Spring Boot
Auto Configuration

Auto-configuration

- Spring Boot auto-configuration attempts to automatically configure your Spring application **based on the jar dependencies that you have added.**

- For example, if HSQLDB is on your classpath, and you have not manually configured any database connection beans, then Spring Boot auto-configures an in-memory database.

- You need to opt-in to auto-configuration by adding the **@EnableAutoConfiguration** or **@SpringBootApplication** annotations to one of your @Configuration classes.

- You should only ever add one **@SpringBootApplication** or **@EnableAutoConfiguration** annotation. We generally recommend that you add one or the other to your primary @Configuration class only.

Customizing Auto-configuration

- you can start to define your own configuration to replace specific parts of the auto-configuration.
- For example, if you add your own DataSource bean, the default embedded database support backs away.
- If you need to find out what auto-configuration is currently being applied, and why, start your application with the **--debug** switch.
- ***logging.level.org.springframework=debug***
- Doing so enables debug logs for a selection of core loggers and logs a conditions report to the console.

Disabling Auto-configuration

- If you find that specific auto-configuration classes that you do not want are being applied, you can use the exclude attribute of @EnableAutoConfiguration to disable them.

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

```
@SpringBootApplication(exclude={DataSourceAutoConfiguration.class})
public class MyConfiguration {
}
```

Disabling Auto-configuration

- you can also control the list of auto-configuration classes to exclude by using the spring.autoconfigure.exclude property.

Application.properties

- spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration

Under the Covers of Spring Boot

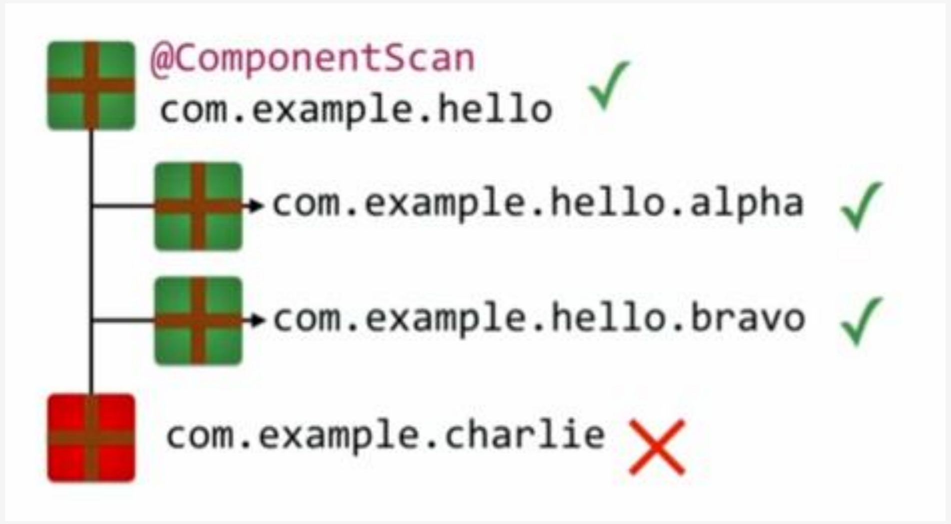
Key Annotations

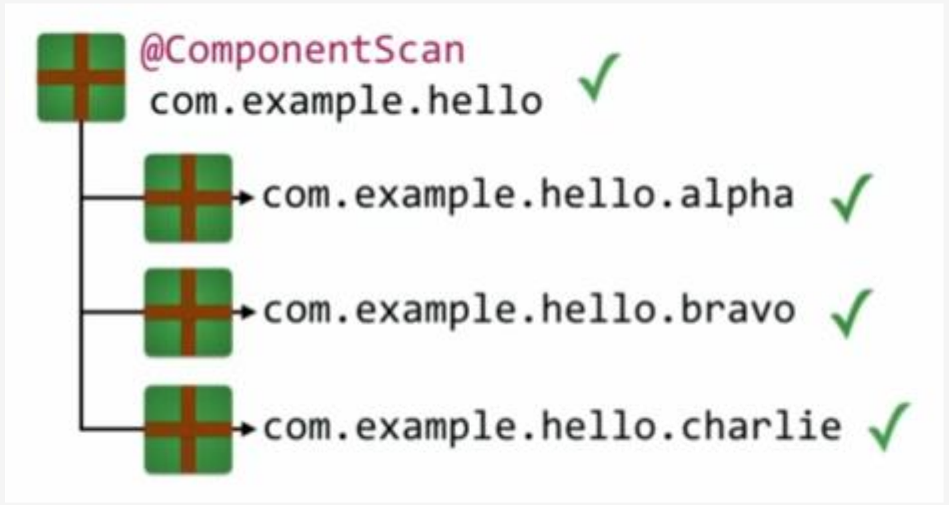
- SpringBootApplication
  - @SpringBootConfiguration
  - @ComponentScan
  - **@EnableAutoConfiguration**
    - Triggers All the Auto Configuration for spring.

Understanding Auto-configured Beans

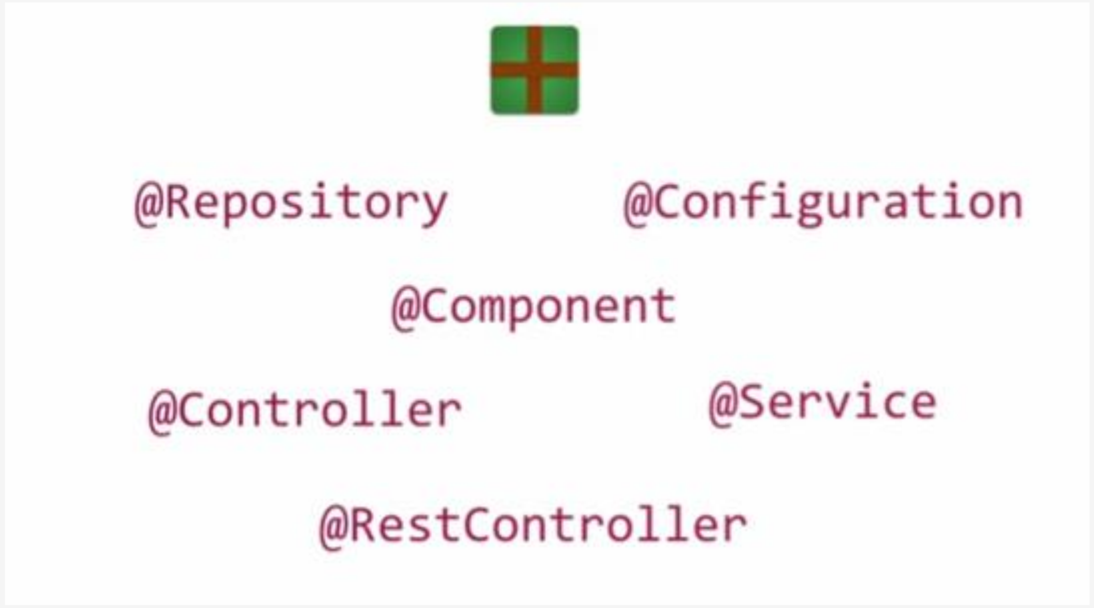
- Under the hood, auto-configuration is implemented with standard @Configuration classes.
- Additional @Conditional annotations are used to constrain when the auto-configuration should apply.
- Usually, auto-configuration classes use @ConditionalOnClass and @ConditionalOnMissingBean annotations.
- This ensures that auto-configuration applies only when relevant classes are found and when you have not declared your own @Configuration.

Component Scanning




@ComponentScan
com.example.hello ✓
com.example.hello.alpha ✓
com.example.hello.bravo ✓
com.example.hello.charlie ✓

Other Stereotypes



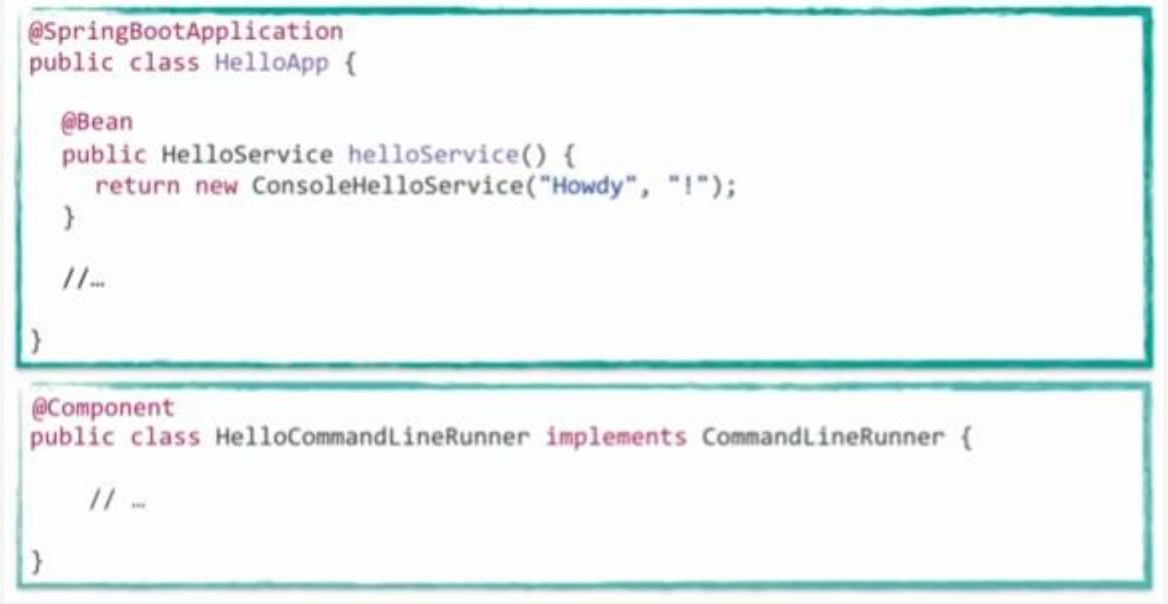
@Repository @Configuration

@Component

@Controller @Service

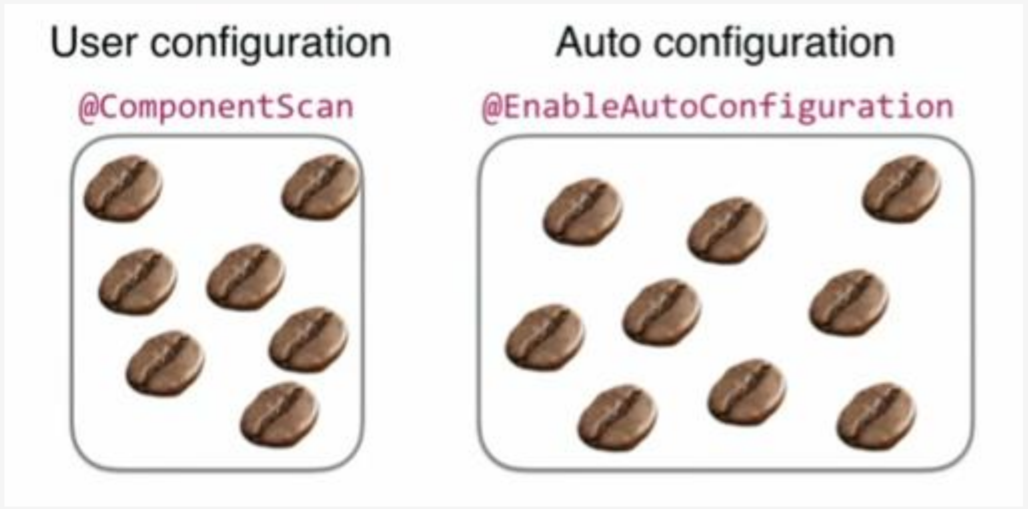
@RestController

Under the Covers of Spring Boot



```
@SpringBootApplication
public class HelloApp {

  @Bean
  public HelloService helloService() {
    return new ConsoleHelloService("Howdy", "!");
  }

  //…

}
```

```
@Component
public class HelloCommandLineRunner implements CommandLineRunner {

    // …

}
```



Locating Auto-configuration Candidates

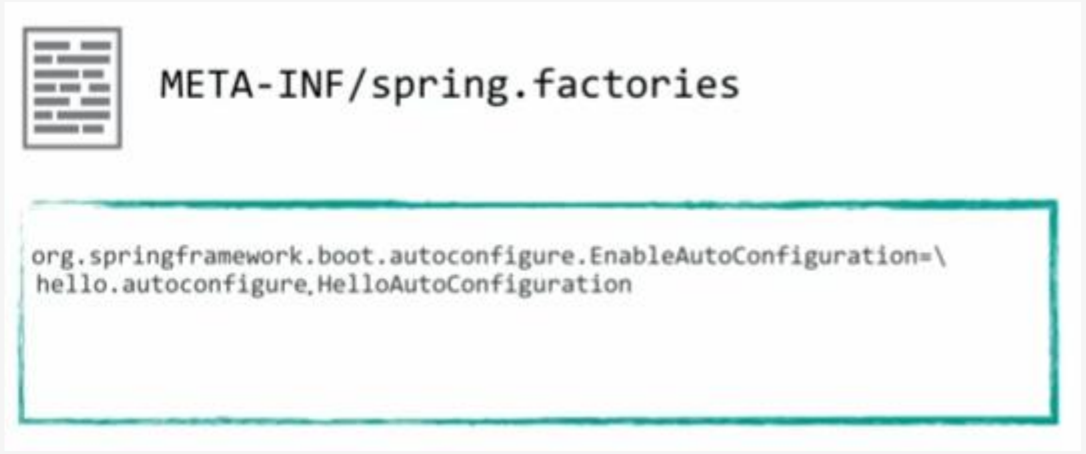
- Spring Boot checks for the presence of a META-INF/spring.factories file within your published jar. The file should list your configuration classes under the EnableAutoConfiguration key,

- You can use the @AutoConfigureAfter or @AutoConfigureBefore annotations if your configuration needs to be applied in a specific order.

- If you want to order certain auto-configurations that should not have any direct knowledge of each other, you can also use @AutoConfigureOrder.

Locating Auto-configuration Candidates

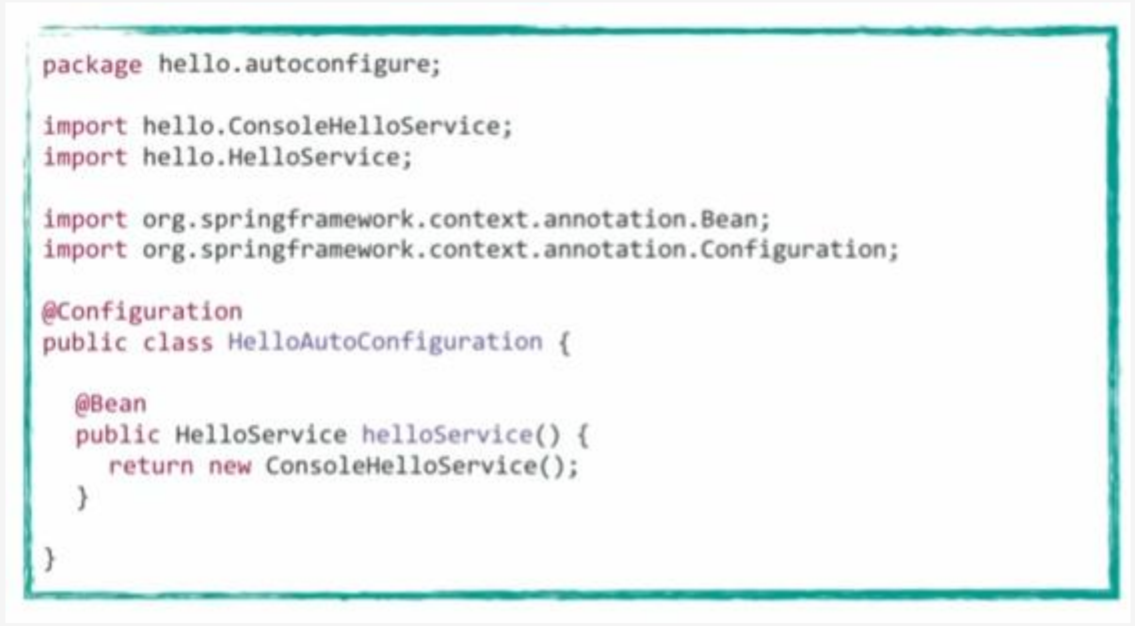
- Difference between Configuration and Autoconfiguration is their life cycle
- Register autoconfiguration in a /META-INF/spring.factories file
- org.spirngframework.boot.autoconfigure.EnableAutoConfiguration=hello.autoconfigure.HelloAutoConfiguration



META-INF/spring.factories

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
hello.autoconfigure.HelloAutoConfiguration
```

Custom Auto Configuration



```
package hello.autoconfigure;

import hello.ConsoleHelloService;
import hello.HelloService;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class HelloAutoConfiguration {

  @Bean
  public HelloService helloService() {
    return new ConsoleHelloService();
  }

}
```

Auto Configuration Reports

Configuration Properties

How to Provide customization on Auto configured object?

```
@ConfigurationProperties("hello")
public class HelloProperties{
        private String prefix = "Hello";
        private String suffix = "!";

        // getters and setters

}
```

Configuration Properties

```
@Configuration
@ConditionalOnClass(HelloService.class)
@EnableConfigurationProperties(HelloProperties.class)
public class HelloAutoConfiguration {
@Bean
@ConditionalOnMissingBean
public HelloService helloService(HelloProperties properties) {
System.out.println("creating Default Bean");
return new ConsoleHelloService(properties.getPrefix() ,
properties.getSuffix());
}
}
```

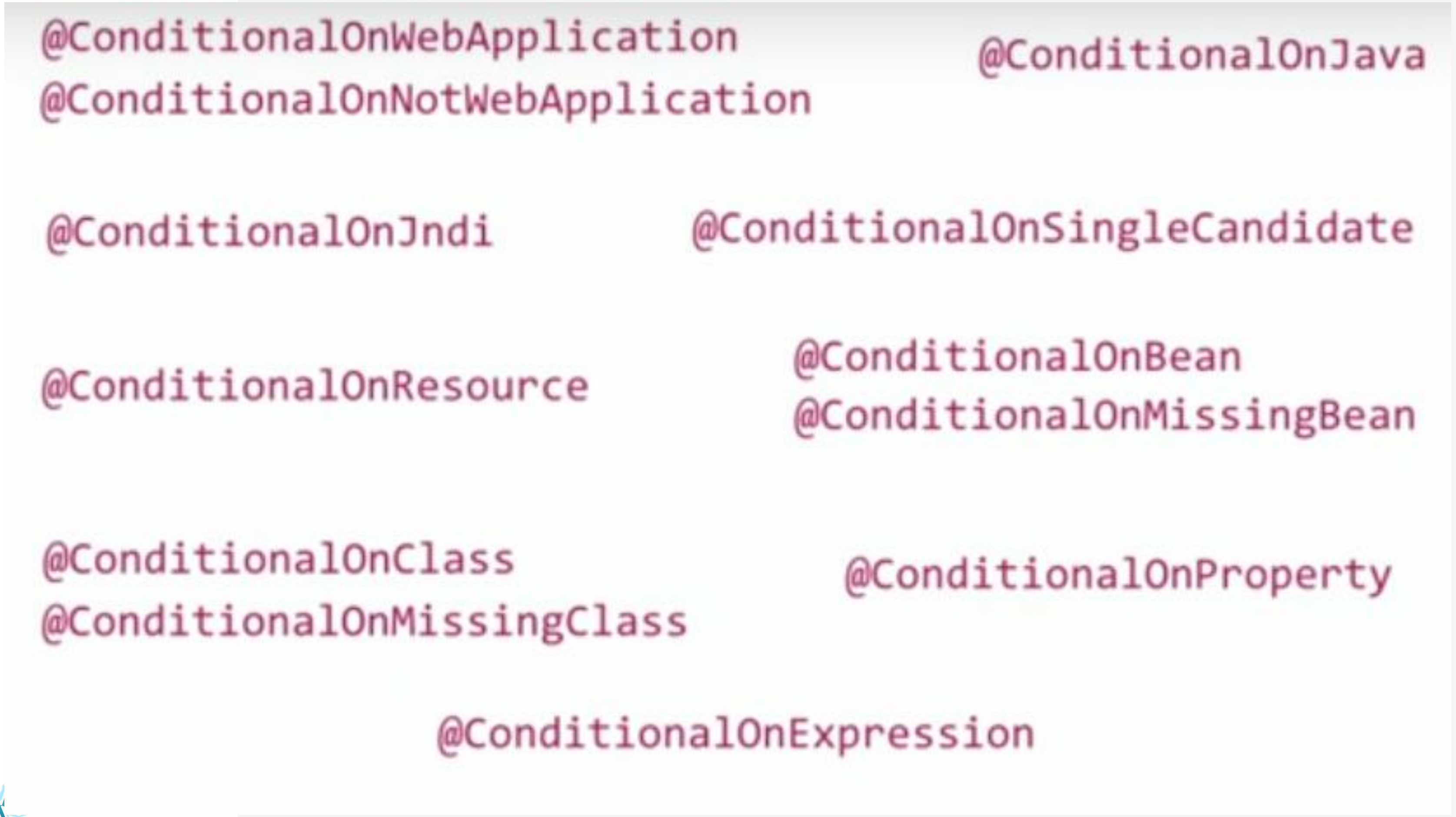
Now we can use this properties in application.properties file

Configuration Properties

- To get Auto-completion support in application.properties file
- Just include the following dependencies in pom.xml
- Note : The java doc comment used on the properties will be used as help description in application.properties file

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-configuration-
processor</artifactId>
        <optional>true</optional>
</dependency>
```

More Condition



@ConditionalOnWebApplication
@ConditionalOnNotWebApplication

@ConditionalOnJava

@ConditionalOnJndi

@ConditionalOnSingleCandidate

@ConditionalOnResource

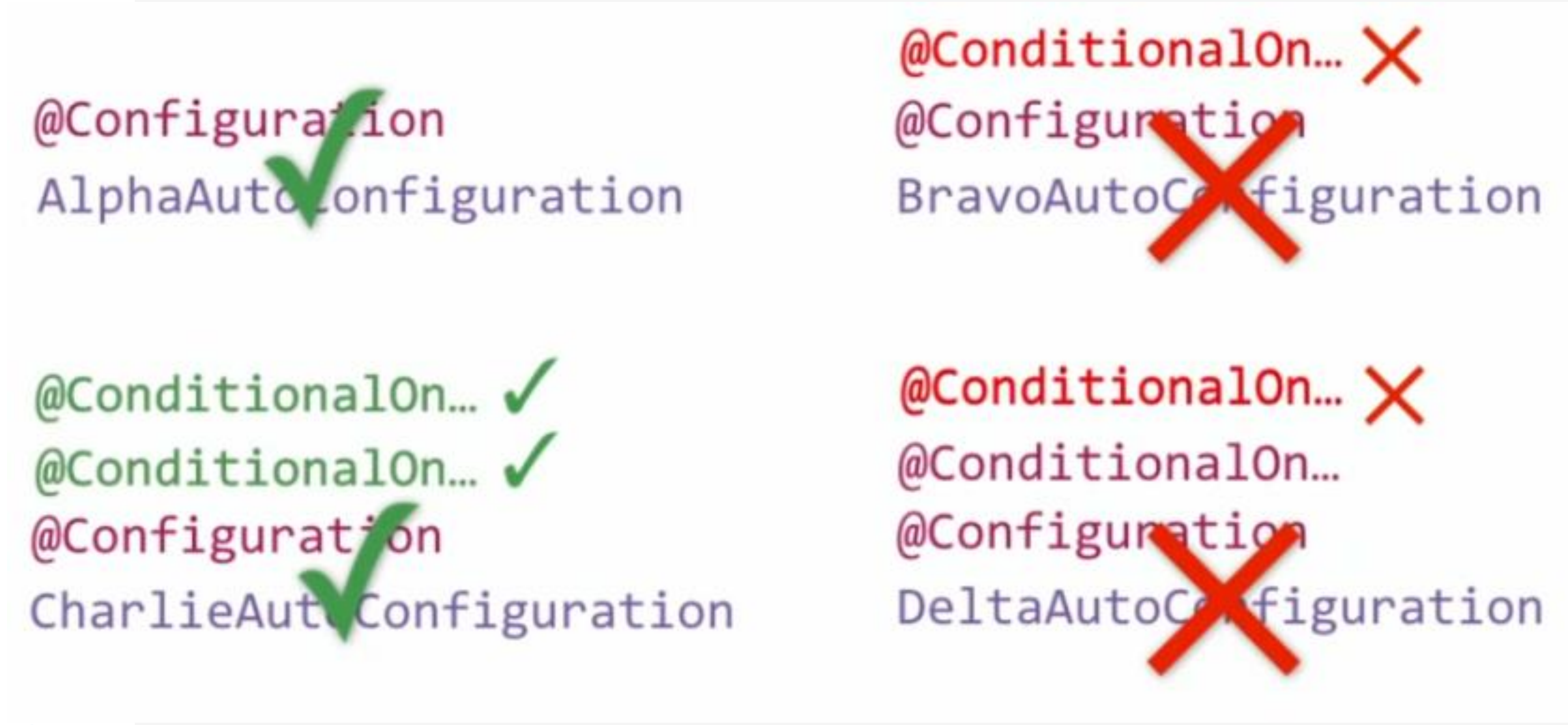
@ConditionalOnBean
@ConditionalOnMissingBean

@ConditionalOnClass
@ConditionalOnMissingClass

@ConditionalOnProperty

@ConditionalOnExpression

More Condition



Q & A

Thank You!