JAR vs WAR

In Java packaging, **JAR** (Java ARchive) and **WAR** (Web ARchive) are both archive file formats based on the ZIP structure, but they serve distinct purposes and have different characteristics. Below is a detailed explanation of their differences:

---

1. Purpose

- **JAR**: A JAR file is a general-purpose archive used to package Java classes, libraries, or standalone applications. It's commonly employed to distribute reusable libraries (e.g., utility classes) or executable Java programs that can run on the Java Virtual Machine (JVM).
- **WAR**: A WAR file is specifically designed for web applications. It packages all components needed for a web app, such as servlets, JSPs (JavaServer Pages), HTML, CSS, JavaScript, and configuration files, into a single unit for deployment on a web server or application server (e.g., Apache Tomcat, JBoss).

---

2. Contents

- **JAR**: Contains Java class files (`.class`), a `MANIFEST.MF` file (under the `META-INF` directory) that provides metadata (e.g., the main class for executable JARs), and optional resources like configuration files, images, or property files.
- **WAR**: Includes web-specific components with a defined structure:
  - **WEB-INF/**: A mandatory directory containing:
    - `web.xml` (deployment descriptor for configuring servlets, mappings, etc.),
    - `classes/` (compiled Java classes),
    - `lib/` (JAR files used as dependencies by the web app).
  - Static resources (e.g., HTML, CSS, JavaScript) typically reside in the root directory or subdirectories outside `WEB-INF`, though JSPs may be placed inside `WEB-INF` to restrict direct access.

---

3. Structure

- **JAR**: Has a flat structure, primarily consisting of class files and resources, with the manifest file specifying metadata. Example:

```
myapp.jar
  META-INF/
      MANIFEST.MF
  com/
      example/
          MyClass.class
  resources/
       config.properties
```

- **WAR**: Follows a hierarchical structure tailored for web applications. Example:

```
mywebapp.war
  index.html
  css/
      style.css
  WEB-INF/
      web.xml
      classes/
          com/
              example/
                  MyServlet.class
      lib/
          dependency.jar
```

---

4. Deployment and Usage

- **JAR**:
  - Included in an application's classpath to provide libraries or reusable code.
  - If executable (with a `Main-Class` specified in the manifest), it can be run directly using `java -jar myapp.jar`.
- **WAR**:
  - Deployed to a servlet container or application server (e.g., Tomcat), which unpacks it and makes the web application accessible at a specific URL (context path).
  - Not executable standalone; it relies on the server to manage its lifecycle.

---

5. Dependency Handling

- **JAR**: Does not inherently contain other JARs; dependencies are managed externally (e.g., via the classpath or build tools like Maven).
- **WAR**: Can bundle dependent JARs inside the `WEB-INF/lib/` directory, ensuring the web application uses specific versions of libraries, isolated from other applications on the same server.

---

6. Creation

Both JAR and WAR files can be created using the `jar` command-line tool from the JDK or modern build tools like Maven or Gradle: - **JAR Example**: `jar cvf myapp.jar -C classes .` - **WAR Example**: `jar cvf mywebapp.war .` (assuming the directory is structured with `WEB-INF`, etc.). - In **Maven**, the packaging type is specified in the `pom.xml`: - `<packaging>jar</packaging>` for JAR. - `<packaging>war</packaging>` for WAR.

Summary Table

Feature	JAR	WAR
Full Name	Java ARchive	Web ARchive
Purpose	Libraries, standalone apps	Web applications
Contents	Classes, manifest, resources	Servlets, JSPs, HTML, WEB-INF, etc.
Structure	Flat	Hierarchical (WEB-INF, etc.)
Deployment	Classpath or `java -jar`	Web server (e.g., Tomcat)
Dependencies	External	Bundled in `WEB-INF/lib/`

Key Takeaway

In essence, a **JAR** is a versatile archive for general Java code, while a **WAR** is a specialized archive for web applications, complete with a structure and components tailored for deployment on a web server. Understanding their distinct roles ensures proper use in Java development and deployment scenarios.