# CLI

Spring Boot’s CLI leverages starter dependencies and auto-configuration to let you

focus on writing code.

the CLI detected the types that are being used, and it

knows which starter dependencies to add to the classpath to make it work. Once those

dependencies are in the classpath, a series of auto-configuration kicks in and ensures

that DispatcherServlet and Spring MVC are enabled so that the controller can

respond to HTTP requests. Spring Boot’s CLI is an optional piece of Spring Boot’s power.

# THE ACTUATOR

inspect the inner workings of your application, including details such as

What beans have been configured in the Spring application context

■ What decisions were made by Spring Boot’s auto-configuration

■ What environment variables, system properties, configuration properties, and command-line arguments are available to your application

■ The current state of the threads in and supporting your application

■ A trace of recent HTTP requests handled by your application

■ Various metrics pertaining to memory usage, garbage collection, web requests, and data source usage

The Actuator exposes this information in two ways: via web endpoints or via a shell

interface. In the latter case, you can actually open a secure shell (SSH) into your application and issue commands to inspect your application as it runs

# What Spring Boot isn’t

not an application server.doesn’t implement any enterprise Java specifications such as

JPA or JMS.doesn’t employ any form of code generation to accomplish its

magic. Instead, it leverages conditional configuration features from Spring 4, along

with transitive dependency resolution offered by Maven and Gradle, to automatically

configure beans in the Spring application context.

In short, at its heart, Spring Boot is just Spring. Inside, Spring Boot is doing the

same kind of bean configuration in Spring that you might do on your own if Spring

Boot didn’t exist. Thankfully, because Spring Boot does exist, you’re freed from dealing

with explicit boilerplate configuration and are able to focus on the logic that

makes your application unique.

The @SpringBootApplication enables Spring component-scanning and Spring Boot auto-configuration.

@SpringBootApplication combines three other useful annotations:

■ Spring’s @Configuration—Designates a class as a configuration class using

Spring’s Java-based configuration. Although we won’t be writing a lot of configuration

in this book, we’ll favor Java-based configuration over XML configuration

when we do.

■ Spring’s @ComponentScan—Enables component-scanning so that the web controller

classes and other components you write will be automatically discovered

and registered as beans in the Spring application context. A little later in this

chapter, we’ll write a simple Spring MVC controller that will be annotated with

@Controller so that component-scanning can find it.

■ Spring Boot’s @EnableAutoConfiguration—This humble little annotation might

as well be named @Abracadabra because it’s the one line of configuration that

enables the magic of Spring Boot auto-configuration. This one line keeps you

from having to write the pages of configuration that would be required otherwise.

2.2.2 statr from

2.3.1 atart from