Spring Recipes: A Collection of Common-Sense Solutions

Nathaniel Schutta (@ntschutta)
ntschutta.io
Dan Vega (@therealdanvega)
danvega.dev

https://github.com/danvega/spring-recipes

Who are these guys?

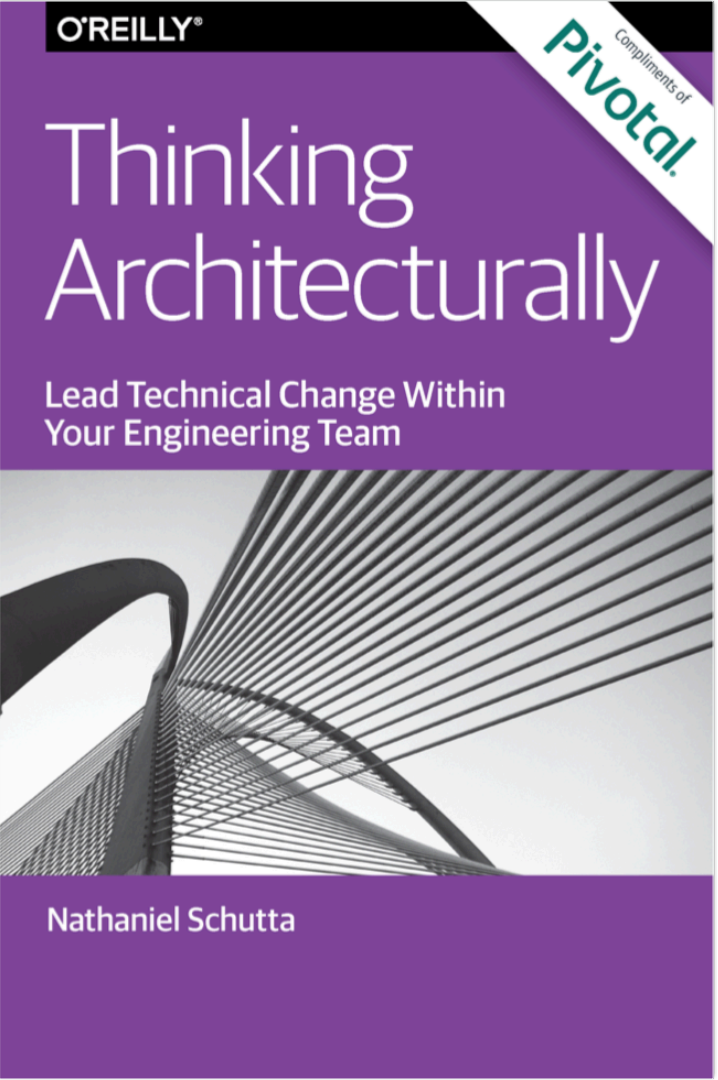
ABOUT ME

- Husband & Father
- Twin Cites, MN
- Software Development 20+ Years
- Architect as a Service
- Developer Advocate
- Author
- Adjunct Professor
- Blogger, Twitch, YouTube
- Golfer, cyclist
- @ntschutta
- http://ntschutta.io

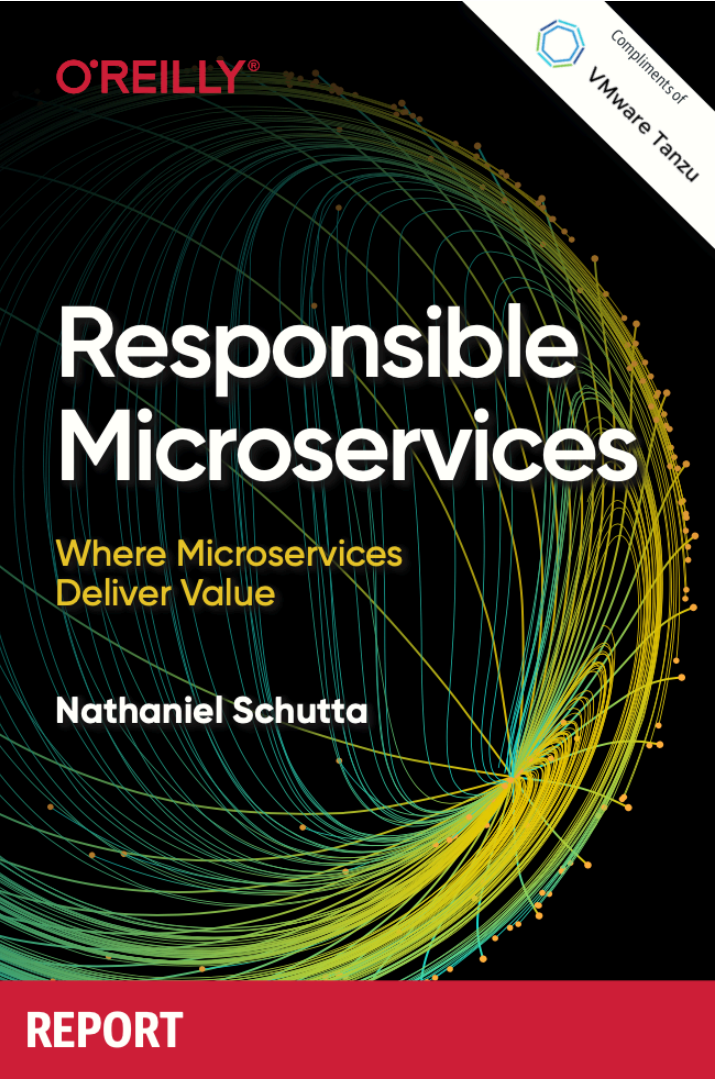
Between Chair and Keyboard

Most Mondays,
around noon Central
https://www.twitch.tv/vmwaretanzu

Nate Schutta
Software Architect
VMware
@ntschutta



https://tanzu.vmware.com/content/ebooks/thinking-architecturally



https://tanzu.vmware.com/content/ebooks/responsible-microservices-ebook

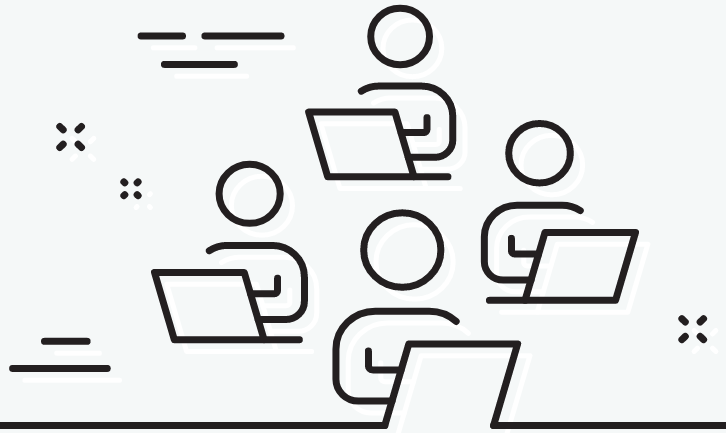
ABOUT ME

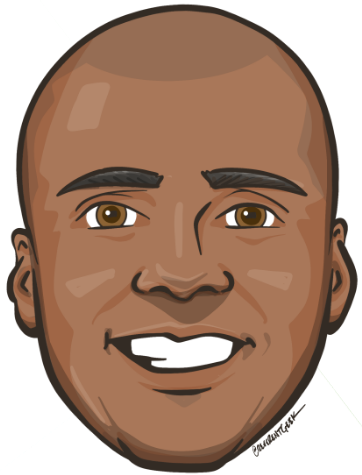
- Husband & Father
- Cleveland, OH
- Software Development 20+ Years
- Spring Developer Advocate
- Content Creator (www.danvega.dev)
  - Blogger
  - YouTuber
  - Course Creator
- @therealdanvega

spring

OFFICE HOURS

https://bit.ly/spring-office-hours





https://tanzu.vmware.com/developer/tv/spring-office-hours/


spring®
RECIPES


Java™


spring®

Spring Projects


Spring Boot


Spring Cloud


Spring Framework

Spring Cloud Data Flow

Spring Tool Suite


Spring LDAP


Spring Cloud Gateway


Spring Security


Spring Data


Spring Batch


Spring Integration


Project Reactor


Spring Kafka


Spring for GraphQL


Spring Web Services


Spring Web Flow


Spring Hateoas


Spring AMQP

Agenda

- Getting Started with Spring Boot
- Building Web Applications
- Working with Databases
- Spring Cloud
- Testing
- Production
- Q+A

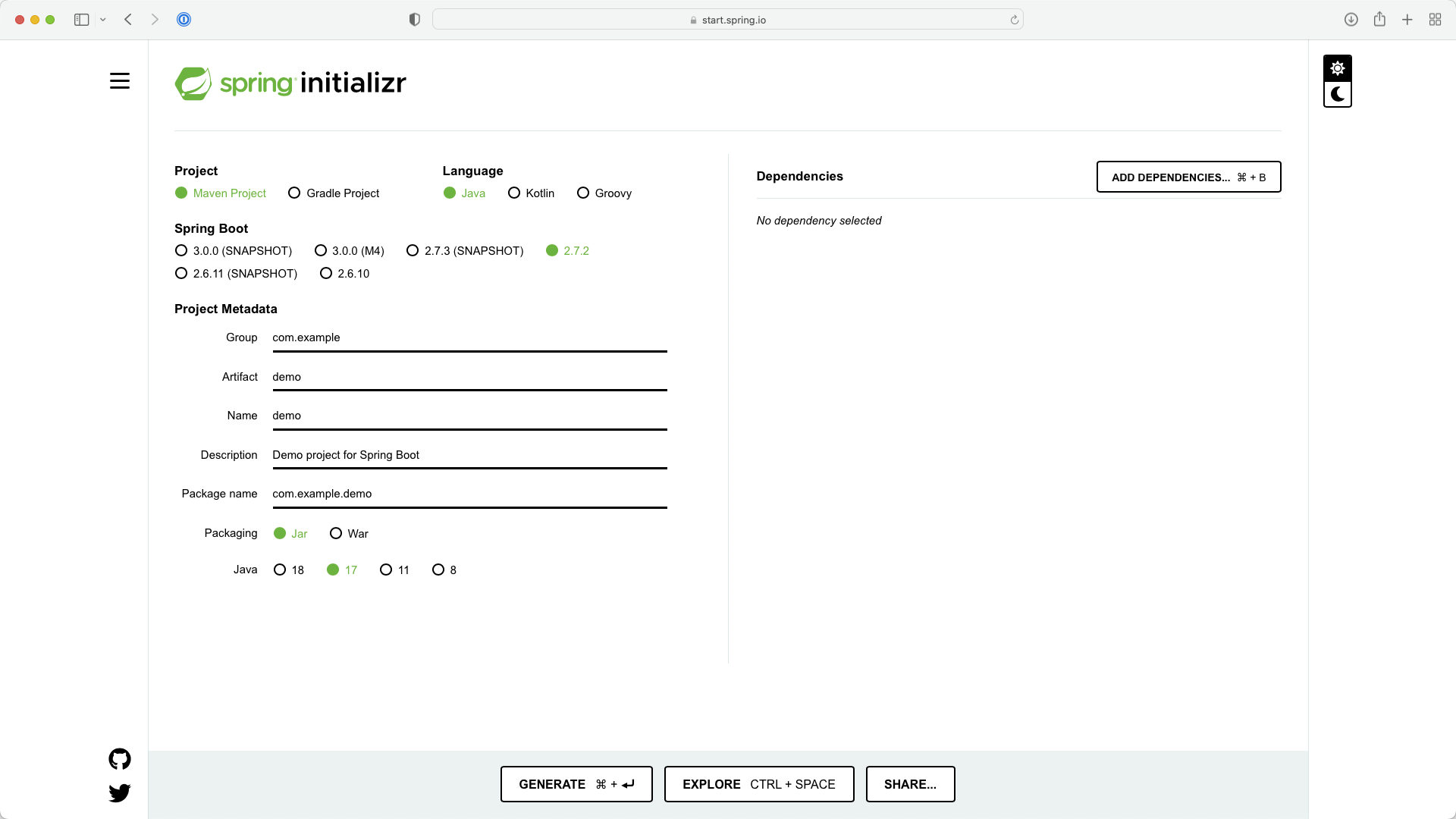
Getting Started with Spring Boot

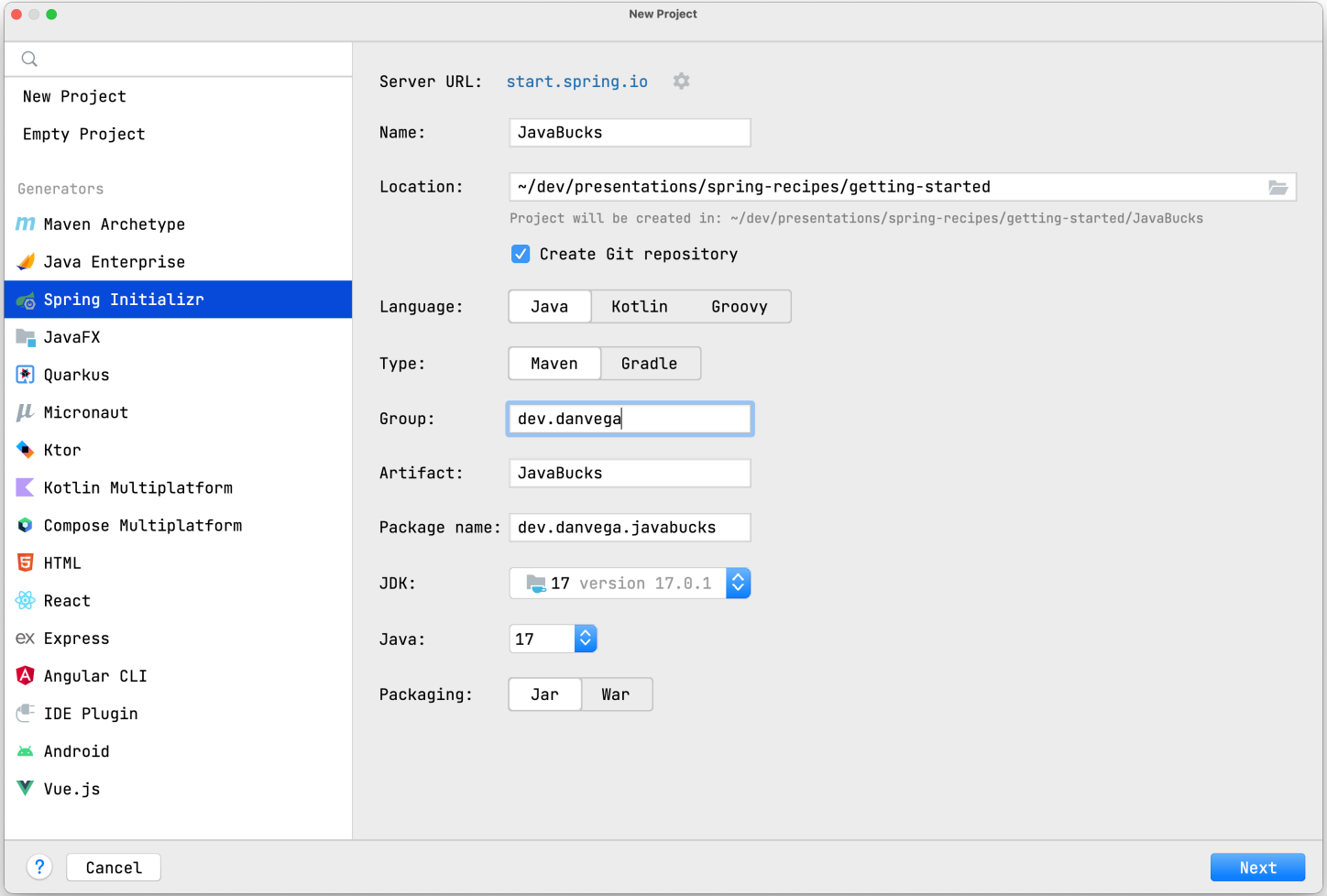
Problem

You want to create a new Spring project but you don't want to start from scratch.

Solution

If you want to create your own Spring Boot-based project, visit Spring Initializr, fill in your project details, pick your options, and download a bundled up project as a zip file.


start.spring.io
spring initializr
Project
Maven Project
Gradle Project
Language
Java
Kotlin
Groovy
Spring Boot
3.0.0 (SNAPSHOT)
3.0.0 (M4)
2.7.3 (SNAPSHOT)
2.7.2
2.6.11 (SNAPSHOT)
2.6.10
Project Metadata
Group
com.example
Artifact
demo
Name
demo
Description
Demo project for Spring Boot
Package name
com.example.demo
Packaging
Jar
War
Java
18
17
11
8
Dependencies
ADD DEPENDENCIES... ⌘ + B
No dependency selected
GENERATE ⌘ + ↵
EXPLORE CTRL + SPACE
SHARE...


New Project
New Project
Empty Project
Generators
Maven Archetype
Java Enterprise
Spring Initializr
JavaFX
Quarkus
Micronaut
Ktor
Kotlin Multiplatform
Compose Multiplatform
HTML
React
Express
Angular CLI
IDE Plugin
Android
Vue.js
Server URL: start.spring.io
Name: JavaBucks
Location: ~/dev/presentations/spring-recipes/getting-started
Project will be created in: ~/dev/presentations/spring-recipes/getting-started/JavaBucks
Create Git repository
Language: Java Kotlin Groovy
Type: Maven Gradle
Group: dev.danvega
Artifact: JavaBucks
Package name: dev.danvega.javabucks
JDK: 17 version 17.0.1
Java: 17
Packaging: Jar War
Cancel
Next

Problem

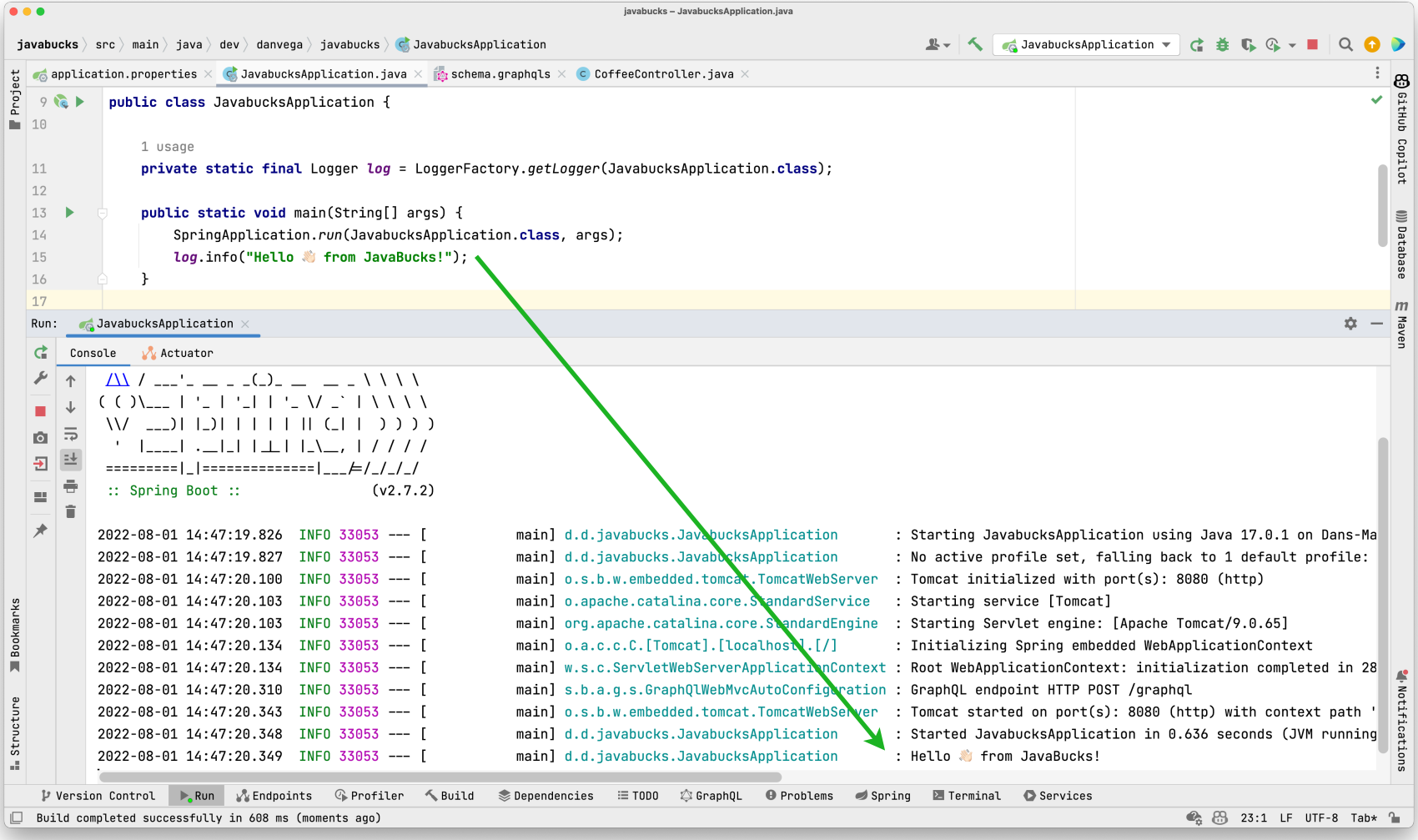
You want to log information about the state of your application.

Solution

Default configurations are provided for Java Util Logging, Log4J2, and Logback. In each case, loggers are pre-configured to use console output with optional file output also available.

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.2)

2022-07-27 15:48:37.847  INFO 7838 --- [           main] d.d.javabucks.JavabucksApplication       : Starting JavabucksApplication using Java 17.0.1 on Dans-MacBook-Pro-M
2022-07-27 15:48:37.848  INFO 7838 --- [           main] d.d.javabucks.JavabucksApplication       : No active profile set, falling back to 1 default profile: "default"
2022-07-27 15:48:38.111  INFO 7838 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2022-07-27 15:48:38.113  INFO 7838 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-07-27 15:48:38.113  INFO 7838 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-07-27 15:48:38.145  INFO 7838 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2022-07-27 15:48:38.145  INFO 7838 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 280 ms
2022-07-27 15:48:38.367  INFO 7838 --- [           main] s.b.a.g.s.GraphQlWebMvcAutoConfiguration : GraphQL endpoint HTTP POST /graphql
2022-07-27 15:48:38.405  INFO 7838 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2022-07-27 15:48:38.415  INFO 7838 --- [           main] d.d.javabucks.JavabucksApplication       : Started JavabucksApplication in 0.675 seconds (JVM running for 1.012)
2022-07-27 15:48:45.655  INFO 7838 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-07-27 15:48:45.655  INFO 7838 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2022-07-27 15:48:45.655  INFO 7838 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 0 ms
```

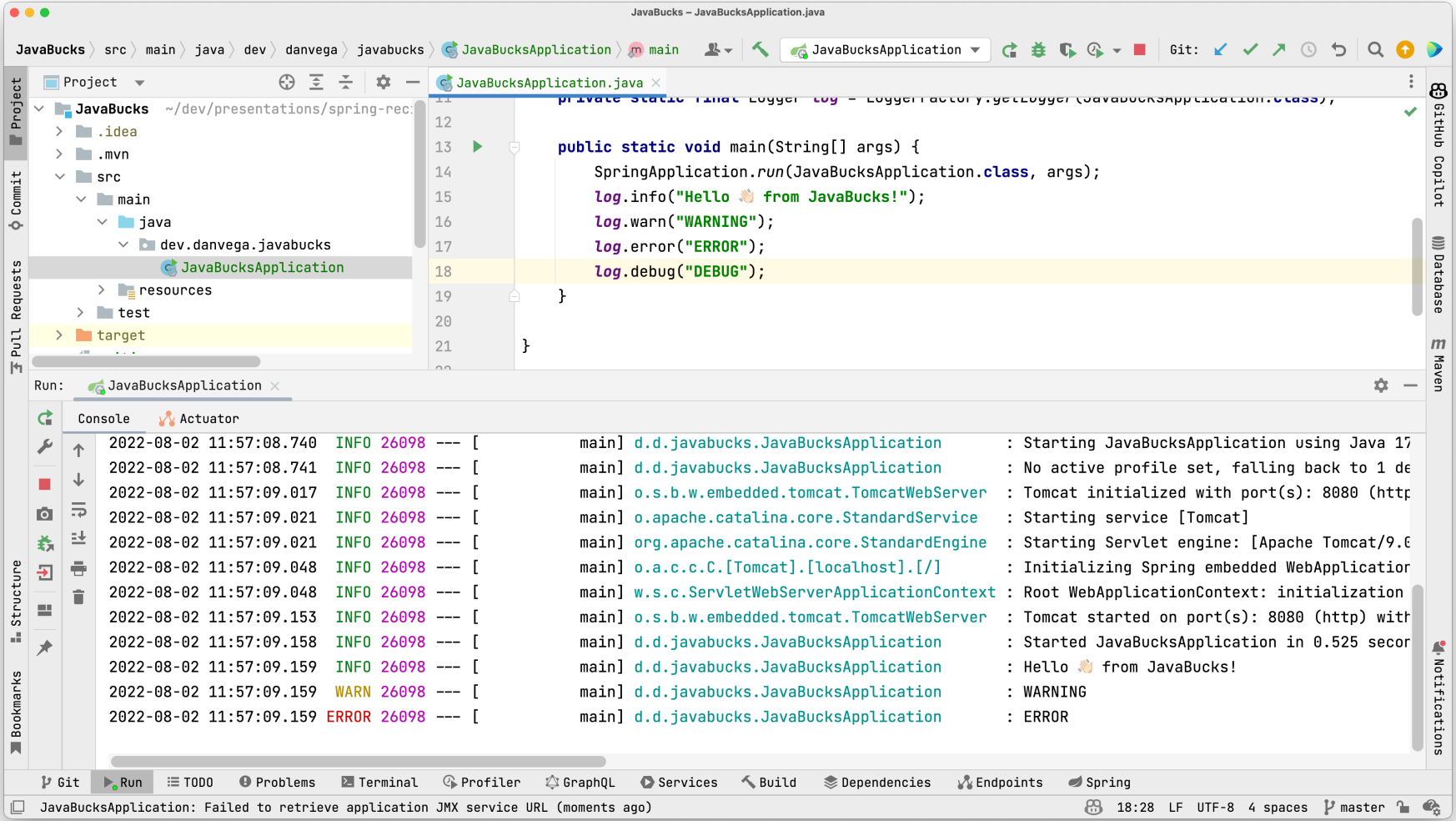

javabucks – JavabucksApplication.java
javabucks › src › main › java › dev › danvega › javabucks › JavabucksApplication
JavabucksApplication
application.properties
JavabucksApplication.java
schema.graphqls
CoffeeController.java
public class JavabucksApplication {
1 usage
private static final Logger log = LoggerFactory.getLogger(JavabucksApplication.class);
public static void main(String[] args) {
SpringApplication.run(JavabucksApplication.class, args);
log.info("Hello 👋 from JavaBucks!");
}
Run: JavabucksApplication
Console
Actuator
:: Spring Boot :: (v2.7.2)
2022-08-01 14:47:19.826 INFO 33053 --- [main] d.d.javabucks.JavabucksApplication : Starting JavabucksApplication using Java 17.0.1 on Dans-Ma
2022-08-01 14:47:19.827 INFO 33053 --- [main] d.d.javabucks.JavabucksApplication : No active profile set, falling back to 1 default profile:
2022-08-01 14:47:20.100 INFO 33053 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-08-01 14:47:20.103 INFO 33053 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-08-01 14:47:20.103 INFO 33053 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-08-01 14:47:20.134 INFO 33053 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-08-01 14:47:20.134 INFO 33053 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 28
2022-08-01 14:47:20.310 INFO 33053 --- [main] s.b.a.g.s.GraphQlWebMvcAutoConfiguration : GraphQL endpoint HTTP POST /graphql
2022-08-01 14:47:20.343 INFO 33053 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path '
2022-08-01 14:47:20.348 INFO 33053 --- [main] d.d.javabucks.JavabucksApplication : Started JavabucksApplication in 0.636 seconds (JVM running
2022-08-01 14:47:20.349 INFO 33053 --- [main] d.d.javabucks.JavabucksApplication : Hello 👋 from JavaBucks!
Version Control
Run
Endpoints
Profiler
Build
Dependencies
TODO
GraphQL
Problems
Spring
Terminal
Services
Build completed successfully in 608 ms (moments ago)
23:1 LF UTF-8 Tab*

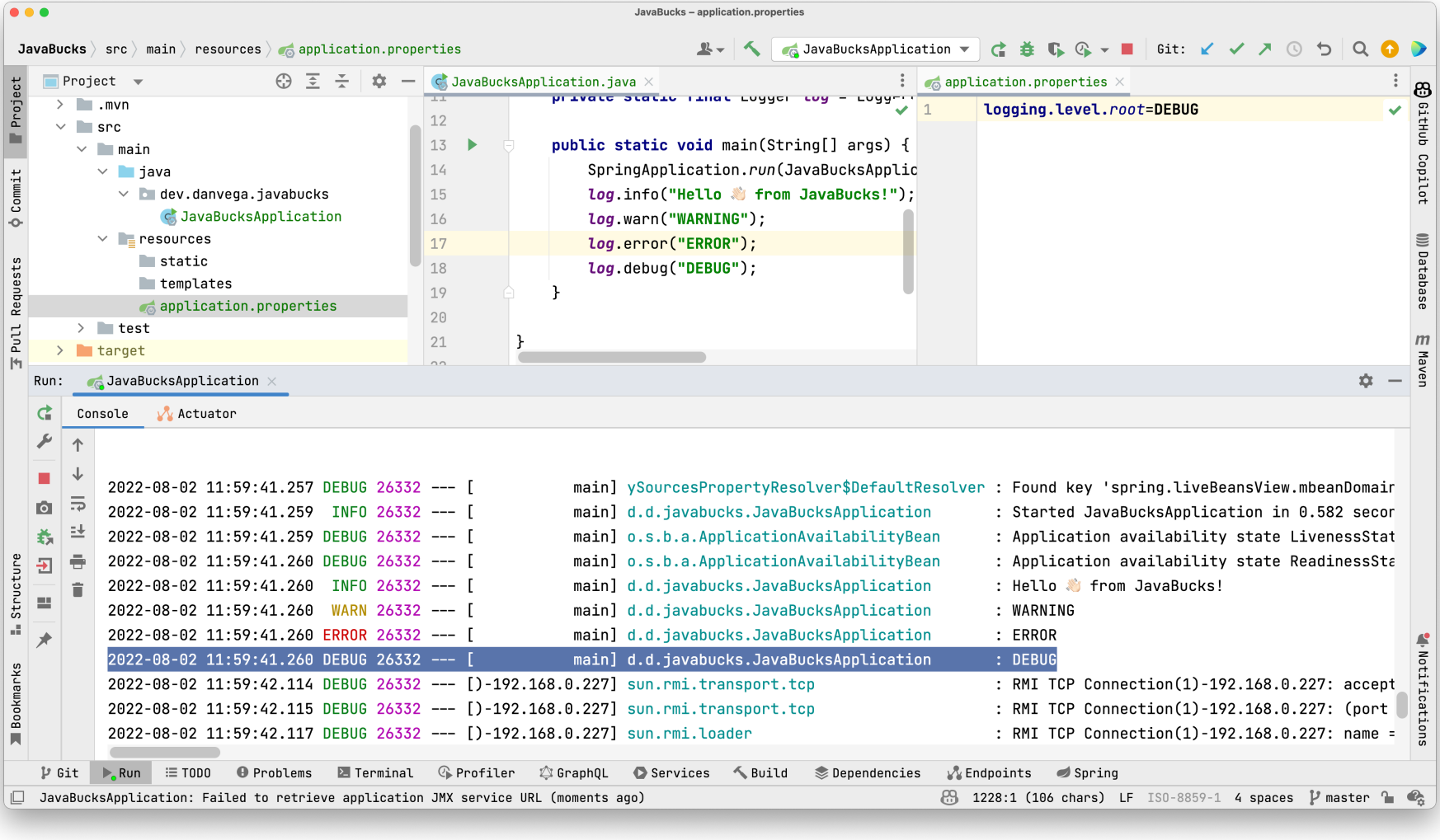
Problem

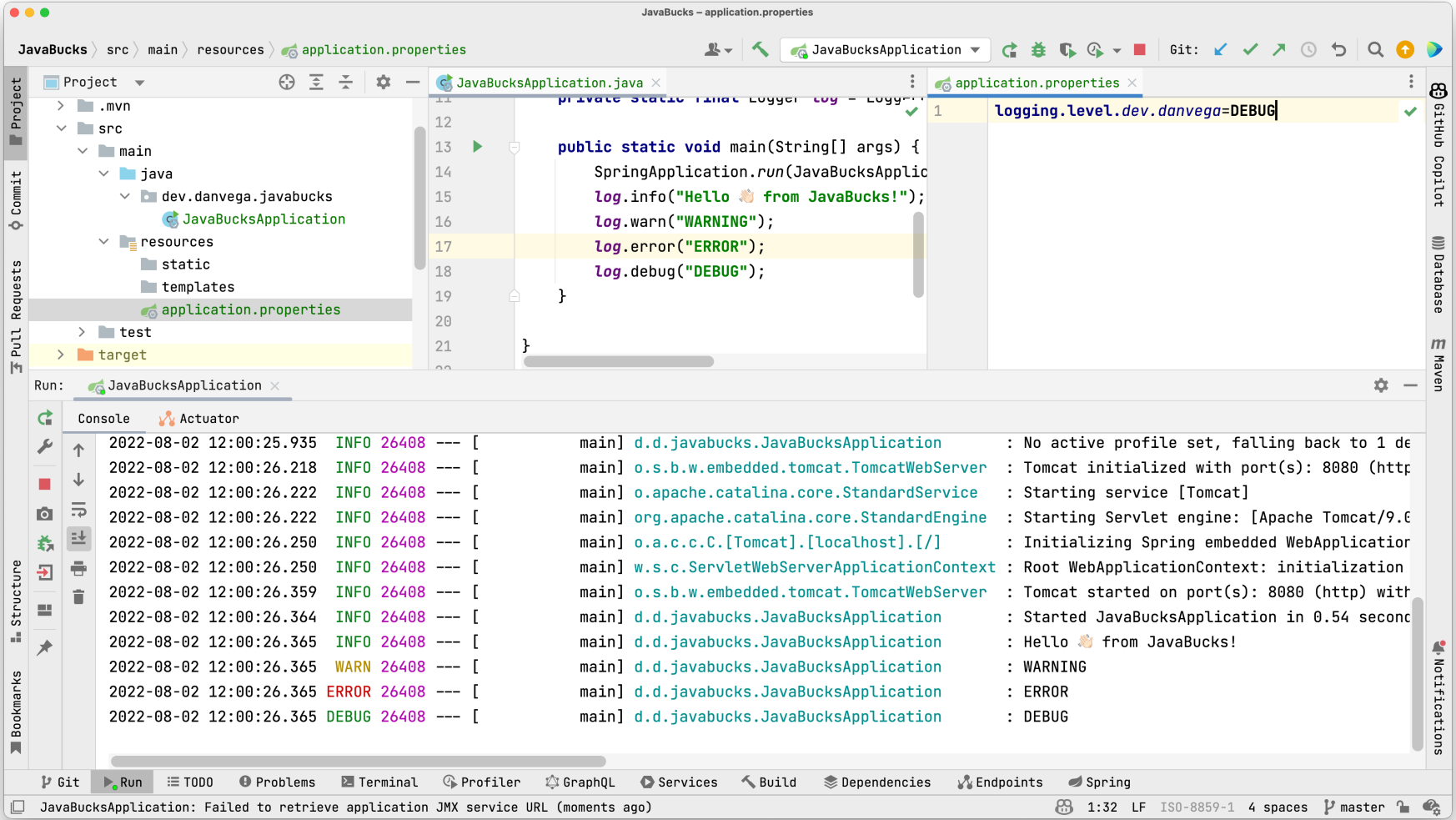
You want to update the logging levels so that a specific log level and above are logged.

Solution

You can configure the logging levels in application.properties.


JavaBucks – JavaBucksApplication.java
JavaBucks › src › main › java › dev › danvega › javabucks › JavaBucksApplication › main
JavaBucksApplication
Git:
Project
JavaBucks ~/dev/presentations/spring-rec
.idea
.mvn
src
main
java
dev.danvega.javabucks
JavaBucksApplication
resources
test
target
JavaBucksApplication.java
12
13 public static void main(String[] args) {
14 SpringApplication.run(JavaBucksApplication.class, args);
15 log.info("Hello 👋 from JavaBucks!");
16 log.warn("WARNING");
17 log.error("ERROR");
18 log.debug("DEBUG");
19 }
20
21 }
Run: JavaBucksApplication
Console
Actuator
2022-08-02 11:57:08.740 INFO 26098 --- [main] d.d.javabucks.JavaBucksApplication : Starting JavaBucksApplication using Java 17
2022-08-02 11:57:08.741 INFO 26098 --- [main] d.d.javabucks.JavaBucksApplication : No active profile set, falling back to 1 de
2022-08-02 11:57:09.017 INFO 26098 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http
2022-08-02 11:57:09.021 INFO 26098 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-08-02 11:57:09.021 INFO 26098 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0
2022-08-02 11:57:09.048 INFO 26098 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicatior
2022-08-02 11:57:09.048 INFO 26098 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization
2022-08-02 11:57:09.153 INFO 26098 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with
2022-08-02 11:57:09.158 INFO 26098 --- [main] d.d.javabucks.JavaBucksApplication : Started JavaBucksApplication in 0.525 secor
2022-08-02 11:57:09.159 INFO 26098 --- [main] d.d.javabucks.JavaBucksApplication : Hello 👋 from JavaBucks!
2022-08-02 11:57:09.159 WARN 26098 --- [main] d.d.javabucks.JavaBucksApplication : WARNING
2022-08-02 11:57:09.159 ERROR 26098 --- [main] d.d.javabucks.JavaBucksApplication : ERROR
Git
Run
TODO
Problems
Terminal
Profiler
GraphQL
Services
Build
Dependencies
Endpoints
Spring
JavaBucksApplication: Failed to retrieve application JMX service URL (moments ago)
18:28 LF UTF-8 4 spaces master
GitHub Copilot
Database
Maven
Notifications
Structure
Bookmarks
Pull Requests
Commit


JavaBucks – application.properties
JavaBucks 〉 src 〉 main 〉 resources 〉 application.properties
JavaBucksApplication
Git:
Project
.mvn
src
main
java
dev.danvega.javabucks
JavaBucksApplication
resources
static
templates
application.properties
test
target
JavaBucksApplication.java
12
13 public static void main(String[] args) {
14 SpringApplication.run(JavaBucksApplic
15 log.info("Hello 👋 from JavaBucks!");
16 log.warn("WARNING");
17 log.error("ERROR");
18 log.debug("DEBUG");
19 }
20
21 }
application.properties
1 logging.level.root=DEBUG
Run: JavaBucksApplication
Console
Actuator
2022-08-02 11:59:41.257 DEBUG 26332 --- [main] ySourcesPropertyResolver$DefaultResolver : Found key 'spring.liveBeansView.mbeanDomair
2022-08-02 11:59:41.259 INFO 26332 --- [main] d.d.javabucks.JavaBucksApplication : Started JavaBucksApplication in 0.582 secor
2022-08-02 11:59:41.259 DEBUG 26332 --- [main] o.s.b.a.ApplicationAvailabilityBean : Application availability state LivenessStat
2022-08-02 11:59:41.260 DEBUG 26332 --- [main] o.s.b.a.ApplicationAvailabilityBean : Application availability state ReadinessSta
2022-08-02 11:59:41.260 INFO 26332 --- [main] d.d.javabucks.JavaBucksApplication : Hello 👋 from JavaBucks!
2022-08-02 11:59:41.260 WARN 26332 --- [main] d.d.javabucks.JavaBucksApplication : WARNING
2022-08-02 11:59:41.260 ERROR 26332 --- [main] d.d.javabucks.JavaBucksApplication : ERROR
2022-08-02 11:59:41.260 DEBUG 26332 --- [main] d.d.javabucks.JavaBucksApplication : DEBUG
2022-08-02 11:59:42.114 DEBUG 26332 --- [)-192.168.0.227] sun.rmi.transport.tcp : RMI TCP Connection(1)-192.168.0.227: accept
2022-08-02 11:59:42.115 DEBUG 26332 --- [)-192.168.0.227] sun.rmi.transport.tcp : RMI TCP Connection(1)-192.168.0.227: (port
2022-08-02 11:59:42.117 DEBUG 26332 --- [)-192.168.0.227] sun.rmi.loader : RMI TCP Connection(1)-192.168.0.227: name =
Git
Run
TODO
Problems
Terminal
Profiler
GraphQL
Services
Build
Dependencies
Endpoints
Spring
JavaBucksApplication: Failed to retrieve application JMX service URL (moments ago)
1228:1 (106 chars) LF ISO-8859-1 4 spaces master
GitHub Copilot
Database
Maven
Notifications
Structure
Bookmarks
Commit
Pull Requests

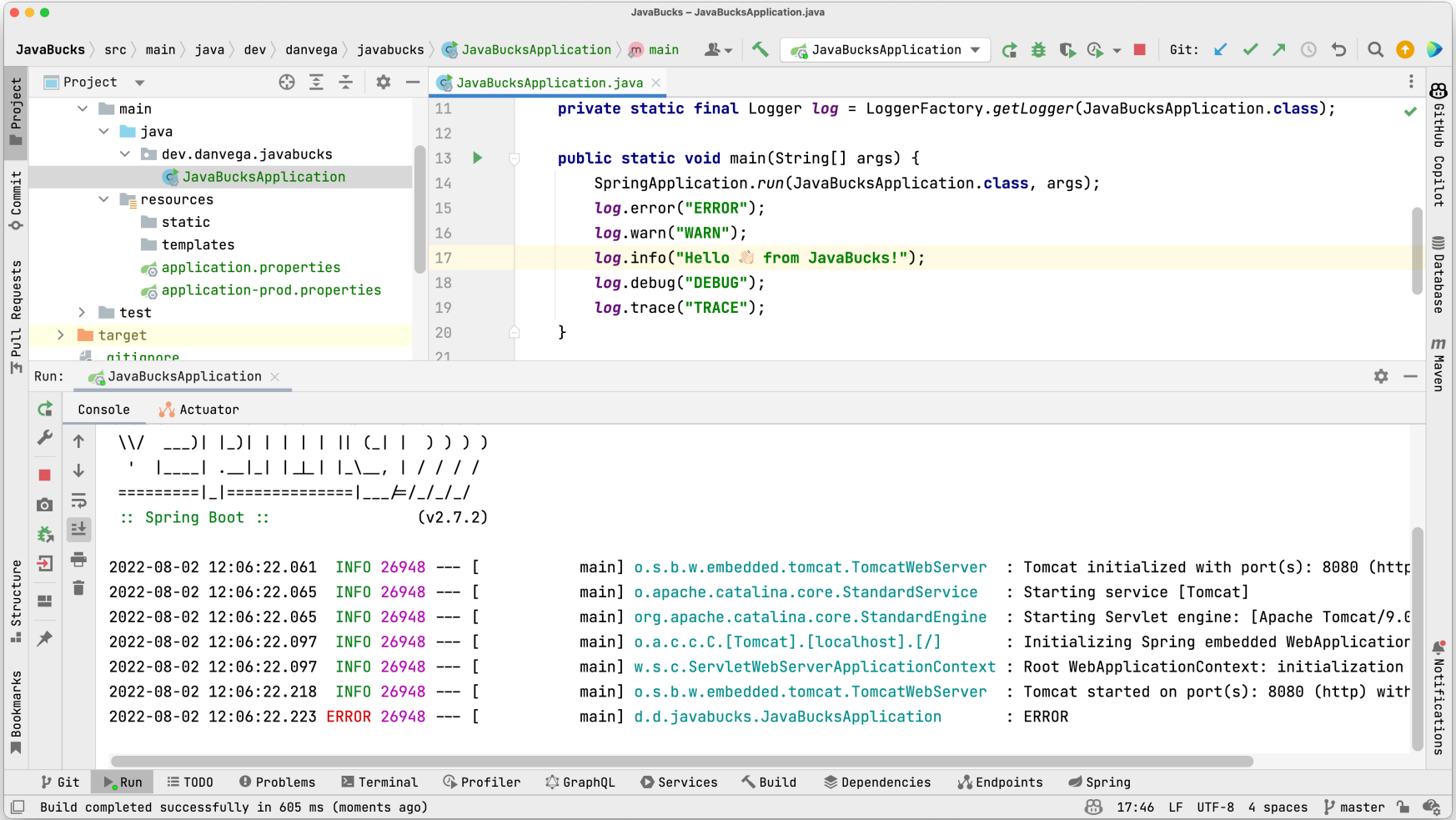

JavaBucks – application.properties
JavaBucks 〉 src 〉 main 〉 resources 〉 application.properties
JavaBucksApplication
Git:
Project
.mvn
src
main
java
dev.danvega.javabucks
JavaBucksApplication
resources
static
templates
application.properties
test
target
JavaBucksApplication.java
12
13 public static void main(String[] args) {
14 SpringApplication.run(JavaBucksApplic
15 log.info("Hello 👋 from JavaBucks!");
16 log.warn("WARNING");
17 log.error("ERROR");
18 log.debug("DEBUG");
19 }
20
21 }
application.properties
1 logging.level.dev.danvega=DEBUG
Run: JavaBucksApplication
Console
Actuator
2022-08-02 12:00:25.935 INFO 26408 --- [main] d.d.javabucks.JavaBucksApplication : No active profile set, falling back to 1 de
2022-08-02 12:00:26.218 INFO 26408 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http
2022-08-02 12:00:26.222 INFO 26408 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-08-02 12:00:26.222 INFO 26408 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0
2022-08-02 12:00:26.250 INFO 26408 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicatior
2022-08-02 12:00:26.250 INFO 26408 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization
2022-08-02 12:00:26.359 INFO 26408 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with
2022-08-02 12:00:26.364 INFO 26408 --- [main] d.d.javabucks.JavaBucksApplication : Started JavaBucksApplication in 0.54 second
2022-08-02 12:00:26.365 INFO 26408 --- [main] d.d.javabucks.JavaBucksApplication : Hello 👋 from JavaBucks!
2022-08-02 12:00:26.365 WARN 26408 --- [main] d.d.javabucks.JavaBucksApplication : WARNING
2022-08-02 12:00:26.365 ERROR 26408 --- [main] d.d.javabucks.JavaBucksApplication : ERROR
2022-08-02 12:00:26.365 DEBUG 26408 --- [main] d.d.javabucks.JavaBucksApplication : DEBUG
Git
Run
TODO
Problems
Terminal
Profiler
GraphQL
Services
Build
Dependencies
Endpoints
Spring
JavaBucksApplication: Failed to retrieve application JMX service URL (moments ago)
1:32 LF ISO-8859-1 4 spaces master
GitHub Copilot
Database
Maven
Notifications
Structure
Bookmarks
Commit
Pull Requests

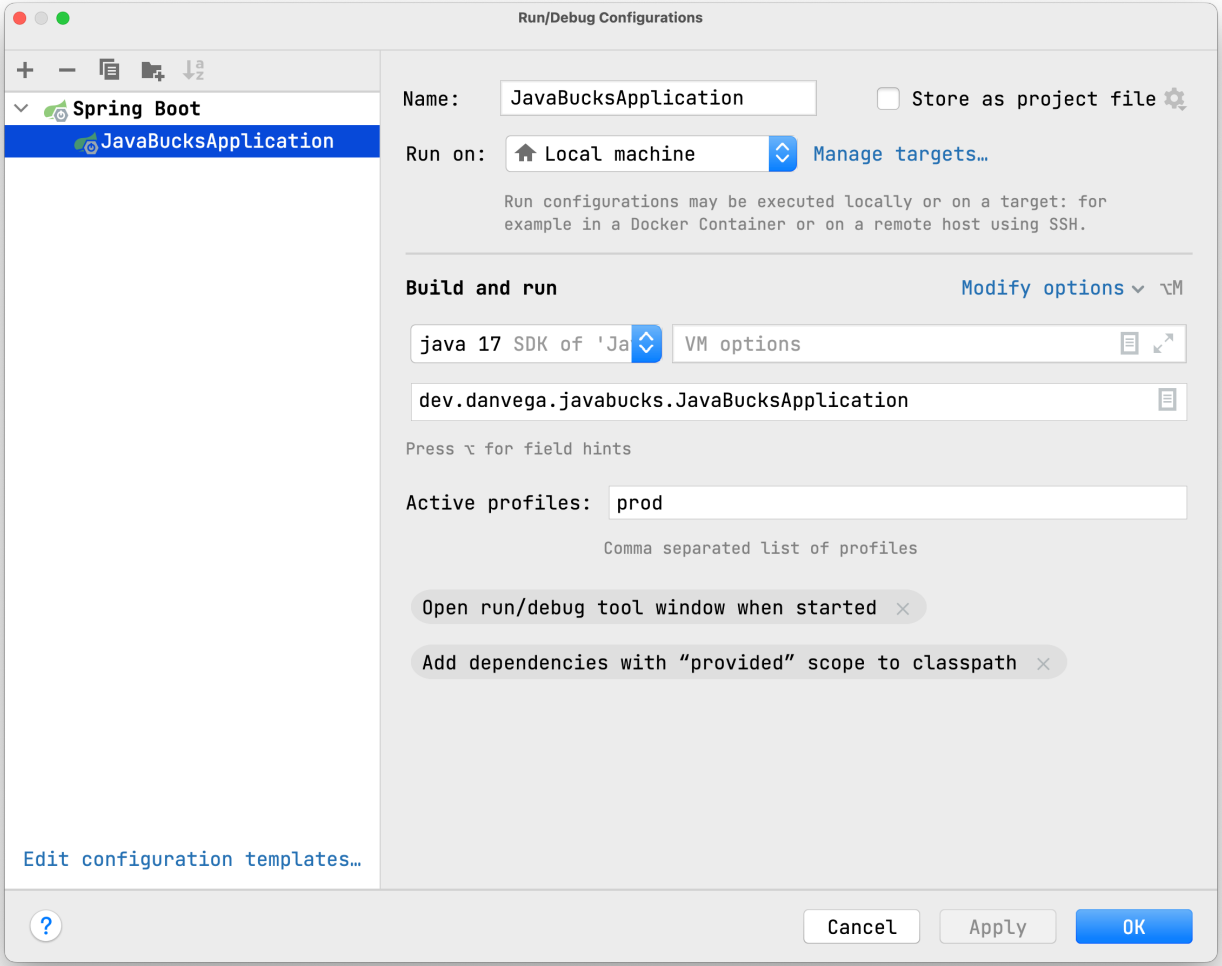
Problem

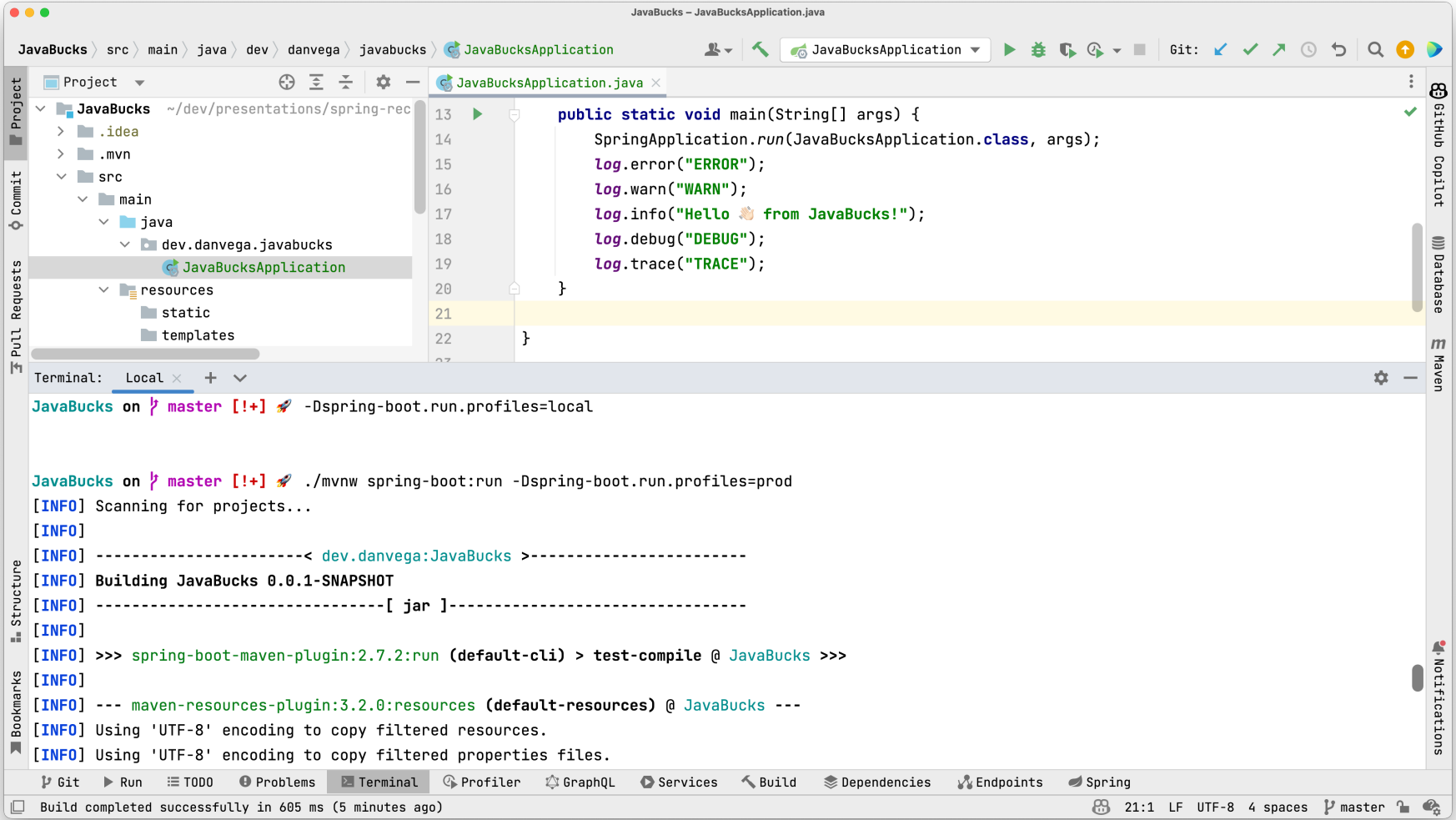
You need configuration in your application to change based on the environment the application is running in.

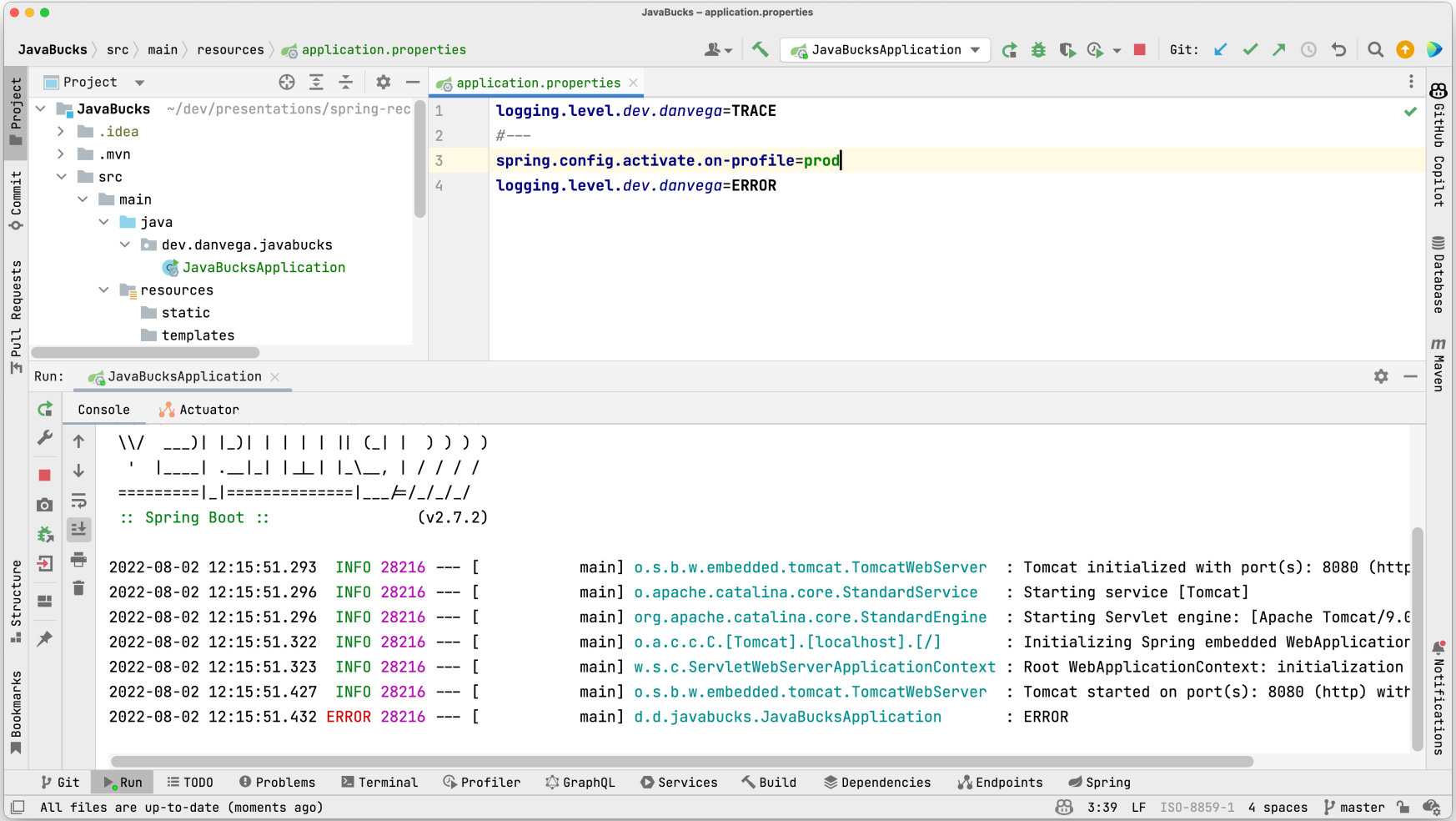
Solution

Spring Boot lets you externalize your configuration so that you can work with the same application code in different environments. You can use a variety of external configuration sources, include Java properties files, YAML files, environment variables, and command-line arguments.


JavaBucks – JavaBucksApplication.java
JavaBucks › src › main › java › dev › danvega › javabucks › JavaBucksApplication › main
JavaBucksApplication
Git:
Project
JavaBucksApplication.java
main
java
dev.danvega.javabucks
JavaBucksApplication
resources
static
templates
application.properties
application-prod.properties
test
target
.gitignore
11 private static final Logger log = LoggerFactory.getLogger(JavaBucksApplication.class);
13 public static void main(String[] args) {
14 SpringApplication.run(JavaBucksApplication.class, args);
15 log.error("ERROR");
16 log.warn("WARN");
17 log.info("Hello 👋 from JavaBucks!");
18 log.debug("DEBUG");
19 log.trace("TRACE");
20 }
Run: JavaBucksApplication
Console
Actuator
:: Spring Boot :: (v2.7.2)
2022-08-02 12:06:22.061 INFO 26948 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http
2022-08-02 12:06:22.065 INFO 26948 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-08-02 12:06:22.065 INFO 26948 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0
2022-08-02 12:06:22.097 INFO 26948 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicatior
2022-08-02 12:06:22.097 INFO 26948 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization
2022-08-02 12:06:22.218 INFO 26948 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with
2022-08-02 12:06:22.223 ERROR 26948 --- [main] d.d.javabucks.JavaBucksApplication : ERROR
Git
Run
TODO
Problems
Terminal
Profiler
GraphQL
Services
Build
Dependencies
Endpoints
Spring
Build completed successfully in 605 ms (moments ago)
17:46 LF UTF-8 4 spaces master
GitHub Copilot
Database
Maven
Notifications
Structure
Bookmarks
Pull Requests
Commit


Run/Debug Configurations
Spring Boot
JavaBucksApplication
Edit configuration templates…
Name:
JavaBucksApplication
Store as project file
Run on:
Local machine
Manage targets…
Run configurations may be executed locally or on a target: for example in a Docker Container or on a remote host using SSH.
Build and run
Modify options
⌥M
java 17 SDK of 'Ja
VM options
dev.danvega.javabucks.JavaBucksApplication
Press ⌥ for field hints
Active profiles:
prod
Comma separated list of profiles
Open run/debug tool window when started
Add dependencies with “provided” scope to classpath
Cancel
Apply
OK


JavaBucks – JavaBucksApplication.java
JavaBucks 〉 src 〉 main 〉 java 〉 dev 〉 danvega 〉 javabucks 〉 JavaBucksApplication
JavaBucksApplication
Git:
Project
JavaBucks ~/dev/presentations/spring-rec
.idea
.mvn
src
main
java
dev.danvega.javabucks
JavaBucksApplication
resources
static
templates
JavaBucksApplication.java
13 public static void main(String[] args) {
14 SpringApplication.run(JavaBucksApplication.class, args);
15 log.error("ERROR");
16 log.warn("WARN");
17 log.info("Hello 👋🏻 from JavaBucks!");
18 log.debug("DEBUG");
19 log.trace("TRACE");
20 }
21
22 }
Terminal: Local
JavaBucks on ᚠ master [!+] 🚀 -Dspring-boot.run.profiles=local
JavaBucks on ᚠ master [!+] 🚀 ./mvnw spring-boot:run -Dspring-boot.run.profiles=prod
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< dev.danvega:JavaBucks >-----------------------
[INFO] Building JavaBucks 0.0.1-SNAPSHOT
[INFO] --------------------------------[jar]---------------------------------
[INFO]
[INFO] >>> spring-boot-maven-plugin:2.7.2:run (default-cli) > test-compile @ JavaBucks >>>
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ JavaBucks ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
Git Run TODO Problems Terminal Profiler GraphQL Services Build Dependencies Endpoints Spring
Build completed successfully in 605 ms (5 minutes ago)
21:1 LF UTF-8 4 spaces master
GitHub Copilot
Database
Maven
Notifications
Project
Commit
Pull Requests
Structure
Bookmarks

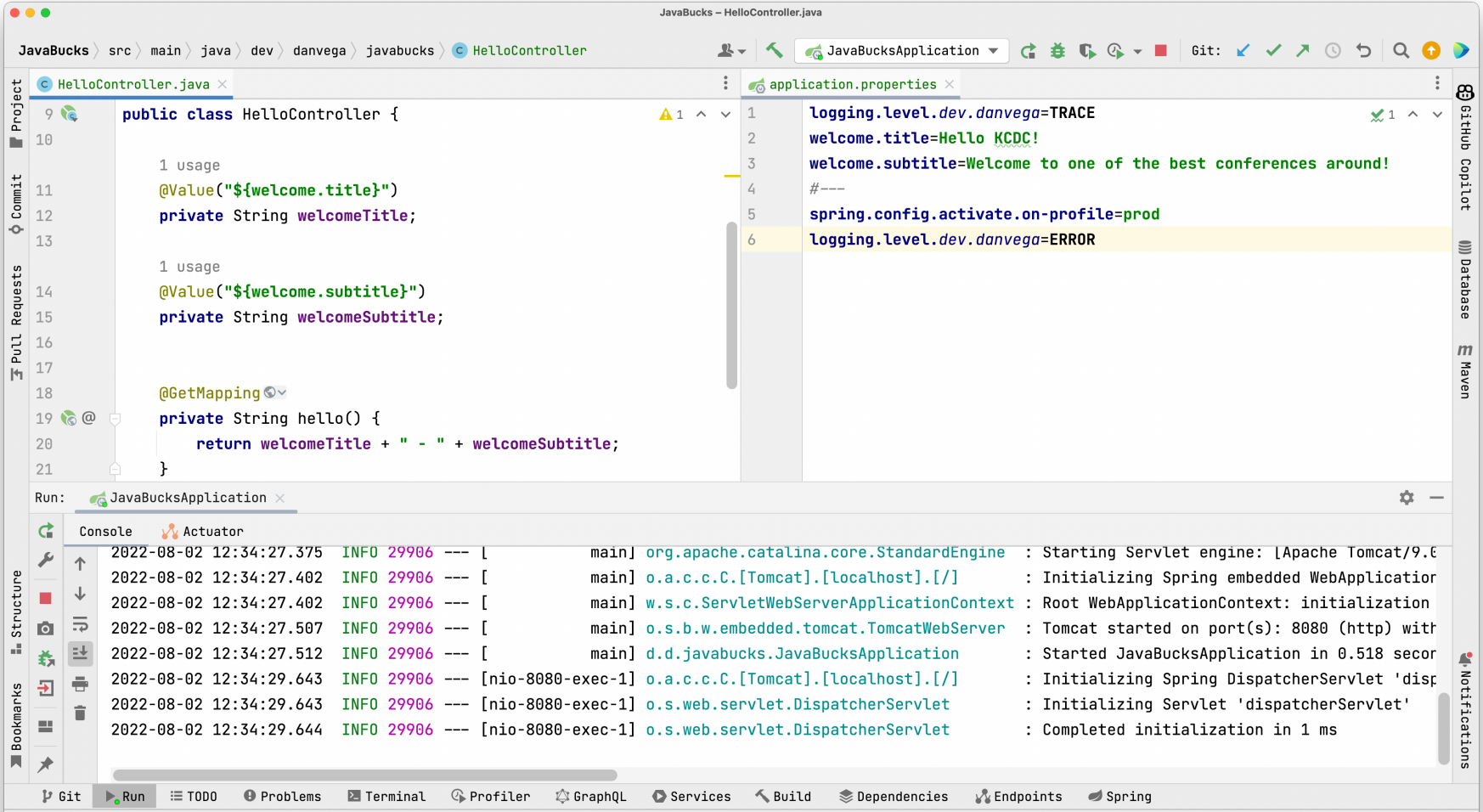

JavaBucks – application.properties
JavaBucks 〉 src 〉 main 〉 resources 〉 application.properties
JavaBucksApplication
Git:
Project
JavaBucks ~/dev/presentations/spring-rec
.idea
.mvn
src
main
java
dev.danvega.javabucks
JavaBucksApplication
resources
static
templates
application.properties
1 logging.level.dev.danvega=TRACE
2 #---
3 spring.config.activate.on-profile=prod
4 logging.level.dev.danvega=ERROR
Run: JavaBucksApplication
Console
Actuator
:: Spring Boot :: (v2.7.2)
2022-08-02 12:15:51.293 INFO 28216 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http
2022-08-02 12:15:51.296 INFO 28216 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-08-02 12:15:51.296 INFO 28216 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0
2022-08-02 12:15:51.322 INFO 28216 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicatior
2022-08-02 12:15:51.323 INFO 28216 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization
2022-08-02 12:15:51.427 INFO 28216 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with
2022-08-02 12:15:51.432 ERROR 28216 --- [main] d.d.javabucks.JavaBucksApplication : ERROR
Git
Run
TODO
Problems
Terminal
Profiler
GraphQL
Services
Build
Dependencies
Endpoints
Spring
All files are up-to-date (moments ago)
3:39 LF ISO-8859-1 4 spaces master
GitHub Copilot
Database
Maven
Notifications
Structure
Bookmarks
Pull Requests
Commit

Problem

You want to inject values from configuration into your application.

Solution

If you want to inject a single value into your application you can use the ***@Value*** annotation. If you have a group of related properties you can use ***@ConfigurationProperties*** on a POJO or Record.


JavaBucks – HelloController.java
JavaBucks 〉 src 〉 main 〉 java 〉 dev 〉 danvega 〉 javabucks 〉 HelloController
JavaBucksApplication
Git:
HelloController.java
public class HelloController {
1 usage
@Value("${welcome.title}")
private String welcomeTitle;
1 usage
@Value("${welcome.subtitle}")
private String welcomeSubtitle;
@GetMapping
private String hello() {
return welcomeTitle + " - " + welcomeSubtitle;
}
application.properties
logging.level.dev.danvega=TRACE
welcome.title=Hello KCDC!
welcome.subtitle=Welcome to one of the best conferences around!
#---
spring.config.activate.on-profile=prod
logging.level.dev.danvega=ERROR
Run: JavaBucksApplication
Console
Actuator
2022-08-02 12:34:27.375 INFO 29906 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0
2022-08-02 12:34:27.402 INFO 29906 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicatior
2022-08-02 12:34:27.402 INFO 29906 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization
2022-08-02 12:34:27.507 INFO 29906 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with
2022-08-02 12:34:27.512 INFO 29906 --- [main] d.d.javabucks.JavaBucksApplication : Started JavaBucksApplication in 0.518 secor
2022-08-02 12:34:29.643 INFO 29906 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'disp
2022-08-02 12:34:29.643 INFO 29906 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-08-02 12:34:29.644 INFO 29906 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
Git
Run
TODO
Problems
Terminal
Profiler
GraphQL
Services
Build
Dependencies
Endpoints
Spring
JavaBucksApplication: Failed to retrieve application JMX service URL (3 minutes ago)
24:1 LF UTF-8 4 spaces master
Project
Commit
Pull Requests
Structure
Bookmarks
GitHub Copilot
Database
Maven
Notifications

```java
package dev.danvega.javabucks;

import org.springframework.boot.context.properties.ConfigurationProperties;

@ConfigurationProperties(prefix = "javabucks")
public class JavaBucksProperties {

    private String title;

    private String subtitle;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getSubtitle() {
        return subtitle;
    }

    public void setSubtitle(String subtitle) {
        this.subtitle = subtitle;
    }
}
```

```properties
logging.level.dev.danvega=TRACE
#welcome.title=Hello KCDC!
#welcome.subtitle=Welcome to one of the best conferences around!
javabucks.title=Hello KCDC
javabucks.subtitle=Welcome to one of the best conferences around!
#---
spring.config.activate.on-profile=prod
logging.level.dev.danvega=ERROR
```

```java
@ConfigurationProperties(prefix = "javabucks")
public record JavaBucksProperties(String title, String subtitle) {

}
```

Problem

I don’t want to have to restart my application after every single change.

Solution

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

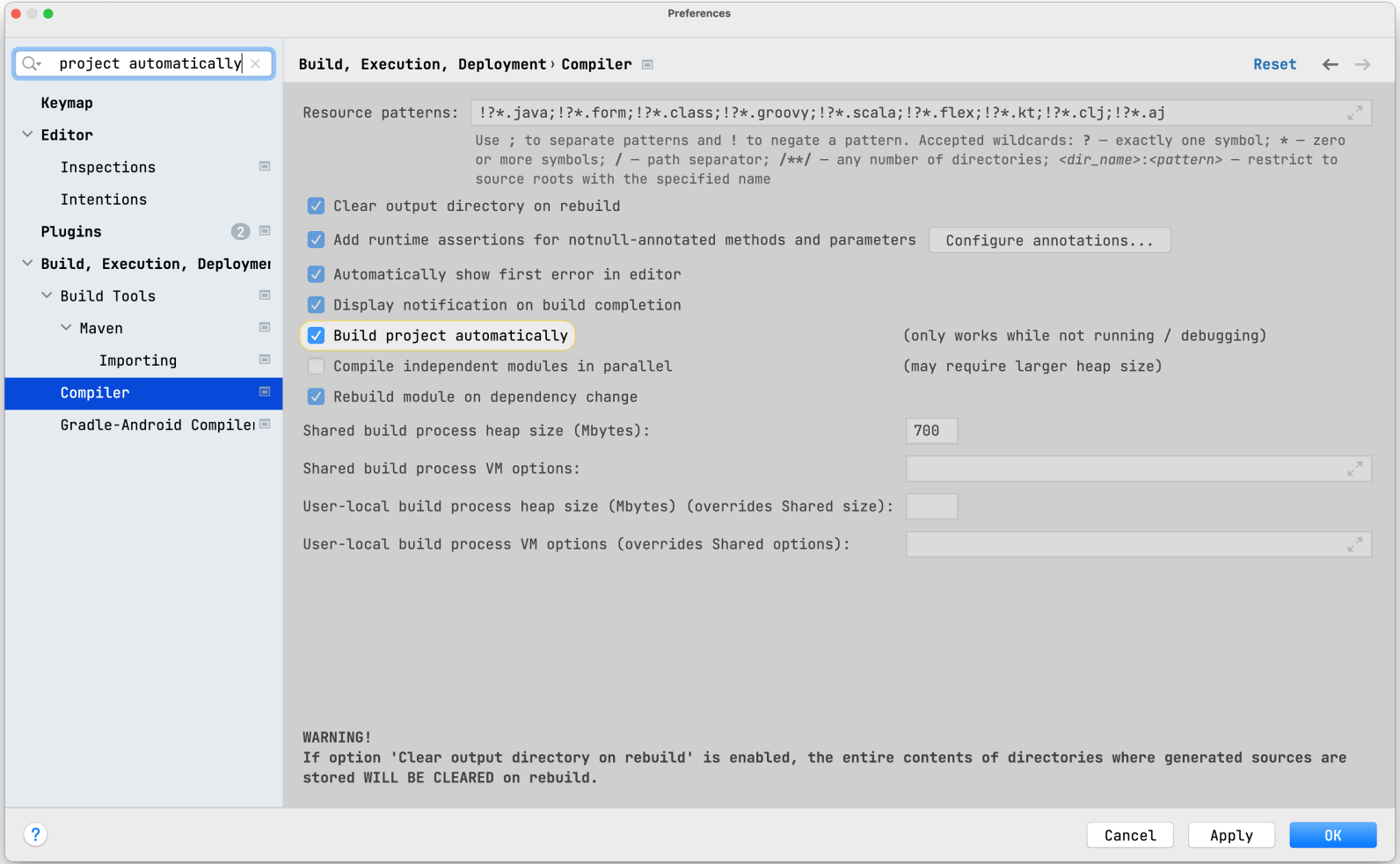
Maven

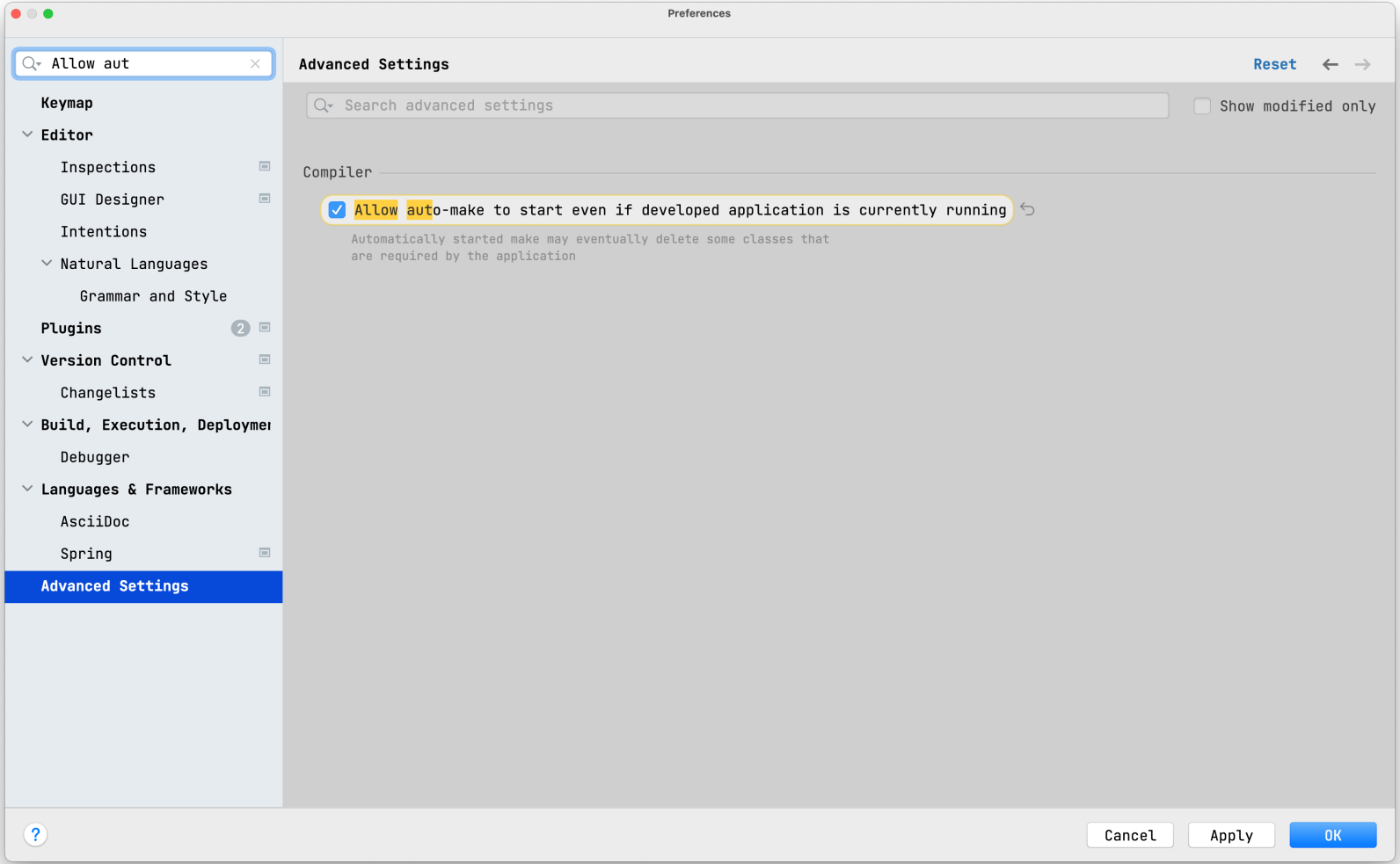
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <optional>true</optional>
</dependency>
```

Gradle

```
dependencies {
    developmentOnly("org.springframework.boot:spring-boot-devtools")
}
```

Name	Default Value
server.error.include-binding-errors	always
server.error.include-message	always
server.error.include-stacktrace	always
server.servlet.jsp.init-parameters.development	TRUE
server.servlet.session.persistent	TRUE
spring.freemarker.cache	FALSE
spring.graphql.graphiql.enabled	TRUE
spring.groovy.template.cache	FALSE
spring.h2.console.enabled	TRUE
spring.mustache.servlet.cache	FALSE
spring.mvc.log-resolved-exception	TRUE
spring.reactor.debug	TRUE
spring.template.provider.cache	FALSE
spring.thymeleaf.cache	FALSE
spring.web.resources.cache.period	0


Preferences
project automatically
Keymap
Editor
Inspections
Intentions
Plugins
2
Build, Execution, Deployme
Build Tools
Maven
Importing
Compiler
Gradle-Android Compile
Build, Execution, Deployment › Compiler
Reset
Resource patterns:
!?*.java;!?*.form;!?*.class;!?*.groovy;!?*.scala;!?*.flex;!?*.kt;!?*.clj;!?*.aj
Use ; to separate patterns and ! to negate a pattern. Accepted wildcards: ? — exactly one symbol; * — zero or more symbols; / — path separator; /**/ — any number of directories; <dir_name>:<pattern> — restrict to source roots with the specified name
Clear output directory on rebuild
Add runtime assertions for notnull-annotated methods and parameters
Configure annotations...
Automatically show first error in editor
Display notification on build completion
Build project automatically
(only works while not running / debugging)
Compile independent modules in parallel
(may require larger heap size)
Rebuild module on dependency change
Shared build process heap size (Mbytes):
700
Shared build process VM options:
User-local build process heap size (Mbytes) (overrides Shared size):
User-local build process VM options (overrides Shared options):
WARNING!
If option 'Clear output directory on rebuild' is enabled, the entire contents of directories where generated sources are stored WILL BE CLEARED on rebuild.
Cancel
Apply
OK


Preferences
Allow aut
Keymap
Editor
Inspections
GUI Designer
Intentions
Natural Languages
Grammar and Style
Plugins
2
Version Control
Changelists
Build, Execution, Deploymer
Debugger
Languages & Frameworks
AsciiDoc
Spring
Advanced Settings
Advanced Settings
Reset
Search advanced settings
Show modified only
Compiler
Allow auto-make to start even if developed application is currently running
Automatically started make may eventually delete some classes that
are required by the application
Cancel
Apply
OK

Building Web Applications

Problem

I want to create a REST API that any number of clients can talk to.

Solution

You can create a REST API using the ***@RestController*** annotation.

```java
public class CoffeeController {

    public String home() {
        return "Hello 👋 KCDC";
    }

}
```

```java
@RestController
public class CoffeeController {

    @GetMapping
    public String home() {
        return "Hello 👋 KCDC";
    }

}
```

```java
@RestController
public class CoffeeController {

    public List<String> drinks = new ArrayList<>();

    public CoffeeController() {
        drinks.add("Caffè Americano - GRANDE");
        drinks.add("Caffè Latte - VENTI");
        drinks.add("Caffè Caramel Macchiato - TALL");
    }

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public List<String> findAll() {
        return drinks;
    }

    @GetMapping("/{name}")
    public String findOne(@PathVariable String name) {
        return drinks.stream()
                .filter(s -> s.startsWith(name))
                .findFirst()
                .orElseThrow(() -> new RuntimeException("Not Found"));
    }

}
```

```java
@RestController
@RequestMapping("/api/coffee")
public class CoffeeController {

    public List<String> drinks = new ArrayList<>();

    public CoffeeController() {
        drinks.add("Caffè Americano - GRANDE");
        drinks.add("Caffè Latte - VENTI");
        drinks.add("Caffè Caramel Macchiato - TALL");
    }

    @GetMapping
    public List<String> findAll() {
        return drinks;
    }

    @GetMapping("/{name}")
    public String findOne(@PathVariable String name) {
        return drinks.stream()
                .filter(s -> s.startsWith(name))
                .findFirst()
                .orElseThrow(() -> new RuntimeException("Not Found"));
    }

}
```

Problem

I need to validate the data that a client is sending to my REST API.

Solution

The Spring Framework provides support for the Java Bean Validation API.

```java
@PostMapping
public Coffee create(@RequestBody Coffee coffee) {
    return coffeeService.create(coffee);
}
```

```
POST http://localhost:8080/api/coffee
Content-Type: application/json

{
  "id": null,
  "name": "Cafè Latte",
  "size": "SHORT",
  "price": 3.99,
  "cost": 1.99
}
```

```java
@PostMapping
public Coffee create(@RequestBody Coffee coffee) {
    return coffeeService.create(coffee);
}
```

```
POST http://localhost:8080/api/coffee
Content-Type: application/json

{
  "id": null,
  "name": "",
  "size": "SHORT",
  "price": 199.99,
  "cost": 1.99
}
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

```java
@PostMapping
public Coffee create(@Valid @RequestBody Coffee coffee) {
    return coffeeService.create(coffee);
}
```

```java
package dev.danvega.javabucksrest.model;

import javax.validation.constraints.Max;
import javax.validation.constraints.NotEmpty;
import java.math.BigDecimal;

public record Coffee(
        Integer id,
        @NotEmpty
        String name,
        Size size,
        @Max(10)
        BigDecimal price,
        BigDecimal cost) {
}
```

```
{
  "timestamp": "2022-08-03T14:00:30.111+00:00",
  "status": 400,
  "error": "Bad Request",
  "message": "Validation failed for object='coffee'. Error count: 2",
  "path": "/api/coffee"
}
```

```
server.error.include-message=always
server.error.include-binding-errors=always
```

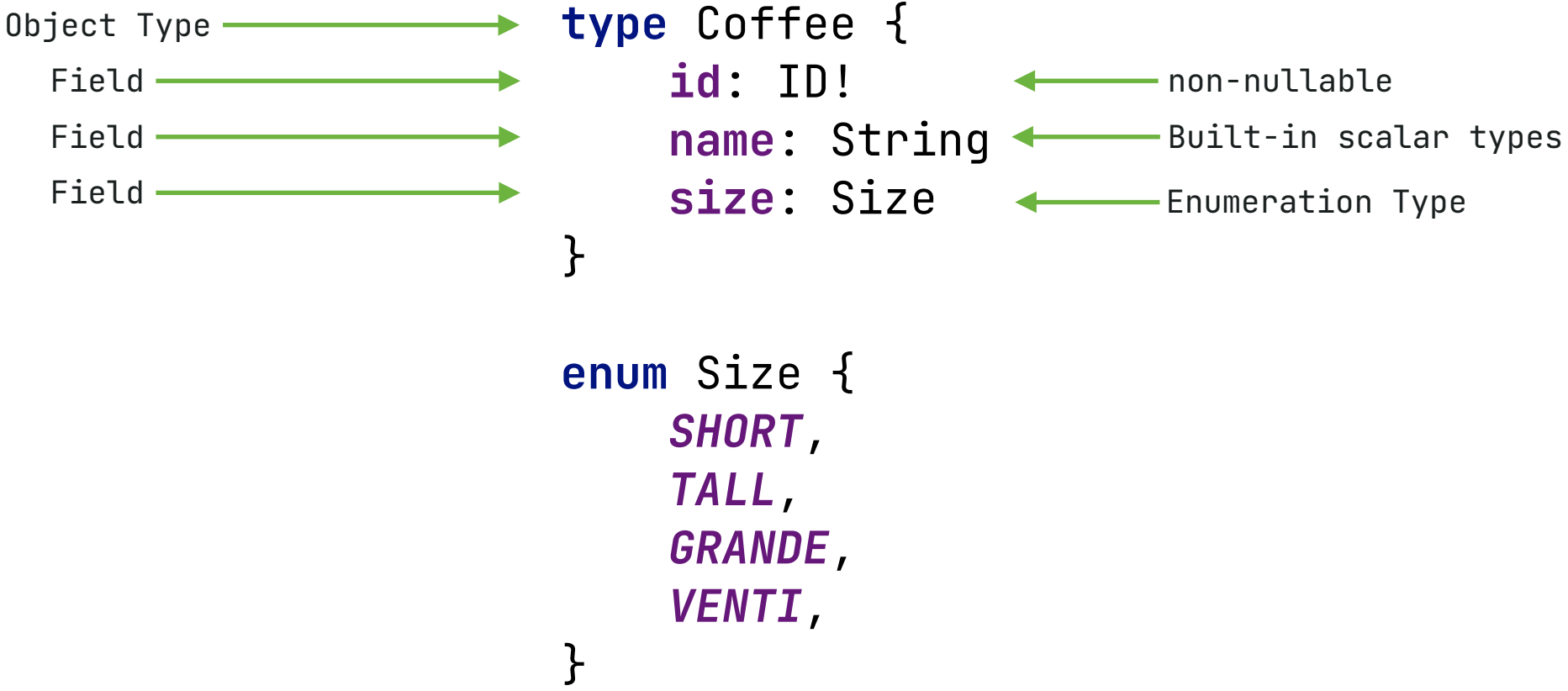
```
{
  "timestamp": "2022-08-03T13:59:36.327+00:00",
  "status": 400,
  "error": "Bad Request",
  "message": "Validation failed for object='coffee'. Error count: 2",
  "errors": [
    {
      "codes": [
        "Max.coffee.price",
        "Max.price",
        "Max.java.math.BigDecimal",
        "Max"
      ],
      "arguments": [
        {
          "codes": [
            "coffee.price",
            "price"
          ],
          "arguments": null,
          "defaultMessage": "price",
          "code": "price"
        },
        10
```

Problem

I want to build a GraphQL API with Spring Boot.

Solution

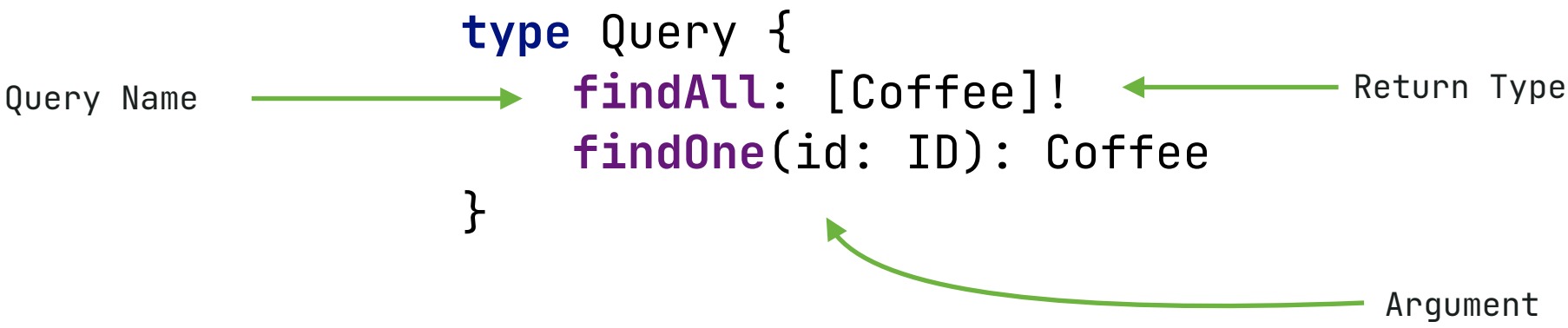
Spring for GraphQL provides support for Spring applications built on GraphQL Java.



src/main/resources/graphql/schema.graphqls

Operation Types

Most types in your schema will just be normal object types, but there are three types that are special within a schema:


type Query {
findAll: [Coffee]!
findOne(id: ID): Coffee
}
Query Name
Return Type
Argument

```java
@Controller
public class CoffeeController {

    private final CoffeeService coffeeService;

    public CoffeeController(CoffeeService coffeeService) {
        this.coffeeService = coffeeService;
    }

    public List<Coffee> list() {
        return coffeeService.findAll();
    }

    public Optional<Coffee> findOne(Integer id) {
        return coffeeService.findOne(id);
    }

}
```

```java
@Controller
public class CoffeeController {

    private final CoffeeService coffeeService;

    public CoffeeController(CoffeeService coffeeService) {
        this.coffeeService = coffeeService;
    }

    @SchemaMapping(typeName = "Query", value = "findAll")
    public List<Coffee> list() {
        return coffeeService.findAll();
    }

    public Optional<Coffee> findOne(Integer id) {
        return coffeeService.findOne(id);
    }

}
```

```java
@Controller
public class CoffeeController {

    private final CoffeeService coffeeService;

    public CoffeeController(CoffeeService coffeeService) {
        this.coffeeService = coffeeService;
    }

    @QueryMapping
    public List<Coffee> findAll() {
        return coffeeService.findAll();
    }

    @QueryMapping
    public Optional<Coffee> findOne(@Argument Integer id) {
        return coffeeService.findOne(id);
    }

}
```

Problem

I need to create a client to call another service.

Solution

There are many solutions to make service to service calls in Java and Spring. In the current versions of Spring you have access to Rest Template and Web Client. In Spring Framework 6 and Spring Boot 3 you will have access to declarative HTTP Clients.

```java
@RestController
@RequestMapping("/api/coffee")
public class CoffeeController {

    private final CoffeeService coffeeService;
    private final RestTemplate restTemplate;

    public CoffeeController(CoffeeService coffeeService, RestTemplate restTemplate) {
        this.coffeeService = coffeeService;
        this.restTemplate = restTemplate;
    }

    @GetMapping
    public List<Coffee> findAll() {
        return coffeeService.findAll();
    }

    @GetMapping("/random")
    public Coffee random() {
        return coffeeService.create(
                restTemplate.getForObject("https://coffee.alexflipnote.dev/random.json", Coffee.class)
        );
    }
}
```

```java
@RestController
@RequestMapping("/api/coffee")
public class CoffeeController {

    WebClient client = WebClient.create("https://coffee.alexflipnote.dev");

    @GetMapping("/random")
    public Mono<Coffee> random() {
        return client.get()
                .uri("/random.json")
                .retrieve()
                .bodyToMono(Coffee.class);
    }
}
```

```java
@SpringBootApplication
public class Application {

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

	@Bean
	CoffeeClient coffeeClient() throws Exception {
		WebClient webClient = WebClient.builder()
				.baseUrl("https://coffee.alexflipnote.dev")
				.build();

		HttpServiceProxyFactory factory = WebClientAdapter.createHttpServiceProxyFactory(webClient);
		factory.afterPropertiesSet();

		return factory.createClient(CoffeeClient.class);
	}
}
```

```java
package dev.danvega.randomcoffeedeclarative.client;

import dev.danvega.randomcoffeedeclarative.model.Coffee;
import org.springframework.web.service.annotation.GetExchange;

public interface CoffeeClient {

    @GetExchange("/random.json")
    Coffee random();

}
```

```java
@RestController
@RequestMapping("/api/coffee")
public class CoffeeController {

    private final CoffeeClient coffeeClient;

    public CoffeeController(CoffeeClient coffeeClient) {
        this.coffeeClient = coffeeClient;
    }

    @GetMapping("/random")
    public Coffee random() {
        return coffeeClient.random();
    }
}
```

Working with Databases

Problem

I need to create an application that can connect to a database.

Solution

Connecting to a database with Spring Boot is really easy. The quickest way to get up and running is by selecting an in-memory database.

spring initializr

Project

- (●) Maven Project
- ( ) Gradle Project

Language

- (●) Java
- ( ) Kotlin
- ( ) Groovy

Spring Boot

- ( ) 3.0.0 (SNAPSHOT)
- (●) 3.0.0 (M4)
- ( ) 2.7.3 (SNAPSHOT)
- ( ) 2.7.2
- ( ) 2.6.11 (SNAPSHOT)
- ( ) 2.6.10

Project Metadata

Group	dev.danvega
Artifact	h2-demo
Name	h2-demo
Description	Demo project for Spring Boot
Package name	dev.danvega.h2-demo
Packaging	(●) Jar ( ) War
Java	( ) 18 (●) 17 ( ) 11 ( ) 8

Dependencies ADD DEPENDENCIES... ⌘ + B

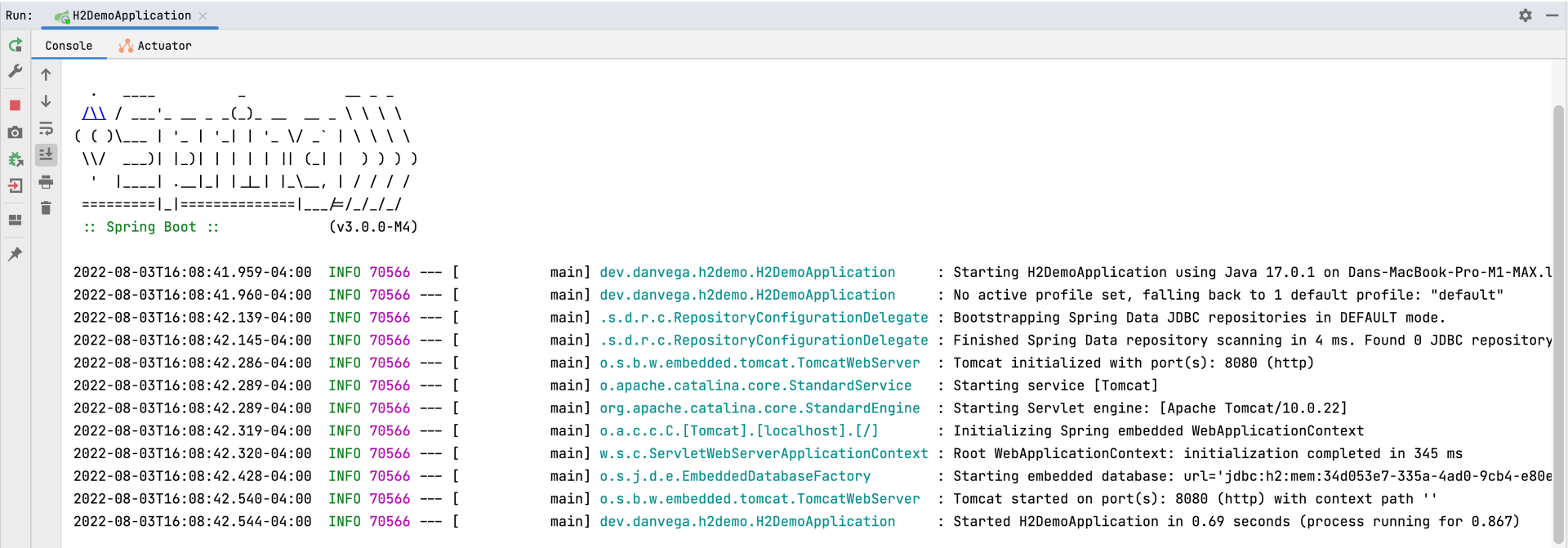
Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JDBC SQL
Persist data in SQL stores with plain JDBC using Spring Data.

H2 Database SQL
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

GENERATE ⌘ + ↵ EXPLORE CTRL + SPACE SHARE...

Starting embedded database: url='jdbc:h2:mem:89060c30-14bd-4f81-8606-3c6f274c94bf;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=false', username='sa'



```
spring.datasource.generate-unique-name=false
spring.datasource.name=coffee
spring.h2.console.enabled=true
```

application.properties

Starting embedded database: url='jdbc:h2:mem:coffee;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=false', username='sa'

H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:coffee'

English Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:coffee

User Name: sa

Password:

Connect Test Connection

Test successful

jdbc:h2:mem:coffee
INFORMATION_SCHEMA
Users
H2 2.1.214 (2022-06-13)

Run | Run Selected | Auto complete | Clear | SQL statement:

```
select * from coffee
```

Important Commands

		Displays this Help Page
		Shows the Command History
	Ctrl+Enter	Executes the current SQL statement
	Shift+Enter	Executes the SQL statement defined by the text selection
	Ctrl+Space	Auto complete
		Disconnects from the database

Sample SQL Script

Delete the table if it exists	DROP TABLE IF EXISTS TEST;
Create a new table with ID and NAME columns	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Add a new row	INSERT INTO TEST VALUES(1, 'Hello');
Add another row	INSERT INTO TEST VALUES(2, 'World');
Query the table	SELECT * FROM TEST ORDER BY ID;

Problem

That was a great example but how can I connect to a production ready database?

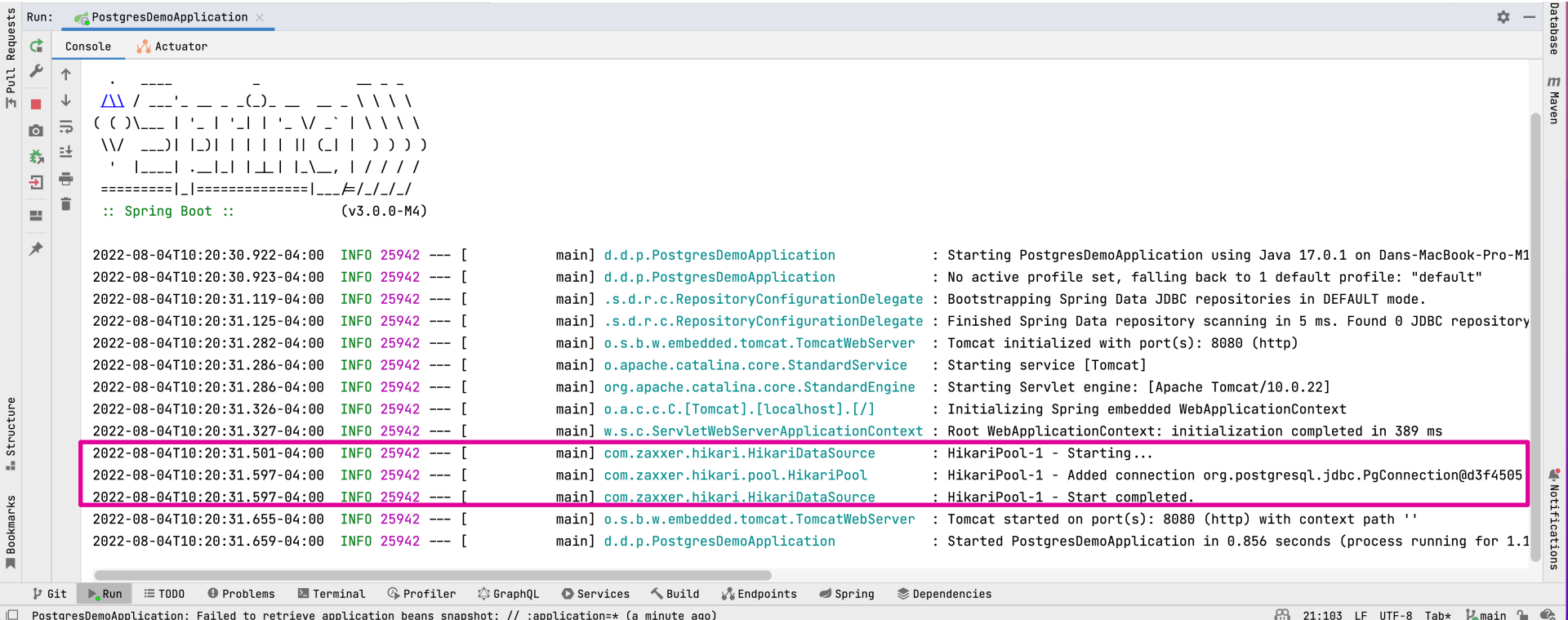
Solution

Connecting to a database like Postgres is just a few more lines of configuration.

```yaml
version: '3.8'

services:
  db:
    image: postgres:alpine
    ports:
      - "5432:5432"
    environment:
      POSTGRES_PASSWORD: password
      POSTGRES_DB: coffee
```

```
spring.datasource.url=jdbc:postgresql://localhost:5432/coffee
spring.datasource.username=postgres
spring.datasource.password=password
```


Run: PostgresDemoApplication
Console
Actuator
:: Spring Boot :: (v3.0.0-M4)
2022-08-04T10:20:30.922-04:00 INFO 25942 --- [main] d.d.p.PostgresDemoApplication : Starting PostgresDemoApplication using Java 17.0.1 on Dans-MacBook-Pro-M1
2022-08-04T10:20:30.923-04:00 INFO 25942 --- [main] d.d.p.PostgresDemoApplication : No active profile set, falling back to 1 default profile: "default"
2022-08-04T10:20:31.119-04:00 INFO 25942 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JDBC repositories in DEFAULT mode.
2022-08-04T10:20:31.125-04:00 INFO 25942 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 5 ms. Found 0 JDBC repository
2022-08-04T10:20:31.282-04:00 INFO 25942 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-08-04T10:20:31.286-04:00 INFO 25942 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-08-04T10:20:31.286-04:00 INFO 25942 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.0.22]
2022-08-04T10:20:31.326-04:00 INFO 25942 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-08-04T10:20:31.327-04:00 INFO 25942 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 389 ms
2022-08-04T10:20:31.501-04:00 INFO 25942 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2022-08-04T10:20:31.597-04:00 INFO 25942 --- [main] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection org.postgresql.jdbc.PgConnection@d3f4505
2022-08-04T10:20:31.597-04:00 INFO 25942 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2022-08-04T10:20:31.655-04:00 INFO 25942 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2022-08-04T10:20:31.659-04:00 INFO 25942 --- [main] d.d.p.PostgresDemoApplication : Started PostgresDemoApplication in 0.856 seconds (process running for 1.1
Git
Run
TODO
Problems
Terminal
Profiler
GraphQL
Services
Build
Endpoints
Spring
Dependencies
PostgresDemoApplication: Failed to retrieve application beans snapshot: // :application=* (a minute ago)
21:103 LF UTF-8 Tab* main

Problem

Now that I am connected to a database how can I run a DDL script to create tables, columns and data?

Solution

You can create a SQL script by convention in *src/main/resources/schema.sql*.

```sql
DROP TABLE IF EXISTS coffee;

CREATE TABLE coffee(
    id SERIAL NOT NULL PRIMARY KEY,
    name varchar(255) NOT NULL,
    size varchar(15) NOT NULL
);
```

src/main/resources/schema.sql

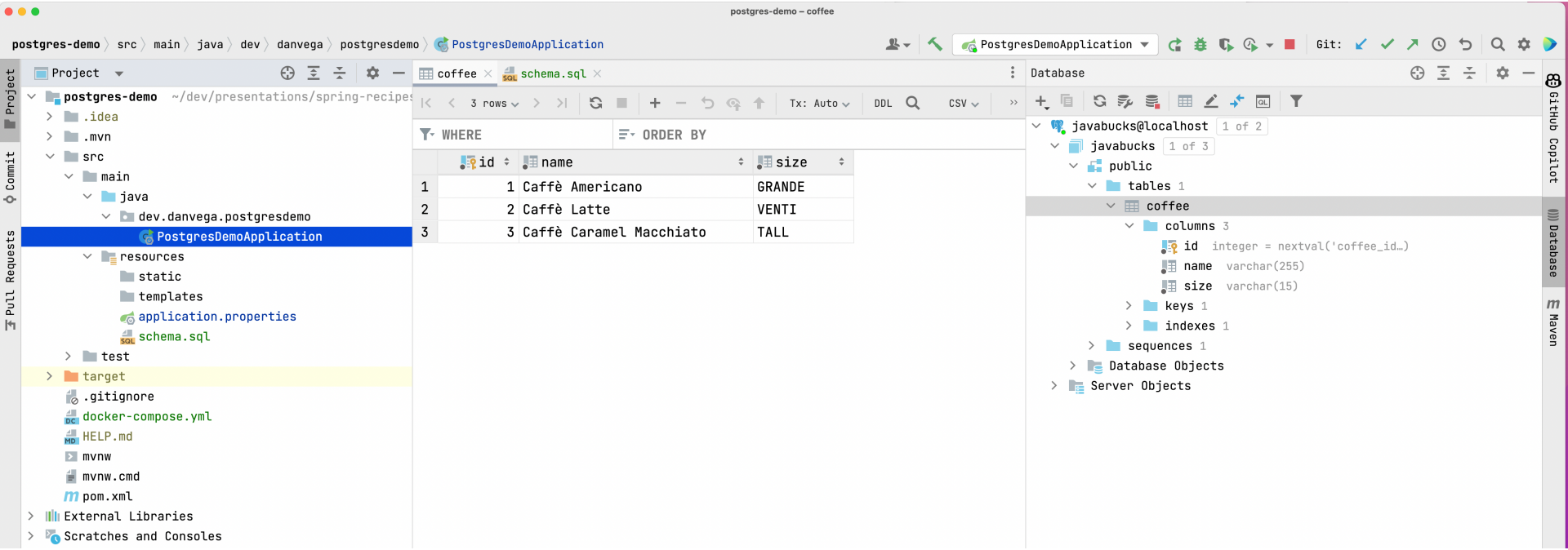
```
spring.datasource.url=jdbc:postgresql://localhost:5432/javabucks
spring.datasource.username=postgres
spring.datasource.password=password

spring.sql.init.mode=always
```

```sql
DROP TABLE IF EXISTS coffee;

CREATE TABLE coffee(
    id SERIAL NOT NULL PRIMARY KEY,
    name varchar(255) NOT NULL,
    size varchar(15) NOT NULL
);
```

```sql
INSERT INTO coffee(name,size) VALUES('Caffè Americano','GRANDE');
INSERT INTO coffee(name,size) VALUES('Caffè Latte','VENTI');
INSERT INTO coffee(name,size) VALUES('Caffè Caramel Macchiato','TALL');
```


postgres-demo – coffee
postgres-demo 〉 src 〉 main 〉 java 〉 dev 〉 danvega 〉 postgresdemo 〉 PostgresDemoApplication
PostgresDemoApplication
Git:
Project
postgres-demo ~/dev/presentations/spring-recipes
.idea
.mvn
src
main
java
dev.danvega.postgresdemo
PostgresDemoApplication
resources
static
templates
application.properties
schema.sql
test
target
.gitignore
docker-compose.yml
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles
coffee
schema.sql
3 rows
Tx: Auto
DDL
CSV
WHERE
ORDER BY
id
name
size
1 1 Caffè Americano GRANDE
2 2 Caffè Latte VENTI
3 3 Caffè Caramel Macchiato TALL
Database
javabucks@localhost 1 of 2
javabucks 1 of 3
public
tables 1
coffee
columns 3
id integer = nextval('coffee_id…)
name varchar(255)
size varchar(15)
keys 1
indexes 1
sequences 1
Database Objects
Server Objects
Project
Commit
Pull Requests
GitHub Copilot
Database
Maven

Problem

You want to be able to insert data programmatically using Java & Spring.

Solution

You can use JDBC Template or Spring Data to persist data. The ***ApplicationRunner*** and ***CommandLineRunner*** interfaces in Spring are a great way to bootstrap data.

org.springframework.jdbc.core

Class JdbcTemplate

java.lang.Object
  org.springframework.jdbc.support.JdbcAccessor
    org.springframework.jdbc.core.JdbcTemplate

All Implemented Interfaces:

`InitializingBean`, `JdbcOperations`

```
public class JdbcTemplate
extends JdbcAccessor
implements JdbcOperations
```

This is the central class in the JDBC core package. It simplifies the use of JDBC and helps to avoid common errors. It executes core JDBC workflow, leaving application code to provide SQL and extract results. This class executes SQL queries or updates, initiating iteration over ResultSets and catching JDBC exceptions and translating them to the generic, more informative exception hierarchy defined in the `org.springframework.dao` package.

Code using this class need only implement callback interfaces, giving them a clearly defined contract. The `PreparedStatementCreator` callback interface creates a prepared statement given a Connection, providing SQL and any necessary parameters. The `ResultSetExtractor` interface extracts values from a ResultSet. See also `PreparedStatementSetter` and `RowMapper` for two popular alternative callback interfaces.

Can be used within a service implementation via direct instantiation with a DataSource reference, or get prepared in an application context and given to services as bean reference. Note: The DataSource should always be configured as a bean in the application context, in the first case given to the service directly, in the second case to the prepared template.

Because this class is parameterizable by the callback interfaces and the `SQLExceptionTranslator` interface, there should be no need to subclass it.

All SQL operations performed by this class are logged at debug level, using "org.springframework.jdbc.core.JdbcTemplate" as log category.

NOTE: An instance of this class is thread-safe once configured.

Since:

May 3, 2001

Author:

Rod Johnson, Juergen Hoeller, Thomas Risberg

See Also:

`PreparedStatementCreator`, `PreparedStatementSetter`, `CallableStatementCreator`, `PreparedStatementCallback`, `CallableStatementCallback`, `ResultSetExtractor`, `RowCallbackHandler`, `RowMapper`, `SQLExceptionTranslator`

Field Summary

Fields inherited from class org.springframework.jdbc.support.JdbcAccessor

`logger`

```java
public class CoffeeDAO {

    private static final Logger LOG = LoggerFactory.getLogger(CoffeeDAO.class);
    private final JdbcTemplate jdbc;

    public CoffeeDAO(JdbcTemplate jdbc) {
        this.jdbc = jdbc;
    }

    public void create(Coffee coffee) {
        String sql = "insert into coffee(name,size) values(?,?)";
        int insert = jdbc.update(sql,coffee.name(),coffee.size().toString());
        if(insert == 1) {
            LOG.info("New Coffee Created: " + coffee.name());
        }
    }

}
```

org.springframework.boot

Interface CommandLineRunner

Functional Interface:

This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

```
@FunctionalInterface
public interface CommandLineRunner
```

Interface used to indicate that a bean should *run* when it is contained within a `SpringApplication`. Multiple `CommandLineRunner` beans can be defined within the same application context and can be ordered using the `Ordered` interface or `@Order` annotation.

If you need access to `ApplicationArguments` instead of the raw String array consider using `ApplicationRunner`.

Since:

1.0.0

Author:

Dave Syer

See Also:

`ApplicationRunner`

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method and Description
void	`run(String... args)` Callback used to run the bean.

Method Detail

run

```
void run(String... args)
  throws Exception
```

Callback used to run the bean.

Parameters:

```java
@SpringBootApplication
public class PostgresDemoApplication {


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

    @Bean
    CommandLineRunner runner(CoffeeDAO dao) {
        return args -> {
            dao.create(new Coffee(null,"Caffè Americano", Size.GRANDE));
            dao.create(new Coffee(null,"Caffè Latte", Size.VENTI));
            dao.create(new Coffee(null,"Caffè Caramel Macchiato", Size.TALL));
        };
    }

}
```

```java
@Component
public class DatabaseLoader implements CommandLineRunner {

    private static final Logger LOG = LoggerFactory.getLogger(DatabaseLoader.class);

    private final CoffeeRepository repository;

    public DatabaseLoader(CoffeeRepository coffeeRepository) {
        this.repository = coffeeRepository;
    }

    @Override
    public void run(String... args) throws Exception {
        List<Coffee> coffees = List.of(new Coffee(null, "Caffè Americano", Size.GRANDE),
                new Coffee(null, "Caffè Latte", Size.VENTI),
                new Coffee(null, "Caffè Caramel Macchiato", Size.TALL));
        repository.saveAll(coffees);
    }

}
```

Problem

I want to retrieve and persist information from a database that I am connected to.

Solution

You already saw a solution for interacting with your database using ***JDBCTemplate***. Spring Data's mission is to provide a familiar and consistent, Spring-based programming model for data access while still retaining the special traits of the underlying data store.

```java
@Component
public class CoffeeDAO {

    private static final Logger LOG = LoggerFactory.getLogger(CoffeeDAO.class);
    private final JdbcTemplate jdbc;

    public CoffeeDAO(JdbcTemplate jdbc) {
        this.jdbc = jdbc;
    }

    RowMapper<Coffee> rowMapper = (rs, rowNum) -> {
        return new Coffee(
                rs.getInt("id"),
                rs.getString("name"),
                Size.valueOf(rs.getString("size"))
        );
    };

    public List<Coffee> list() {
        String sql = "SELECT id,name,size from coffee";
        return jdbc.query(sql,rowMapper);
    }

}
```

```
@Bean
CommandLineRunner runner(CoffeeDAO dao) {
    return args -> {
        dao.create(new Coffee(null,"Caffè Americano", Size.GRANDE));
        dao.create(new Coffee(null,"Caffè Latte", Size.VENTI));
        dao.create(new Coffee(null,"Caffè Caramel Macchiato", Size.TALL));

        LOG.info("Fetching all rows from coffee table ------------------------");
        dao.list().forEach(System.out::println);
    };
}
```

Spring Data 2021.2.2

OVERVIEW LEARN

Spring Data's mission is to provide a familiar and consistent, Spring-based programming model for data access while still retaining the special traits of the underlying data store.

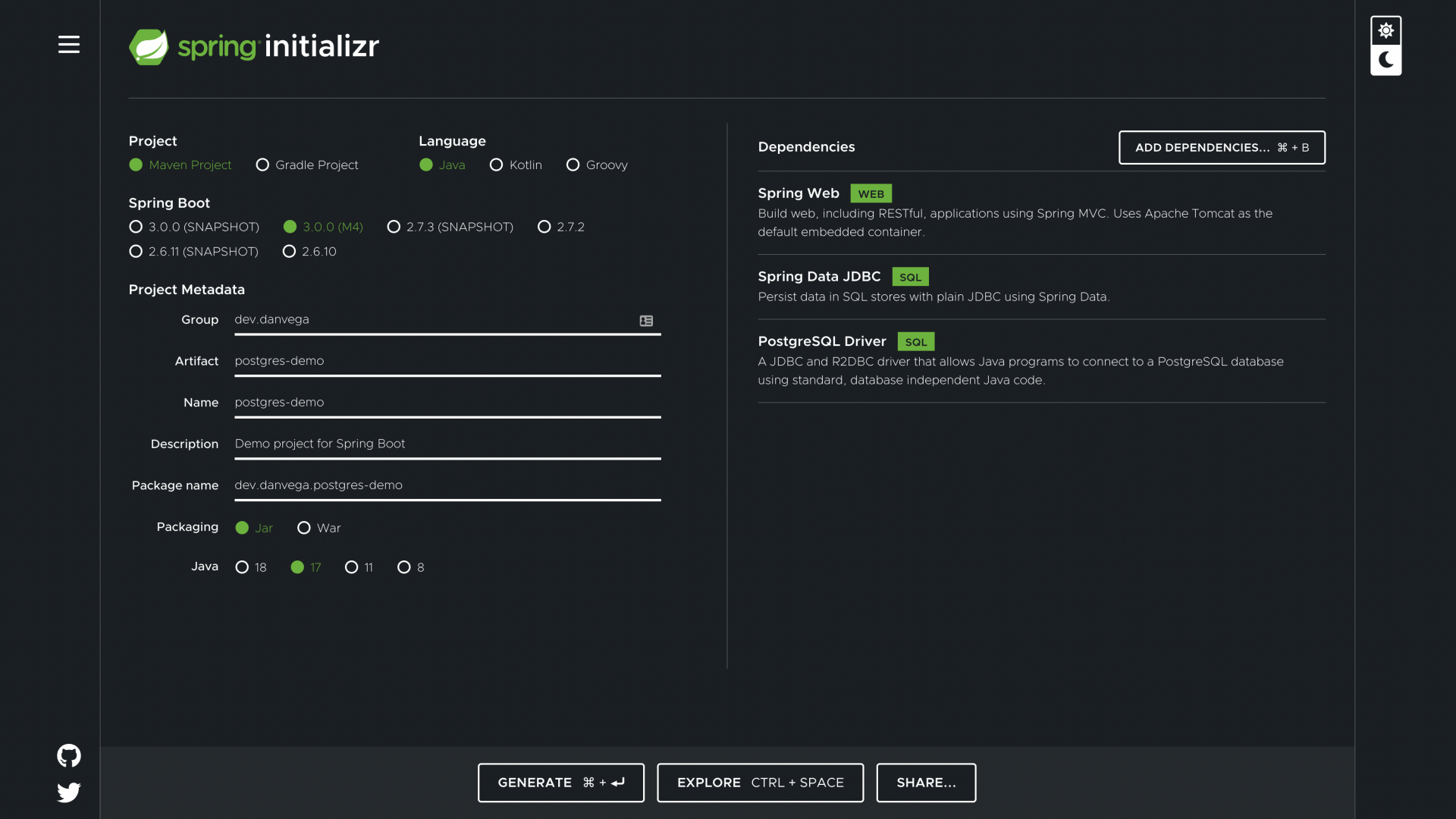
It makes it easy to use data access technologies, relational and non-relational databases, map-reduce frameworks, and cloud-based data services. This is an umbrella project which contains many subprojects that are specific to a given database. The projects are developed by working together with many of the companies and developers that are behind these exciting technologies.

Features

- Powerful repository and custom object-mapping abstractions
- Dynamic query derivation from repository method names
- Implementation domain base classes providing basic properties
- Support for transparent auditing (created, last changed)
- Possibility to integrate custom repository code
- Easy Spring integration via JavaConfig and custom XML namespaces
- Advanced integration with Spring MVC controllers
- Experimental support for cross-store persistence

Main modules

- Spring Data Commons - Core Spring concepts underpinning every Spring Data module.


spring initializr
Project
Maven Project
Gradle Project
Language
Java
Kotlin
Groovy
Spring Boot
3.0.0 (SNAPSHOT)
3.0.0 (M4)
2.7.3 (SNAPSHOT)
2.7.2
2.6.11 (SNAPSHOT)
2.6.10
Project Metadata
Group
dev.danvega
Artifact
postgres-demo
Name
postgres-demo
Description
Demo project for Spring Boot
Package name
dev.danvega.postgres-demo
Packaging
Jar
War
Java
18
17
11
8
Dependencies
ADD DEPENDENCIES... ⌘ + B
Spring Web
WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
Spring Data JDBC
SQL
Persist data in SQL stores with plain JDBC using Spring Data.
PostgreSQL Driver
SQL
A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.
GENERATE ⌘ + ↵
EXPLORE CTRL + SPACE
SHARE...

```java
import dev.danvega.postgresdemo.model.Coffee;
import org.springframework.data.repository.CrudRepository;

public interface CoffeeRepository extends CrudRepository<Coffee,Integer> {


}
```

```java
@Bean
CommandLineRunner runner(CoffeeRepository repository) {
     return args -> {
          List<Coffee> coffees = List.of(new Coffee(null, "Caffè Americano", Size.GRANDE),
                    new Coffee(null, "Caffè Latte", Size.VENTI),
                    new Coffee(null, "Caffè Caramel Macchiato", Size.TALL));
          repository.saveAll(coffees);

          LOG.info("Fetching all rows from coffee table ------------------------");
          repository.findAll().forEach(System.out::println);
     };
}
```

Problem

The CrudRepository returns `Iterable` where I would like to work with a `List`.

Solution

Spring Data 3.0.0 now offers a `ListCrudRepository` which returns a `List` where `CrudRepository` returns an `Iterable`.

```java
@NoRepositoryBean
public interface CrudRepository<T, ID> extends Repository<T, ID> {
    <S extends T> S save(S entity);

    <S extends T> Iterable<S> saveAll(Iterable<S> entities);

    Optional<T> findById(ID id);

    boolean existsById(ID id);

    Iterable<T> findAll();

    Iterable<T> findAllById(Iterable<ID> ids);

    long count();

    void deleteById(ID id);

    void delete(T entity);

    void deleteAllById(Iterable<? extends ID> ids);

    void deleteAll(Iterable<? extends T> entities);

    void deleteAll();
```

```java
import dev.danvega.postgresdemo.model.Coffee;
import org.springframework.data.repository.ListCrudRepository;

public interface CoffeeRepository extends ListCrudRepository<Coffee,Integer> {


}
```

```java
@NoRepositoryBean
public interface ListCrudRepository<T, ID> extends CrudRepository<T, ID> {
    <S extends T> List<S> saveAll(Iterable<S> entities);

    List<T> findAll();

    List<T> findAllById(Iterable<ID> ids);
}
```

Problem

How can I create a Read Only Repository?

Solution

You can extend the Repository interface and only provide read-only methods.

```java
package org.springframework.data.repository;

import org.springframework.stereotype.Indexed;

/**
 * Central repository marker interface. Captures the domain type to manage as well as the domain type's id type. General
 * purpose is to hold type information as well as being able to discover interfaces that extend this one during
 * classpath scanning for easy Spring bean creation.
 * <p>
 * Domain repositories extending this interface can selectively expose CRUD methods by simply declaring methods of the
 * same signature as those declared in {@link CrudRepository}.
 *
 * @see CrudRepository
 * @param <T> the domain type the repository manages
 * @param <ID> the type of the id of the entity the repository manages
 * @author Oliver Gierke
 */
@Indexed
public interface Repository<T, ID> {

}
```

```java
import dev.danvega.postgresdemo.model.Coffee;
import org.springframework.data.repository.Repository;

import java.util.List;
import java.util.Optional;

public interface CoffeeReadOnlyRepository extends Repository<Coffee,Integer> {

    List<Coffee> findAll();

    Optional<Coffee> findById(Integer id);

}
```

Spring Cloud

Problem

You need to monitor various metrics, information or the health information of a service or application.

Solution

Spring Boot Actuator allows you to monitor and interact with your application via a number of pre built endpoints. Developers can also create custom endpoints.

Back to index

1. Enabling Production-ready Features
2. Endpoints
2.1. Enabling Endpoints
2.2. Exposing Endpoints
2.3. Securing HTTP Endpoints
2.4. Configuring Endpoints
2.5. Hypermedia for Actuator Web Endpoints
2.6. CORS Support
2.7. Implementing Custom Endpoints
2.8. Health Information
2.9. Kubernetes Probes
2.10. Application Information
3. Monitoring and Management over HTTP
4. Monitoring and Management over JMX
5. Loggers
6. Metrics
7. Auditing
8. HTTP Tracing
9. Process Monitoring
10. Cloud Foundry Support
11. What to Read Next

2. 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 and exposed (made remotely accessible) over HTTP or JMX. 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.

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
</dependencies>
```

```
management:
  endpoint:
    shutdown:
      enabled: true
```

Problem

Your application needs to degrade gracefully when services fail.

Solution

Spring Cloud Circuit Breaker provides an abstraction over various implementations of the circuit breaker pattern allowing your application to avoid cascading failures.

Spring Cloud Circuit Breaker 1.0.4.RELEASE

Introduction

Spring Cloud Circuit breaker provides an abstraction across different circuit breaker implementations. It provides a consistent API to use in your applications allowing you the developer to choose the circuit breaker implementation that best fits your needs for your app.

Supported Implementations

- Netfix Hystrix
- Resilience4J
- Sentinel
- Spring Retry

Core Concepts

To create a circuit breaker in your code you can use the `CircuitBreakerFactory` API. When you include a Spring Cloud Circuit Breaker starter on your classpath a bean implementing this API will automatically be created for you. A very simple example of using this API is given below

COPY

```
@Service
public static class DemoControllerService {
        private RestTemplate rest;
        private CircuitBreakerFactory cbFactory;

        public DemoControllerService(RestTemplate rest, CircuitBreakerFactory cbFactory) {
                this.rest = rest;
                this.cbFactory = cbFactory;
        }

        public String slow() {
                return cbFactory.create("slow").run(() -> rest.getForObject("/slow",
```

```xml
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>
</dependency>
```

```
CircuitBreakerConfig circuitBreakerConfig = CircuitBreakerConfig.custom()
        .failureRateThreshold(5)
        .waitDurationInOpenState(Duration.ofMillis(300))
        .slidingWindowSize(2)
        .build();
```

```
WaffleController(MacronutrientsProvider macronutrientsProvider, CircuitBreakerFactory circuitBreakerFactory) {
    this.macronutrientsProvider = macronutrientsProvider;
    this.circuitBreaker = circuitBreakerFactory.create("waffles");
}

@GetMapping("/create")
Waffle create() {
    return new Waffle(circuitBreaker.run(macronutrientsProvider::protein));
}
```

Problem

To meet your application's scaling needs you've adopted an event driven approach but you don't want to tie yourself to one particular messaging broker.

Solution

Spring Cloud Stream provides binders for multiple message brokers from RabbitMQ to Apache Kafka to various cloud based pub/sub implementations.

Spring Cloud Stream 3.2.4

Spring Cloud Stream is a framework for building highly scalable event-driven microservices connected with shared messaging systems.

The framework provides a flexible programming model built on already established and familiar Spring idioms and best practices, including support for persistent pub/sub semantics, consumer groups, and stateful partitions.

Binder Implementations

Spring Cloud Stream supports a variety of binder implementations and the following table includes the link to the GitHub projects.

- RabbitMQ
- Apache Kafka
- Kafka Streams
- Amazon Kinesis
- Google PubSub *(partner maintained)*
- Solace PubSub+ *(partner maintained)*
- Azure Event Hubs *(partner maintained)*
- Azure Service Bus *(partner maintained)*
- AWS SQS *(partner maintained)*
- AWS SNS *(partner maintained)*

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-stream-binder-rabbit</artifactId>
</dependency>
```

Problem

Your service is used by a variety of consumers and you cannot introduce breaking changes to downstream systems.

Solution

Spring Cloud Contract simplifies consumer driven contracts bringing test driven development to your application architecture.

Spring Cloud Contract 3.1.3

Spring Cloud Contract is an umbrella project holding solutions that help users in successfully implementing the Consumer Driven Contracts approach. Currently Spring Cloud Contract consists of the Spring Cloud Contract Verifier project.

Spring Cloud Contract Verifier is a tool that enables Consumer Driven Contract (CDC) development of JVM-based applications. It is shipped with Contract Definition Language (DSL) written in Groovy or YAML. Contract definitions are used to produce following resources:

- by default JSON stub definitions to be used by WireMock (HTTP Server Stub) when doing integration testing on the client code (client tests). Test code must still be written by hand, test data is produced by Spring Cloud Contract Verifier.
- Messaging routes if you're using one. We're integrating with Spring Integration, Spring Cloud Stream and Apache Camel. You can however set your own integrations if you want to.
- Acceptance tests (by default in JUnit or Spock) used to verify if server-side implementation of the API is compliant with the contract (server tests). Full test is generated by Spring Cloud Contract Verifier.

Spring Cloud Contract Verifier moves TDD to the level of software architecture.

To see how Spring Cloud Contract supports other languages just check out this blog post.

Features

When trying to test an application that communicates with other services then we could do one of two things:

- deploy all microservices and perform end to end tests
- mock other microservices in unit / integration tests

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-contract-verifier</artifactId>
    <scope>test</scope>
</dependency>
```

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-contract-stub-runner</artifactId>
    <scope>test</scope>
</dependency>
```

Testing

Problem

I want to write a full integration test against my REST API.

Solution

The `@SpringBootTest` annotation provides support for different `webEnvironment` modes, including the ability to start a fully running web server listening on a `defined` or `random` port.

SpringBootTest.java

44

Annotation that can be specified on a test class that runs Spring Boot based tests. Provides the following features over and above the regular *Spring TestContext Framework*:

- Uses SpringBootContextLoader as the default ContextLoader when no specific @ContextConfiguration (loader=...) is defined.
- Automatically searches for a @SpringBootConfiguration when nested @Configuration is not used, and no explicit classes are specified.
- Allows custom Environment properties to be defined using the properties attribute.
- Allows application arguments to be defined using the args attribute.
- Provides support for different webEnvironment modes, including the ability to start a fully running web server listening on a defined or random port.
- Registers a TestRestTemplate and/or WebTestClient bean for use in web tests that are using a fully running web server.

```java
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
class CoffeeControllerIntTest {

    @Autowired
    private TestRestTemplate testRestTemplate;

    @Test
    void findAll() {
        ResponseEntity<List<Coffee>> exchange = testRestTemplate.exchange("/api/coffee",
                HttpMethod.GET,
                null,
                new ParameterizedTypeReference<List<Coffee>>() {});

        assertEquals(3,exchange.getBody().size());
    }
}
```

Problem

As the size and complexity of my application increases my tests are taking longer and longer to run. How can I write tests that run faster?

Solution

A slice test allows you to focus on just a particular “slice” (web, db, etc…) of your application.

Test slice
@DataCassandraTest
@DataCouchbaseTest
@DataElasticsearchTest
@DataJdbcTest
@DataJpaTest
@DataLdapTest
@DataMongoTest
@DataNeo4jTest
@DataR2dbcTest
@DataRedisTest

Test slice
@GraphQlTest
@JdbcTest
@JooqTest
@JsonTest
@RestClientTest
@WebFluxTest
@WebMvcTest
@WebServiceClientTest
@WebServiceServerTest

org.springframework.boot.test.autoconfigure.web.servlet

Annotation Type WebMvcTest

```
@Target(value=TYPE)
 @Retention(value=RUNTIME)
 @Documented
 @Inherited
 @BootstrapWith(value=org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTestContextBootstrapper.class)
 @ExtendWith(value=org.springframework.test.context.junit.jupiter.SpringExtension.class)
 @OverrideAutoConfiguration(enabled=false)
 @TypeExcludeFilters(value=WebMvcTypeExcludeFilter.class)
 @AutoConfigureCache
 @AutoConfigureWebMvc
 @AutoConfigureMockMvc
 @ImportAutoConfiguration
public @interface WebMvcTest
```

Annotation that can be used for a Spring MVC test that focuses **only** on Spring MVC components.

Using this annotation will disable full auto-configuration and instead apply only configuration relevant to MVC tests (i.e. `@Controller`, `@ControllerAdvice`, `@JsonComponent`, `Converter/GenericConverter`, `Filter`, `WebMvcConfigurer` and `HandlerMethodArgumentResolver` beans but not `@Component`, `@Service` or `@Repository` beans).

By default, tests annotated with `@WebMvcTest` will also auto-configure Spring Security and `MockMvc` (include support for HtmlUnit WebClient and Selenium WebDriver). For more fine-grained control of MockMVC the `@AutoConfigureMockMvc` annotation can be used.

Typically `@WebMvcTest` is used in combination with `@MockBean` or `@Import` to create any collaborators required by your `@Controller` beans.

If you are looking to load your full application configuration and use MockMVC, you should consider `@SpringBootTest` combined with `@AutoConfigureMockMvc` rather than this annotation.

When using JUnit 4, this annotation should be used in combination with `@RunWith(SpringRunner.class)`.

Since:

1.4.0

Author:

Phillip Webb, Artsiom Yudovin

See Also:

`AutoConfigureWebMvc`, `AutoConfigureMockMvc`, `AutoConfigureCache`

Optional Element Summary

Optional Elements

Modifier and Type	Optional Element and Description
`Class<?>[]`	`controllers`

```java
@WebMvcTest(CoffeeController.class)
class CoffeeControllerTest {

    @Autowired
    private MockMvc mvc;

    @MockBean
    private CoffeeRepository coffeeRepository;

    @Test
    void findAllShouldReturn200Status() throws Exception {
        Coffee coffee = new Coffee(null,"TEST", Size.TALL);
        given(coffeeRepository.findAll()).willReturn(List.of(coffee));
        mvc.perform(get("/api/coffee"))
                .andExpect(MockMvcResultMatchers.status().isOk())
                .andExpect(MockMvcResultMatchers.jsonPath("$.*",hasSize(1)));
    }

}
```

Problem

How can I test my GraphQL API independent of the underlying server transport.

Solution

Spring for GraphQL provides dedicated support for testing GraphQL requests over HTTP, WebSocket, and RSocket, as well as for testing directly against a server.

Auto-configured Spring GraphQL Tests

Maven

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.graphql</groupId>
        <artifactId>spring-graphql-test</artifactId>
        <scope>test</scope>
    </dependency>
    <!-- Unless already present in the compile scope -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-webflux</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Gradle

```gradle
dependencies {
    testImplementation("org.springframework.graphql:spring-graphql-test")
    // Unless already present in the implementation configuration
    testImplementation("org.springframework.boot:spring-boot-starter-webflux")
}
```

Spring for GraphQL: Testing spring-graphql-test

GraphQlTester

- Workflow for testing GraphQL
- Fluent API
- Automatic checks to verify no errors in response
- Use JsonPath to specify data to inspect
- Data Decoding

WebGraphQlTester

- Extension of GraphQL tests over web transports
- Specific HTTP specific inputs
- Uses WebTestClient

```java
@GraphQlTest(CoffeeController.class)
class CoffeeControllerTest {

    @Autowired
    private GraphQlTester graphQlTester;

    @MockBean
    private CoffeeService coffeeService;

    private List<Coffee> coffees = new ArrayList<>();

    @BeforeEach
    void setUp() {
        coffees.add(new Coffee(1, "Caffè Americano", Size.GRANDE));
        coffees.add(new Coffee(2, "Caffè Latte", Size.VENTI));
        coffees.add(new Coffee(3, "Caffè Caramel Macchiato", Size.TALL));
    }
```

```java
@Test
void findAll() {
    // language=GraphQL
    String document = """
    query {
        findAll {
            id
            name
            size
        }
    }
    """;

    when(coffeeService.findAll()).thenReturn(coffees);

    graphQlTester.document(document)
            .execute()
            .path("findAll")
            .entityList(Coffee.class)
            .hasSize(3);
}
```

```java
@Test
void findOne() {
    // language=GraphQL
    String document = """
    query findOne($id: ID){
        findOne(id: $id) {
            id
            name
            size
        }
    }
    """;

    when(coffeeService.findOne(1)).thenReturn(Optional.of(coffees.get(0)));

    graphQlTester.document(document)
            .variable("id",1)
            .execute()
            .path("findOne")
            .entity(Coffee.class)
            .satisfies(coffee -> {
                assertEquals("Caffè Americano",coffee.name());
                assertEquals(Size.GRANDE,coffee.size());
            });

}
```

Problem

I want to write tests against the same database we use in production but I need that process to be fast and easy.

Solution

Testcontainers is a Java library that supports JUnit tests, providing lightweight, throwaway instances of common databases, Selenium web browsers, or anything else that can run in a Docker container.

Testing is important!

Testing can be hard!

“You have to make the right thing to do, the easy thing to do.”

- Cora Iberkleid

```java
@SpringBootTest
@Testcontainers
class BookRepositoryIntegrationTest {

    @Container
    static PostgreSQLContainer psql = new PostgreSQLContainer("postgres:alpine");

    @Autowired
    BookRepository repository;

    @Test
    void findAll_shouldReturn1Books() {
        Integer mappedPort = psql.getMappedPort(5432);
        System.out.println("Mapped Port: " + mappedPort);
        assertEquals(1, repository.findAll().size());
    }

}
```

Production


Production

Problem

How can I package my application for production?

Solution

Use the Spring Boot Maven/Gradle Plugin to create an executable JAR.

```
<build>
    <plugins>
```

```
plugins {
    id 'org.springframework.boot' version '1.5.22.RELEASE'
}
```

```
    </plugins>
</build>
```

Problem

Your organization uses Docker to containerize your applications.

Solution

You can leverage Cloud Native Buildpacks to simplify and ensure consistency in your applications ecosystem.

2.1. Getting started with Buildpacks

This section gives you a practical overview of building a Spring Boot native application using Cloud Native Buildpacks. This is a practical guide that uses the RESTful Web Service getting started guide.

2.1.1. System Requirements

Docker should be installed, see Get Docker for more details. Configure it to allow non-root user if you are on Linux.

You can run `docker run hello-world` (without `sudo`) to check the Docker daemon is reachable as expected. Check the Maven or Gradle Spring Boot plugin documentation for more details.

On MacOS, it is recommended to increase the memory allocated to Docker to at least `8GB`, and potentially add more CPUs as well. See this Stackoverflow answer for more details. On Microsoft Windows, make sure to enable the Docker WSL 2 backend for better performance.

2.1.2. Sample Project Setup

The completed "RESTful Web Service" guide can be retrieved using the following commands:

```bash
git clone https://github.com/spring-guides/gs-rest-service
cd gs-rest-service/complete
```

Validate Spring Boot version

Spring Native 0.12.1 only supports Spring Boot 2.7.1, so change the version if necessary.

Maven | Gradle Groovy | Gradle Kotlin

```xml
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
```

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
```

```
bootBuildImage {
    // ...
    buildpacks = ["gcr.io/paketo-buildpacks/java-native-image:7.19.0"]
}
```

```
            </env>
        </image>
    </configuration>
</plugin>
```

Problem

You want your application to startup faster and take up less memory.

Solution

You can compile your application to a native executable using Spring Native which is built in to Spring Boot 3.

spring

Spring Native documentation

Version 0.12.1 - Andy Clement · Sébastien Deleuze · Filip Hanik · Dave Syer · Esteban Ginez · Jay Bryant · Brian Clozel · Stéphane Nicoll · Josh Long

1. Overview

Spring Native provides support for compiling Spring applications to native executables using the GraalVM native-image compiler.

Compared to the Java Virtual Machine, native images can enable cheaper and more sustainable hosting for many types of workloads. These include microservices, function workloads, well suited to containers, and Kubernetes

Using native image provides key advantages, such as instant startup, instant peak performance, and reduced memory consumption.

There are also some drawbacks and trade-offs that the GraalVM native project expect to improve on over time. Building a native image is a heavy process that is slower than a regular application. A native image has fewer runtime optimizations after warmup. Finally, it is less mature than the JVM with some different behaviors.

The key differences between a regular JVM and this native image platform are:

- A static analysis of your application from the main entry point is performed at build time.
- The unused parts are removed at build time.
- Configuration is required for reflection, resources, and dynamic proxies.
- Classpath is fixed at build time.
- No class lazy loading: everything shipped in the executables will be loaded in memory on startup.
- Some code will run at build time.
- There are some limitations around some aspects of Java applications that are not fully supported.

```
<build>
    <plugins>
```

```
plugins {
    id 'org.springframework.boot' version '1.5.22.RELEASE'
}
```

```
    </plugins>
</build>
```

Spring Projects


Spring Boot


Spring Cloud


Spring Framework

Spring Cloud Data Flow

Spring Tool Suite


Spring LDAP


Spring Cloud Gateway


Spring Security


Spring Data


Spring Batch


Spring Integration


Project Reactor


Spring Kafka

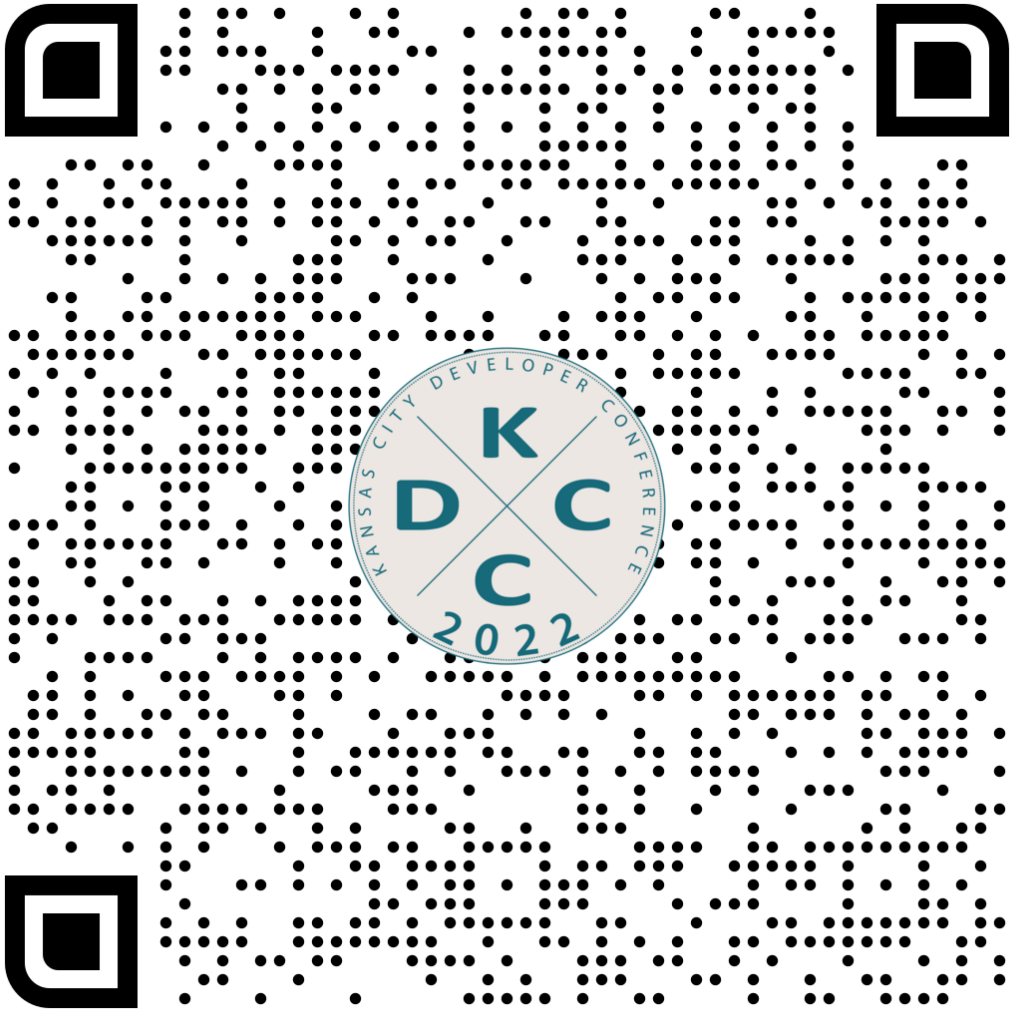

Spring for GraphQL


Spring Web Services


Spring Web Flow


Spring Hateoas


Spring AMQP


KANSAS CITY DEVELOPER CONFERENCE
K
D
C
C
2022

Thank you!

Nate (@ntschutta) www.ntschutta.io

Dan (@therealdanvega) www.danvega.dev