
| org.springframework.data | spring-data-rest-webmvc | 3.3.4.RELEASE |
| org.springframework.data | spring-data-solr | 4.2.4.RELEASE |
| org.springframework.hateoas | spring-hateoas | 1.1.2.RELEASE |
| org.springframework.integratio n | spring-integration-amqp | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-core | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-event | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-feed | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-file | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-ftp | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-gemfire | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-groovy | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-http | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-ip | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-jdbc | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-jms | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-jmx | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-jpa | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-mail | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-mongodb | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-mqtt | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-redis | 5.3.2.RELEASE |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.springframework.integratio n | spring-integration-rmi | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-rsocket | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-scripting | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-security | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-sftp | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-stomp | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-stream | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-syslog | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-test | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-test- support | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-webflux | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-websocket | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-ws | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-xml | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-xmpp | 5.3.2.RELEASE |
| org.springframework.integratio n | spring-integration-zookeeper | 5.3.2.RELEASE |
| org.springframework.kafka | spring-kafka | 2.5.6.RELEASE |
| org.springframework.kafka | spring-kafka-test | 2.5.6.RELEASE |
| org.springframework.ldap | spring-ldap-core | 2.3.3.RELEASE |
| org.springframework.ldap | spring-ldap-core-tiger | 2.3.3.RELEASE |
| org.springframework.ldap | spring-ldap-ldif-batch | 2.3.3.RELEASE |
| org.springframework.ldap | spring-ldap-ldif-core | 2.3.3.RELEASE |
| org.springframework.ldap | spring-ldap-odm | 2.3.3.RELEASE |
| org.springframework.ldap | spring-ldap-test | 2.3.3.RELEASE |
| org.springframework.restdocs | spring-restdocs-asciidoctor | 2.0.5.RELEASE |
| org.springframework.restdocs | spring-restdocs-core | 2.0.5.RELEASE |
| org.springframework.restdocs | spring-restdocs-mockmvc | 2.0.5.RELEASE |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.springframework.restdocs | spring-restdocs-restassured | 2.0.5.RELEASE |
| org.springframework.restdocs | spring-restdocs-webtestclient | 2.0.5.RELEASE |
| org.springframework.retry | spring-retry | 1.2.5.RELEASE |
| org.springframework.security | spring-security-acl | 5.3.4.RELEASE |
| org.springframework.security | spring-security-aspects | 5.3.4.RELEASE |
| org.springframework.security | spring-security-cas | 5.3.4.RELEASE |
| org.springframework.security | spring-security-config | 5.3.4.RELEASE |
| org.springframework.security | spring-security-core | 5.3.4.RELEASE |
| org.springframework.security | spring-security-crypto | 5.3.4.RELEASE |
| org.springframework.security | spring-security-data | 5.3.4.RELEASE |
| org.springframework.security | spring-security-ldap | 5.3.4.RELEASE |
| org.springframework.security | spring-security-messaging | 5.3.4.RELEASE |
| org.springframework.security | spring-security-oauth2-client | 5.3.4.RELEASE |
| org.springframework.security | spring-security-oauth2-core | 5.3.4.RELEASE |
| org.springframework.security | spring-security-oauth2-jose | 5.3.4.RELEASE |
| org.springframework.security | spring-security-oauth2- resource-server | 5.3.4.RELEASE |
| org.springframework.security | spring-security-openid | 5.3.4.RELEASE |
| org.springframework.security | spring-security-remoting | 5.3.4.RELEASE |
| org.springframework.security | spring-security-rsocket | 5.3.4.RELEASE |
| org.springframework.security | spring-security-saml2-service- provider | 5.3.4.RELEASE |
| org.springframework.security | spring-security-taglibs | 5.3.4.RELEASE |
| org.springframework.security | spring-security-test | 5.3.4.RELEASE |
| org.springframework.security | spring-security-web | 5.3.4.RELEASE |
| org.springframework.session | spring-session-core | 2.3.1.RELEASE |
| org.springframework.session | spring-session-data-geode | 2.3.1.RELEASE |
| org.springframework.session | spring-session-data-mongodb | 2.3.0.RELEASE |
| org.springframework.session | spring-session-data-redis | 2.3.1.RELEASE |
| org.springframework.session | spring-session-hazelcast | 2.3.1.RELEASE |
| org.springframework.session | spring-session-jdbc | 2.3.1.RELEASE |
| org.springframework.ws | spring-ws-core | 3.0.10.RELEASE |
| org.springframework.ws | spring-ws-security | 3.0.10.RELEASE |
| org.springframework.ws | spring-ws-support | 3.0.10.RELEASE |
| org.springframework.ws | spring-ws-test | 3.0.10.RELEASE |
| org.springframework.ws | spring-xml | 3.0.10.RELEASE |
| org.synchronoss.cloud | nio-multipart-parser | 1.1.0 |
| org.thymeleaf | thymeleaf | 3.0.11.RELEASE |
| org.thymeleaf | thymeleaf-spring5 | 3.0.11.RELEASE |

|  |  |  |
| --- | --- | --- |
| **Group ID** | **Artifact ID** | **Version** |
| org.thymeleaf.extras | thymeleaf-extras-java8time | 3.0.4.RELEASE |
| org.thymeleaf.extras | thymeleaf-extras- springsecurity5 | 3.0.4.RELEASE |
| org.webjars | hal-browser | 3325375 |
| org.webjars | webjars-locator-core | 0.45 |
| org.xerial | sqlite-jdbc | 3.31.1 |
| org.xmlunit | xmlunit-assertj | 2.7.0 |
| org.xmlunit | xmlunit-core | 2.7.0 |
| org.xmlunit | xmlunit-legacy | 2.7.0 |
| org.xmlunit | xmlunit-matchers | 2.7.0 |
| org.xmlunit | xmlunit-placeholders | 2.7.0 |
| org.yaml | snakeyaml | 1.26 |
| redis.clients | jedis | 3.3.0 |
| wsdl4j | wsdl4j | 1.6.3 |

### Version Properties

The following table provides all properties that can be used to override the versions managed by Spring Boot. Browse the [spring-boot-dependencies build.gradle](https://github.com/spring-projects/spring-boot/tree/v2.3.4.RELEASE/spring-boot-project/spring-boot-dependencies/build.gradle) for a complete list of dependencies.

|  |  |
| --- | --- |
| **Library** | **Version Property** |
| ActiveMQ | activemq.version |
| ANTLR2 | antlr2.version |
| AppEngine SDK | appengine-sdk.version |
| Artemis | artemis.version |
| AspectJ | aspectj.version |
| AssertJ | assertj.version |
| Atomikos | atomikos.version |
| Awaitility | awaitility.version |
| Bitronix | bitronix.version |
| Build Helper Maven Plugin | build-helper-maven-plugin.version |
| Byte Buddy | byte-buddy.version |
| Caffeine | caffeine.version |
| Cassandra Driver | cassandra-driver.version |
| Classmate | classmate.version |
| Commons Codec | commons-codec.version |
| Commons DBCP2 | commons-dbcp2.version |
| Commons Lang3 | commons-lang3.version |
| Commons Pool | commons-pool.version |
| Commons Pool2 | commons-pool2.version |

|  |  |
| --- | --- |
| **Library** | **Version Property** |
| Couchbase Client | couchbase-client.version |
| DB2 JDBC | db2-jdbc.version |
| Dependency Management Plugin | dependency-management-plugin.version |
| Derby | derby.version |
| Dropwizard Metrics | dropwizard-metrics.version |
| Ehcache | ehcache.version |
| Ehcache3 | ehcache3.version |
| Elasticsearch | elasticsearch.version |
| Embedded Mongo | embedded-mongo.version |
| Exec Maven Plugin | exec-maven-plugin.version |
| Flatten Maven Plugin | flatten-maven-plugin.version |
| Flyway | flyway.version |
| FreeMarker | freemarker.version |
| Git Commit ID Plugin | git-commit-id-plugin.version |
| Glassfish EL | glassfish-el.version |
| Glassfish JAXB | glassfish-jaxb.version |
| Groovy | groovy.version |
| Gson | gson.version |
| H2 | h2.version |
| Hamcrest | hamcrest.version |
| Hazelcast | hazelcast.version |
| Hazelcast Hibernate5 | hazelcast-hibernate5.version |
| Hibernate | hibernate.version |
| Hibernate Validator | hibernate-validator.version |
| HikariCP | hikaricp.version |
| HSQLDB | hsqldb.version |
| HtmlUnit | htmlunit.version |
| HttpAsyncClient | httpasyncclient.version |
| HttpClient | httpclient.version |
| HttpCore | httpcore.version |
| Infinispan | infinispan.version |
| InfluxDB Java | influxdb-java.version |
| Jackson Bom | jackson-bom.version |
| Jakarta Activation | jakarta-activation.version |
| Jakarta Annotation | jakarta-annotation.version |
| Jakarta JMS | jakarta-jms.version |
| Jakarta Json | jakarta-json.version |
| Jakarta Json Bind | jakarta-json-bind.version |

|  |  |
| --- | --- |
| **Library** | **Version Property** |
| Jakarta Mail | jakarta-mail.version |
| Jakarta Persistence | jakarta-persistence.version |
| Jakarta Servlet | jakarta-servlet.version |
| Jakarta Servlet JSP JSTL | jakarta-servlet-jsp-jstl.version |
| Jakarta Transaction | jakarta-transaction.version |
| Jakarta Validation | jakarta-validation.version |
| Jakarta WebSocket | jakarta-websocket.version |
| Jakarta WS RS | jakarta-ws-rs.version |
| Jakarta XML Bind | jakarta-xml-bind.version |
| Jakarta XML SOAP | jakarta-xml-soap.version |
| Jakarta XML WS | jakarta-xml-ws.version |
| Janino | janino.version |
| Javax Activation | javax-activation.version |
| Javax Annotation | javax-annotation.version |
| Javax Cache | javax-cache.version |
| Javax JAXB | javax-jaxb.version |
| Javax JAXWS | javax-jaxws.version |
| Javax JMS | javax-jms.version |
| Javax Json | javax-json.version |
| Javax JsonB | javax-jsonb.version |
| Javax Mail | javax-mail.version |
| Javax Money | javax-money.version |
| Javax Persistence | javax-persistence.version |
| Javax Transaction | javax-transaction.version |
| Javax Validation | javax-validation.version |
| Javax WebSocket | javax-websocket.version |
| Jaxen | jaxen.version |
| Jaybird | jaybird.version |
| JBoss Logging | jboss-logging.version |
| JBoss Transaction SPI | jboss-transaction-spi.version |
| JDOM2 | jdom2.version |
| Jedis | jedis.version |
| Jersey | jersey.version |
| Jetty | jetty.version |
| Jetty EL | jetty-el.version |
| Jetty JSP | jetty-jsp.version |
| Jetty Reactive HTTPClient | jetty-reactive-httpclient.version |
| JMustache | jmustache.version |

|  |  |
| --- | --- |
| **Library** | **Version Property** |
| Johnzon | johnzon.version |
| Jolokia | jolokia.version |
| jOOQ | jooq.version |
| Json Path | json-path.version |
| Json-smart | json-smart.version |
| JsonAssert | jsonassert.version |
| JSTL | jstl.version |
| JTDS | jtds.version |
| JUnit | junit.version |
| JUnit Jupiter | junit-jupiter.version |
| Kafka | kafka.version |
| Kotlin | kotlin.version |
| Kotlin Coroutines | kotlin-coroutines.version |
| Lettuce | lettuce.version |
| Liquibase | liquibase.version |
| Log4j2 | log4j2.version |
| Logback | logback.version |
| Lombok | lombok.version |
| MariaDB | mariadb.version |
| Maven AntRun Plugin | maven-antrun-plugin.version |
| Maven Assembly Plugin | maven-assembly-plugin.version |
| Maven Clean Plugin | maven-clean-plugin.version |
| Maven Compiler Plugin | maven-compiler-plugin.version |
| Maven Dependency Plugin | maven-dependency-plugin.version |
| Maven Deploy Plugin | maven-deploy-plugin.version |
| Maven Enforcer Plugin | maven-enforcer-plugin.version |
| Maven Failsafe Plugin | maven-failsafe-plugin.version |
| Maven Help Plugin | maven-help-plugin.version |
| Maven Install Plugin | maven-install-plugin.version |
| Maven Invoker Plugin | maven-invoker-plugin.version |
| Maven Jar Plugin | maven-jar-plugin.version |
| Maven Javadoc Plugin | maven-javadoc-plugin.version |
| Maven Resources Plugin | maven-resources-plugin.version |
| Maven Shade Plugin | maven-shade-plugin.version |
| Maven Source Plugin | maven-source-plugin.version |
| Maven Surefire Plugin | maven-surefire-plugin.version |
| Maven War Plugin | maven-war-plugin.version |
| Micrometer | micrometer.version |

|  |  |
| --- | --- |
| **Library** | **Version Property** |
| MIMEPull | mimepull.version |
| Mockito | mockito.version |
| MongoDB | mongodb.version |
| MSSQL JDBC | mssql-jdbc.version |
| MySQL | mysql.version |
| NekoHTML | nekohtml.version |
| Neo4j OGM | neo4j-ogm.version |
| Netty | netty.version |
| Netty tcNative | netty-tcnative.version |
| Nimbus JOSE JWT | nimbus-jose-jwt.version |
| NIO Multipart Parser | nio-multipart-parser.version |
| OAuth2 OIDC SDK | oauth2-oidc-sdk.version |
| OJDBC | ojdbc.version |
| OkHttp3 | okhttp3.version |
| Oracle Database | oracle-database.version |
| Pooled JMS | pooled-jms.version |
| Postgresql | postgresql.version |
| Prometheus PushGateway | prometheus-pushgateway.version |
| Quartz | quartz.version |
| QueryDSL | querydsl.version |
| R2DBC Bom | r2dbc-bom.version |
| Rabbit AMQP Client | rabbit-amqp-client.version |
| Reactive Streams | reactive-streams.version |
| Reactor Bom | reactor-bom.version |
| REST Assured | rest-assured.version |
| RSocket | rsocket.version |
| RxJava | rxjava.version |
| RxJava Adapter | rxjava-adapter.version |
| RxJava2 | rxjava2.version |
| SAAJ Impl | saaj-impl.version |
| Selenium | selenium.version |
| Selenium HtmlUnit | selenium-htmlunit.version |
| SendGrid | sendgrid.version |
| Servlet API | servlet-api.version |
| SLF4J | slf4j.version |
| SnakeYAML | snakeyaml.version |
| Solr | solr.version |
| Spring AMQP | spring-amqp.version |

|  |  |
| --- | --- |
| **Library** | **Version Property** |
| Spring Batch | spring-batch.version |
| Spring Data Releasetrain | spring-data-releasetrain.version |
| Spring Framework | spring-framework.version |
| Spring HATEOAS | spring-hateoas.version |
| Spring Integration | spring-integration.version |
| Spring Kafka | spring-kafka.version |
| Spring LDAP | spring-ldap.version |
| Spring RESTDocs | spring-restdocs.version |
| Spring Retry | spring-retry.version |
| Spring Security | spring-security.version |
| Spring Session Bom | spring-session-bom.version |
| Spring WS | spring-ws.version |
| SQLite JDBC | sqlite-jdbc.version |
| Sun Mail | sun-mail.version |
| Thymeleaf | thymeleaf.version |
| Thymeleaf Extras Data Attribute | thymeleaf-extras-data-attribute.version |
| Thymeleaf Extras Java8Time | thymeleaf-extras-java8time.version |
| Thymeleaf Extras SpringSecurity | thymeleaf-extras-springsecurity.version |
| Thymeleaf Layout Dialect | thymeleaf-layout-dialect.version |
| Tomcat | tomcat.version |
| UnboundID LDAPSDK | unboundid-ldapsdk.version |
| Undertow | undertow.version |
| Versions Maven Plugin | versions-maven-plugin.version |
| WebJars HAL Browser | webjars-hal-browser.version |
| WebJars Locator Core | webjars-locator-core.version |
| WSDL4j | wsdl4j.version |
| XML Maven Plugin | xml-maven-plugin.version |
| XmlUnit2 | xmlunit2.version |