

Pivotal.

Spring Boot



Agenda

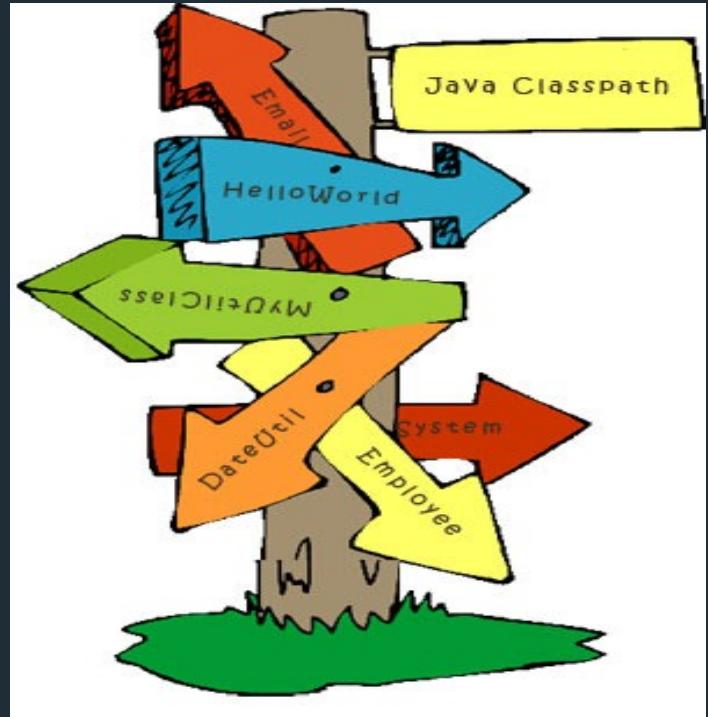
1. Challenges building non-Boot applications
2. What is Spring Boot?
3. Capabilities

Agenda

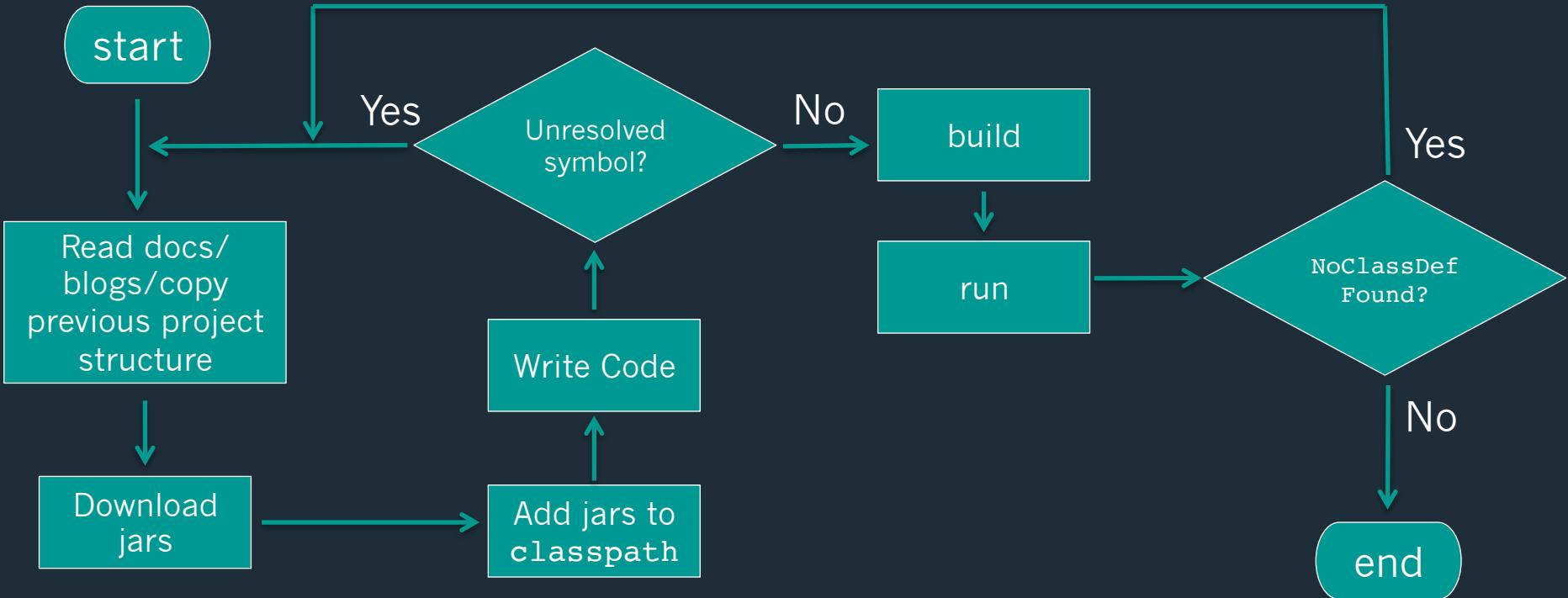
1. Challenges building non-Boot applications
2. What is Spring Boot?
3. Capabilities

Classpath Hell

- A Jar is missing
- There is one Jar too many
- A class is not visible where it should be



Resolving Classpath Hell



Complex Application Server Setup

Setup

1. Perform setup operations as “root”
2. Update `/etc/hosts` with entries for “localhost”
3. Create a new group (`oinstall`) and user (`oracle`)
4. Create directories in which Oracle will be installed
5. Add `ORACLE_BASE`, `ORACLE_HOME`, `MW_HOME`, `WLS_HOME`, `WL_HOME`, `DOMAIN_BASE`, `DOMAIN_HOME`, `JAVA_HOME` to `bash_profile`

Installation

6. Download the installer
7. Run the installer as “oracle” user
8. Choose Inventory Directory and Operating System Group
9. Choose `MIDDLEWARE_HOME`
10. Install

Configuration

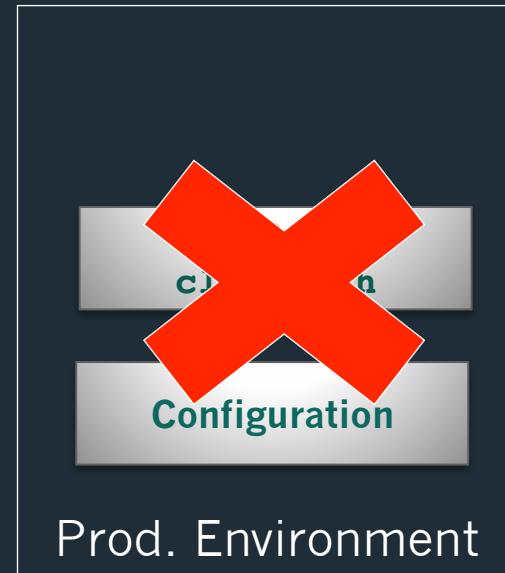
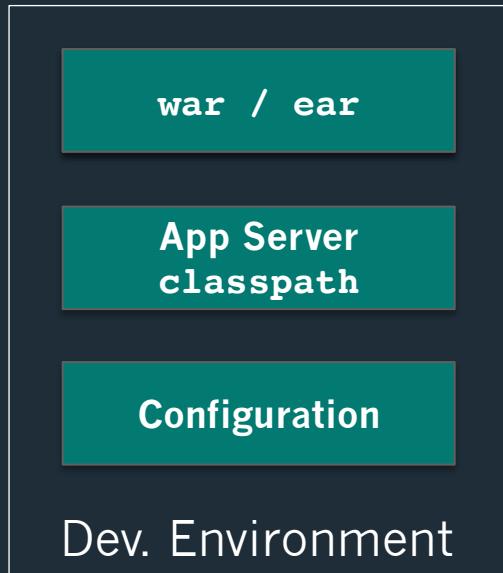
11. Choose a Domain Location
12. Create a Domain using product templates
13. Provide administrator credentials
14. Choose JDK
15. Configure Admin Server
16. Configure Node Manager
17. Configure Managed Servers
18. Configure Clusters

Post Installation

19. Start WebLogic server
20. Manage and monitor through an administrator console

Environment Drift

Standard war/ear packaging causes drift across environments



Boilerplate code

Every application that accesses a relational database with JDBC
needs to configure a `JdbcTemplate` and a `DataSource`

```
@Bean
public JdbcTemplate jdbcTemplate(
    DataSource dataSource) {
    return new JdbcTemplate(dataSource);
}
```

```
@Bean
public DataSource dataSource() {
    return new EmbeddedDatabaseBuilder()
        .setType(EmbeddedDatabaseType.H2)
        .addScripts("schema.sql", "data.sql")
        .build();
}
```

Boilerplate code

To get a db connection

- Set Driver
- Set Connection URL
- Set Username
- Set Password
- Catch Exceptions

```
public static Connection getConnection() {  
    if (dbConnection != null) {  
        return dbConnection;  
    } else {  
        try {  
            InputStream inputStream =  
                DbUtil.class.getClassLoader().  
                getResourceAsStream("db.properties");  
            Properties properties = new Properties();  
            if (properties != null) {  
                properties.load(inputStream);  
  
                String dbDriver = properties.getProperty("dbDriver");  
                String connectionUrl = properties.getProperty("connectionUrl");  
                String userName = properties.getProperty("userName");  
                String password = properties.getProperty("password");  
  
                Class.forName(dbDriver).newInstance();  
                dbConnection = DriverManager.  
                    getConnection(connectionUrl, userName, password);  
            }  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
    return dbConnection;  
}
```

Boilerplate code

For CRUD Operations

- Prepare Statement
- Set Parameters
- Execute Statement
- Catch Exceptions

```
public void save(String userName, String password, String firstName,
    String lastName, String dateOfBirth, String emailAddress) {
    try {
        PreparedStatement prepStatement = dbConnection.prepareStatement(
            "insert into student(userName, password, " +
            "firstName, lastName, dateOfBirth, emailAddress) " +
            "values (?, ?, ?, ?, ?, ?)");
        prepStatement.setString(1, userName);
        prepStatement.setString(2, password);
        prepStatement.setString(3, firstName);
        prepStatement.setString(4, lastName);
        prepStatement.setDate(5, new java.sql.Date(
            new SimpleDateFormat("MM/dd/yyyy").
            parse(dateOfBirth.substring(0, 10)).getTime()));
        prepStatement.setString(6, emailAddress);

        prepStatement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (ParseException e) {
        e.printStackTrace();
    }
}
```

Debugging

Scouring log files

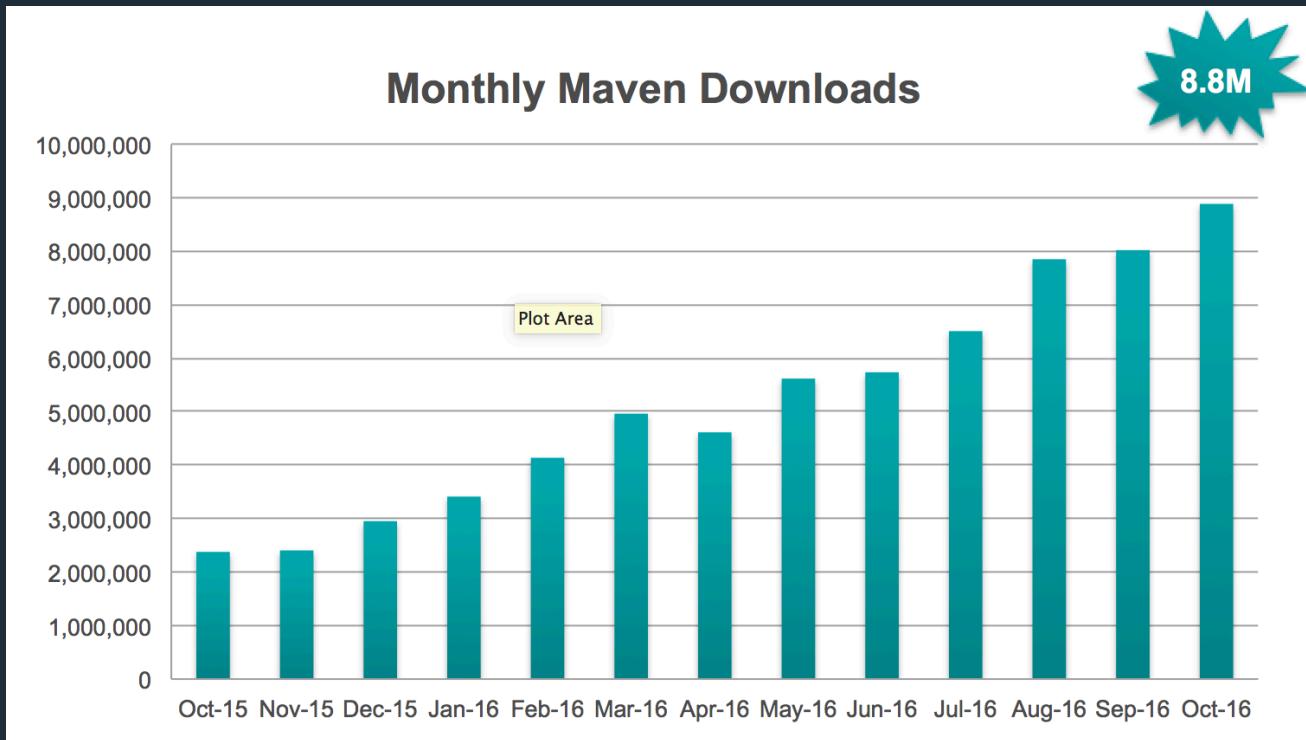
```
[2011-02-10 14:14:40,013][?] INFO:server:database user - openpg
[2011-02-10 14:14:40,013][?] INFO:server:initialising distributed
objects services
[2011-02-10 14:14:45,200][?] INFO:server:OpenERP version - 6.0.1
[2011-02-10 14:14:45,210][?] INFO:server:addons_path - C:\Program
Files\OpenERP 6.0\Server\addons
[2011-02-10 14:14:45,210][?] INFO:server:database hostname - localhost
[2011-02-10 14:14:45,210][?] INFO:server:database port - 5432
[2011-02-10 14:14:45,210][?] INFO:server:database user - openpg
[2011-02-10 14:14:45,210][?] INFO:server:initialising distributed
objects services
[2011-02-10 14:14:46,361][?] INFO:web-services:starting HTTP service at
0.0.0.0 port 8069
[2011-02-10 14:14:46,371][?] INFO:web-services:starting HTTPS service
at 0.0.0.0 port 8071
[2011-02-10 14:14:46,371][?] INFO:web-services:Registered XML-RPC over
HTTP
```

Agenda

1. Challenges building non-Boot applications
2. What is Spring Boot?
3. Capabilities

Spring Boot is an opinionated framework to
simplify bootstrapping and development of new
Spring Applications

Spring Boot Adoption



Agenda

1. Challenges building non-Boot applications
2. What is Spring Boot?
3. Capabilities

Capabilities

- Quick start project generation
- Automatic project dependency management
- Configuration drift prevention
- Conditional configuration

- Developer Productivity Tooling
- Auto-configuration
- Monitoring and management endpoints
- Microservices-friendliness

Spring Initializr

(Quick start project generation)

Pivotal™

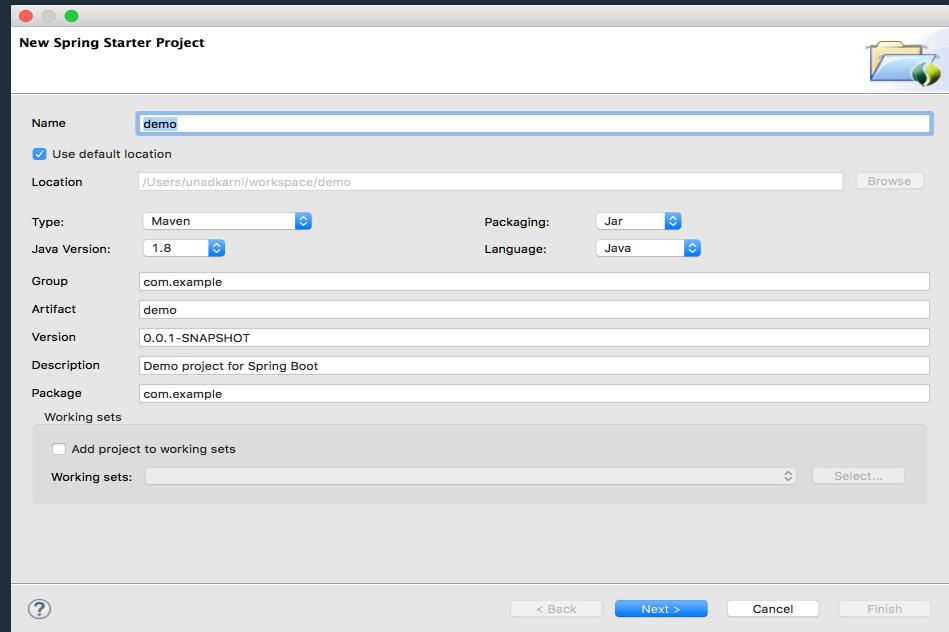
Spring Initializr is a configurable service to
consistently and easily generate a quick start
project

- Generates a Spring Boot project structure
- Provides a Maven/Gradle build specification
- Doesn't generate application code
- You can customize the Spring Initializr
 - <https://github.com/spring-io/initializr/>

Spring Initializr has three supported interfaces

1. IDE

- Eclipse
- IntelliJ



2. Web-based Interface

SPRING INITIALIZR bootstrap your application now

Generate a with Spring Boot

Project Metadata

Artifact coordinates

Group

Artifact

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web
Full-stack web development with Tomcat and Spring MVC

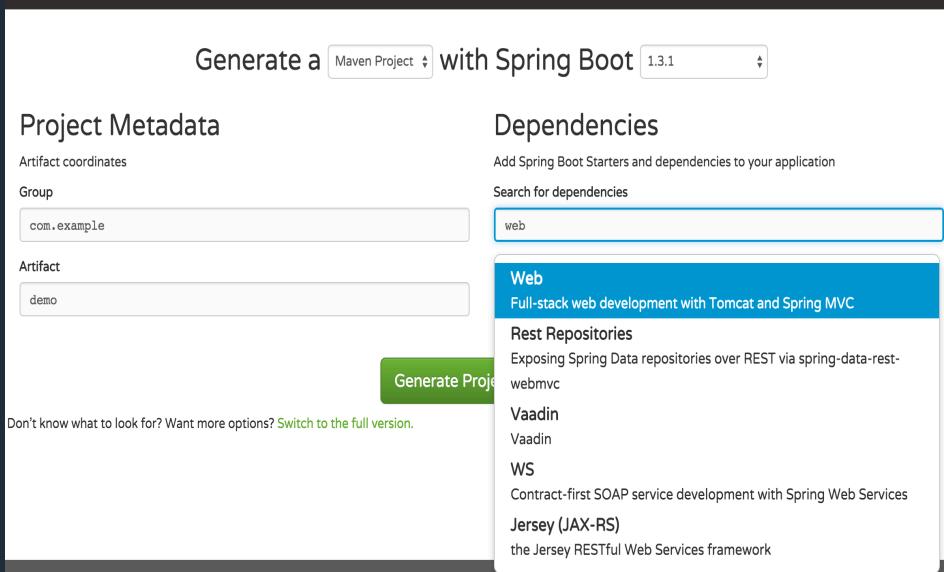
Rest Repositories
Exposing Spring Data repositories over REST via `spring-data-rest-webmvc`

Vaadin
Vaadin

WS
Contract-first SOAP service development with Spring Web Services

Jersey (JAX-RS)
the Jersey RESTful Web Services framework

Don't know what to look for? Want more options? [Switch to the full version.](#)



3. Command Line Interface

The screenshot shows a terminal window titled "workspace — bash — 80x27". The user runs the command `spring init --dependencies=web,data-jpa my-project`. The output shows the project was extracted to `'/Users/unadkarni/workspace/my-project'`. A red box highlights this command. Below it, the `tree` command shows the directory structure of the project:

```
my-project
├── mvnw
├── mvnw.cmd
└── pom.xml
src
└── main
    ├── java
    │   └── com
    │       └── example
    │           └── DemoApplication.java
    ├── resources
    │   ├── application.properties
    │   ├── static
    │   └── templates
    └── test
        └── java
            └── com
                └── example
                    └── DemoApplicationTests.java
```

An orange arrow points from the text "Directory structure of Boot Maven project" to the `src/main/java/com/example/DemoApplication.java` file in the tree output.

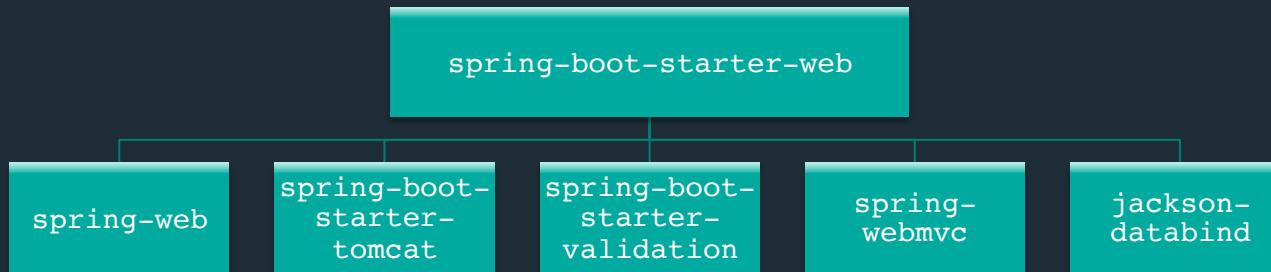
12 directories, 6 files

Directory structure of Boot Maven project

Spring Boot starters

(Automatic project dependency management)

- Are virtual packages deployed to Maven central
- They pull in other dependencies while containing no code of their own



Project pom.xml

- Set parent of your pom.xml as spring-boot-starter-parent
 - Inherit preset version numbers

```
<?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-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>demo</name>
  <description>Demo project for Spring Boot</description>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.3.1.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
```

Starters vs Current Practice

Starter versions determined by version of Spring Boot used.

Dependencies guaranteed to be compatible.

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

Concise. Boot pom.xml without version numbers.

Are the version numbers compatible with each other?

Is the list of dependencies complete?

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>4.1.6.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>1.9.2.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>5.0.6.Final</version>
  </dependency>
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.4.190</version>
  </dependency>
</dependencies>
```

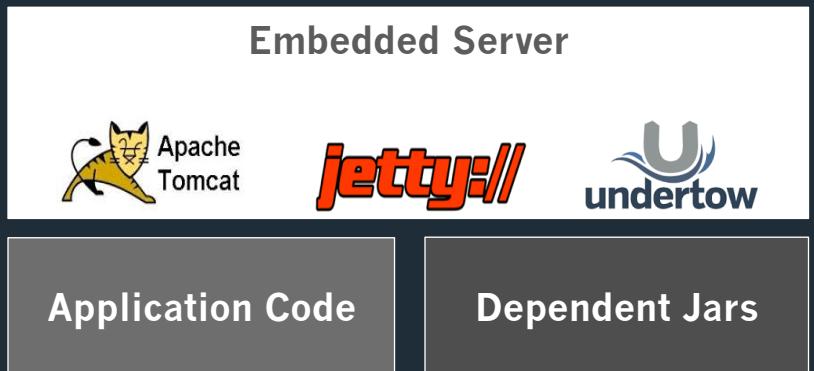
Verbose. Non-boot pom.xml with version numbers.

Packaging

(Configuration drift prevention)

Make Jar not War!

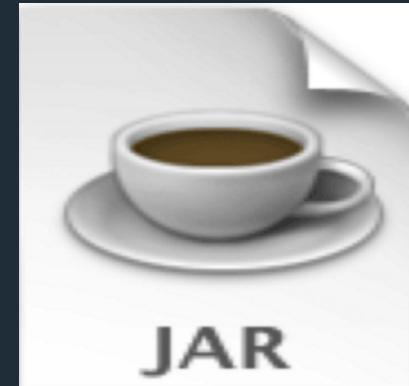
Boot's Maven / Gradle plugins produce an executable “*fat jar*”



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

Benefits

- No installation of application servers
- No setting classpath
- Promotes consistency across environments
- Can be started as Unix services using
 - init.d
 - systemd



The Traditional JEE App Server is Dead!



Setup

1. Perform setup operations as “root”
2. Update `/etc/hosts` with entries for “localhost”
3. Create a new group (`oinstall`) and user (`oracle`)
4. Create directories in which Oracle will be installed
5. Add `ORACLE_BASE`, `ORACLE_HOME`, `MW_HOME`, `WLS_HOME`, `WL_HOME`, `DOMAIN_BASE`, `DOMAIN_HOME`, `JAVA_HOME` to bash profile

Installation

6. Download the installer
7. Run the installer as “oracle” user
8. Choose Inventory Directory and Operating System Group
9. Choose `MIDDLEWARE_HOME`
10. Install

Configuration

11. Choose Domain Location
12. Create a domain using product templates
13. Provide administrator credentials
14. Choose JDK
15. Configure Admin Server
16. Configure Node Manager
17. Configure Managed Servers
18. Configure Clusters

Post Installation

19. Start WebLogic server
20. Manage and monitor through an administrator console

Profiles

(Conditional configuration)

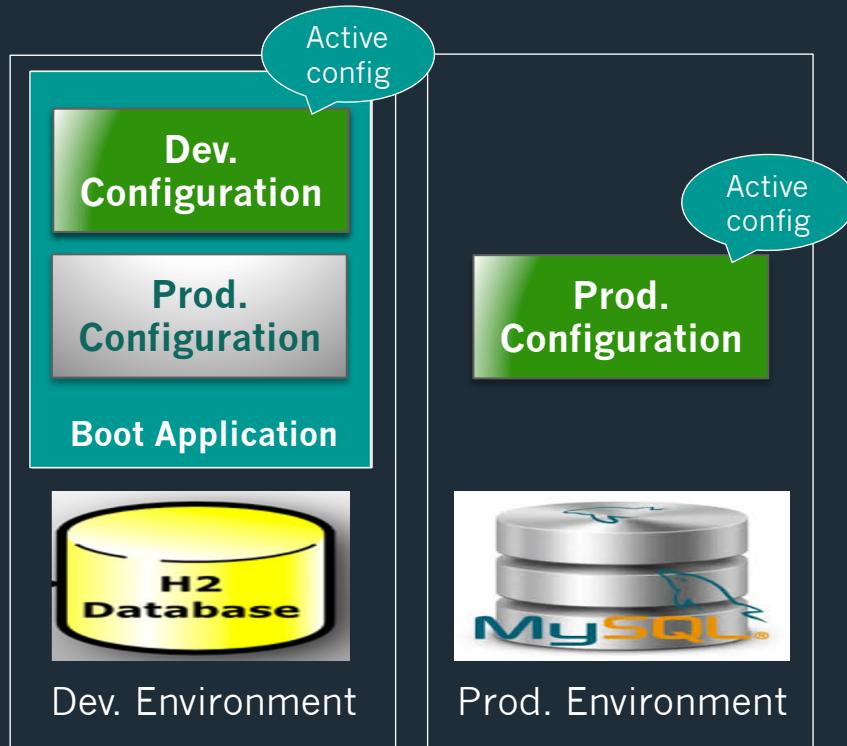
Segregate parts of the application configuration
and make it available in certain environments via

- Annotations
- Properties file

Mark @Component or @Configuration with @Profile to limit when it is loaded.

```
@Configuration  
@Profile("development")  
public class DevelopmentConfiguration {  
    // The @Profile requires that the "development"  
    // profile be active at runtime  
    // for this configuration to be applied  
    // Activate this profile in the properties  
    // file with spring.profile.active=development  
    // command line with --spring.profiles.active=development  
}
```

```
@Configuration  
@Profile("production")  
public class ProductionConfiguration {  
    // The @Profile requires that the "production"  
    // profile be active at runtime  
    // for this configuration to be applied  
    // Activate this profile in the properties  
    // file with spring.profile.active=production  
    // command line with --spring.profiles.active=production  
}
```



Checkout code from source control
Build an executable jar
Configuration activated based on active profile
Deploy anywhere
No Code Changes
No Configuration Changes

Precedence of externalized configuration

1. Command line arguments.
2. Properties from `SPRING_APPLICATION_JSON` (inline JSON embedded in an environment variable or system property)
3. JNDI attributes from `java:comp/env`.
4. Java System properties (`System.getProperties()`).
5. OS environment variables.
6. A `RandomValuePropertySource` that only has properties in `random.*`.
7. Profile-specific application properties outside of your packaged jar (`application-{profile}.properties` and YAML variants)
8. Profile-specific application properties packaged inside your jar (`application-{profile}.properties` and YAML variants)
9. Application properties outside of your packaged jar (`application.properties` and YAML variants).
10. Application properties packaged inside your jar (`application.properties` and YAML variants).
11. `@PropertySource` annotations on your `@Configuration` classes.
12. Default properties (specified using `SpringApplication.setDefaultProperties()`).

Developer Tools

(Developer Productivity Tools)

- Automatic restart
- *LiveReload*

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

Automatic Restart

- Restarts a running application when files are changed in the classpath
- Excludes static resources from restart considerations
 - /META-INF/resources
 - /resources
 - /static
 - /public
 - /templates

LiveReload support

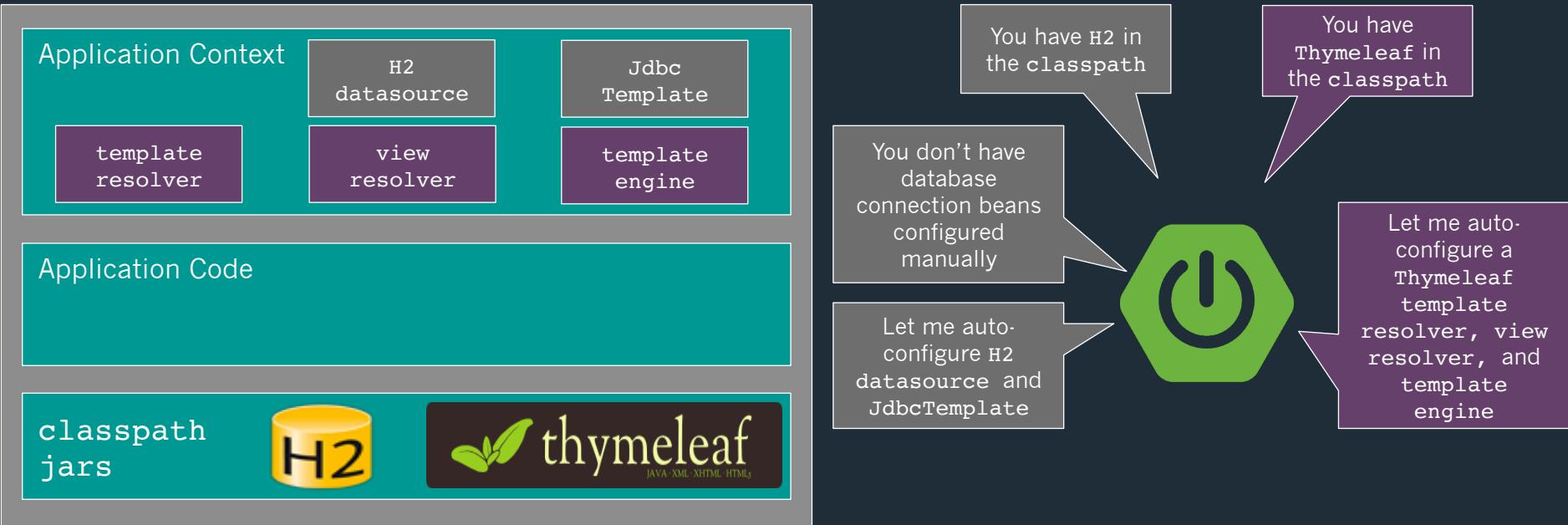
- Install the *LiveReload* plugin into your browser
- Boot starts an embedded *LiveReload* server
- Changes to resources trigger a browser refresh automatically



Auto-Configuration

Auto-Configuration

Automatically configures application based on the `classpath`



Explicitly exclude Auto-Configuration

Application Context

template
resolver

view
resolver

template
engine

Application Code

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

classpath



You have H2 in
your classpath

You want to
explicitly exclude
auto-configuration
of the DataSoure

I'll step aside.

You have
Thymeleaf in
your classpath

Let me auto-
configure a
Thymeleaf
template
resolver, view
resolver, and
template
engine

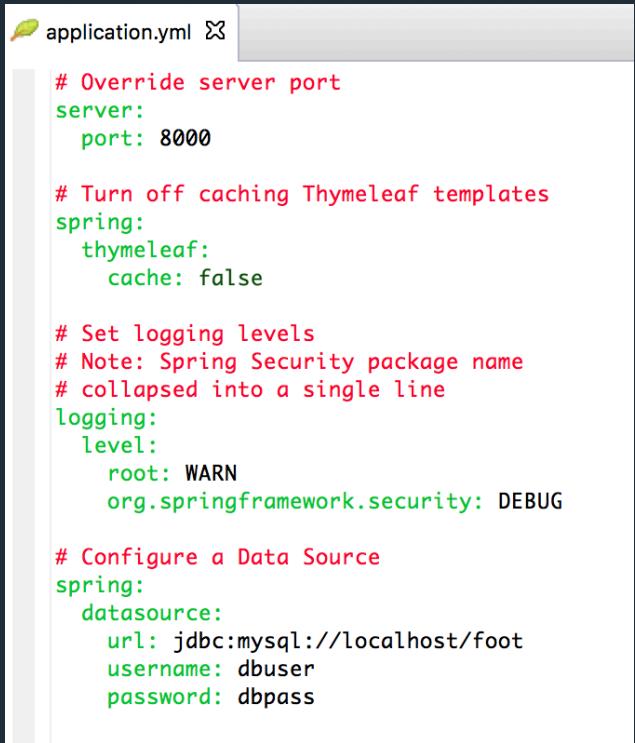


Pivotal™

Fine Tuning Auto-Configuration

Over 300 properties to tweak beans in a Boot application via

- Command line arguments
- Environment variables
- `.properties / .yml file`



```
# Override server port
server:
  port: 8000

# Turn off caching Thymeleaf templates
spring:
  thymeleaf:
    cache: false

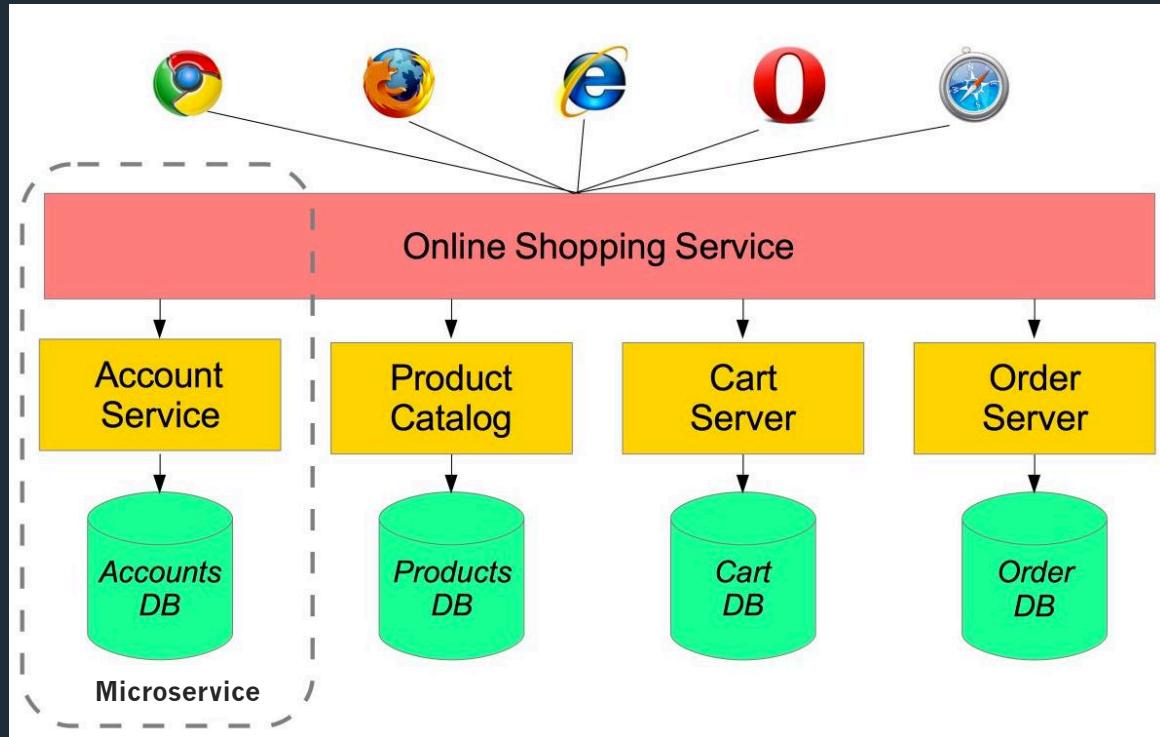
# Set logging levels
# Note: Spring Security package name
# collapsed into a single line
logging:
  level:
    root: WARN
    org.springframework.security: DEBUG

# Configure a Data Source
spring:
  datasource:
    url: jdbc:mysql://localhost/foot
    username: dbuser
    password: dbpass
```

Microservices

(Microservices-friendliness)

Each microservice is a
Spring Boot fat jar



Pivotal.

Thank You

$$I(x) = \begin{cases} h e^{bx} & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} P_{X_0}(t) &= j[x(s)] = P_{X_0(t-s)} = j[x_0] e^{-s \lambda_x} \\ X_{j(t)} &= P_{\{X_{t-s}=j\}} X_0 \\ X_0 &= 6 \\ 0 &= x + C \quad dx = 1 \\ \int x \, dx &= \int dt \quad x = \int dt = 1 \\ x &= t + C \\ X_0 &= 6 \end{aligned}$$

$$\begin{aligned} X_H &= 1 + \frac{1}{2} X_0 \\ P_{X_0}(t) &= j[x(s)] = P_{X_0(t-s)} = j[x_0] e^{-s \lambda_x} \\ P_{X_0}(t) &= j[X_0] = P_{\{X(t) = j\}} \end{aligned}$$

