Deutsche Bank
Corporate Division

Unveiling Spring Boot & Building Your First Application

PPT-Master Version 01. There will be an extensive update with the new Deutsche Bank font and sample charts at the end of August 2024.

8 May 2024, Speaker name

Introduction

Open Source Java Framework for WebApps & microservices

a. Autoconfiguration

b. Pre-configuried dependency bundle

c. Spring + Web Server = Spring Boot

d. Streamlines Spring application setup

Why not Spring?

1. Simplifies significantly, need one dependency: spring-boot-starter-web

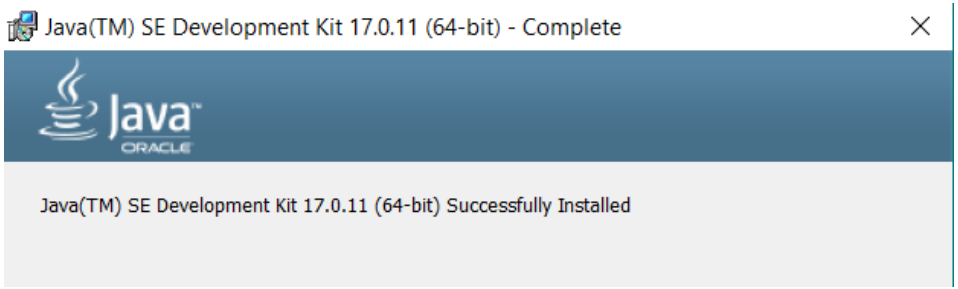
2. It automatically add necessary dependencies during the build.
3. Convention over Configuration.
4. Features include embedded servers, metrics, health checks.

Setting up Java

1. Download Java JDK 17 from official Java website (Link below).

https://www.oracle.com/java/technologies/downloads/#java17

2. Double click downloaded file and follow the onscreen instructions for installation.



Setting up Apache Maven

Download the binary file from : https://maven.apache.org/download.cgi

	Link
Binary tar.gz archive	apache-maven-3.9.8-bin.tar.gz
Binary zip archive	apache-maven-3.9.8-bin.zip

Step 1: Extract the downloaded ZIP file

Step 2: Move the extracted folder to the desired location

(e.g., C:\apache-maven-3.9.8)

Step 3: Add apache-maven-3.9.8 to System PATH

Getting SpringBoot CLI

Prerequisites:-

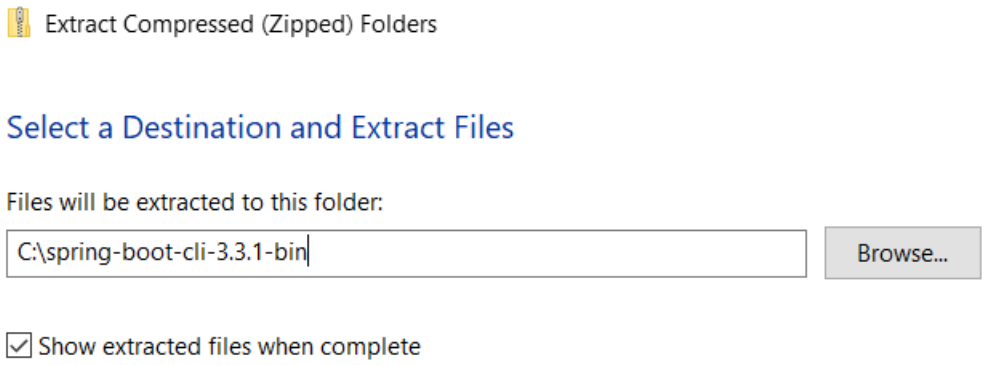
- Java Development Kit (JDK): Ensure JDK 17
- Apache Maven 3.9.8
- Homebrew (for Mac): Optional but recommended for easy installation

Installation:-

- Navigate to the Spring Boot CLI section on the Spring website
- Download Link:

https://docs.spring.io/spring-boot/installing.html#getting-started.installing.cli

Installation on Windows



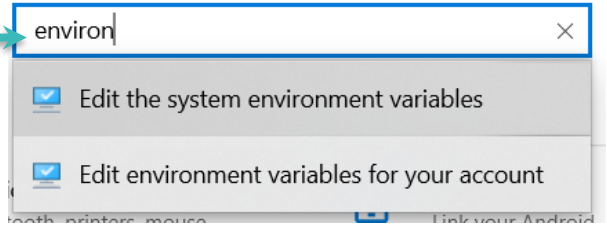
Step 1: Extract the downloaded ZIP file

Step 2: Move the extracted folder to the desired location (e.g., C:\SpringBootCLI)

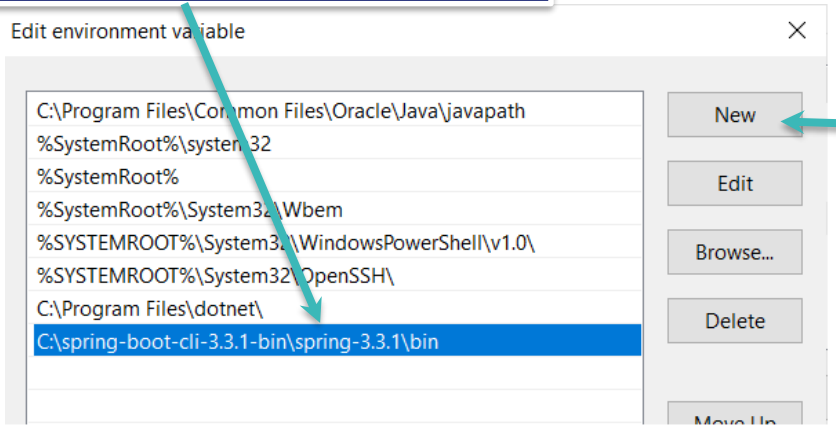
Step 3: Add Spring Boot CLI to System PATH

Add Spring Boot CLI to System PATH

Type environment variables on search bar



Path to your spring boot cli installation



Click on New to add

Post-Installation Steps

Command Prompt

```
C:\spring-boot-cli-3.3.1-bin\spring-3.3.1\bin>spring --version
Spring CLI v3.3.1
C:\spring-boot-cli-3.3.1-bin\spring-3.3.1\bin>
```

Testing: Run a simple Spring Boot command to ensure the CLI is working correctly

Example Command:

spring --version

On Your First “Hello World” :)

What is Spring Initializr?

— An online tool to generate Spring Boot project structures

— Key features : dependency management and project configuration

Accessing Spring Initializr

— Navigate to the Spring Initializr website (https://start.spring.io)

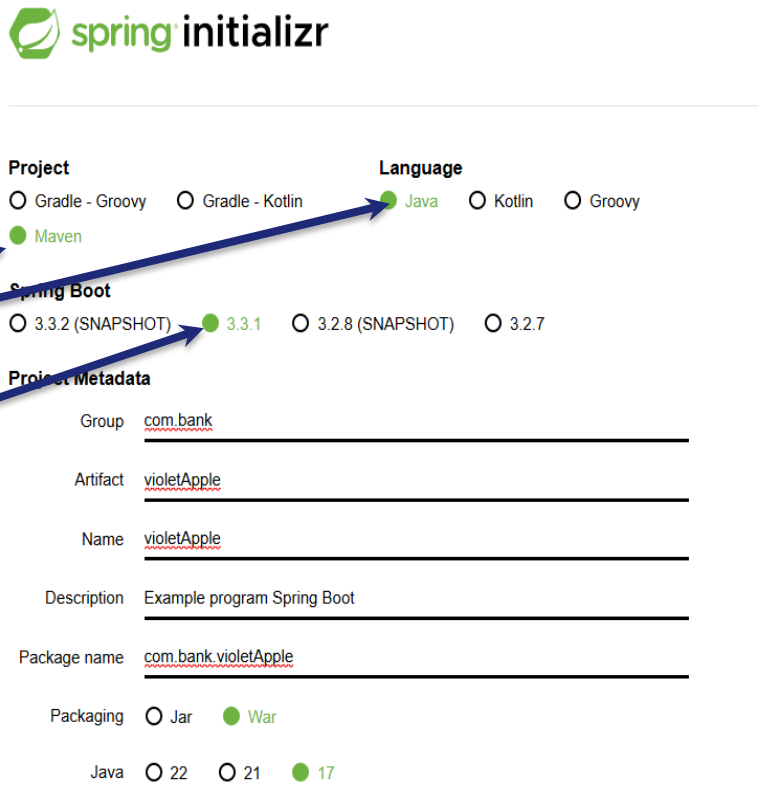
— Note : Spring Initializr is also integrated into IDEs such as IntelliJ IDEA or VScode

Start New Project

Project Type: Choose Maven

Language: Select Java

Spring Boot Version: Select the appropriate version of Spring Boot eg. 3.3.1



Metadata Fields

Group: Typically the domain of your organization (e.g., com.banking)

Artifact: The name of your project

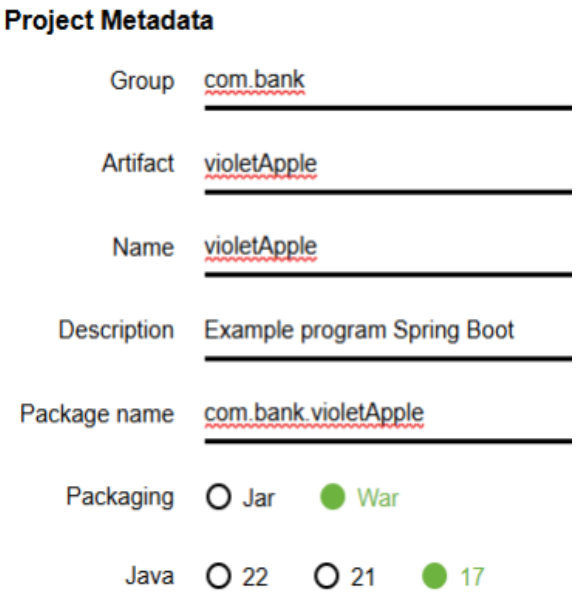
Name: The display name for your project

Description: A brief description of your project

Package Name: The base package for your application

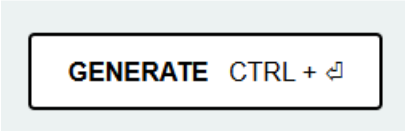
Packaging: Choose War

Java Version: Select the version of Java you will be using (e.g., Java JDK 17)

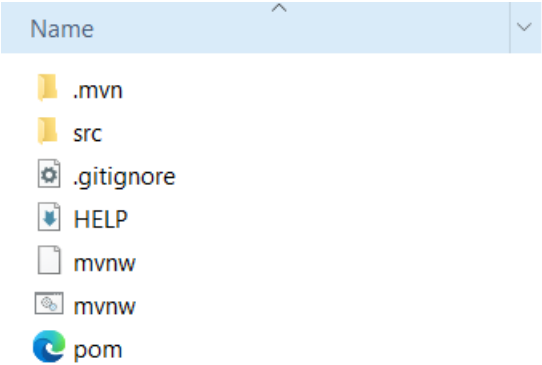


Generating the Project

1. Generate Button: Click on the "Generate" button to download the project zip file



2. Download and Extract: Download & extract the generated zip file



The Project Structure

Overview of the key files and folders in the generated project

src/main/java:

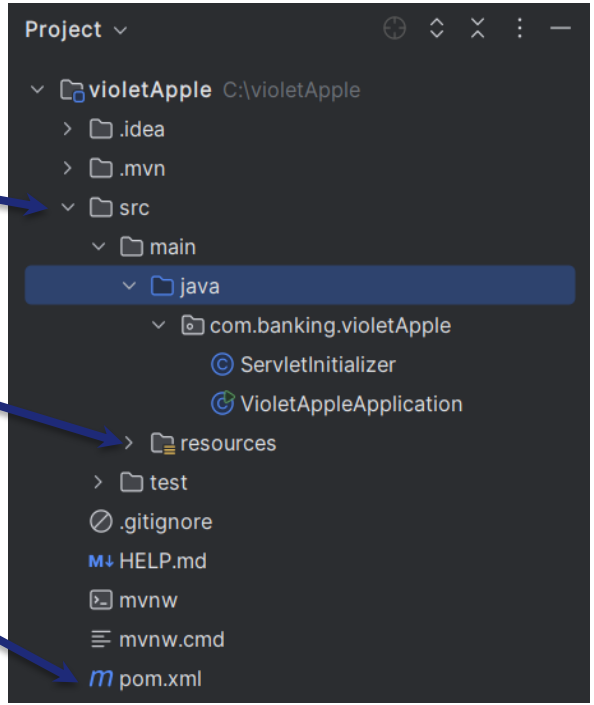
— Contains application code

src/main/resources:

— Contains configuration files and static resources

pom.xml:

— Dependency management files



Importing the Project into an IDE

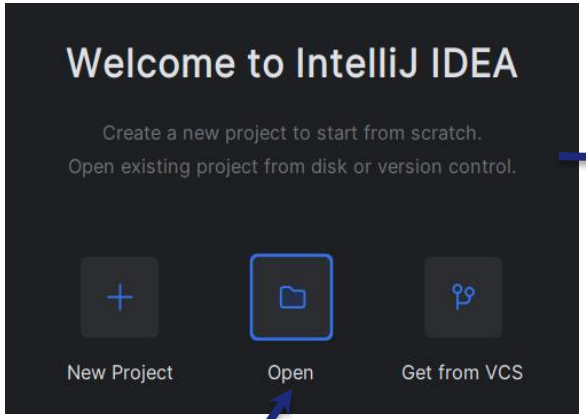
Step-by-Step Guide: Instructions for importing the project into popular IDEs

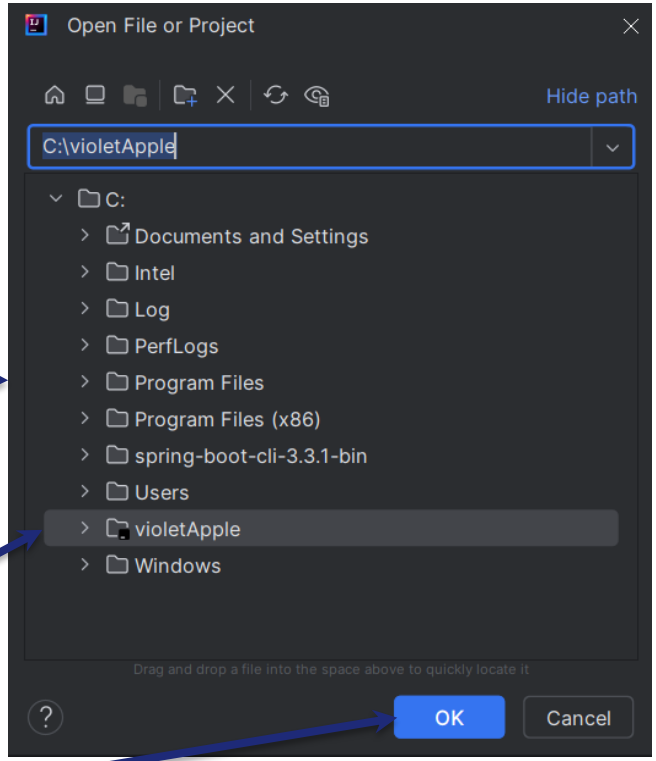
IntelliJ IDEA: File -> New -> Project from Existing Sources -> Select the project

Eclipse: File -> Import -> Existing Maven/Gradle Project -> Select the project

VS Code: Open Folder -> Select the project folder

IntelliJ IDEA

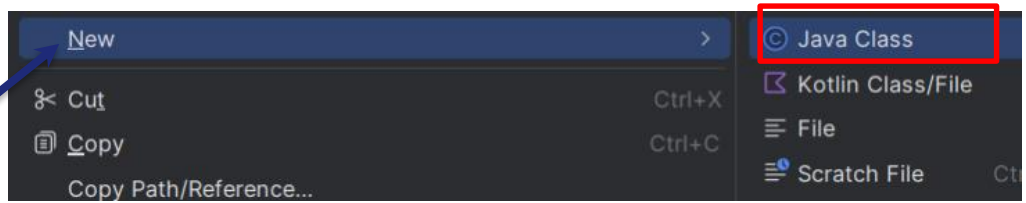


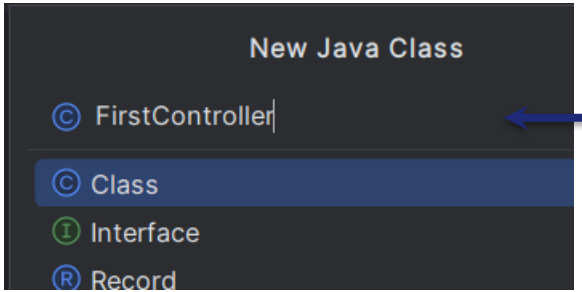


Open -> Select the folder of project -> Ok

Get into first code

Create a new Java Class file

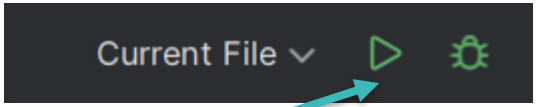




Give a meaningful name

First Code

```
@RestController  no usages
public class FirstController {
    @RequestMapping("/")  no usages
    public String apple() {
        return "What is color of my apple?";
    }
}
```

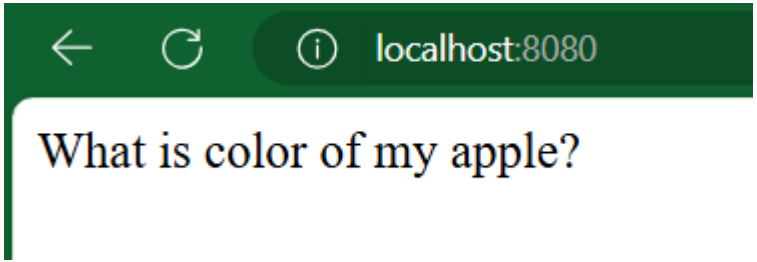


Click on run buttion to run your first application

Default run on localhost:8080

```
main] c.b.violetApple.VioletAppleApplication    : Starting VioletAppleApplication using Java 17.0.11 wi
main] c.b.violetApple.VioletAppleApplication    : No active profile set, falling back to 1 default prof
main] o.s.b.w.embedded.tomcat.TomcatWebServer   : Tomcat initialized with port 8080 (http)
main] o.apache.catalina.core.StandardService    : Starting service [Tomcat]
main] o.apache.catalina.core.StandardEngine     : Starting Servlet engine: [Apache Tomcat/10.1.25]
main] o.a.c.c.C.[Tomcat].[localhost].[/]        : Initializing Spring embedded WebApplicationContext
main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed
main] o.s.b.w.embedded.tomcat.TomcatWebServer   : Tomcat started on port 8080 (http) with context path
main] c.b.violetApple.VioletAppleApplication    : Started VioletAppleApplication in 4.1 seconds (proces
xec-2] o.a.c.c.C.[Tomcat].[localhost].[/]        : Initializing Spring DispatcherServlet 'dispatcherServ
```

Output on web browser



Inversion of Control (IoC)

Definition:

— IoC transfers control of objects or program portions to a container or framework.

Traditional vs. IoC:

— In traditional programming, custom code controls program flow.

— In IoC, an external entity (the IoC container) manages flow and object creation.

IoC Example

manually instantiate the Service and Controller classes

```java
public class TraditionalFruitApp {
    public static void main(String[] args) {
        Service service = new Service();
        Controller controller = new Controller(service);
        controller.doSomething();
    }
}

class Controller {  2 usages
    private final Service Fruitservice;  2 usages

    public Controller(Service service) {  1 usage
        this.Fruitservice = service;
    }

    public void doSomething() {  1 usage
        Fruitservice.perform();
    }
}
```

Automatically injects the dependencies using annotations like @Autowired

```java
@SpringBootApplication
public class FruitIocApp implements CommandLineRunner {

    @Autowired  1 usage
    private Controller controller;

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

- Decoupling the Class from the creation process

Dependency Injection (DI)

Definition:

— Dependency Injection (DI) is a design pattern that implements IoC.

Benifits:

— DI makes the code more modular and easier to test by allowing dependencies to be swapped out.

— It also promotes loose coupling

Dependency Injection Example

A Spring Boot example where a service class is injected into a controller class using the @Autowired annotation

```
@SpringBootApplication
public class FruitIocApp implements CommandLineRunner {

    @Autowired  1 usage
    private Controller controller;

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

Loose Coupling

Loose coupling refers to a design principle where components or classes are minimally dependent on each other.

1. Easier Maintenance: Components can be updated or replaced with minimal impact on others.
2. Better Scalability: Systems can be scaled more easily because components are independent.
3. Flexibility: New features or changes can be integrated without significant refactoring.

Loose Coupling Example

MyService is an interface, and MyServiceImplementation is its implementation.

```
interface MyService {
    void perform();
}
```

The MyController class depends on the MyService interface rather than a concrete implementation, promoting loose coupling.

```
@Component
class MyController {
    private final MyService service;

    @Autowired
    public MyController(MyService service) {
        this.service = service;
    }
```

```
@Component
class MyServiceImplementation implements MyService {
    @Override
    public void perform() {
        System.out.println("Asking price of fruit...");
    }
}

@Configuration
class AppConfig {

    @Bean
    public MyService myService() {
        return new MyServiceImplementation();
    }
}
```

Tight Coupling

Tight coupling refers to a design where components or classes are highly dependent on each other.

1. Reduced Flexibility: Difficult to modify or replace components without affecting others.
2. Harder Maintenance: More effort is required to update and maintain the system.
3. Increased Risk of Changes Propagating: Changes in one part of the system can cause issues in other parts, making debugging and testing more complex.

Tight Coupling Example

The MyController class directly creates an instance of MyService, leading to tight coupling.

If the implementation of MyService changes or if you want to swap it out for another implementation, you would need to modify the MyController class.

```java
class MyController {
    private final MyService service;

    public MyController() {
        this.service = new MyService();
    }

    public void doSomething() {
        service.perform();
    }
}

class MyService {
    public void perform() {
        System.out.println("Fruit price is ok...");
    }
}
```

Deutsche Bank
Corporate Division

Unveiling Spring Boot & Building Your First Application

Module 2

PPT-Master Version 01. There will be an extensive update with the new Deutsche Bank font and sample charts at the end of August 2024.

8 May 2024, Speaker name

Spring Boot Autoconfiguration: Unveiling the Magic

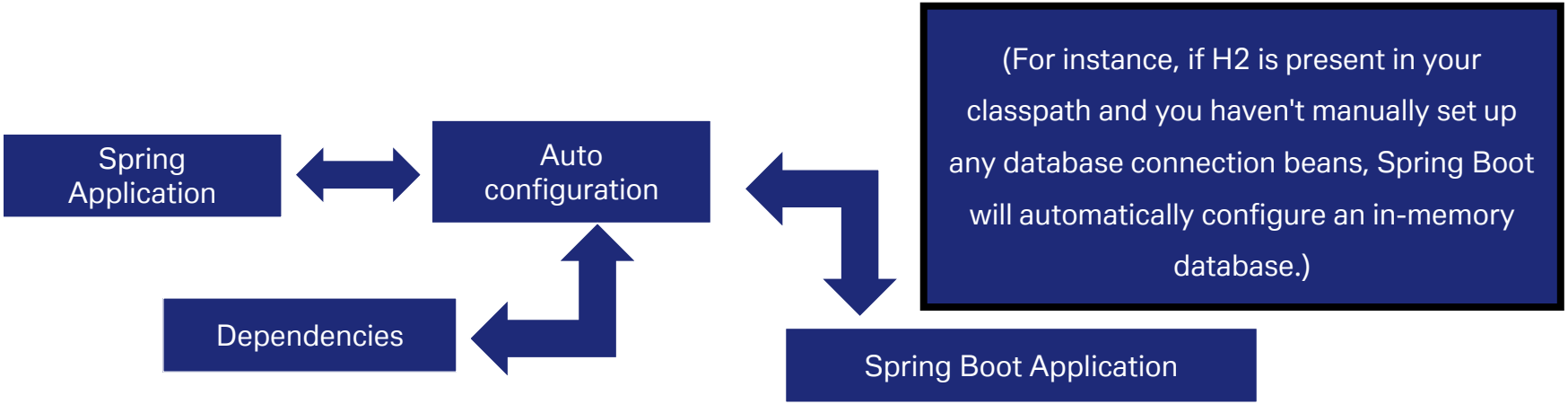
How Spring Boot Automatically Configures Beans Based on Project Dependencies

Spring Boot

Open-source Java framework

What is Spring Boot Autoconfiguration?

1. Autoconfiguration simplifies Spring application setup.
2. Automatically configures Spring beans based on classpath settings, other beans, and property settings.
3. Reduces boilerplate configuration.



How Does It Work?

1. Spring Boot scans the classpath for dependencies.
2. Conditional configuration based on available libraries and classes.
3. Utilizes @Conditional annotations for decision-making.

Conditionally configure beans based on certain conditions

```
@Bean  no usages
@Conditional(EmbeddedDataSourceCondition.class)
public DataSource embeddedDataSource() {
    JdbcDataSource dataSource = new JdbcDataSource();
    dataSource.setURL("jdbc:h2:mem:testdb");
    dataSource.setUser("sa");
    dataSource.setPassword("password");
    return dataSource;
}
```

Example: DataSource Autoconfiguration

1. Detects DataSource class in the classpath.
2. Configures DataSource bean if not already defined..
3. Annotated with @SpringBootApplication to enable auto-configuration.

```
@SpringBootApplication
public class VioletAppleApplication {

    public static void main(String[] a
```

Maven Dependency (pom.xml)

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
     <scope>runtime</scope>
</dependency>
```

application.properties

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.h2.console.enabled=true
```

Utilizing Powerful Annotations in Spring Boot

Introduction to Annotations in Spring Boot

1. Annotations simplify configuration and setup.
2. Central to Spring Boot's philosophy of convention over configuration.
3. Focus on three powerful annotations:

— @SpringBootApplication, @Configuration, @Bean.

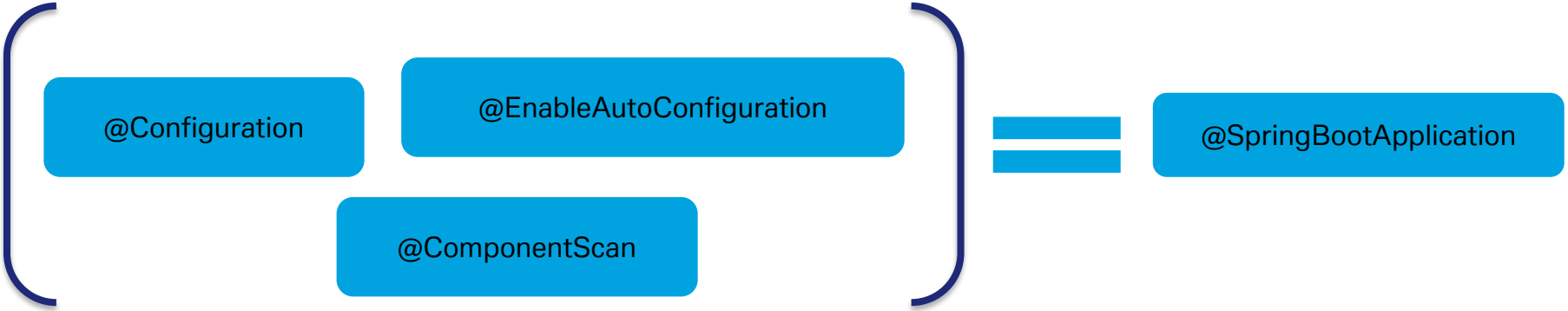
Understanding @SpringBootApplication

- Combines @Configuration, @EnableAutoConfiguration, and @ComponentScan.
- Marks the main class of a Spring Boot application.
- Simplifies application setup.

```
@SpringBootApplication
public class VioletAppleApplication {
    public static void main(String[] args) { SpringApplication.run(VioletAppleA

}
```

Breaking Down @SpringBootApplication

- @Configuration: Marks a class as a source of bean definitions.
- @EnableAutoConfiguration: Enables Spring Boot's autoconfiguration mechanism.
- @ComponentScan: Scans for Spring components in the package.



The @Configuration Annotation

- Indicates that a class declares one or more @Bean methods.
- Used for defining beans and configuration settings.
- Essential for Java-based configuration.

@Configuration: Indicates that the class contains one or more bean definitions

@Bean: Marks a method as a bean producer, and the method's return value will be managed by the Spring container.

```
@Configuration  no usages
public class AppConfig {

    @Bean  no usages
    public MyService myService() {
        return new MyService();
    }

    @Bean  no usages
    public MyRepository myRepository() {
        return new MyRepository();
    }
}
```

The @Bean Annotation

- Marks a method as a bean definition..
- Beans are managed by the Spring container.
- Allows custom bean creation and configuration.

@Bean: Marks the myService method as a bean producer, meaning that Spring will manage the instance returned by this method.

```
@Bean  no usages
public MyService myService() {
    return new MyService();
}
```

This snippet defines a single bean, MyService, which will be instantiated and managed by Spring. The MyService class includes a simple method performService.

```
class MyService {  2 usages
    public void performService() {  no usages
        System.out.println("Apple are fresh.");
    }
}
```

Deutsche Bank
Corporate Division

Understanding Maven in Spring Boot

8 May 2024, Speaker name

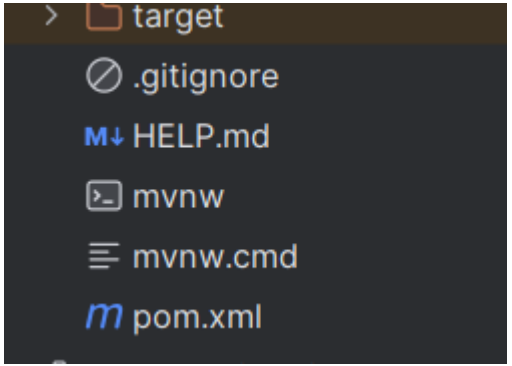
What is Maven?

- Maven is a powerful project management tool.
- Manages project dependencies and builds lifecycle.
- Ensures consistency and reproducibility in builds.

Maven in Spring Boot

- Default dependency management tool in Spring Boot.
- Simplifies configuration and dependency management.
- Uses pom.xml to manage project dependencies.

Spring Boot gives you a bunch of starter pom which make it super easy to add jars to your ClassPath.



Understanding pom.xml

- Project Object Model (POM) file.
- Central configuration file for Maven projects.
- Defines project dependencies, plugins, and configurations.

snippet of a pom.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.3.1</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
```

Key Sections in pom.xml

<dependencies>: Lists all project dependencies.

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

<properties>: Defines project properties

```
<properties>
    <java.version>17</java.version>
</properties>
```

<build>: Configures build process and plugins.

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

Managing Dependencies

Adding dependency in pom.xml file

```
<!-- Spring Boot Starter Web dependency for building web applications -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<!-- Spring Boot Starter Data JPA for database access -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Specifying versions

```
<properties>
    <java.version>17</java.version>
</properties>
```

Understanding Dependency Scope

- compile: Dependency is available in all classpaths.
- provided: It is available only at compile-time, not runtime.
- runtime: It is available only at runtime.
- test: Only available in testing.

Snippet showing scope

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

Deutsche Bank
Corporate Division

Introduction to Spring Boot Logging

- Logging is the process of recording information about a program's execution, which helps in debugging and monitoring.
- Configures Logback as the default logging framework.
- Includes Logback and SLF4J dependencies by default.

Configuring Logging in Spring Boot

- Logging settings can be configured in application.properties
- Each log level indicates the severity of the messages: ERROR (most severe), WARN, INFO, DEBUG, TRACE (most detailed)

```
<!-- Spring Boot Starter for logging -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-logging</artifactId>
</dependency>
```

Adding Spring boot Logging dependency

Example configuration in the application.properties

```
# Logging configuration
logging.level.org.springframework=INFO
logging.level.com.myapp=DEBUG
```

Example usage of logging

Depending upon the severity, use the appropriate logging level

```
@GetMapping("/log")  no usages
public String logExample() {
    // Log msg at different levels
    logger.trace("This is a TRACE level message");
    logger.debug("This is a DEBUG level message");
    logger.info("This is an INFO level message");
    logger.warn("This is a WARN level message");
    logger.error("This is an ERROR level message");
}
```

Deutsche Bank
Corporate Division

Introduction to Spring Boot Admin

- An open-source project designed to help manage and monitor Spring Boot applications
- Keep track of various aspects of applications in real time.
- Managing Application Properties and Environment.

Setting Up Spring Boot Admin

```
spring.boot.admin.client.url=http://localhost:8080/
```

Add to application.properties

Add the @EnableAdminServer annotation to your main application class to enable admin server features.

```
@SpringBootApplication
@EnableAdminServer
@EnableWebMvc
public class VioletAppleApplication {
    public static void main(String[] args) {
        SpringApplication.run(VioletAppleApplication.cla
    }
```

Add Dependency: Include the spring-boot-admin-starter-server dependency in your Spring Boot application's pom.xml

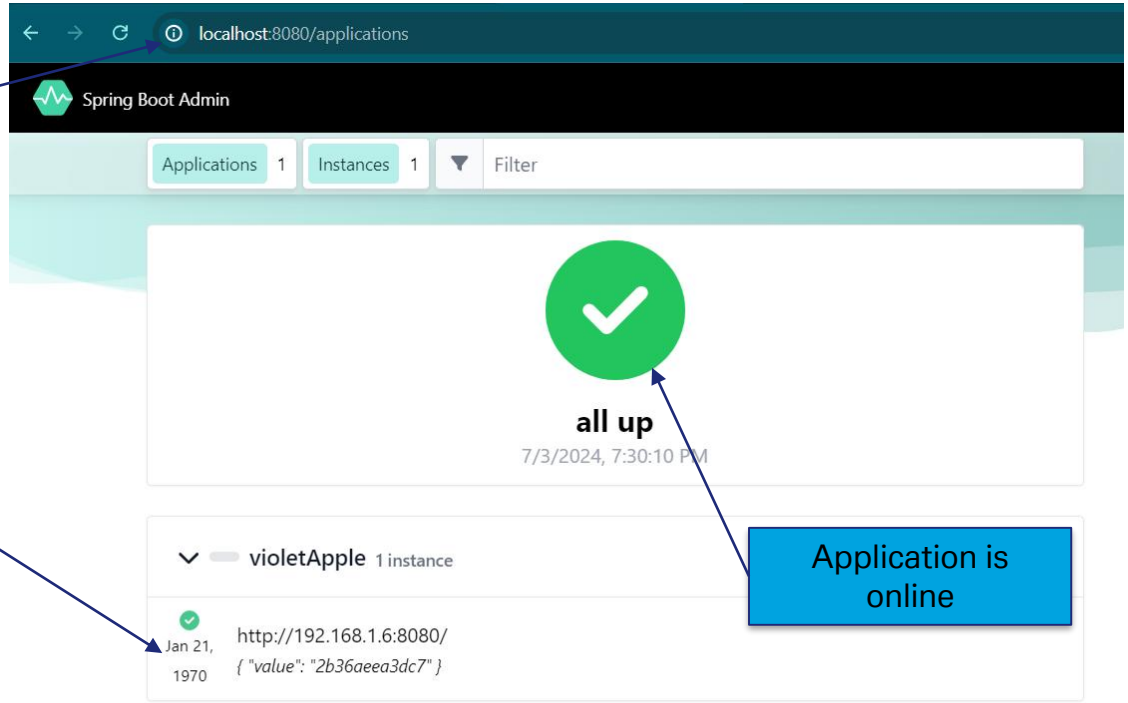
```
<!-- https://mvnrepository.com/artifact/de.codecentric/spring-b
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-starter-client</artifactId>
    <version>3.3.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/de.codecentric/spring-b
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-starter-server</artifactId>
    <version>3.3.2</version>
</dependency>
```

Monitoring and Managing

By default runs on localhost:8080

Registered application will be seen

Application is online



Deutsche Bank
Corporate Division

Starting with Zipkin

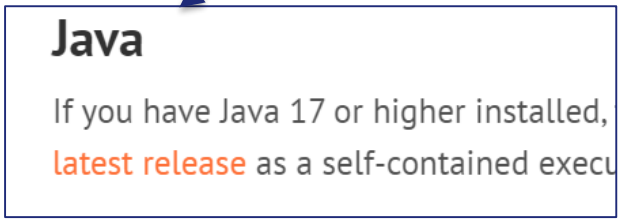
- Zipkin is an open-source distributed tracing system that helps gather timing data
- Distributed tracing allows tracking the entire journey of a request, identifying bottlenecks

8 May 2024, Speaker name

Setting Up Zipkin Server

— Download Zipkin server from https://zipkin.io/pages/quickstart

Click on latest release to download



— Run the zipkin server by running following in power shell.

```
PS C:\> java -jar .\zipkin-server-3.4.0-exec.jar
```

Setting Up Zipkin Server

Add the following dependency to pom.xml

```
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-tracing-bridge-brave</artifactId>
</dependency>
<dependency>
    <groupId>io.zipkin.reporter2</groupId>
    <artifactId>zipkin-reporter-brave</artifactId>
</dependency>
```

Add the following to application.properties

```
#Zepkin
management.tracing.sampling.probability=1.0
```

Zipkin Dashboard

After doing API calls click on run query

Zipkin | Find a trace | Dependencies | Search by trace ID | EN

RUN QUERY

Results | EXPAND ALL | COLLAPSE ALL | Service filters

Root	Start Time	Spans	Duration	
violetapple: http get /fruit/{name}	a few seconds ago (07/08 18:41:51:820)	1	119.196ms	SHOW
violetapple: http get /fruit/{name}	a few seconds ago (07/08 18:42:06:307)	1	3.201ms	SHOW
violetapple: http get /fruit/{name}	a few seconds ago (07/08 18:41:54:109)	1	2.411ms	SHOW

Api calls

URL for accessing Zipkin server:
http://127.0.0.1:9411/zipkin/

Duration of api call

Deutsche Bank
Corporate Division

Introduction to Swagger

1. An open-source framework for API development
2. Provides a standardized way to describe, produce, consume, and visualize RESTful web services
3. The Swagger ecosystem includes several tools: Swagger UI, Swagger Editor, Swagger Codegen

8 May 2024, Speaker name

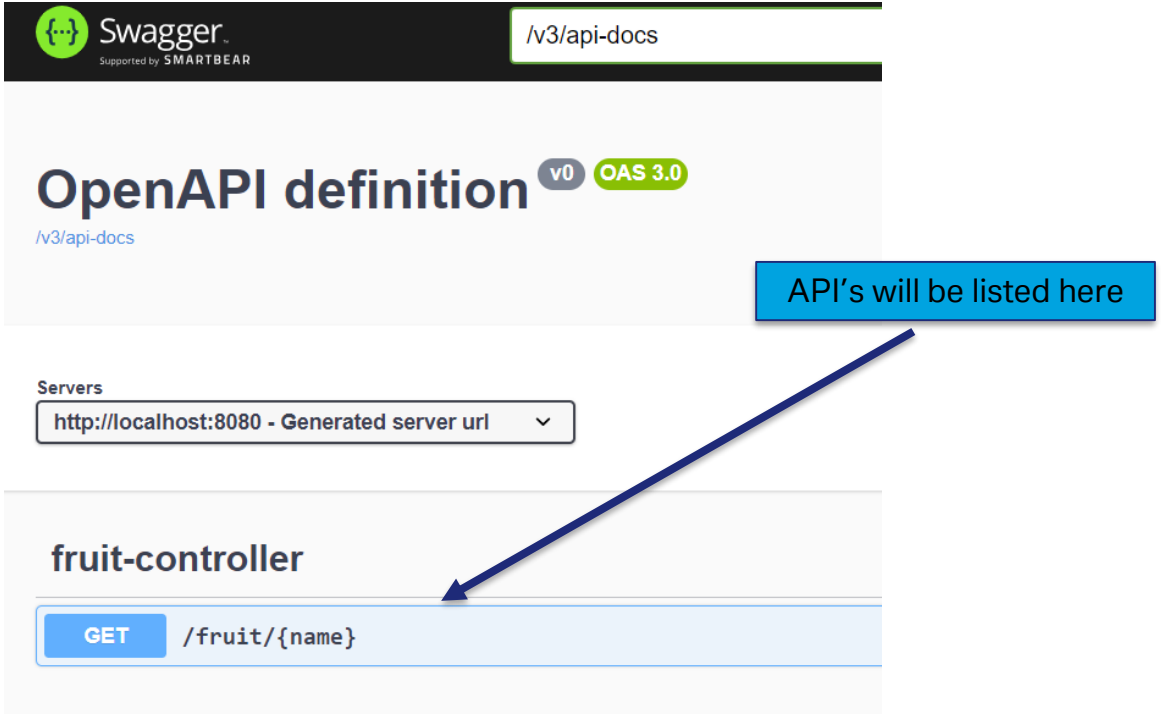
Setting Up Swagger

Add the dependency to pom.xml

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.6.0</version>
</dependency>
```

The view

Url for swagger: http://localhost:8080/swagger-ui/index.html



Deutsche Bank
Corporate Division

Spring Basics

Deutsche Bank
Corporate Division

Blog Application

Deutsche Bank
Corporate Division

Spring Data

Deutsche Bank
Corporate Division

Spring Boot Creating REST APIs

Deutsche Bank
Corporate Division

Spring Boot Testing Lab: Testing Fitness Application

Deutsche Bank
Corporate Division

Programmer to API Developer Sandbox