n this quick tutorial, we'll learn about the differences between [*@ComponentScan*](https://www.baeldung.com/spring-component-scanning)and *@EnableAutoConfiguration*annotations in the Spring Framework.

**2. Spring Annotations**

Annotations make it easier to configure the dependency injection in Spring. **Instead of using XML configuration files, we can use**[**Spring Bean**](https://www.baeldung.com/spring-bean-annotations)**annotations on classes and methods to define beans**. After that, the Spring IoC container configures and manages the beans.

Here's an overview of the annotations that we are going to discuss in this article:

* *@ComponentScan*scans for annotated Spring components
* *@EnableAutoConfiguration*is used to enable the auto-configuration

Let's now look into the difference between these two annotations.

**3. How They Differ**

**The main difference between these annotations is that *@ComponentScan* scans for Spring components while *@EnableAutoConfiguration* is used for auto-configuring beans present in the classpath in**[**Spring Boot**](https://www.baeldung.com/spring-boot)**applications**.

Now, let's go through them in more detail.

**3.1. *@ComponentScan***

While developing an application, we need to tell the Spring framework to look for Spring-managed components. ***@ComponentScan*enables Spring to scan for things like configurations, controllers, services, and other components we define**.

In particular, the *@ComponentScan*annotation is used with *@Configuration*annotation to specify the package for Spring to scan for components:

@Configuration

@ComponentScan

**public** **class** EmployeeApplication {

**public** **static** **void** main(String[] args) {

ApplicationContext context = SpringApplication.run(EmployeeApplication.class, args);

// ...

}

}

**Alternatively, Spring can also start scanning from the specified package, which we can define using *basePackageClasses()*or *basePackages().*If no package is specified, then it considers the package of the class declaring the *@ComponentScan*** annotation as the starting package**:**

**package** com.baeldung.annotations.componentscanautoconfigure;

// ...

@Configuration

@ComponentScan(basePackages = {"com.baeldung.annotations.componentscanautoconfigure.healthcare",

"com.baeldung.annotations.componentscanautoconfigure.employee"},

basePackageClasses = Teacher.class)

**public** **class** EmployeeApplication {

**public** **static** **void** main(String[] args) {

ApplicationContext context = SpringApplication.run(EmployeeApplication.class, args);

// ...

}

}

In the example, Spring will scan the *healthcare* and *employee*packages and the *Teacher*class for components.

Spring searches the specified packages along with all its sub-packages for classes annotated with *@Configuration.***Additionally*,*the *Configuration* classes can contain *@Bean*annotations, which register the methods as beans in the Spring application context**. After that, the @*ComponentScan*annotation can auto-detect such beans:

@Configuration

**public** **class** Hospital {

@Bean

**public** Doctor getDoctor() {

**return** **new** Doctor();

}

}

**Furthermore, the @*ComponentScan*annotation can also scan, detect, and register beans for classes annotated with *@Component, @Controller, @Service*, and *@Repository***.

For example, we can create an *Employee*class as a component which can be scanned by the @*ComponentScan*annotation:

@Component("employee")

**public** **class** Employee {

// ...

}

**3.2. *@EnableAutoConfiguration***

**The *@EnableAutoConfiguration*annotation enables Spring Boot to auto-configure the application context**. **Therefore, it automatically creates and registers beans based on both the included jar files in the classpath and the beans defined by us.**

For example, when we define the [*spring-boot-starter-web*](https://www.baeldung.com/spring-boot-starters) dependency in our classpath, Spring boot auto-configures [Tomcat](https://www.baeldung.com/tomcat) and [Spring MVC](https://www.baeldung.com/spring-mvc-tutorial). However, this auto-configuration has less precedence in case we define our own configurations.

**The package of the class declaring the *@EnableAutoConfiguration* annotation is considered as the default**. Therefore, we should always apply the *@EnableAutoConfiguration* annotation in the root package so that every sub-packages and class can be examined:

@Configuration

@EnableAutoConfiguration

**public** **class** EmployeeApplication {

**public** **static** **void** main(String[] args) {

ApplicationContext context = SpringApplication.run(EmployeeApplication.class, args);

// ...

}

}

Furthermore, the *@EnableAutoConfiguration* annotation provides two parameters to manually exclude any parameter:

We can use *exclude* to disable a list of classes that we do not want to be auto-configured:

@Configuration

@EnableAutoConfiguration(exclude={JdbcTemplateAutoConfiguration.class})

**public** **class** EmployeeApplication {

**public** **static** **void** main(String[] args) {

ApplicationContext context = SpringApplication.run(EmployeeApplication.class, args);

// ...

}

}

We can use *excludeName*to define a fully qualified list of class names that we want to exclude from the auto-configuration:

@Configuration

@EnableAutoConfiguration(excludeName = {"org.springframework.boot.autoconfigure.jdbc.JdbcTemplateAutoConfiguration"})

**public** **class** EmployeeApplication {

**public** **static** **void** main(String[] args) {

ApplicationContext context = SpringApplication.run(EmployeeApplication.class, args);

// ...

}

}

Since Spring Boot 1.2.0, we can use the ***@SpringBootApplication*annotation, which is a combination of the three annotations *@Configuration, @EnableAutoConfiguration,*and*@ComponentScan*with their default attributes**:

@SpringBootApplication

**public** **class** EmployeeApplication {

**public** **static** **void** main(String[] args) {

ApplicationContext context = SpringApplication.run(EmployeeApplication.class, args);

// ...

}

}

**4. Conclusion**

In this article, we learned about the differences between *@ComponentScan*and *@EnableAutoConfiguration* in Spring Boot.