Spring Boot + Servlet Container + Servlet

Embedded Server

Spring Boot by defaults includes an embedded web server (Tomcat). Without the embedded web server, the traditional way deploying to web servers would be to have Apache Tomcat be installed on a different server. You would package your application into a `.war` file, then move that `.war` file into the Tomcat's `webapp/` directory in which then it will be able to serve your application.

As you can see there is decoupling of the application code and the server code. Spring Boot including the embedded server allows your to run both the application code and the webserver code altogether, allowing you to run your application with just `java -jar app.jar` . NEAT!

Tomcat

Tomcat is the default embedded server included by Spring Boot, it listens on a port (8080 by default), and manages the thread pool for handling the request that reaches the server (by default 200 threads), it parses the raw TCP bytes into high level HTTP request/response objects, manages the connection for you, and runs the servlet lifecycle.

The embedded server used by Spring Boot can be swapped, Tomcat is the default one but there are other alternatives:

- Jetty - Lighter, good for async and websockets
- Undertow - High-performance and non-blocking

Jakarta?

Okay, how does Jakarta ties into this?

Jakarta is the rename of Java EE, it is a specification, and actually a bunch of specification. It tells you how to handle certain things for example:

1. `jakarta.servlet`: Tells you how to handle HTTP. **Tomcat/Jetty/Undertow**

2. `jakarta.ws.rs`: Tells you how to handle JAX-RS REST (Rest API) **Jersey/RESTEasy**
3. `jakarta.persistent`: JPA, tells you how to handle ORM **Hibernate**

And a bunch more, and because it is only a specification, you will need an implementation of the specification in order for it to be useful! You will see Jakarta annotation being used, but underneath it is power by an actual implementation for it to actually work.

Servlet Container and Servlet

A Servlet is just a Java class that handles HTTP request and produces HTTP responses back. It is the lowest-level bridge between HTTP server and your application code.

```
public classs MyServlet extends HttpServlet {
  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res) {
    res.getWriter().write("hi");
  }
}
```

Now what the hell is a Servlet container? Well it manages and run your servlet! In our case, Tomcat would be both the webserver and the servlet container, it handles both of the roles! It accepts the request from client, it figures out which registered servlet to run, and then route the request to that servlet, where the servlet then routes the request to the actual framework.

How to Register Servlet in Spring Boot?

Servlet are registered by the framework, if you're using Spring MVC then it will register a servlet named **DispatcherServlet,** if you're using Jersey then it will register a servlet named **ServletContainer**.

Jersey vs Spring MVC

Jersey is a JAX-RS Java REST implementation. As we have discussed before, Jakarta specification without implementation is useless, Jersey provides the actual implementation on how to handle REST API implementation. It provides `@Path`, `@POST`... implementation.

Spring MVC is NOT a JAX-RS implementation, but it is Spring Boot's framework and an opinionated approach on how to implement REST API through `@GetMapping, @PostMapping`... annotation.

Spring MVC is the default framework used for implementing REST API with Spring Boot, but Spring Boot also supports Jersey by you including `spring-boot-starter-jersey` dependency! That's how I'm able to use it with Chassis!

How does Request / Response work?

Ever wonder how the JSON that you send to the Spring Boot server gets deserialized into POJO? And how the POJO that you returned as response gets serialized back as JSON?

Well the magic lies within the Servlet! Jersey OR Spring MVC!

Let's talk about Spring MVC first. With Spring MVC the interface that handles the Serialization and deserialization would be HttpMessageConverter, and the Jackson (JSON library) implementation would be the class **MappingJackson2HttpMessageConverter**. This class is responsible for handling the serialization and deserialization of your HTTP request / response when you get a HTTP request and return a POJO. This is provided by the `spring-boot-starter-web` library.

On the other hand with Jersey it is done by the interface called **MessageBodyWriter / MessageBodyReader**. The Jackson implementation of the class is called **JacksonJaxbJsonProivder** and it is provided through the `jersey-media-json-jackson` library.

If you don't use `jersey-media-json-jackson` Jersey actually has it's own JSON provider called MOXy. It is it's own implementation. It is XML first, and JSON bolted on library. It serialize / deserialize through the `@XmlRootElement` annotations, while with Jackson it uses `@JsonProperty` annotations.

Web Architecture

Your application Code

Spring MVC | Jersey

Jakart Servlet API

Tomcat / Jetty / Undertow

TCP / HTTP

Revision #1
Created 2026-05-09 03:45:51 UTC by Tamarine

Updated 2026-05-09 04:16:20 UTC by Tamarine