APACHE TOMCAT SERVER

SPRING BOOT

THAPAROHAN2019@GMAIL.COM | ROHAN THAPA

What is Apache Tomcat?

Apache Tomcat is an **open-source Java Servlet Container** developed by the **Apache Software Foundation (ASF).** It implements the **Java Servlet, JavaServer Pages (JSP),** and **WebSocket technologies**, providing an environment where Java-based web applications can run.

Tomcat is often referred to as both a **web server** and a **servlet container**. It is designed to execute Java servlets and render web pages that use JSP. While Tomcat can handle static pages like **HTML, CSS, or JavaScript**, its primary use case is serving dynamic Java applications.

Tomcat as a Web Server vs. Servlet Container

- **Web Server:** Tomcat handles **HTTP** requests and serves responses (much like other web servers such as Apache **HTTPD or NGINX**). However, unlike traditional web servers, which serve static content (HTML, images, etc.), Tomcat primarily serves dynamic content by interpreting Java servlets or JSP files.
- **Servlet Container (Web Container):** Tomcat provides the environment for executing Java servlets, JSP pages, and other web technologies. It manages the lifecycle of these components (initiation, request processing, destruction).

Tomcat Architecture

Tomcat is built on a **layered and modular architecture**. Here's a breakdown of each critical component:

1. Catalina - The Servlet Container

Catalina is the core component of Tomcat, responsible for managing the **lifecycle of servlets**, mapping requests to servlets, and managing session tracking. It adheres to the Java Servlet and **JavaServer Pages (JSP**) specifications, ensuring proper handling of dynamic content.

Key Features of Catalina:

- Manages the execution of servlets.
- Creates and destroys servlet objects.
- Provides HTTP request/response handling.
- Deals with session management and concurrency.

Tomcat Architecture

2. Coyote - The HTTP Connector

Coyote is the component that handles the **HTTP protocol in Tomcat**. It listens to the incoming HTTP requests on a port (usually port **8080** by default) and forwards them to the **Catalina servlet container**. Coyote is essential for making Tomcat function as a web server.

- HTTP/1.1 Connector: Implements HTTP/1.1 protocol to allow Tomcat to act as a web server.
- AJP Connector: Communicates with another web server like Apache HTTPD to balance load or serve static files.

Tomcat Architecture

3. Jasper - JSP Engine

Jasper is Tomcat's engine responsible for **compiling JSP files into Servlets**. The compiled servlets are then executed by Catalina. JSP pages are written in HTML with embedded Java code; when a request for a JSP page is made, Jasper compiles the JSP into a servlet, which Catalina then processes.

Tomcat Architecture

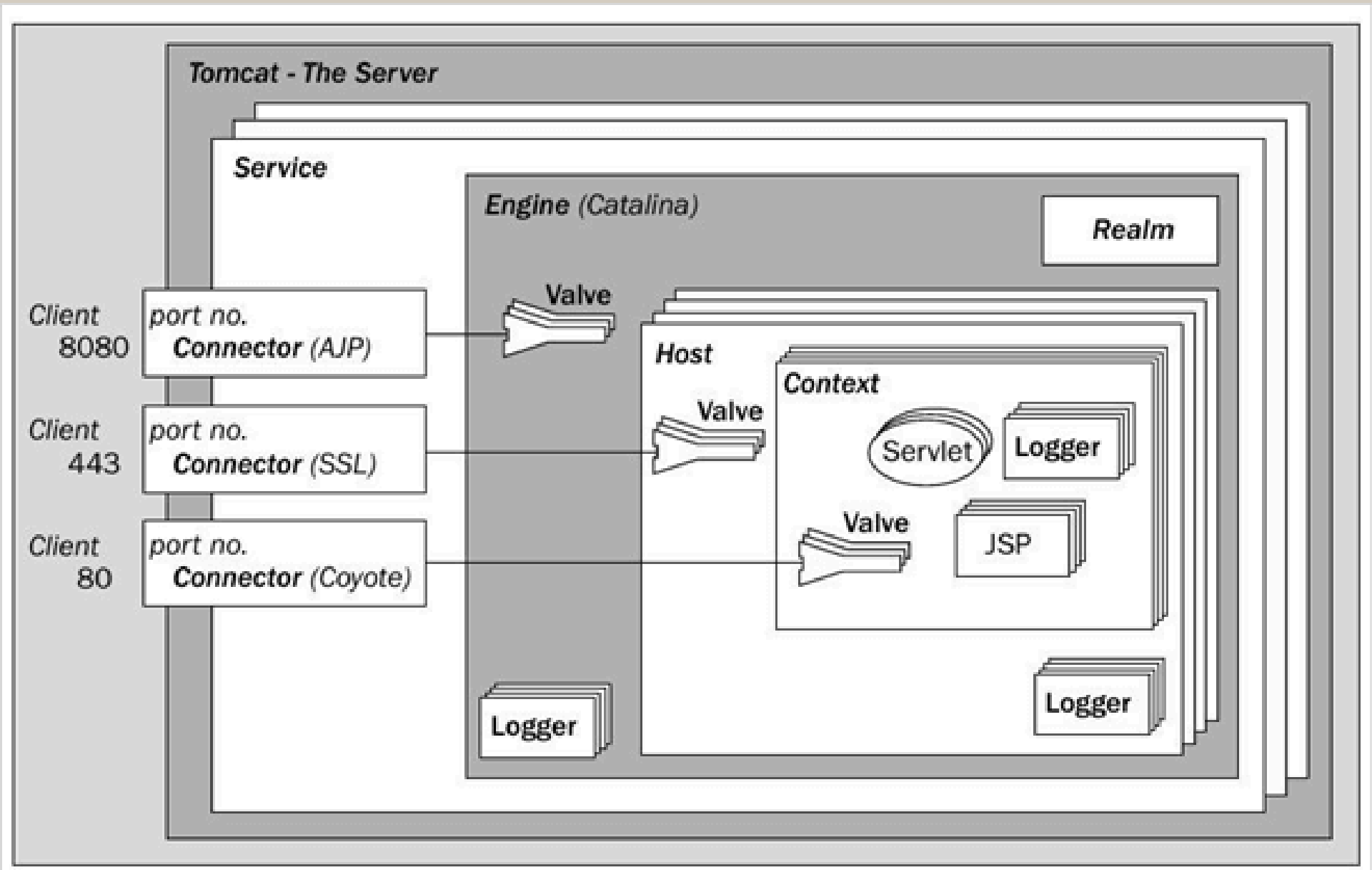
4. Realm - Authentication and Authorization

A Realm in Tomcat represents a **"database"** of **usernames, passwords, and roles**. When a user attempts to access a protected resource, Tomcat can consult the **Realm to authenticate** the user and determine if they have the necessary permissions.

5. Valves and Filters

Valves are similar to Servlet Filters but are tied to the Tomcat server itself and can be configured at a broader level (engine, host, or context). They are often used for logging, request filtering, or access control at the Tomcat level.

Tomcat Architecture



Tomcat's architecture.

Tomcat's Lifecycle and Request Flow

1. Initialization:
When Tomcat starts, it initializes its key components, including the connectors (e.g., **Coyote for HTTP**), **servlet containers (Catalina), and web applications**. It also loads any necessary configuration, such as **web.xml**, which defines how servlets and JSP pages are mapped to URLs.

Tomcat's Lifecycle and Request Flow

2. Request Handling:
Once initialized, Tomcat **begins listening for incoming requests**. For an HTTP request:

- **Coyote** receives the HTTP request and parses it.
- **Catalina** receives the request from Coyote and looks at the URL mapping (defined in **web.xml** or annotations) to route it to the appropriate servlet.
- If the request targets a **JSP file, Jasper compiles** the JSP into a servlet (if not already compiled) and sends it to Catalina for execution.

Tomcat's Lifecycle and Request Flow

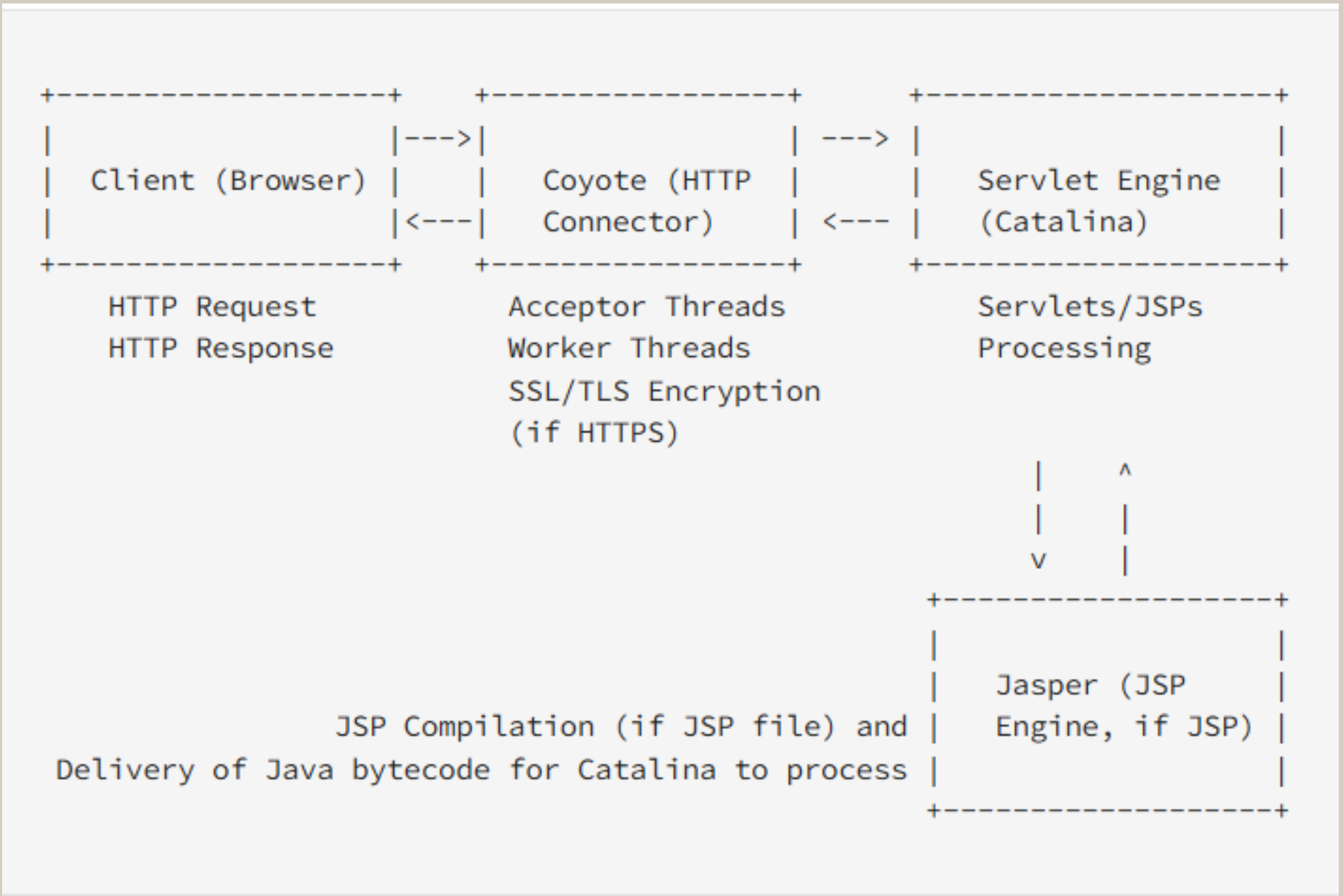
3. Session Management:
Tomcat manages user sessions through cookies or URL rewriting, allowing the server to maintain state between requests.

4. Request Processing:
Once the request reaches the servlet, the servlet processes it (e.g., fetching data from a database, performing business logic) and generates a response, usually HTML or JSON, which is then sent back to Coyote to deliver to the client.

Tomcat's Lifecycle and Request Flow

5. Shutdown:
When Tomcat is shut down, it stops accepting new requests, processes any pending requests, and then shuts down the connectors (e.g., Coyote) and web applications, releasing resources.

Tomcat's Lifecycle and Request Flow



Tomcat Configuration Files

Tomcat has several key configuration files that control its behavior:

1. **server.xml:** Defines global server settings, including **connectors, ports** (e.g., 8080 for HTTP, 8443 for HTTPS), and thread pools.
2. **web.xml:** The global deployment descriptor for all web applications, defining default servlets and error handling.
3. **context.xml:** Defines application-specific configurations like resource lookups, security, and database connections.
4. **logging.properties:** Configures logging settings for Tomcat

How Tomcat Handles Java Servlets

- **Servlet Loading:** When Tomcat starts, it reads the deployment descriptor **(web.xml)** or servlet annotations to determine which servlets to load.

- **Instantiation:** Tomcat creates an **instance of the servlet** class using reflection when the servlet is loaded for the first time or upon startup, if specified.

- **Initialization:** After instantiating the servlet, Tomcat calls the **init()** method of the servlet, which is where any servlet-specific initialization logic should be placed.

How Tomcat Handles Java Servlets

- **Request Handling:** For each incoming HTTP request, Tomcat invokes the servlet's **service()** method, which then calls the appropriate **doGet(), doPost(),** etc., based on the HTTP method of the request.

- **Destruction:** When the servlet is no longer needed (e.g., when Tomcat is shutting down), the **destroy()** method is called, giving the servlet a chance to clean up resources.

Advanced Features of Tomcat

1. Clustering:
Tomcat supports clustering, which allows it to distribute load across multiple instances, improving scalability and fault tolerance. Sessions can be replicated across the cluster to provide session failover in case of instance failure.

2. SSL Configuration:
Tomcat can be configured to handle HTTPS requests by setting up SSL/TLS certificates in the server.xml configuration. This provides encrypted communication between clients and the server.

Advanced Features of Tomcat

3. JNDI (Java Naming and Directory Interface):

Tomcat provides support for **JNDI lookups**, commonly used for resource lookups like **database connections**. JNDI allows you to define global resources in the **context.xml** file, such as database pools or mail sessions, which can be shared across applications.

Advanced Features of Tomcat

4. Security Features:

Tomcat provides several layers of security, including:

- **Realms** for managing user authentication and authorization.
- **SSL** support for encrypted communication.
- Security constraints defined in **web.xml** to protect resources.

5. Hot Deployment:

Tomcat supports hot deployment, which allows you to **redeploy or update web applications without restarting the entire server**. This feature is extremely useful for reducing downtime in production .

Performance Optimization

Tomcat performance can be optimized by:

- **Thread Pooling:** Adjusting the number of worker threads in the connector configuration (e.g., **maxThreads, minSpareThreads**).
- **Connection Timeout:** Setting appropriate connectionTimeout values to close idle connections.
- **Memory Tuning:** Configuring the **Java Virtual Machine (JVM)** parameters (e.g., **heap size, garbage collection**) to match the expected load.
- **Compression:** Enabling GZIP compression for responses to reduce the size of data sent over the network.

Tomcat in Modern Applications

- **Spring Boot: Tomcat** is the **default embedded servlet container** in Spring Boot applications. When you create a Spring Boot application, Tomcat is **automatically included** and runs as part of your application. It allows easy development and deployment of Java-based web applications.

- **Docker: Tomcat can be containerized using Docker**. Running Tomcat in a Docker container provides the benefits of containerization, such as portability, isolation, and scalability.

Tomcat in Modern Applications

- **Cloud Deployment:** Many cloud providers, like **AWS, Google Cloud,** and **Azure,** support deploying **Tomcat-based applications**. This simplifies infrastructure management, allowing developers to focus more on code.

Conclusion

Apache Tomcat is a **robust, lightweight,** and widely used **web server** and **servlet container** for **Java-based web applications**.

Its modular design, ease of configuration, and extensive feature set make it an ideal choice for developers working with Java Servlets, JSP, or frameworks like Spring Boot.

Understanding Tomcat's internal components like **Catalina, Coyote, Jasper,** and its **configuration files like server.xml**, allows you to efficiently deploy and manage Java web applications in production environments.

Thank You

Rohan Thapa

thaparohan2019@gmail.com