Spring Boot Configuration Magic

Spring Boot Starters

Starter POM

- Standard Maven POMs
- Contains transient dependencies we need
- Parent POM optional / import via <DependencyManagement>
- Available for web, batch, integration, data, security, aop, jdbc, ...

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

AutoConfiguration

@SpringBootApplication

- Creates a running ApplicationContext

```
@SpringBootApplication
public class DemoApplication {

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

- Or one without Spring Boot Banner

```
@SpringBootApplication
public class DemoApplication {

    public static void main(String[]args) {
        SpringApplication context =
            new SpringApplication(DemoApplication.class);
        context.setBannerMode(Mode.OFF);
        context.run(args);
    }}
```

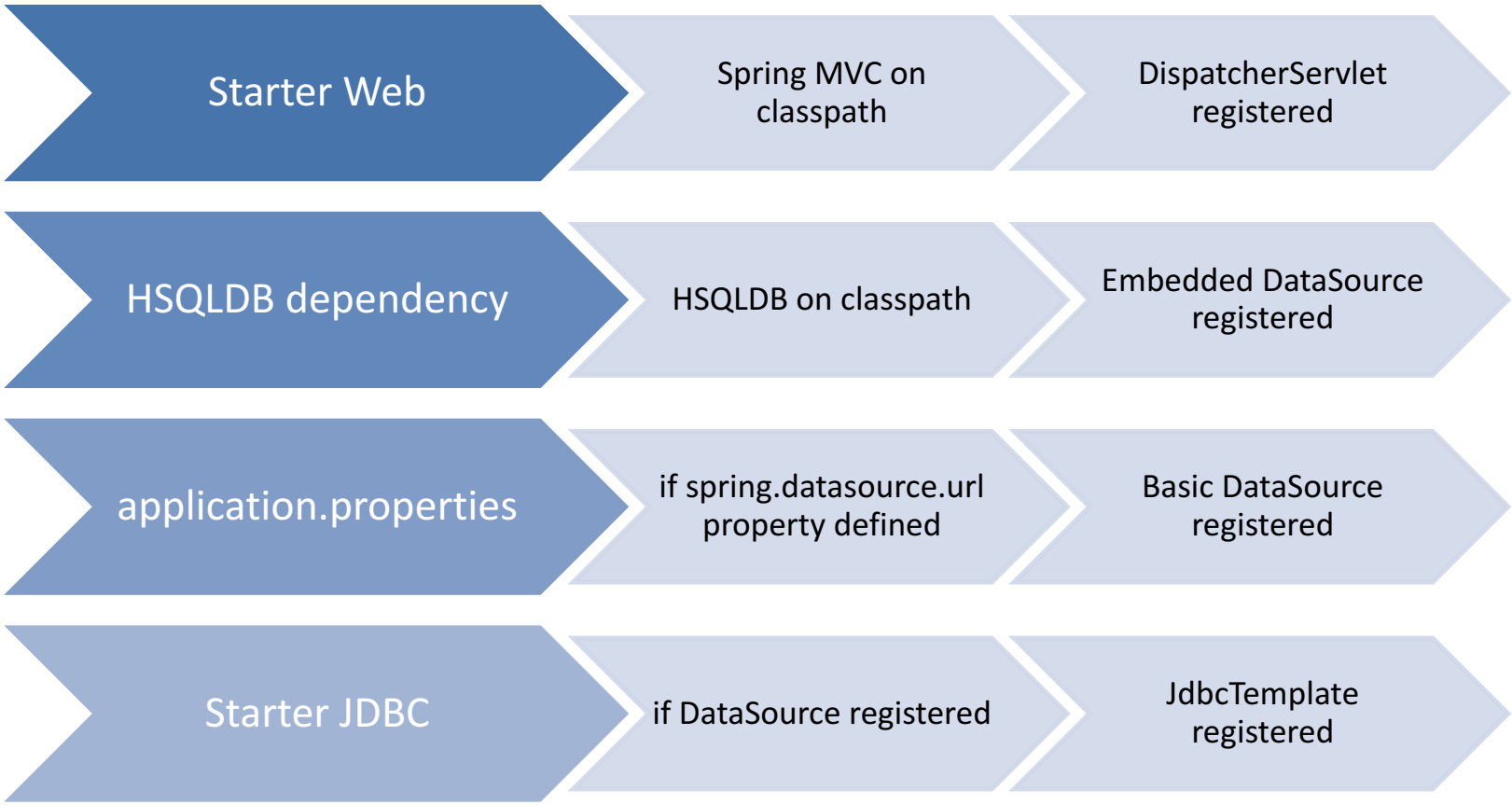
@EnableAutoConfiguration

```
@SpringBootApplication
public class DemoApplication {}
```

```
@Configuration
@ComponentScan
@EnableAutoConfiguration
public class DemoApplication {}
```

- Tries to auto-configure your application
- Does not do anything if you define your own beans
- Regular @Configuration classes
- Usually done with @ConditionalOnClass and @ConditionalOnMissingBean or @ConditionalOnProperty

AutoConfiguration



Currently Available AutoConfigured Behavior (1)

- Embedded servlet container (Tomcat, Jetty or Undertow)
- DataSource (Tomcat, Hikari, Commons DBCP)
- SQL and NoSQL stores: Spring Data
- Messaging: JMS (HornetQ, ActiveMQ), AMQP (Rabbit)
- Thymeleaf, Groovy templates, Freemarker, Mustache and Velocity
- Batch processing - Spring Integration
- Cloud connectors
- Rest repositories
- Spring Security

Currently Available AutoConfigured Behavior (2)

- Data grid: Spring Data Gemfire, Solr and Elasticsearch
- Websocket
- Web services
- Mobile & Social (Facebook, Twitter and LinkedIn)
- Reactor for events and async processing
- Jersey
- JTA
- Email, CRaSH, AOP (AspectJ)
- Actuator features (Security, Audit, Metrics, Trace)
- And many more ...

PropertiesSources

Environment and Profile

- Every ApplicationContext has an Environment
- Spring Environment available since Spring 3.1
- Abstraction for key/value pairs from multiple sources
- Used to manage @Profile switching
- @Value reads property from Environment
- Always available: System properties and OS ENV variables

Externalizing Configuration to Properties

- Put application.properties in one of the following locations:
  - A /config sub-directory of the current directory
  - The current directory
  - classpath /config package
  - The root classpath
- Properties can be overridden
  - command line arg > file > classpath
  - locations higher in the list override lower items

application.properties

```
spring.application.name = DemoApplication
server.port = 9000
```

Binding Configuration To Beans

- MyProperties.java

```
@Component
public class MyProperties {

    @Value("${private.location}")
    private Resource location;

    @Value("${private.skip:true}")
    private boolean skip;
    // ... getters and setters
}
```

- application.properties

```
private.location = classpath:mine.xml
private.skip = false
```

Binding Configuration To Beans

- @Configuration

```
@EnableConfigurationProperties
```

- MyProperties.java

```
@ConfigurationProperties(prefix="private")
public class MyProperties {
    private Resource location;
    private boolean skip = true;
    // ... getters and setters
}
```

- application.properties

```
private.location = classpath:mine.xml
private.skip = false
```

Configuration Name & Location

- spring.config.name - default application
  - can be comma-separated list

- spring.config.location - a Resource path
  - Ends with / to define a directory
  - Otherwise overrides

```
$ java -jar myApp.jar --spring.config.name=production
$ java -jar myApp.jar --spring.config.location=classpath:/cfg/
$ java -jar myApp.jar --spring.config.location=classpath:/cfg.yml
```

Disabling AutoConfiguration

Disabling specific AutoConfiguration

- Via Annotation

```
@Configuration
@EnableAutoConfiguration(excludeName="…",
    exclude={DataSourceAutoConfiguration.class})
public class AppConfiguration {
}
```

- Or via property

```
spring.autoconfigure.exclude =
….boot.autoconfigure.jdbc.DataSourceAutoConfiguration
```