CONSULTING.TECHNOLOGY.OUTSOURCING

# Spring and Cloud Applications

People matter, results count.

## Objectives

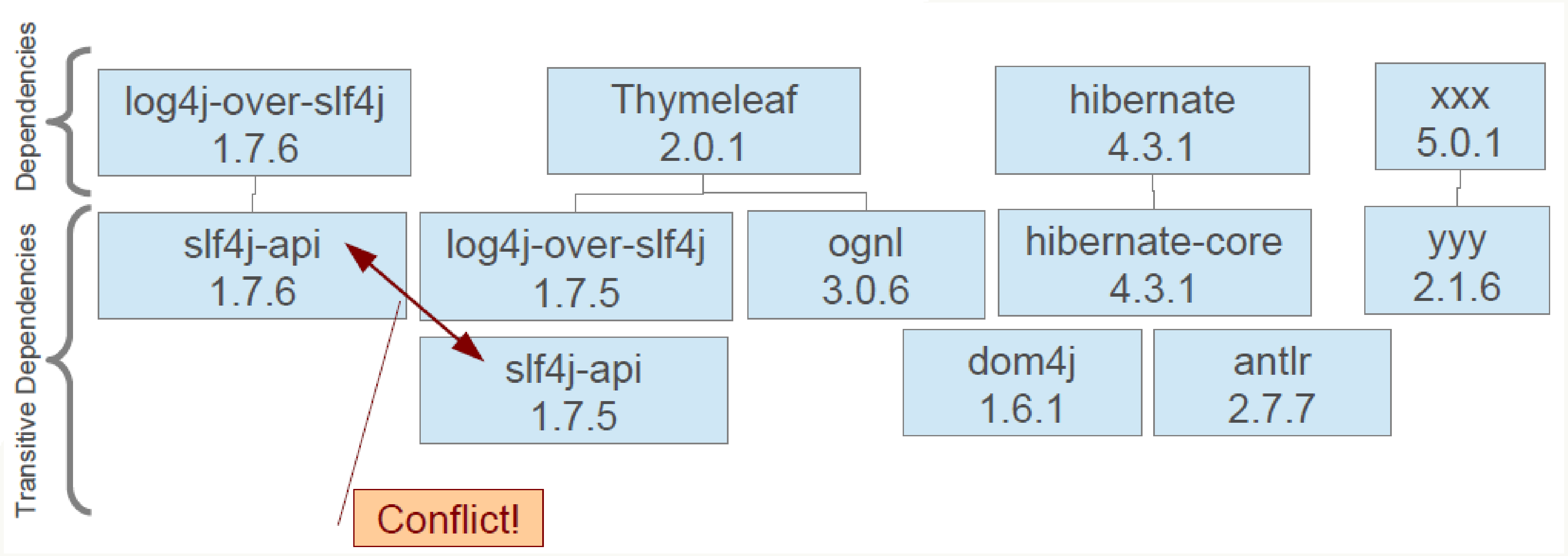
- Purpose:
  - Use Spring IO and Spring Boot
  - Configure and run applications using Profiles
- Product:
  - Spring IO, Spring Boot are not essentials for Cloud application
  - But they simplify code and dependency management
  - So cloud deployment is much easier
- Process:
  - Making JVM Cloud application eaiser.

## Table of Contents

## SPRING IO

# What is Spring IO?

- Working with open source libraries can be challenging
  - Transitive Dependencies - OS libraries that depend on other libraries
  - Need Maven, Gradle, etc. to manage, but still difficult



## Spring IO

- Defines a set of Maven library dependencies
  - For Spring and other commonly used JARs
  - Version-set known to work together

"Bill of Materials"

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>io.spring.platform</groupId>
            <artifactId>platform-bom</artifactId>
            <version>1.0.1.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Use Gradle if you prefer

## Using Spring IO

- Now define Maven dependencies in usual way
  - No need for <version>, Spring IO will decide

```
<dependencies>
    <dependency>
        <groupId>org.springframework.data</groupId>
        <artifactId>spring-data-commons</artifactId>
    </dependency>

    <dependency>
        <groupId>org.hsqldb</groupId>
        <artifactId>hsqldb</artifactId>
    </dependency>
    ...
</dependencies>
```

No version needed!

## what Dependencies are Available?

- Many, many of the JARs commonly used in Spring JVM applications
  - Spring, Groovy core and project JARs, ...
  - Apache Commons, Tiles, Solr, Velocity, Tomcat, ...
  - JPA, hibernate, EclipseLink, MyBatis, NoSQL DBs...
  - Logging, OXM, JSON, Metrics, ...
  - Web, Servlets, Jetty, Thymeleaf ...

- Presently, 480+ JAR versions are managed

http://docs.spring.io/platform/docs/current/refernce/htmlsingle/#appendix-dependency-version

# SPRING BOOT

## what is Spring Boot?

- A quick way to start building a Spring project
- An opinionated runtime for Spring projects
- Supports different project types, like Web and Batch
- Can be used to create containerless apps

- It is not:
  - A code generator
  - An IDE plug-in

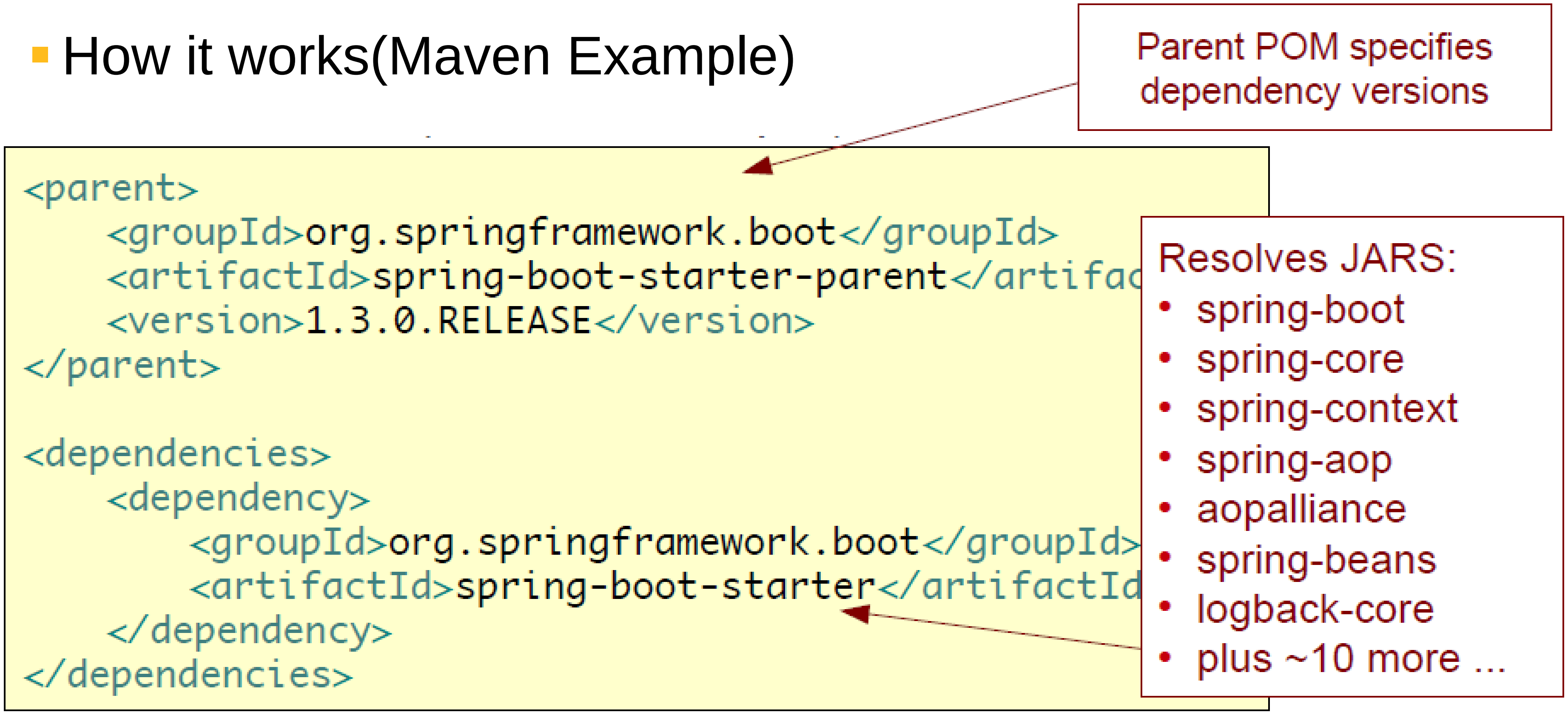
## Opinionated Runtime?

- Spring Boot uses sensible defaults, mostly based on the classpath contents
- For example:
  - Sets up a JPA Entity Manager Factory if a JPA implementation is on the classpath
  - Creates a default Spring MVC setup, is Spring MVC is on the classpath
- Everything can be overridden very easily
  - But most of the time not needed
  - Relies heavily on Spring 4 @ Conditional annotation

## SPRING BOOT DEMO

# Spring Boot Dependency Management

- How it works(Maven Example)



Parent POM specifies dependency versions

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifac
    <version>1.3.0.RELEASE</version>
</parent>

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

Resolves JARS:
- spring-boot
- spring-core
- spring-context
- spring-aop
- aopalliance
- spring-beans
- logback-core
- plus ~10 more ...

- Eliminates need to document standard dependencies
- See http://projects.spring.io/spring-boot for latest version

## Spring Boot - Adding Dependencies

- To add capabilities, add additional "started" dependencies:

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

Adds JARS:
- spring-web
- spring-webmvc
- jackson-databind
- hibernate-validator
- tomcat-embed
- *plus transitive dependencies*

- For full list of Spring boot Starter dependencies see:
- https://githuh.com/spring-projects/spring-boot/tree/master/spring-boot-starters

## Spring Boot JPA Example

- Dependencies

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

    <dependency>
        <groupId>org.hsqldb</groupId>
        <artifactId>hsqldb</artifactId>
    </dependency>
</dependencies>
```

*Starter* dependency set for RDBMS application: JDBC, JPA, Hibernate

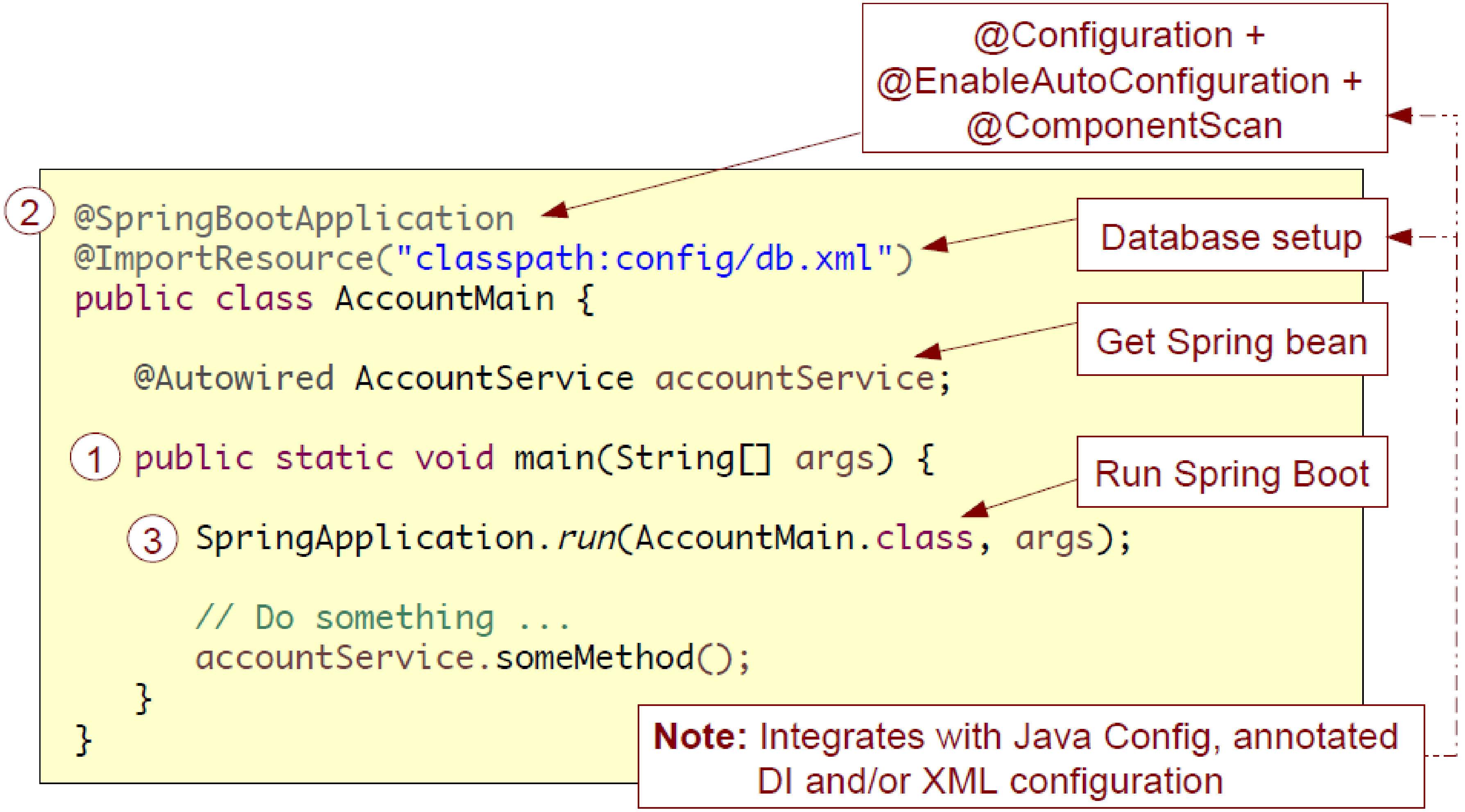
HSQL Database

# Programming Spring Boot

- Run application using a main() method ①
  - Like we used to before containers!
- @SpringBootApplication ②
  - Enables automatic configuration: @EnableAutoConfiguration
    - Spring Boot scans classpath, setting up typical defaults
  - marks class as a @Configuration class
  - Enabled @ComponentScan from current base package
- SpringApplication class ③
- Initiates Spring Boot
- Tells Spring Boot which class to start with
  - Usually this class

① ② ③ see next slide ….

# Example Spring Boot Application



```java
@SpringBootApplication
@ImportResource("classpath:config/db.xml")
public class AccountMain {

   @Autowired AccountService accountService;

   public static void main(String[] args) {

      SpringApplication.run(AccountMain.class, args);

      // Do something ...
      accountService.someMethod();
   }
}
```

**Note:** Integrates with Java Config, annotated DI and/or XML configuration

# Web application using Spring Boot

- Define Spring Controllers in usual way
- Add spring-boot-starter-web dependency

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

– The main() program starts up embedded Tomcat

```
2 @SpringBootApplication
  public class WebMain {
1   public static void main(String[] args) {
        SpringApplication.run (WebMain.class, args);   3
    }
}
```

- Spring Boot detects web-artifacts, starts Tomcat
- This method *only* returns when Tomcat shuts down

# Run as a War

- Two Steps:
  - Change Packaging to war: <packaging>war</packaging>
  - Extend Spring boot's Servlet Initializer

```
@SpringBootApplication
public class WebMain extends SpringBootServletInitializer {

   // Can still have main() if you like
   public static void main(String[] args) { ... }

   @Override
   protected SpringApplicationBuilder
         configure(SpringApplicationBuilder application) {
      return application.sources(WebMain.class);
   }

}
```

No WEB.XML!  (Unless you want)

## Overriding Defaults

- Spring Boot takes an opinionated approach to application decisions
  - Example Opinion: Web applications should use Tomcat
  - Example Opinion: JPA application should use Hibernate
- what if you have different opinions?
  - Use Jetty, use EclipseLink
- No Problem!
  - Simply override the dependencies (See next)

## Overriding Defaults - Option 1

- Use Jetty instead of Tomcat
  - Just add its starter dependency

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

   <!-- Use Jetty as embedded servlet container not Tomcat -->
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-jetty</artifactId>
   </dependency>

   ...
</dependencies>
```

# Overriding Defaults - Option 2a

- Use EclipseLink instead of Hibernate
  1. Change maven dependency
  2. Override the default entityManagerFactory bean
     - Normally created by Spring Boot, will use yours instead

persistence.xml

```xml
<!-- Minimal persistence.xml, not needed with Hibernate, but
     EclipseLink adheres to the specification. -->
<persistence xmlns="http://java.sun.com/xml/ns/persistence" ...>
   <persistence-unit name="account">
   </persistence-unit>
</persistence>
```

Maven

```xml
<!-- Add EclipseLink dependency -->
<dependency>
  <groupId>org.eclipse.persistence</groupId>
  <artifactId>org.eclipse.persistence.jpa</artifactId>
</dependency>
```

## Overriding Defaults - Option 2b

```
@Configuration
public class JpaConfig {
   @Bean
   public EntityManagerFactoryBean emf (DataSource dataSource) {
      JpaVendorAdapter adapter = new EclipseLinkJpaVendorAdapter();
      // Set desired properties.

      Properties props = new Properties();
      // Set desired properties

      LocalContainerEntityManagerFactoryBean emfb =
        new LocalContainerEntityManagerFactoryBean();
      emfb.setDataSource(dataSource);
      emfb.setPersistenceUnitName("account");
      emfb.setJpaProperties(props);
      emfb.setJpaVendorAdapter(adapter);
      return emfb;
   }
}
```

Spring Java Config – remember to enable component-scanning

## Spring Boot Maven Properties

- Can override product versions using properties in your Maven POM or Gradle build file

```
<properties>
    <!-- Get Spring Boot to set Java version -->
    <java.version>1.7</java.version>

    <!-- Class containing main() -->
    <start-class>io.pivotal.cf.Application</start-class>

    <!-- Servlet version – downgrade to 2.x if you prefer -->
    <servlet-api.version>3.0.1</servlet-api.version>

    <!-- Tomcat version -->
    <tomcat.version>7.0.55</tomcat.version>
</properties>
```

https://github.com/spring-projects/spring-boot/blob/master/spring-boot-dependencies/pom.xml

## Spring Boot Application Properties

- Spring Boot automatically looks for application.properties
  - Or use application.yml if you prefer YAML
- Use predefined properties to control Spring Boot

```
spring.datasource.url=jdbc:mysql://localhost/test
spring.datasource.username=dbuser
spring.datasource.password=dbpass
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

application.properties

```
spring:
  datasource:
    url: jdbc:mysql://localhost/test
    username: dbuser
    password: dbpass
    driver-class-name: com.mysql.jdbc.Driver
```

application.yml

No tabs!

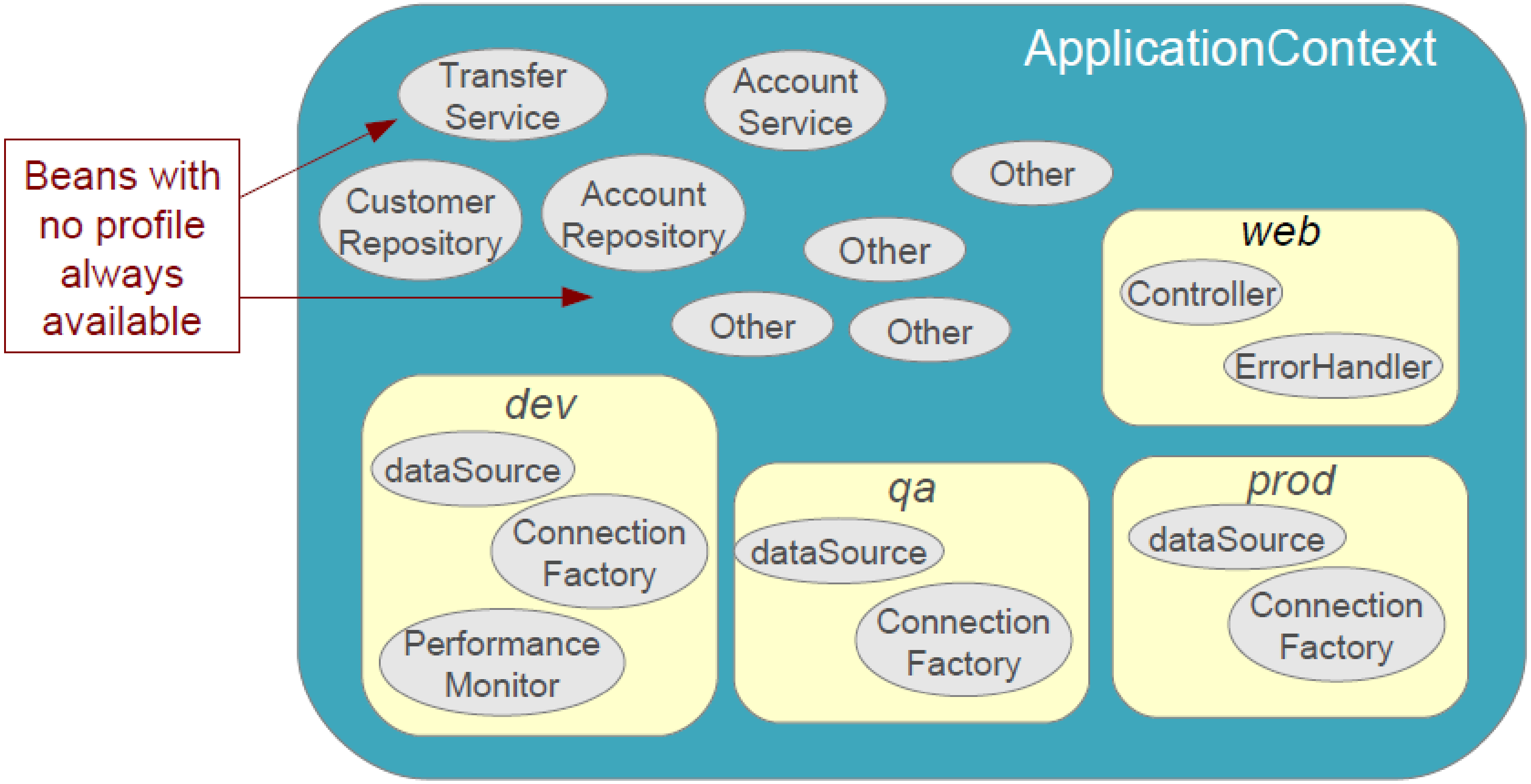
## SPRING PROFILES

## Spring Profiles

- Part of Spring Framework since 3.0
  - Allow multiple different configurations
  - Select the ones you want by selecting one or more profiles
  - integrated into Spring Testing framework also

- Beans can be grouped into Profiles
  - Profiles can represent purpose: "web", "offline"
  - Or environment: "dev","qa","uat","prod","cloud"
  - Or implementation: "jdbc","jpa"
  - Beans included /excluded based on profile membership

## Example Profiles



## Defining profiles

- Add @Profile annotation to component or configuration
- Or qualify <beans> in XML

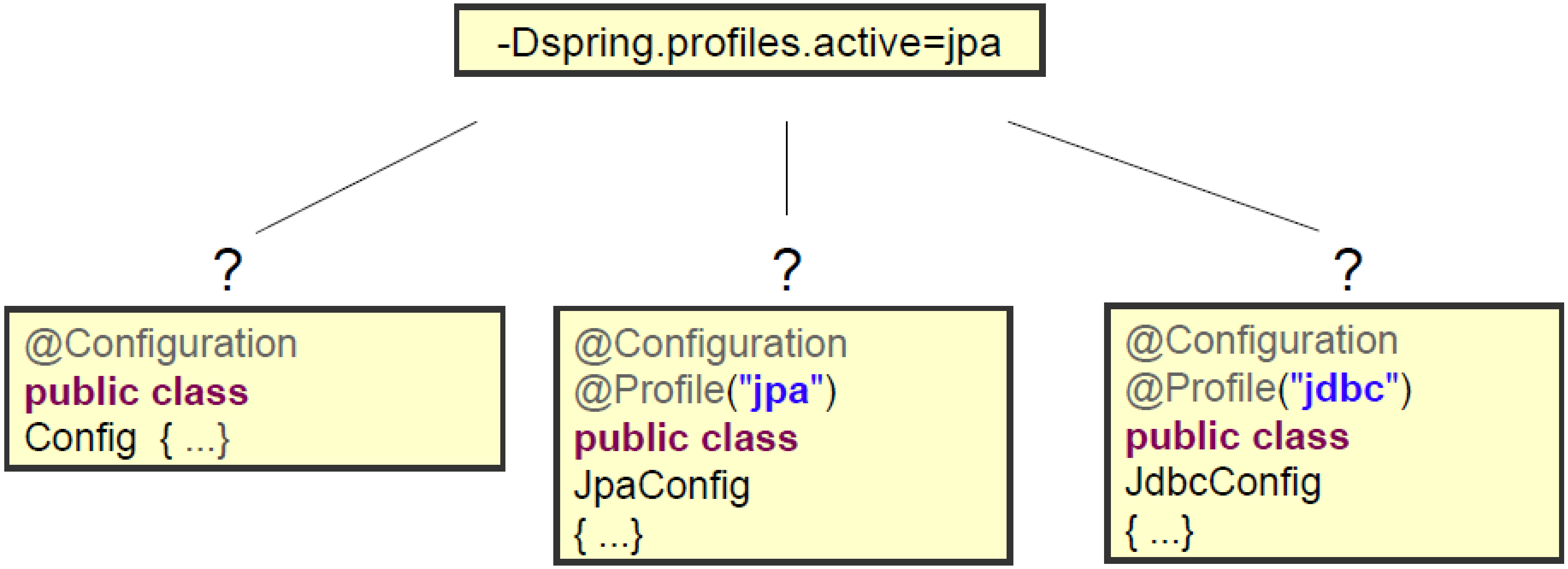
```
@Configuration
@Profile("dev")
public class DevConfig {

    @Bean
    public DataSource dataSource() {
        ...
    }
}
```

```
@Repository
@Profile("jdbc")
public class
JdbcAccountRepository
   { ...}
```

```
<beans xmlns=...>
   <!-- Available to all profiles →
   <bean id="transferService" … />
   ...
   <beans profile="jdbc">
      <bean id="dataSource" … />
   </beans>
   <beans profile="jpa"> ... </beans>
</beans>
```

## QUIZ

- Which of the following is/are selected?



```
-Dspring.profiles.active=jpa
```

```
@Configuration
public class
Config { ...}
```

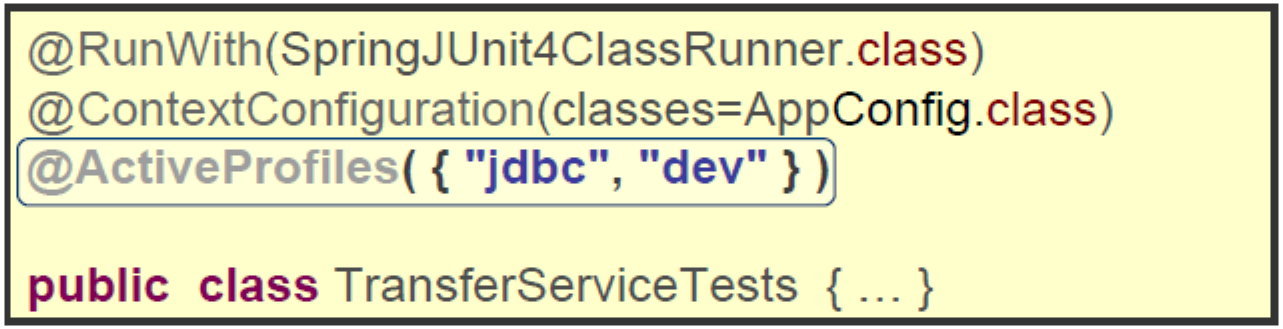
```
@Configuration
@Profile("jpa")
public class
JpaConfig
{ ...}
```

```
@Configuration
@Profile("jdbc")
public class
JdbcConfig
{ ...}
```

## Activating Profiles For a Test

- @ActiveProfiles inside Spring-driven test class
  - Define one or more profiles
  - Beans associated with that profile are instantiated
  - Also beans not associated with any profile

- Example: Two profiles activated -jdbc and dev



```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes=AppConfig.class)
@ActiveProfiles( { "jdbc", "dev" } )

public  class TransferServiceTests  { … }
```

## Active Profiles

- Profiles may be activated at execution-time
  - System property

```
-Dspring.profiles.active=dev,jpa
```

- Profiles activated via Cloud Foundry environment variable
  - CLI or manifest

```
cf  set-env <app>  spring.profiles.active  dev
```

- Java buildpack automatically activates "cloud" profile
  - You can activate additional profiles as needed

## SPRING CLOUD

## Spring Cloud

- Umbrella project for several sub-projects
  - Implement useful patterns required when building distributed, cloud-based applications
  - ***Cloud Connectors:*** access bound service information
  - ***Cloud Starters:*** for Spring Boot support
  - ***Cloud Config:*** centralized configuration management
  - ***Cloud Netflix:*** integration with Netflix OSS components
    - Eureka, Hystrix, Zuul, Achaius, ...
  - ***Cloud Bus***: distributed messaging for services instances
  - ***Spring Cloud*** for Cloud Foundry
  - Spring Cloud *for Amazon Web Services*

## Spring Cloud Connectors Summary

```
// Obtain the Cloud abstraction:
Cloud cloud = new CloudFactory().getCloud();

// Obtain a bound service (no JSON parsing of VCAP_SERVICES):
DataSource ds1, ds2;
ds1 = cloud.getSingletonServiceConnector(DataSource.class, null);

// Obtain a specific bound service by name:
ds2 = cloud.getServiceConnector("mydb", DataSource.class, null);

// Information about this instance:
ApplicationInstanceInfo info = cloud.getApplicationInstanceInfo();
logger.info("App id=" + info.getAppId()
                      + ", instance=" + info.getInstanceId());

// Cloud properties:
Properties p = cloud.getCloudProperties();
```

## Recap

Spring

IO

Profile

Boot

Cloud

Netflix

Umbrella

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

People matter, results count.

## About Capgemini

With more than 130,000 people in 44 countries, Capgemini is one of the world's foremost providers of consulting, technology and outsourcing services. The Group reported 2012 global revenues of EUR 10.3 billion.
Together with its clients, Capgemini creates and delivers business and technology solutions that fit their needs and drive the results they want. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience$^{TM}$, and draws on Rightshore$^{®}$, its worldwide delivery model.

www.capgemini.com