Spring Boot Feature Introduction

Spring Boot simplifies application development

1.18.5

Objectives

After completing this lesson, you should be able to do the following

- Explain what Spring Boot is and how it simplifies application development
- Explain and use Spring Boot features

Agenda

- **What is and Why Spring Boot?**
- Spring Boot Features
  - Dependency management
  - Auto-Configuration
  - Packaging and Runtime
  - Integration Testing
- Getting Started with Spring Boot
- Summary

What is Spring Boot?

- Takes “opinionated” view of the Spring platform and third-party libraries
- Supports different project types like Web or Batch
- Handles most low-level, predictable set-up for you

- It is *NOT*
  - A code generator
  - An IDE plug-in

See: Spring Boot Reference
http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle

Why Spring Boot?

- Provide a radically faster and widely accessible getting-started experience for all Spring development
- Be opinionated out of the box but get out of the way quickly as requirements start to diverge from the defaults
- Provide a range of non-functional features that are common to large classes of projects
    - Embedded servers, metrics, health checks, externalized configuration, containerization, etc.

Agenda

- What is and Why Spring Boot?
- **Spring Boot Features**
  - **Dependency management**
  - Auto-Configuration
  - Packaging and Runtime
  - Integration Testing
- Getting Started with Spring Boot
- Summary

How do you manage Dependencies?

- Modern Java application require a large number of dependencies - How do you make sure they are compatible?
  - Spring Boot JARs, Spring JARs, common 3rd party JARs, etc.

- Spring Boot’s parent or Starters to the rescue
  - Leverages existing dependency management schemes

- Fine-grained dependency management still possible
  - Exclude dependencies you do not use
  - Define the dependencies explicitly yourself - find the correct version from the Starters

Spring Boot Parent POM

- Defines versions of key dependencies
  - Uses a **`dependencyManagement`** section internally
  - Through **`spring-boot-dependencies`** as a parent

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.7.5</version>
</parent>
```

Defines properties for dependencies, for example:
`${spring-framework.version}= 5.3.23`

- Defines Maven plugins
- Sets up Java version

Spring Boot “*Starter*” Dependencies

- Easy way to bring in multiple coordinated dependencies
  - Including “*Transitive*” Dependencies

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

Version not needed!
Defined by parent

Resolves ~ 18 JARs!

spring-boot-.jar* *spring-core-*.jar*
spring-context-.jar* *spring-aop-*.jar*
spring-beans-.jar* **-slf4j-*.jar*
...

Test “*Starter*” Dependencies

- Common test libraries

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

Resolves
spring-test-.jar*
junit-.jar*
mockito-.jar*
…

Many Starters are available out of the box

- Not essential but *strongly* recommended for getting started
- Coordinated dependencies for common Java enterprise frameworks
  - Pick the starters you need in your project
- To name a few:
  - **`spring-boot-starter-jdbc`**
  - **`spring-boot-starter-data-jpa`**
  - **`spring-boot-starter-web`**
  - **`spring-boot-starter-batch`**

See: Spring Boot Reference, Starter POMs
https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-starter

Agenda

- What is and Why Spring Boot?
- **Spring Boot Features**
  - Dependency management
  - **Auto-Configuration**
  - Packaging and Runtime
  - Integration Testing
- Getting Started with Spring Boot
- Summary

Auto-configuration enabled by @EnableAutoConfiguration

- Spring Boot automatically creates beans it thinks you need based on some conditions
- *`@EnableAutoConfiguration`* annotation on a Spring Java configuration class enables auto-configuration

```
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan
public class Application {
   public static void main(String[] args) {
            SpringApplication.run(Application.class, args);
      }
}
```

`SpringApplication` is actually a Spring Boot class

Shortcut: @SpringBootApplication

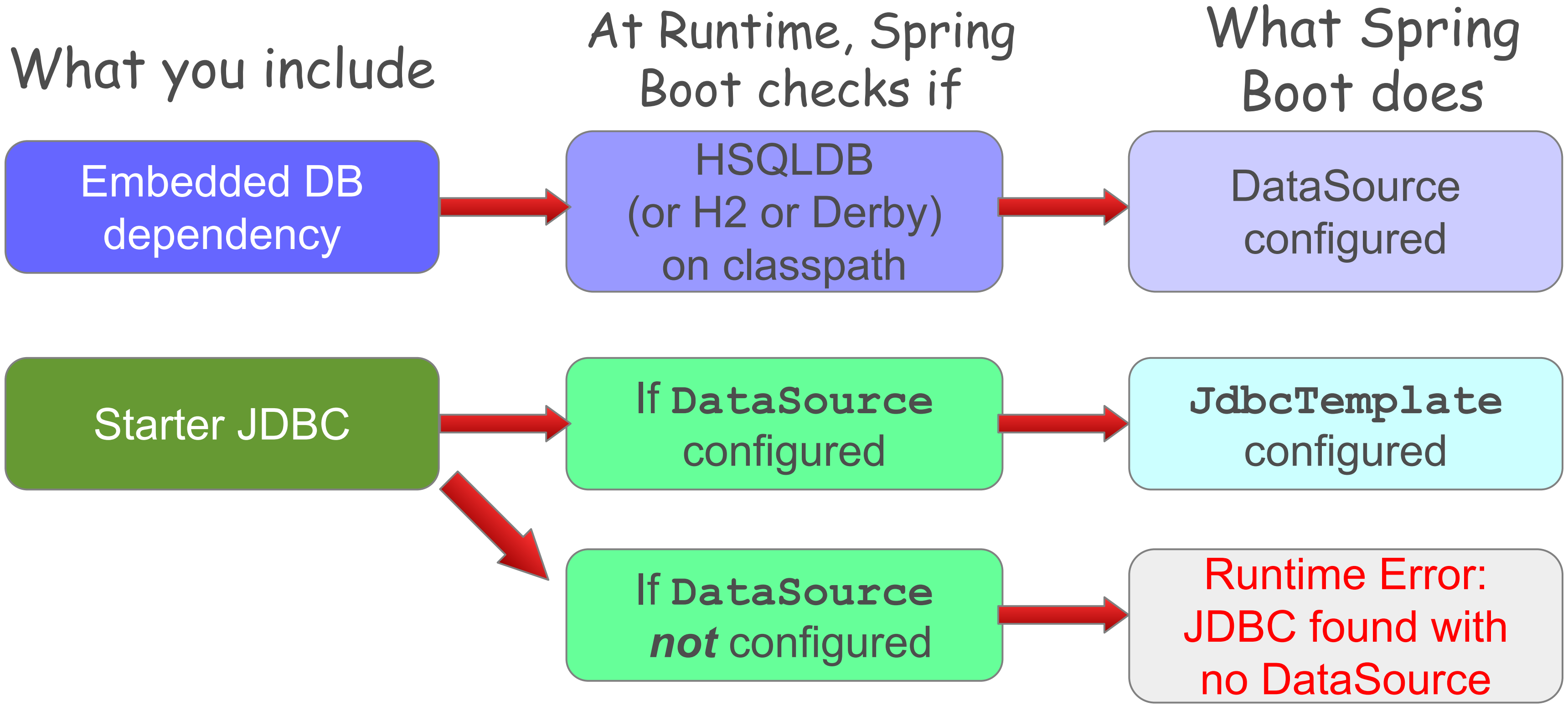
- Very common to use @SpringBootConfiguration, @EnableAutoConfiguration, and @ComponentScan together

```
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan("example.config")
 public class Application {
     ...
 }
```

```
@SpringBootApplication
(scanBasePackages="example.config")
public class Application {
   ...
}
```

@SpringBootConfiguration simply extends **@Configuration** – see **@SpringBootTest** for how it is used in testing - will be covered later

Examples of Auto-configuration: DataSource, JdbcTemplate


What you include
At Runtime, Spring Boot checks if
What Spring Boot does
Embedded DB dependency
HSQLDB (or H2 or Derby) on classpath
DataSource configured
Starter JDBC
If **DataSource** configured
JdbcTemplate configured
If **DataSource** ***not*** configured
Runtime Error: JDBC found with no DataSource

Agenda

- What is and Why Spring Boot?
- **Spring Boot Features**
  - Dependency management
  - Auto-Configuration
  - **Packaging and Runtime**
  - Integration Testing
- Getting Started with Spring Boot
- Summary

Fat JAR is created through the Spring Boot Plugin

- A “fat” JAR contains all dependencies including Tomcat for web application
- Can be run directly using **`java -jar`** command

- To create a fat JAR
  - Add Spring Boot plugin to your Maven POM or Gradle Build file
  - Build JAR in usual way
    - **`gradle assemble`** or **`mvn package`**
  - Two JARs are created
    - **`my-app.jar`** the executable “fat” JAR
    - **`my-app.jar.original`** the “usual” JAR

Spring Boot Plugin - Maven

- What it does
  - Extend **`package`** goal to create fat JAR
  - Add **`spring-boot:run`** goal to run your application

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

Packaging Result

- "**`mvn package`**" execution produces (in **`target`**)

```
22M  yourapp-0.0.1-SNAPSHOT.jar
 5K  yourapp-0.0.1-SNAPSHOT.jar.original
```

  – **`.jar.original`** contains only your code (a traditional JAR file)
  – **`.jar`** contains your code *and* all dependencies – executable
    - *Notice that it is much bigger*

Packaging as a “Container Image”

Fat jar packaging may not be sufficient when running on modern cloud platforms:

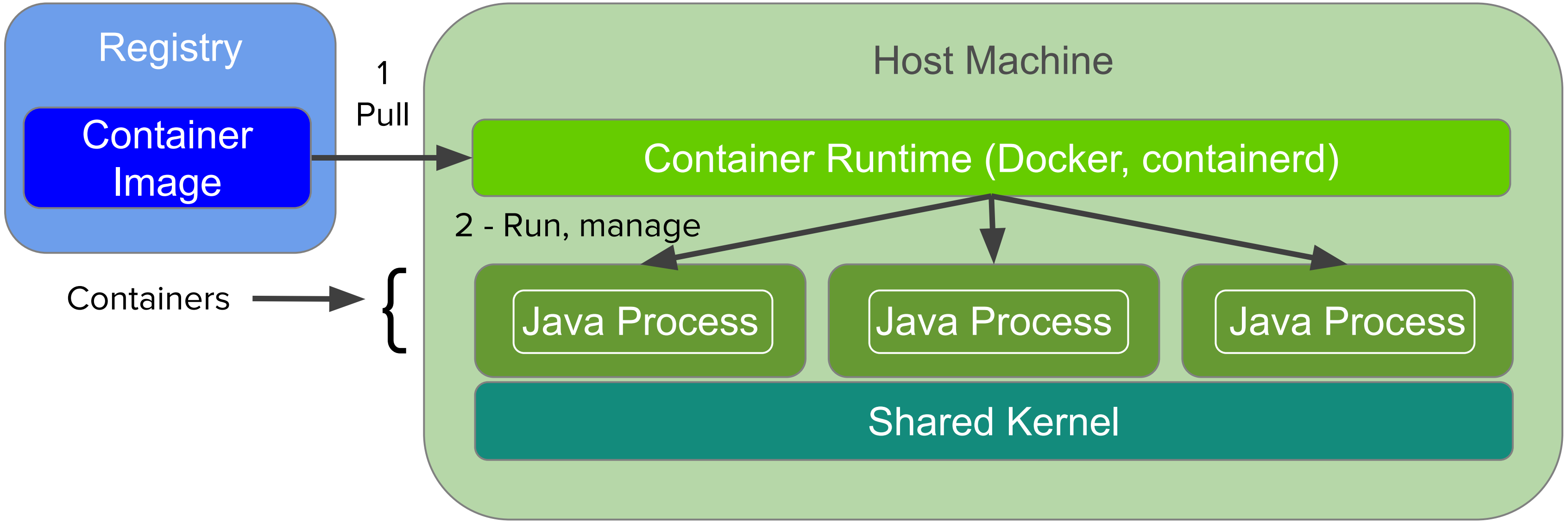
- The fat jar still needs a java runtime to run it, and modern cloud platforms may not give it to you.

- Running the fat jar directly is not the most efficient or secure way to run the Spring Boot app on modern *container* platforms.

Spring Boot supports building “container images”, that solve both of these problems.

What is a "Container"?

A *container* is the set of one or more processes isolated from the rest of the system. A *container runtime* starts one or more container by sourcing a *container image*, configuring it on its host machine, and running it. The containers are limited in the amount of resources they can use.



What is a “Container Image” ?

A *container image* is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, and a command to run a worker process

File System:

- Java runtime
- Spring framework libraries
- Spring boot libraries
- Application java classes
- Spring Boot app launcher libraries
- Other shared libraries

Entry point:

- Java command to start application

Identified by a location coordinate:

```
{registry}/{namespace}/{repository}:{version
}
```

Example: `gcr.io/myproject/myapp:1.0.0`

Build a container image with Docker

1. Source code built with Maven or Gradle

2. “Dockerfile” (source control)

```
FROM adoptopenjdk:11-jre-hotspot
VOLUME /tmp
COPY target/*.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

Base image (includes a linux distro and jdk and runtime)

Copy the maven built fat jar to image file system (Layer)

Command to start the application java process

2. Build a container image and tag with its registry location, using Dockerfile at current directory:

```
docker build -t gcr.io/myproject/myapp:1.0.0 .
```

3. Publish the container image to the container registry (namespaced repository) at version 1.0.0:

```
docker push gcr.io/myproject/myapp.1.0.0
```

List container image built from Dockerfile

1. List the container images:

```
docker images
```

2. Review the output:

```
REPOSITORY                TAG        IMAGE ID       CREATED          SIZE
gcr.io/myproject/myapp    1.0.0      53b1978fe80e   14 minutes ago   253MB
```

Unique image SHA

Run container from Dockerfile built image

1. Run the container with docker:

```
docker run gcr.io/myproject/myapp:1.0.0
```

2. Review the output:

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.5)


2022-03-22 21:55:39.791  INFO 1 --- [           main] myproject.myapp.MyappApplication         : Starting MyappApplication using Java 11.0.11 on
50429f733ff1 with PID 1 (/myapp.jar started by root in /)
2022-03-22 21:55:39.795  INFO 1 --- [           main] myproject.myapp.MyappApplication         : No active profile set, falling back to 1 default profile:
"default"
2022-03-22 21:55:40.269  INFO 1 --- [           main] myproject.myapp.MyappApplication         : Started MyappApplication in 0.881 seconds (JVM running for
1.261)
```

Agenda

- What is and Why Spring Boot?
- **Spring Boot Features**
  - Dependency management
  - Auto-Configuration
  - Packaging and Runtime
  - **Integration Testing**
- Getting Started with Spring Boot
- Summary

Test: *@SpringBootTest*

- Alternative to @SpringJUnitConfig

```
@SpringBootTest(classes=Application.class)
public class TransferServiceTests {
    @Autowired
    private TransferService transferService;

    @Test
    public void successfulTransfer() {
        TransferConfirmation conf = transferService. transfer(...);
        ...
    }

}
```

Loads the specified configuration applying same Spring Boot defaults

```
@SpringBootApplication(scanBasePackages="transfers")
public class Application {
        // Bean methods
}
```

Testing: *@SpringBootConfiguration*

- @SpringBootTest searches for @SpringBootConfiguration class
  - Creates application context for the test
  - Provided the configuration is in a package *above* the test
  - Only one `@SpringBootConfiguration` allowed in a hierarchy

```
@SpringBootTest  // classes not needed
public class TransferServiceTests {
    // Same tests as previous slide

}
```

```
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan("transfers")
public class Application {
        // Bean methods
}
```

Agenda

- What is and Why Spring Boot?
- Spring Boot Features
  - Dependency management
  - Auto-Configuration
  - Packaging and Runtime
  - Integration Testing
- **Getting Started with Spring Boot**
- Summary

Hello World example

- Just three files to get a running Spring application

pom.xml

Setup Spring Boot (and any other) dependencies

application.properties

General configuration

Application class

Application launcher

Maven is just one option. You can also use Gradle or Ant/Ivy.
Our slides will use Maven.

Spring Initializr - What is it?

- Framework, API, and default implementation to generate initial Spring Boot application projects

- Spring's public web-site: http://start.spring.io

- Or build your own: https://github.com/spring-io/initializr

Spring Initializr - What is its value?

- Constructs starting template of Spring Boot projects
  - Mainly folder structure, Maven/Gradle build files
- Simplify and curate dependency management
  - Gradle or Maven supported
  - Java, Groovy or Kotlin
- Accessible as a “New Project” wizard in STS, IntelliJ IDE (Ultimate version only)

Hello World (1a) - Maven descriptor

pom.xml

```xml
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.5</version>
</parent>

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

Parent POM

Defines dependencies for:
Spring JDBC, JDBC Connection Pool,
Spring Boot itself

Embedded
SQL Database

No versions –
defined by parent
POM

Hello World (2) - application.properties

- Properties can be defined to supplement autoconfiguration or override autoconfiguration

application.properties

```
# Set the log level for all modules to 'ERROR'
logging.level.root=ERROR

# Tell Spring JDBC Embedded DB Factory where
# to obtain DDM and DML files
spring.sql.init.schema-locations=classpath:rewards/schema.sql
spring.sql.init.data-locations=classpath:rewards/data.sql
```

Hello World (3) - Application Class

```
@SpringBootApplication
public class Application {

    public static final String QUERY = "SELECT count(*) FROM T_ACCOUNT";

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

    @Bean
    CommandLineRunner commandLineRunner(JdbcTemplate jdbcTemplate){

            return args -> System.out.println("Hello, there are "
                    + jdbcTemplate.queryForObject(QUERY, Long.class)
                    + " accounts");
     }
}
```

Application.java

This annotation *turns on* Spring Boot

JdbcTemplate bean is automatically configured through auto-configuration

Main method will be used to run the packaged application from the command line

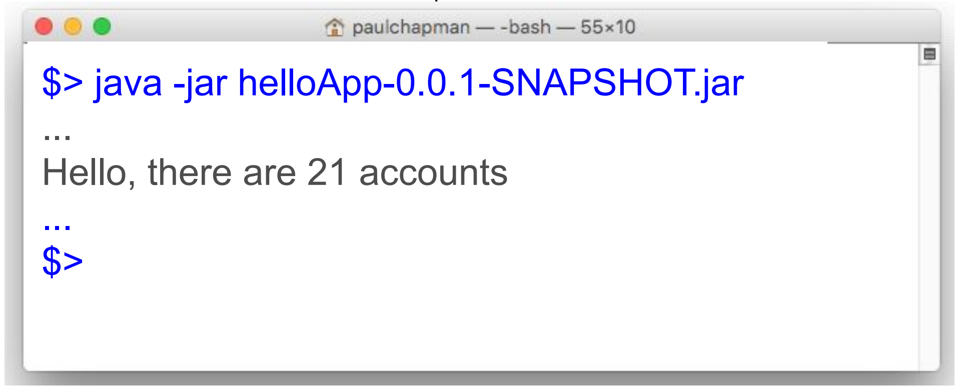
Hello World (4) - Putting it all together

```
mvn package
```

```
helloApp-0.0.1-SNAPSHOT.jar
```

generated file

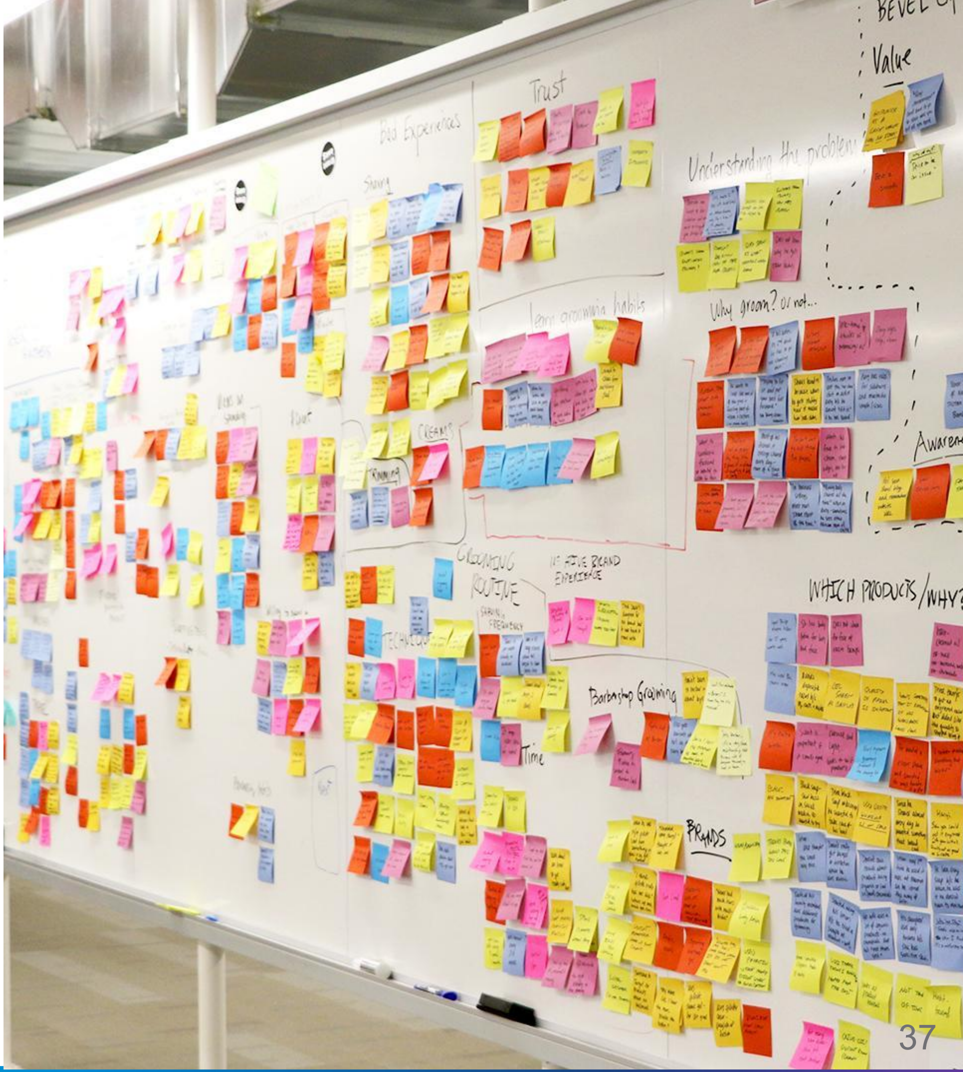
```
java -jar helloApp-0.0.1-SNAPSHOT.jar
```



```
$> java -jar helloApp-0.0.1-SNAPSHOT.jar
...
Hello, there are 21 accounts
...
$>
```

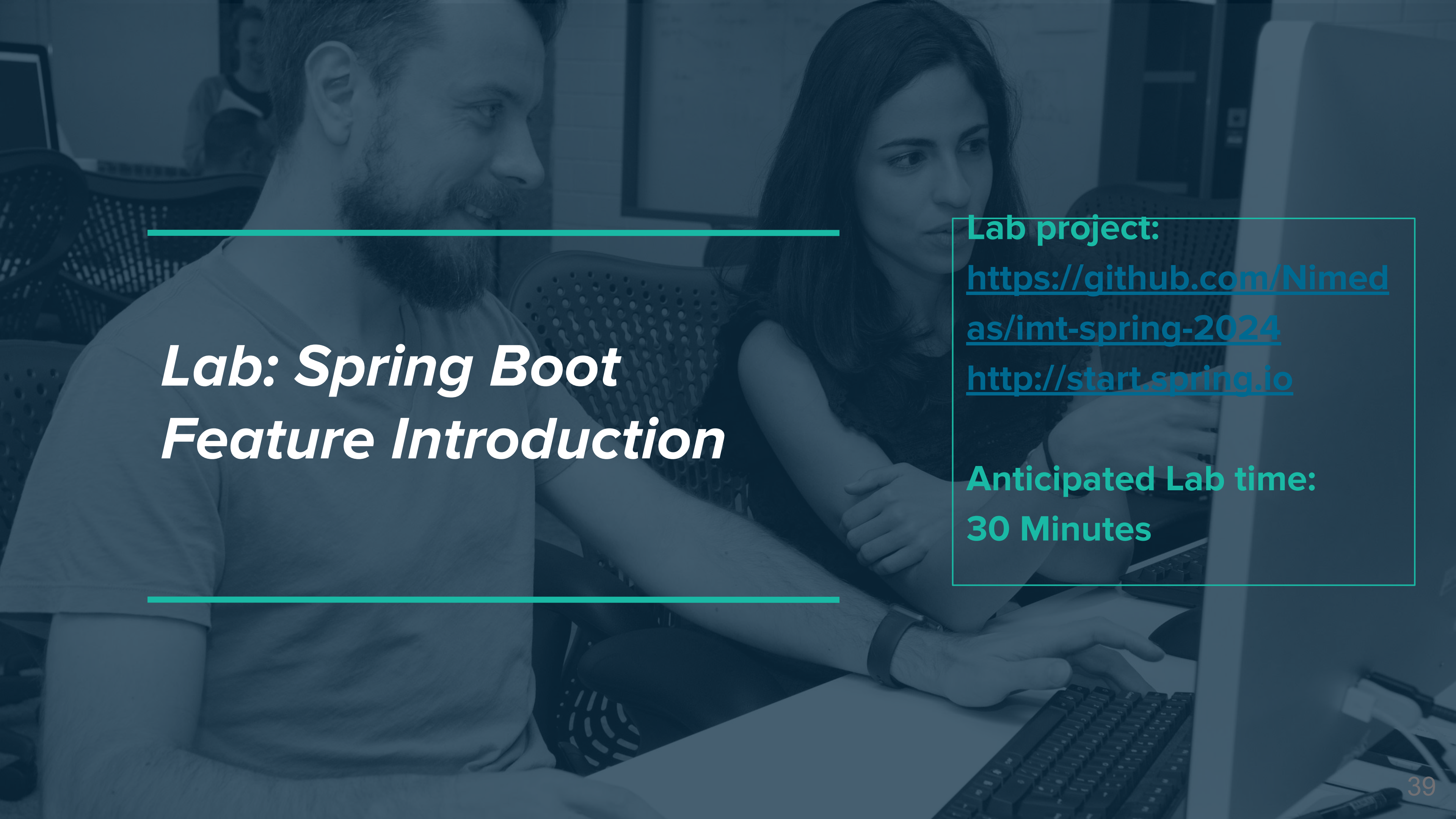
Agenda

- What is and Why Spring Boot?
- Spring Boot Features
  - Dependency management
  - Auto-Configuration
  - Packaging and Runtime
  - Integration Testing
- Getting Started with Spring Boot
- **Summary**



Summary

- Spring Boot significantly simplifies Spring setup
  - Will setup much of your application for you
  - Simplifies dependency management
  - Uses in-built defaults (opinions) to do the obvious setup
    - Automatically creates beans it thinks you need
  - Builds “fat” JAR
  - You can use containers to wrap the Spring Boot application
  - Provides `@SpringBootTest` for enhanced testing features



Lab: Spring Boot Feature Introduction

Lab project:

https://github.com/Nimedas/imt-spring-2024

http://start.spring.io

Anticipated Lab time:

30 Minutes