SpringBoot Auto Configuration

Auto Configuration is an ability of Spring Boot that automatically configures a Spring Application, analyzing the declared JAR dependencies.

Without Spring Boot

Consider a case where we want to use a spring application to query a database. We add a '*spring-jdbc*' dependency to the project and write XML configuration for creating a '*datasource*' and '*jdbcTemplate*'.

Below is an extract of an actual Spring XML configuration which creates a *datasource* and a *jdbcTemplate*.

```
<!-- Create Datasource -->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
  <property name="driverClassName" value="${app.driverClassName}"/>
  <property name="url" value="${app.url}"/>
  <property name="username" value="${app.username}"/>
  <property name="password" value="${app.password}"/>
</bean>

<!-- Load Properties -->
<context:property-placeholder location="app.properties"/>

<!-- Create JdbcTemplate -->
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
  <property name="dataSource" ref="dataSource"/>

</bean>
```

The *jdbcTemplate* bean depends on the *dataSource* bean, which depends on various external configurations. Apart from the external variables, the bean configurations remain the same in every project we use the *JdbcTemplate* instance.

What if the instances were created in Java instead? – We still have repeating code, boilerplate code.

```
public DataSource getDataSource() {
```

```
 DriverManagerDataSource dataSource =
    new DriverManagerDataSource();
 dataSource.setDriverClassName(driverClassName);
 dataSource.setUrl(url);
 dataSource.setUsername(dbUsername);
 dataSource.setPassword(dbPassword);
 return dataSource;
}

public JdbcTemplate getJdbcTemplate(){
 return new JdbcTemplate(getDataSource());
}
```

Without Spring Boot, we write too much Boilerplate code.
Without Spring Boot, we have to add the same dependencies and initialize the same components with the same configurations. Spring Boot helps us avoid a repeating job in every project.

Disabling Specific Auto-configuration

If we don't want specific auto-configure classes to be applied, we can disable them with the exclude attribute of @EnableAutoConfiguration.

The code for this looks like:

```
import org.springframework.boot.autoconfigure.*;

import org.springframework.boot.autoconfigure.jdbc.*;

import org.springframework.context.annotation.*;

@Configuration

@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})

public class MyConfiguration {

}
```

@Conditional

The @Conditional annotation indicates that a component is only registered into the application context when all the specified conditions match. If a @Configuration class is marked

with @Conditional, all of the @Bean methods, @Import annotations,
and @ComponentScan annotations associated with that class will be subject to the conditions.

Spring provides plenty of conditional annotations out-of-the-box,
including @ConditionalOnClass, @ConditionalOnMissingBean, @ConditionalOnBean, @ConditionalOnProperty, @ConditionalOnNotWebApplication, and @ConditionalOnExpression.

@ConditionalOnProperty

The @ConditionalOnProperty annotation allows to load beans conditionally depending on a certain configurational property. i.e.

```
@Bean
@ConditionalOnProperty(
      value = "emp.address.enabled",
      havingValue = "true",
      matchIfMissing = false
)
public Address getAddress(){
   return new Address();
}
```

Above bean will only be loaded if in config some where we have emp.address.enabled flag as true. matchIfMissing says, if not config found then set true by default.

@ConditionalOnExpression

```
@Bean
@ConditionalOnExpression(
      "${module.emp.enabled} and ${module.address.enabled:true}"
)
public Employee getEmployee(){
   return new Employee();
}
```

Employee bean will be loaded ifexpressions module.emp.enabled and module.address.enabled is found true. Note, module.address.enabled is set with default value as true.

@ConditionalOnBean

```
@Bean
@ConditionalOnBean(Address.class)
public Employee getEmployeeWithAddress(){
   return new Employee();
}
```

Employee bean will be build if Address bean class is there in application context. We can also put bean name instead of bean class

@ConditionalOnMissingBean

```
@Bean
@ConditionalOnMissingBean(
      NotABean.class
)
public Name getName(){
   return new Name();
}
```

Name bean will be loaded if NotABean bean class is not found in application context or classpath.

@ConditionalOnResource

```
@Bean
@ConditionalOnResource(
      resources = "/application.properties"
```

```
)
public NoResourceBean getNoResourceBean(){
   return new NoResourceBean();
}
```

The above bean will only be loaded if we have application.properties in our classpath.

@ConditionalOnClass

```
@Bean
@ConditionalOnClass(
 name = "org.springframework.boot.web.embedded.tomcat.TomcatWebServer")
public ClassPathBean getClassPathBean(){
 return new ClassPathBean();
}
```

ClassPathBean will be loaded only if class TomcatWebServer is found in classpath.

@ConditionalOnMissingClass

```
@Bean
@ConditionalOnMissingClass(
      value = "class.not.found"
)
public MissingClassBean getMissingClassBean(){
   return new MissingClassBean();
}
```

MissingClassBean will be loaded only if given class in value attribute is not found in classpath.

@ConditionalOnJndi

```
@Bean
@ConditionalOnJndi(
      value = "java:comp/env/datasource"
)
public JNDIClass getJNDIClass(){
   return new JNDIClass();
}
```

Load the bean only if certain JNDI resource is found in JNDI config.

@ConditionalOnJava

```
@Bean
@ConditionalOnJava(
      value = JavaVersion.EIGHT
)
public JavaBean getJavaBean(){
   return new JavaBean();
}
```

JavaBean will be loaded only if the running Java version in .

@ConditionalOnSingleCandidate

```
@Bean
@ConditionalOnSingleCandidate(
```

```
        value = Employee.class
    )
    public SingleCandidateBean getSingleCandidateBean(){
       return new SingleCandidateBean();
    }
```

SingleCandidateBean will be loaded only if one instance of Employee bean in found in application context.

@ConditionalOnWebApplication

```
    @Bean
    @ConditionalOnWebApplication
    public WebApplicationBean getWebApplicationBean(){
       return new WebApplicationBean();
    }
```

Bean will be loaded only if our application is running in web application container.

@ConditionalOnNotWebApplication

```
    @Bean
    @ConditionalOnNotWebApplication
    public NotWebApplication getNotWebApplication(){
       return new NotWebApplication();
    }
```