Spring & SpringBoot

Power of Spring

JEE

POJO Based

Unobtrusive

AOP/Proxies

Best Practices

Advantages of Spring

Testability

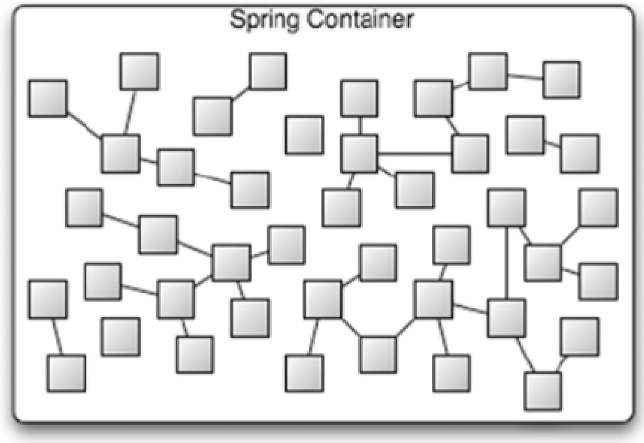
Maintainability

Scalability

Complexity

Business Focus

How Spring Container works?



POJOS

HashMap

Registry

Spring Boot definition

- Spring Boot makes it easy to create stand-alone, production-grade Spring based applications that you can "just run".
- We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration
- Spring Boot enables developers to focus on the business logic behind their microservice. It aims to take care of all the nitty-gritty technical details involved in developing microservices.

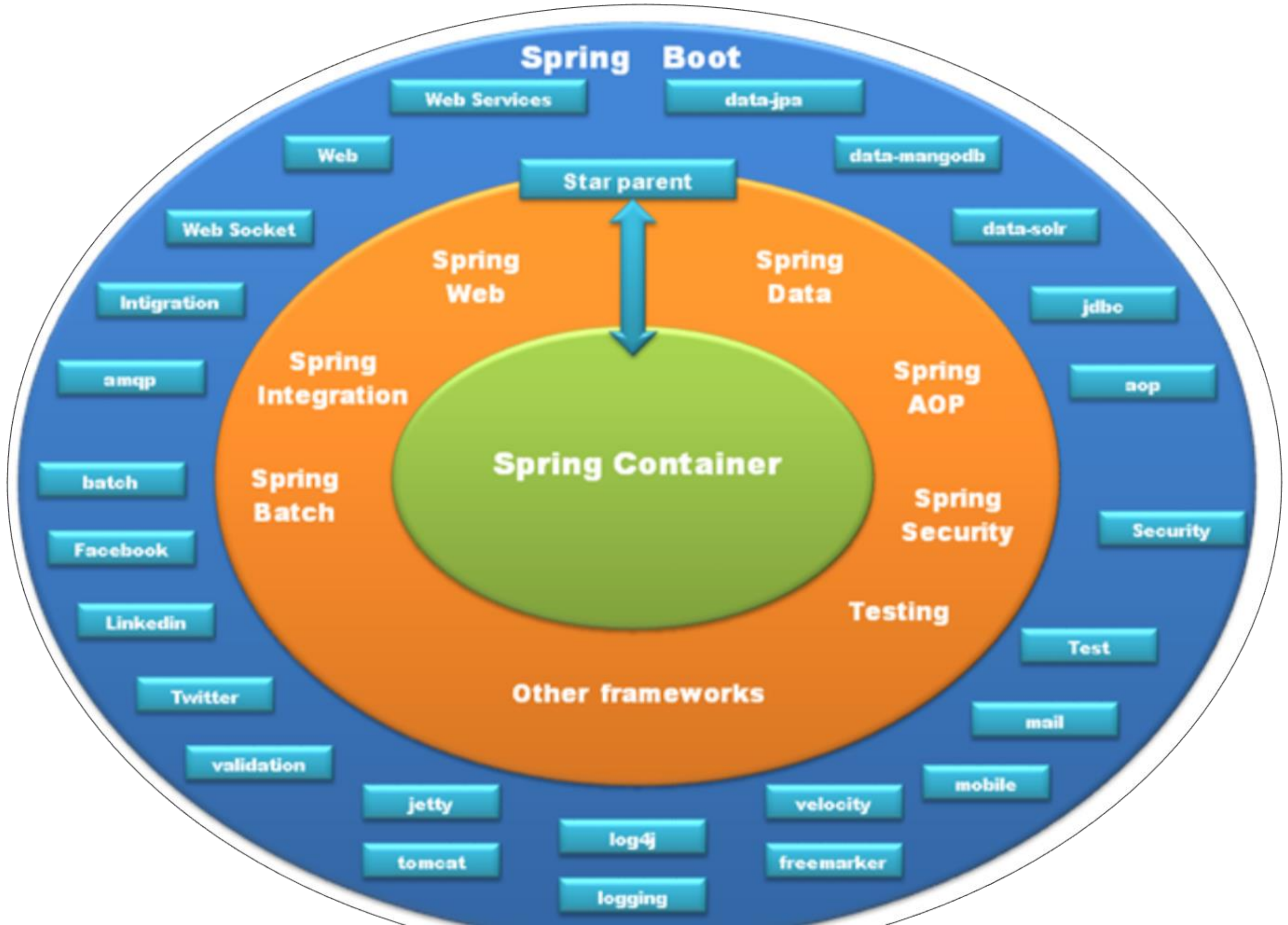
Primary Goals of Spring Boot

- Enable quickly getting off the ground with Spring-based projects.
- Be opinionated. Make default assumptions based on common usage. Provide configuration options to handle deviations from defaults.
- Provide a wide range of nonfunctional features out of the box.
- Do not use code generation and avoid using a lot of XML configuration.

Non Functional Features of Spring Boot

- Default handling of versioning and configuration of a wide range of frameworks, servers, and specifications
- Default options for application security
- Default application metrics with possibilities to extend
- Basic application monitoring using health checks
- Multiple options for externalized configuration

Spring Boot - a gist



Spring Initializr

- Spring Initializr provides a lot of flexibility in creating projects. You have options to do the following:
  - Choose your build tool: Maven or Gradle.
  - Choose the Spring Boot version you want to use.
  - Configure a Group ID and Artifact ID for your component.
  - Choose the starters (dependencies) that you would want for your project.
  - Choose how to package your component: JAR or WAR.
  - Choose the Java version you want to use.
  - Choose the JVM language you want to use.

Hello with Spring Boot

- We will start with building our first Spring Boot application.
- We will use Maven to manage dependencies.
- The following steps are involved in starting up with a Spring Boot application:
  - Configure spring-boot-starter-parent in your pom.xml file.
  - Configure the pom.xml file with the required starter projects.
  - Configure spring-boot-maven-plugin to be able to run the application.
  - Create your first Spring Boot launch class.

spring-boot-starter-parent

- The spring-boot-starter-parent dependency is the parent POM providing dependency and plugin management for Spring Boot-based applications.
- A spring-boot-starter-parent dependency contains the default versions of Java to use, the default versions of dependencies that Spring Boot uses, and the default configuration of the Maven plugins.

```
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version> 2.2.5.RELEASE</version>
<relativePath/>
</parent>
```

Spring Application

- The Spring Application class can be used to Bootstrap and launch a Spring application from a Java main method.
- The following are the steps that are typically performed when a Spring Boot application is bootstrapped:
  - Create an instance of Spring's ApplicationContext.
  - Enable the functionality to accept command-line arguments and expose them as Spring properties.
  - Load all the Spring beans as per the configuration.

@SpringBootApplication

- The **@SpringBootApplication** annotation is a shortcut for three annotations:
  - **@Configuration**: Indicates that this a Spring application context configuration file.
  - **@EnableAutoConfiguration**: Enables auto-configuration, an important feature of Spring Boot. We will discuss auto-configuration later in a separate section.
  - **@ComponentScan**: Enables scanning for Spring beans in the package of this class and all its sub packages.

@Component

- The @Component annotation
  - marks a java class as a bean so
  - the component-scanning mechanism of spring
  - can pick it up and pull it into the application context.

@Component vs @Bean

- @Component and @Bean do two quite different things, and shouldn't be confused.
- @Component (and @Service and @Repository) are used to **auto-detect and auto-configure** beans using classpath scanning. There's an implicit one-to-one mapping between the annotated class and the bean (i.e. one bean per class). Control of wiring is quite limited with this approach, since it's purely declarative.
- @Bean is used to ***explicitly*** declare a single bean, rather than letting Spring do it automatically as above. It decouples the declaration of the bean from the class definition, and lets you create and configure beans exactly how you choose.

@ Autowired (to show DI)

- The **@Autowired** annotation provides more fine-grained control over where and how autowiring should be accomplished.
- The @Autowired annotation can be used to autowire bean on the setter method just like @Required annotation, constructor, a property or methods with arbitrary names and/or multiple arguments.

Starters in Spring Boot

- Starters are simplified dependency descriptors customized for different purposes.
- For example, **spring-boot-starter-web** is the starter for building web application, including RESTful, using Spring MVC. It uses Tomcat as the default embedded container.
- If I want to develop a web application using Spring MVC, all we would need to do is include spring-boot-starter-web in our dependencies, and we get the following automatically pre-configured:
  - Spring MVC
  - Compatible versions of jackson-databind (for binding) and hibernate-validator (for form validation)
  - spring-boot-starter-tomcat (starter project for Tomcat)

spring-boot-starter-web

- This will add Spring MVC capabilities to Spring Boot
- Helps you build Spring MVC-based web applications or RESTful applications. It uses Tomcat as the default embedded servlet container.

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

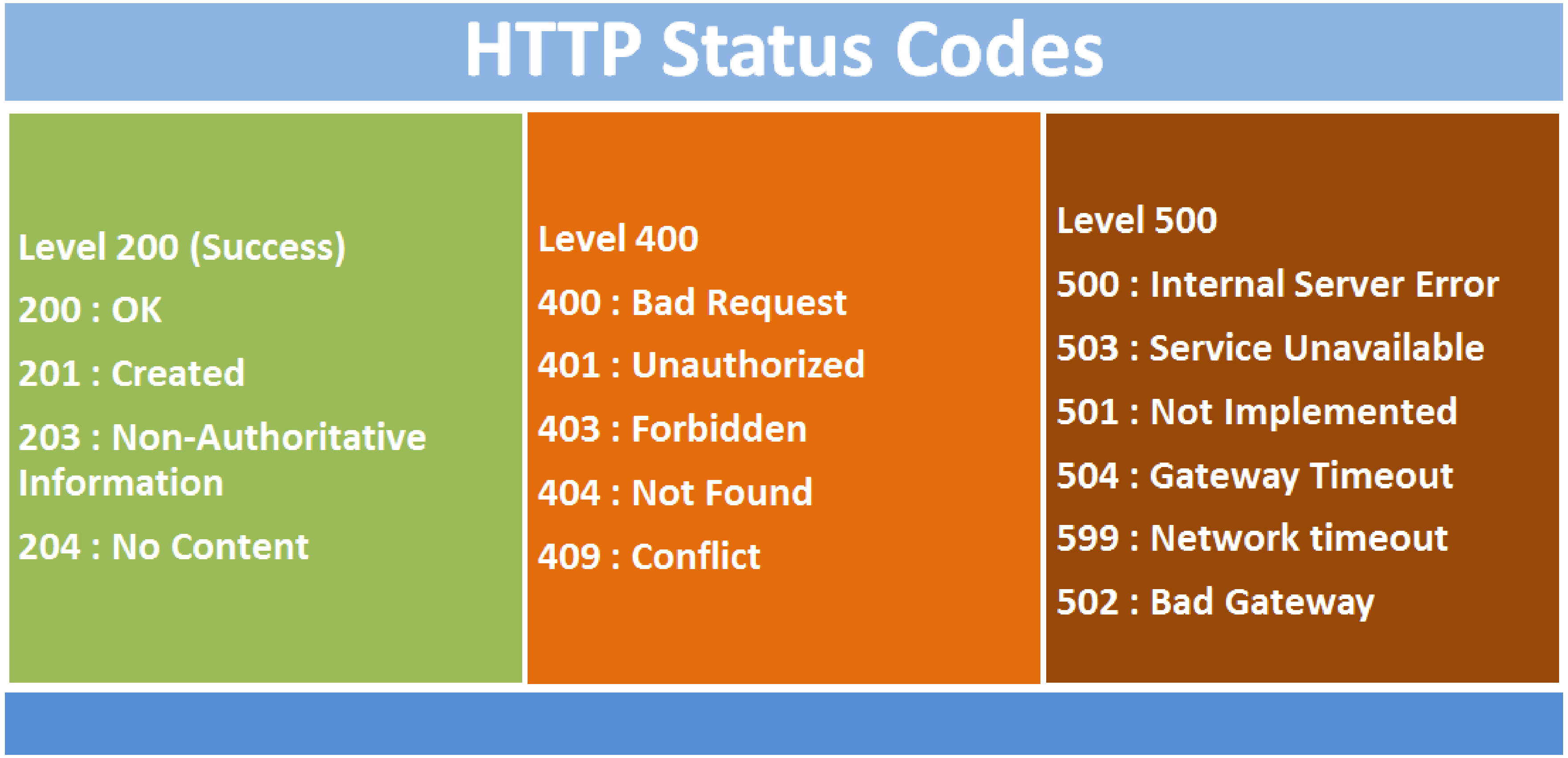
spring-boot-maven-plugin

- When we build applications using Spring Boot, there are a couple of situations that are possible:
  - We would want to run the applications in place without building a JAR or a WAR
  - We would want to build a JAR and a WAR for later deployment
- The **spring-boot-maven-plugin** dependency provides capabilities for both of the preceding situations. The following snippet shows how we can configure spring-boot-maven-plugin in an application:

Http status codes

- HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:
  - informational responses
  - successful responses
  - redirects
  - client errors
  - servers errors.
- Go to: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

HTTP Status codes



HTTP Status Codes

Level 200 (Success)	Level 400	Level 500
200 : OK	400 : Bad Request	500 : Internal Server Error
201 : Created	401 : Unauthorized	503 : Service Unavailable
203 : Non-Authoritative Information	403 : Forbidden	501 : Not Implemented
204 : No Content	404 : Not Found	504 : Gateway Timeout
	409 : Conflict	599 : Network timeout
		502 : Bad Gateway

HTTP-REST Vocabulary

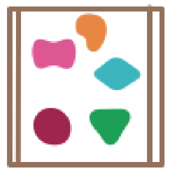
HTTP Methods supported by REST:

- GET – Requests a resource at the request URL
  - Should <u>not</u> contain a request body, as it will be discarded.
  - <u>May</u> be cached locally or on the server.
  - May produce a resource, but should not modify on it.
- POST – Submits information to the service for processing
  - Should typically return the new or modified resource.
- PUT – Add a new resource at the request URL
- DELETE – Removes the resource at the request URL

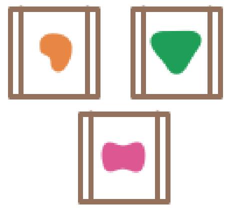
Microservices

Monolithic vs Microservices Architecture

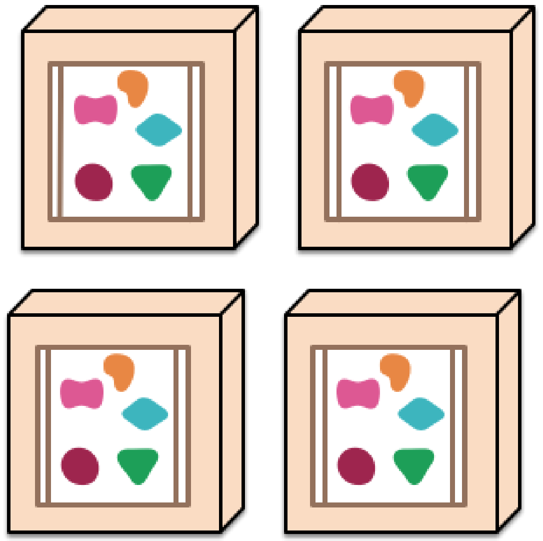
A monolithic application puts all its functionality into a single process…



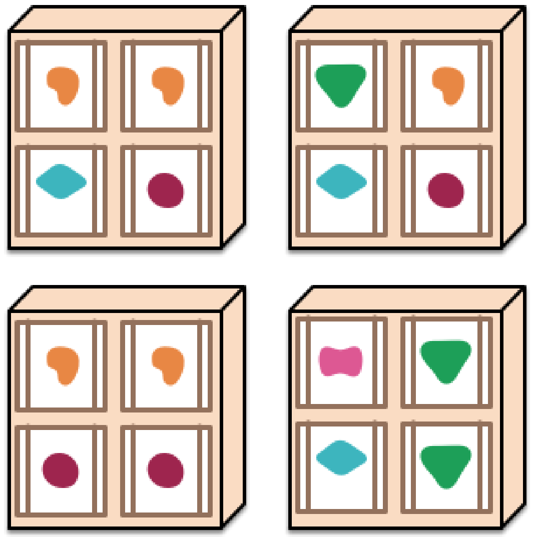
A microservices architecture puts each element of functionality into a separate service…



… and scales by replicating the monolith on multiple servers



… and scales by distributing these services across servers, replicating as needed.



Microservice Architectural Style

- approach to developing a single application
- as a **suite of small services**
- each **running in its own process** and
- communicating with lightweight mechanisms (HTTP)

Microservice Architectural Style

- services are **built around business capabilities**
- **independently deployable**
- **bare minimum of centralized management**
- may be written in different programming languages
- use different data storage technologies.

Microservice Characteristics

- Domain Specific
- Loosely Coupled
- Multiple Independent Development Teams
- Fault Tolerant
- Service Based
- Continuous Delivery

Microservices

MICROSERVICES

Building small,
self-contained,
ready to run applications
can bring great flexibility and
added resilience
to your code

Spring Cloud

Spring Cloud

Tools for developers
to quickly build some of the
common patterns in distributed
systems
(e.g. configuration management,
service discovery)

Thank you