

# SPRING BOOT 2 CHEAT SHEET

Spring Boot 2.1.x - Date : September 2019

## WHAT IS SPRING BOOT ?

- Built on top of a lot of Spring Projects (<https://spring.io>)
- Opinionated configuration
- Wide ecosystem
- Start a project faster with no configuration

## FEATURES

- Bootstrap class `SpringApplication`
- Default logger (@see `spring-jcl`)
- FailureAnalyzers : friendly failure report
- Application Events on Listeners
- Choose the right `ApplicationContext`
- Accessing application arguments
- Control application exit code

## QUICKSTART

```
<parent>

<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.1.8.RELEASE</version>
</parent>

<dependencies>
  <dependency>


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

```
@SpringBootApplication
public class MyApp {

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

}
```

## BEST PRACTICES

 Don't use the `**default**` package. Prefer using the Java convention `*com.example.project*`

### Code structure

```
com
+ example
|   + project
|   |   - MyApp.java
|   |   + player
|   |   |   - Player.java
|   |   |   - PlayerService.java
```

### Configuration

Properties files in folder `src/main/resources/` are loaded automatically

- **YAML** : `**/application.yml` or `**/application.yaml`
- **Properties** : `**/application.properties`

### Auto-configuration

- Enabled by `@SpringBootApplication` or `@EnableAutoConfiguration`
- Spring Boot scans all libs on the classpath and auto-configures them (**if you didn't manually**)

Display Spring Boot Autoconfigure report

```
java -jar myapp.jar --debug
```

Disabling an auto-configuration class with Java conf

```
@EnableAutoConfiguration(exclude=
{DataSourceAutoConfiguration.class})
```

Disabling an auto-configuration class with properties

```
spring.autoconfigure.exclude= \
org.springframework.boot.autoconfigure.XXXX
```

# SPRING BOOT 2 CHEAT SHEET

## BUILD A PRODUCTION JAR

### Maven

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

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

### Gradle

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

### Actuator

Production ready features threw HTTP or JMX

```
<dependency>

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

## RUNNING THE CODE

### Standalone

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

### Standalone with remote debug

```
java -Xdebug \
-Xrunjdwp:server=y,transport=dt_socket\
,address=8000,suspend=n -jar
target/myapplication-0.0.1-SNAPSHOT.jar
```

### Maven

```
mvn spring-boot:run
```

### Gradle

```
gradle bootRun
```

## STARTERS

Official starters : `spring-boot-starter-*`

### Application starters

Name	Function
web	Web applications using Spring MVC (Tomcat embedded)
test	Spring test using JUnit, Hamcrest and Mockito
security	Secured services with Spring Security
webflux	WebFlux applications using Spring Framework's Reactive Web
websocket	WebSocket applications using Spring Framework's WebSocket
data-jdbc	Configured resources to use Spring Data JDBC
data-mongodb	Configured resources to use Spring Data JPA with Hibernate
data-rest	Web applications using Spring Data repositories over REST
actuator	Production ready features using Spring's Actuator (monitor and manage)

### Technical starters

Name	Function
jetty	Using Jetty over the default Tomcat
log4j2	Using Log4j2 for logging over the default Logback
undertow	Using Undertow over the default Tomcat
Netty	Using Netty over the default Tomcat and using WebFlux

## ANNOTATIONS

Annotations	Description
<b>@SpringBootApplication</b>	uses @Configuration
<b>@EnableAutoConfiguration</b>	make Spring guess the configuration based on the classpath.
<b>@SpringBootApplication</b>	uses @Configuration
<b>@Controller</b>	marks the class as web controller, capable of handling the requests. @RestController - a convenience annotation of a @Controller and @ResponseBody.
<b>@ResponseBody</b>	makes Spring bind method's return value to the web response body.
<b>@PathVariable</b>	bind HTTP parameters into method arguments.
<b>@RequestParam</b>	uses @Configuration
<b>@EnableConfigServer</b>	turns your application into a server other apps can get their configuration from.
<b>@EnableEurekaServer</b>	makes your app an Eureka discovery service, other apps can locate services through it.
<b>@EnableDiscoveryClient</b>	makes your app register in the service discovery server and discover other services through it.
<b>@EnableCircuitBreaker</b>	configures Hystrix circuit breaker protocols

## REFERENCES

- <https://spring.io/guides>
- <https://spring.io/projects/spring-framework>
- [Spring Boot documentation](#)