What Is Conditional Loading?

Conditional Loading in Spring Boot means:

Spring creates or loads certain beans only if specific conditions are met.

In other words —

Spring doesn't always load everything.

It checks **certain conditions** (like class presence, property value, or existing bean) before deciding to create a bean.

Why Is It Needed?

Spring Boot has hundreds of **auto-configurations** (like for DataSource, JPA, Web, Security, etc.).

If all of them loaded at startup, your app would be:

- Slow to start \(\sqrt{\infty} \)
- Filled with unused beans 💣
- Hard to customize X

So Spring Boot loads **only the necessary configurations** — conditionally — based on what's present in your project or settings.

How It Works

Spring uses a set of annotations from the package

org.springframework.boot.autoconfigure.condition such as:

Annotation	Description
@ConditionalOnClass	Load bean only if a class is on the classpath
@ConditionalOnMissingBean	Load bean only if another bean is not already defined
@ConditionalOnProperty	Load bean only if a specific property is set in application.properties

```
    @ConditionalOnMissingClass
    @ConditionalOnExpression
    @ConditionalOnWebApplicati
    @ConditionalOnWebApplicati
    @ConditionalOnNotWebApplic
    @ConditionalOnNotWebApplic
    Load bean only if a SpEL (Spring Expression Language) evaluates to true
    @ConditionalOnWebApplicati
    @ConditionalOnNotWebApplic
    Load bean only in a web environment
    ation
```



Example 1 — Conditional Bean Based on Class

Let's say you want to load a bean only if the H2 database is present:

If H2 dependency exists in pom.xm1, this config will load.

If not, it'll be skipped automatically.



Example 2 — Conditional Bean Based on Property

```
@Configuration
public class NotificationConfig {

    @Bean
    @ConditionalOnProperty(name = "app.notifications.enabled",
havingValue = "true", matchIfMissing = false)
    public EmailNotificationService emailNotificationService() {
        return new EmailNotificationService();
    }
```

```
In your application.properties:

app.notifications.enabled=true

Bean loads.

If property is false or missing, the bean isn't created.
```

Example 3 — Conditional Bean Based on Missing Bean

This is useful to avoid bean conflicts.

```
@Configuration
public class DefaultUserServiceConfig {
    @Bean
    @ConditionalOnMissingBean(UserService.class)
    public UserService defaultUserService() {
        return new DefaultUserServiceImpl();
    }
}
```

If another configuration already defines a UserService, Spring will skip this one automatically.

★ Example 4 — How Spring Boot Uses This Internally

Spring Boot's auto-configuration files (like DataSourceAutoConfiguration) are full of such conditions.

For example:

```
@Configuration(proxyBeanMethods = false)
@ConditionalOnClass({ DataSource.class, EmbeddedDatabaseType.class
})
@ConditionalOnProperty(prefix = "spring.datasource", name = "url")
```

```
public class DataSourceAutoConfiguration {
    // Creates DataSource bean only if spring.datasource.url is set
}
```

So, if you include a database dependency and define a property like:

```
spring.datasource.url=jdbc:mysql://localhost:3306/test
```

DataSourceAutoConfiguration loads If you don't — it's skipped.

5 Benefits of Conditional Loading

Advantage Explanation

Smart Configuration Only loads what's needed

Fast Startup Skips unused auto-configurations

Customizability Lets you override default beans easily

Decoupled Design Modules don't depend on each other

unnecessarily

6 Real-Life Example (Your Project)

In your **Digital Wallet Project**, suppose you want:

• To use **RazorpayService** only if the property wallet.razorpay.enabled=true is set.

```
@Configuration
public class PaymentConfig {

    @Bean
    @ConditionalOnProperty(name = "wallet.razorpay.enabled",
havingValue = "true")
    public RazorpayService razorpayService() {
        return new RazorpayService();
    }
}
```

}

Then in your config file:

wallet.razorpay.enabled=true

RazorpayService bean gets loaded.

If you set it to false, Spring skips it entirely.

In Summary

Concept Meaning

Conditional Loading Loading beans/configs only when certain conditions are true

Implemented via @Conditional... annotations

Used in Auto-configuration, custom configs, feature toggles

Benefit Reduces memory, increases flexibility, improves startup

time

Would you like me to show you a **diagram flow (visual)** of how Spring Boot checks all these conditions during startup before deciding which beans to load?

It's super helpful for understanding auto-configuration flow in depth.

You said:

yes

ChatGPT said:

Perfect, Shubham +

Let's go step-by-step through a **visual + explanation** of how **Spring Boot conditional loading works during startup** —

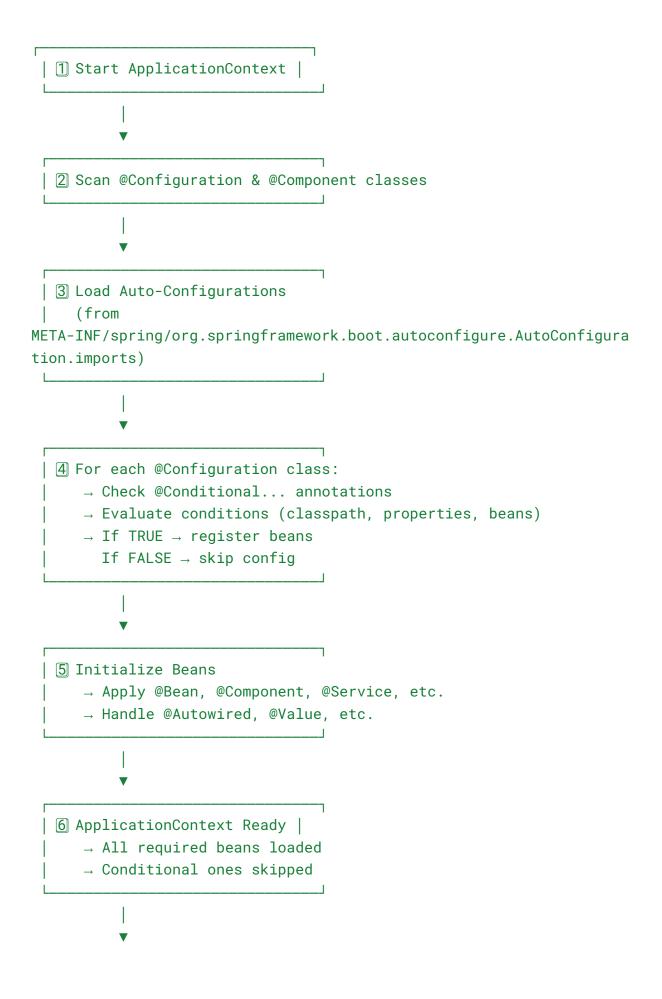
this is the hidden "auto-configuration brain" of Spring Boot in



High-Level Flow: When You Run a Spring Boot App

You run: ./gradlew bootRun OR java -jar yourapp.jar

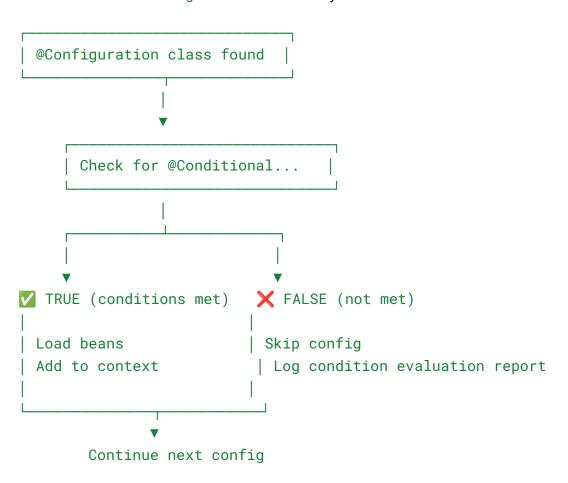
Spring Boot performs the following key steps:





🧩 Zoom Into Step 4 — Conditional Evaluation Flow

Here's what happens internally for every configuration file like DataSourceAutoConfiguration.class or your own:



Example: What Happens When You Have a DataSource Property

Let's say your app has:

spring.datasource.url=jdbc:mysql://localhost:3306/walletdb

Spring Boot goes through this internal process 👇

[AutoConfigurationClass] --> DataSourceAutoConfiguration

Check: @ConditionalOnClass(DataSource.class) ✓ (found)
Check: @ConditionalOnProperty(prefix="spring.datasource",
name="url") ✓ (found)

→ Conditions satisfied → Load DataSource beans

But if that property was **missing**, it would skip:

Skipped DataSourceAutoConfiguration

Behind the Scenes: AutoConfigurationReport

Spring Boot actually keeps a detailed record of why something was loaded or skipped.

If you enable debug logs:

debug=true

Spring prints this in your console:

CONDITIONS EVALUATION REPORT

----Positive matches:

DataSourceAutoConfiguration matched:

- @ConditionalOnClass found required classes
- @ConditionalOnProperty (spring.datasource.url) matched

Negative matches:

SecurityAutoConfiguration did not match:

- @ConditionalOnClass classes not found

This helps developers understand which beans are being conditionally loaded.

So, in Summary

Step	Description
	Spring creates an ApplicationContext
Scan & Evaluate	It scans configs, then evaluates all @Conditional annotations
Decision	Loads only those beans whose conditions match (classpath, property, etc.)
Skip Others	Beans whose conditions fail are skipped
	Lean, efficient, environment-specific configuration

PReal Analogy

Think of Spring Boot's conditional loading like **Google Maps** choosing a route:

It doesn't activate all possible routes — it picks **only the one** that fits your conditions (traffic, distance, etc.).

Similarly, Spring Boot chooses only the configurations that fit **your runtime environment**.