Web Applications Developments

ITWT413

LECTURE 2: INTRODUCTION TO JAKARTA EE 10

DR. HALA SHAARI

Course Structure

- Lecture (Sunday& Tuesday)
- Time: 12:30-14:00
- Microsoft Teams Code (goi2d67)

(Lectures, labs, announcements, References):

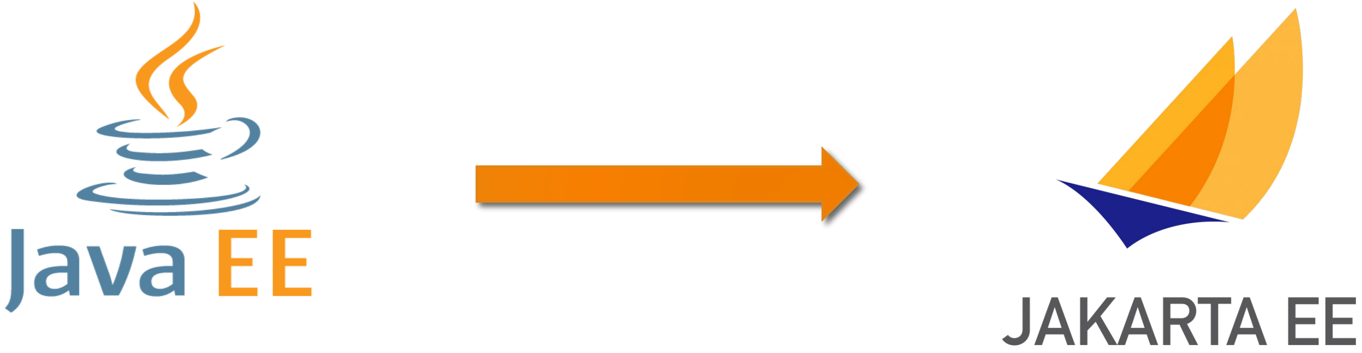
- Grading :
  - 25% Midterm exam.
  - 25% Assignments
  - 25% Group Project (Groups of two).
  - 25% Final Exam.

Lecture Agenda

- Jakarta EE
- The Importance of Jakarta EE
- Jakarta EE Evolution
- What is Jakarta EE
- Jakarta EE Services
- Jakarta EE Containers and Servers
- Jakarta EE Components
- Usage of Java Standard Edition (SE) APIs
- Java EE vs. Spring Boot: Comparing Two Java Framework Giants.

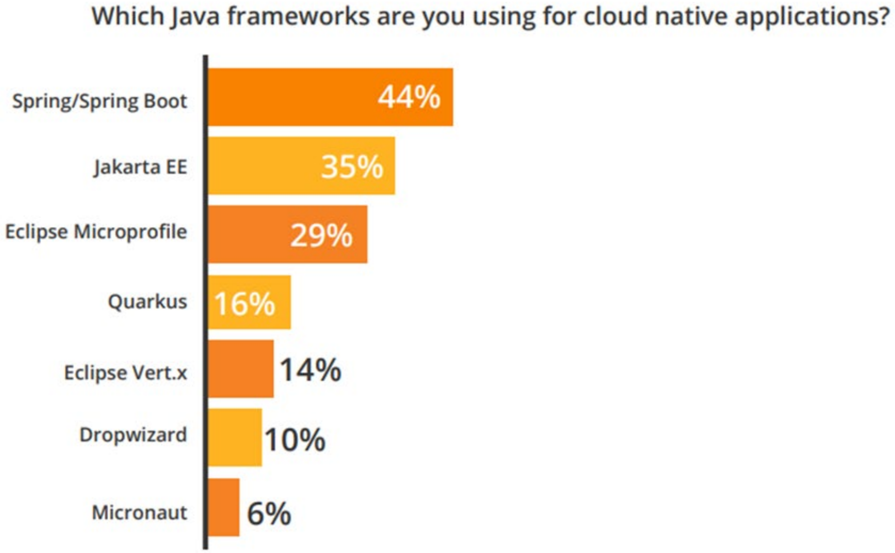
Jakarta EE

- Java EE transitioned from JCP to Eclipse Foundation as Jakarta EE
- Open governance, open source, open compatibility testing
- Well defined specification process, clear IP flow, vendor-neutral open collaboration, level playing field
- Key stakeholders maintained if not expanded including Oracle, IBM, Red Hat, Payara and VMware
- Community participation and contribution key.

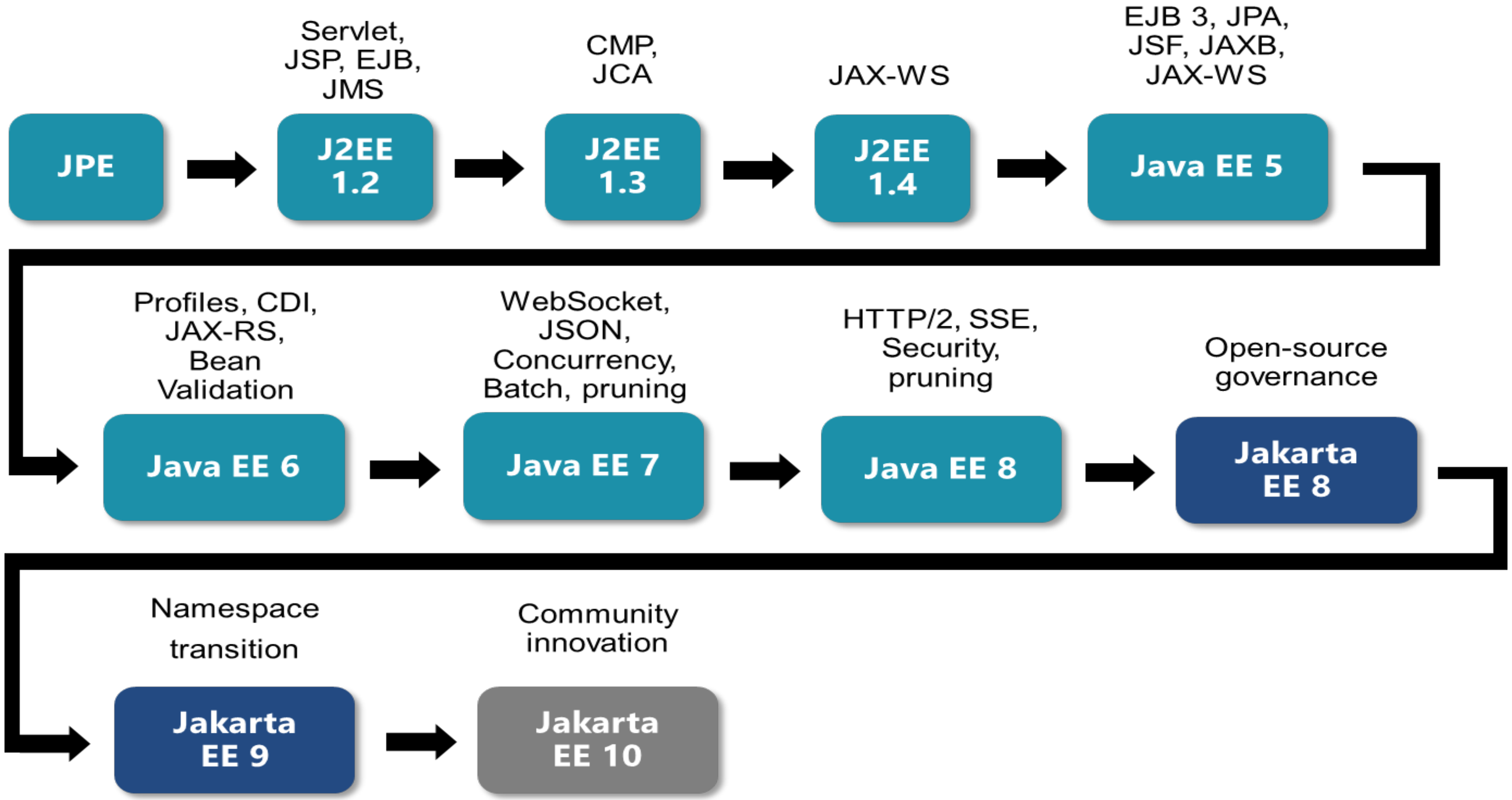


The Importance of Jakarta EE

- Jakarta EE is an important part of the Java ecosystem and cloud
- 25-35% of Java applications run on Jakarta EE application servers
  - WebLogic, WebSphere/Liberty, JBoss EAP, WildFly, Payara
- 70-80% of Java applications depend on at least one or more Jakarta EE APIs
  - Tomcat, Hibernate, ActiveMQ, Jetty, MicroProfile, Spring, Quarkus



Jakarta EE Evolution



What is Jakarta EE (1)

- Jakarta EE is a collection of services that help you write enterprise applications that run on the Java Platform. It provides the infrastructure often required for these applications so you that you can focus on core features and business logic.
- **Enterprise applications** support the high levels of security, scalability, and reliability that large organizations typically require. They can be web services, web applications, batch processes, and so on.
- The **Java Platform** consists of the Java Virtual Machine, compiler, standard APIs, and several tools that help you build, deploy, and debug Java applications. Java is the primary programming language, although the Java Platform supports several others, including Kotlin and Scala. Most Jakarta EE applications, however, are written in Java. The**Java Platform** is also called **Java Standard Edition (SE)**.

What is Jakarta EE (2)

- With Jakarta EE, you can pick and choose which services you need, and you use them with or without other frameworks and libraries. One framework commonly used with Jakarta EE is MicroProfile, which provides additional features commonly used in microservices.
- A key benefit of Jakarta EE is that it’s a set of open source standards supported by multiple vendors. Each vendor has their own implementation of the standards, and the vendors work together to update the standards over time. This means that you aren’t locked into a particular vendor, and each standard is well-thought-out and based on existing patterns and practices.
- Jakarta EE is stable and mature, and used in tens of thousands of mission-critical enterprises applications worldwide; yet it has evolved over time to keep up with modern computing trends, such as cloud computing.

Jakarta EE Services

- Jakarta EE provides a wide range of services, organized into three main categories. Each category is implemented by one of three profiles. The Core profile contains the foundational services. The Web profile contains the Core profile plus services for writing web applications. The Platform contains the Core and Web profiles, plus additional services for mail, batch processing, and messaging, and more.
- Services, Specificiations and Profiles lists each high-level service, the specification that provides it, and the profile that contains it provided in details in : https://jakartaee.github.io/jakartaee-documentation/jakartaee-tutorial/current/intro/overview/overview.html

Jakarta EE Containers and Servers

- The services we discussed in the Jakarta EE Services section are provided by container[1]. A Jakarta
- EE Server is software that runs the container, deploys applications into it, and provides administration and additional features. Jakarta EE Servers run stand-alone, but many implementations also support the ability to generate a single "fat", executable Java Archive (JAR) file that includes all the code necessary to run the application.

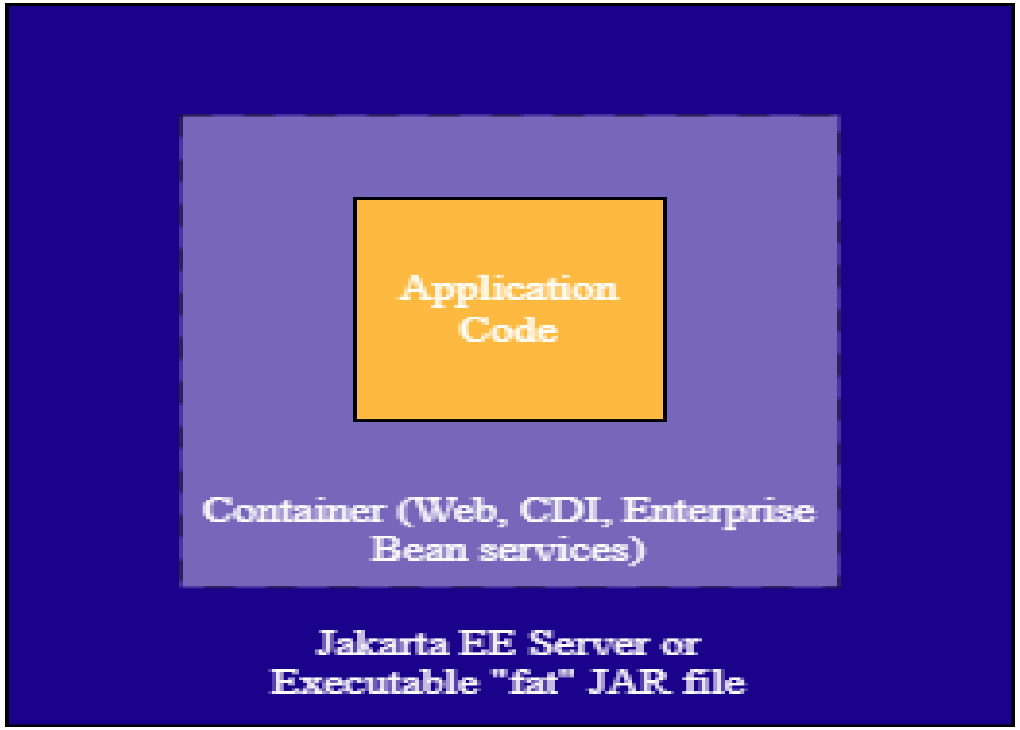
Jakarta EE Containers vs Docker Containers

- *Container* is a commonly used term in computing, but it basically means "that which holds something". In the case of Jakarta EE, the container "holds," or more accurately, executes your code and provides low-level application services, such as dependency injection, transactions, HTTP request handling, REST support, and so on.

- In the case of Docker container (which is technically an Open Container Initiative, or OCI container) your application is executed inside it, but the container provides operating system services, such as disk and network I/O.

- While the two container types are different, it’s entirely possible (and common) to run a Jakarta EE application inside a Docker container.

Relationship between Code, Container, and Server

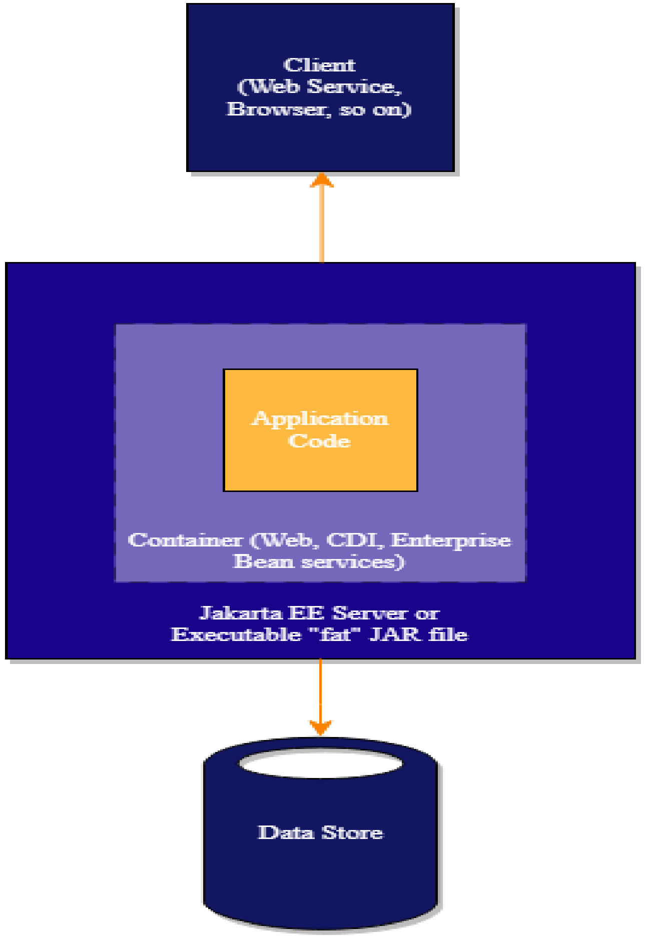


Application Code

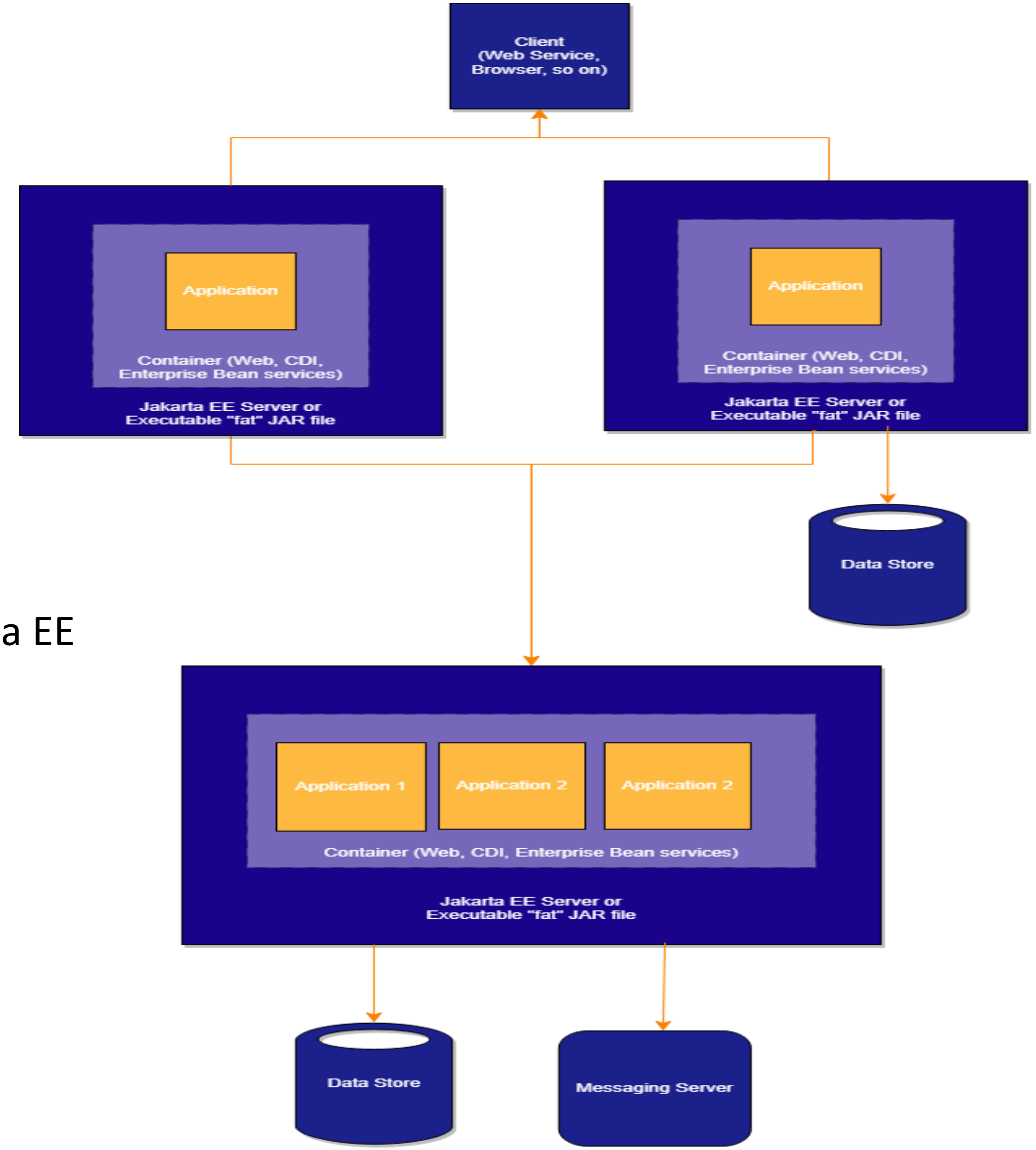
Container (Web, CDI, Enterprise Bean services)

Jakarta EE Server or Executable "fat" JAR file

A Simple Jakarta EE Application



A More Complex Jakarta EE Application


Client (Web Service, Browser, so on)
Application
Container (Web, CDI, Enterprise Bean services)
Jakarta EE Server or Executable "fat" JAR file
Application
Container (Web, CDI, Enterprise Bean services)
Jakarta EE Server or Executable "fat" JAR file
Data Store
Application 1
Application 2
Application 2
Container (Web, CDI, Enterprise Bean services)
Jakarta EE Server or Executable "fat" JAR file
Data Store
Messaging Server

Jakarta EE Components (1)

- Jakarta EE applications are made up of components. A **Jakarta EE component** is a self-contained functional unit that makes use of one or more container services. That usage could be as simple as injecting a single dependency or responding to REST requests, or as complicated as injecting multiple dependencies, querying a database, firing an event, and participating in distributed transactions.

Jakarta EE Components (2)

Jakarta EE supports the following components:

- **Web components** interact with web standards (HTTPS, HTML, WebSocket, and so on) using Jakarta Servlet, Jakarta Faces, Jakarta WebSocket, and related technologies.
- Business components implement logic necessary to support the business domain using either **Enterprise bean components (enterprise beans)** or CDI managed beans.
  - For new applications, business components should be implemented using CDI managed beans, unless you need Enterprise bean-specific features such as transactions, role-based security, messaging driven beans, or remote execution. The two technologies play well together.

All of these components run on the **server**, and are ordinary Java classes written with Jakarta EE annotations (or optionally, external configuration files called deployment descriptors).

Usage of Java Standard Edition (SE) APIs

In addition to many of the core Java programming language APIs, there are a couple of key APIs you may encounter when working with Jakarta EE: Java Database Connectivity (JDBC) and Java Naming and Directory Interface (JNDI).

- **Java Database Connectivity API**

The Java Database Connectivity (JDBC) API lets you work with SQL databases. You can use the JDBC API directly from within a Jakarta EE component, but most Jakarta EE applications use Jakarta Persistence to map between objects and database tables. Jakarta EE servers also support managed JDBC data sources, which can be configured for one or more applications.

- **Java Naming and Directory Interface API**

The Java Naming and Directory Interface (JNDI) API allows you to work with multiple naming and directory services, such as LDAP, DNS, and NIS. The API has methods for performing standard directory operations, such as associating attributes with objects and searching for objects using their attributes.

How do I get Jakarta EE?

Since Jakarta EE is an open-source industry standard, there are multiple implementations. A good place to start is the Jakarta EE Starter, which lets you quickly generate a sample starter app using one of the supported Jakarta EE servers. You can also find the most recent list of servers on the Jakarta EE website.

Java EE(Jakarta EE) Vs Spring Boot

- Java EE and Spring Boot are major frameworks in enterprise Java development.
- Choosing between the two affects an application’s architecture and long-term maintainability.
- Understanding their differences is crucial for developers, architects, and decision-makers.
- Each framework has distinct approaches and benefits depending on project requirements.

Understanding Java EE (Jakarta EE)

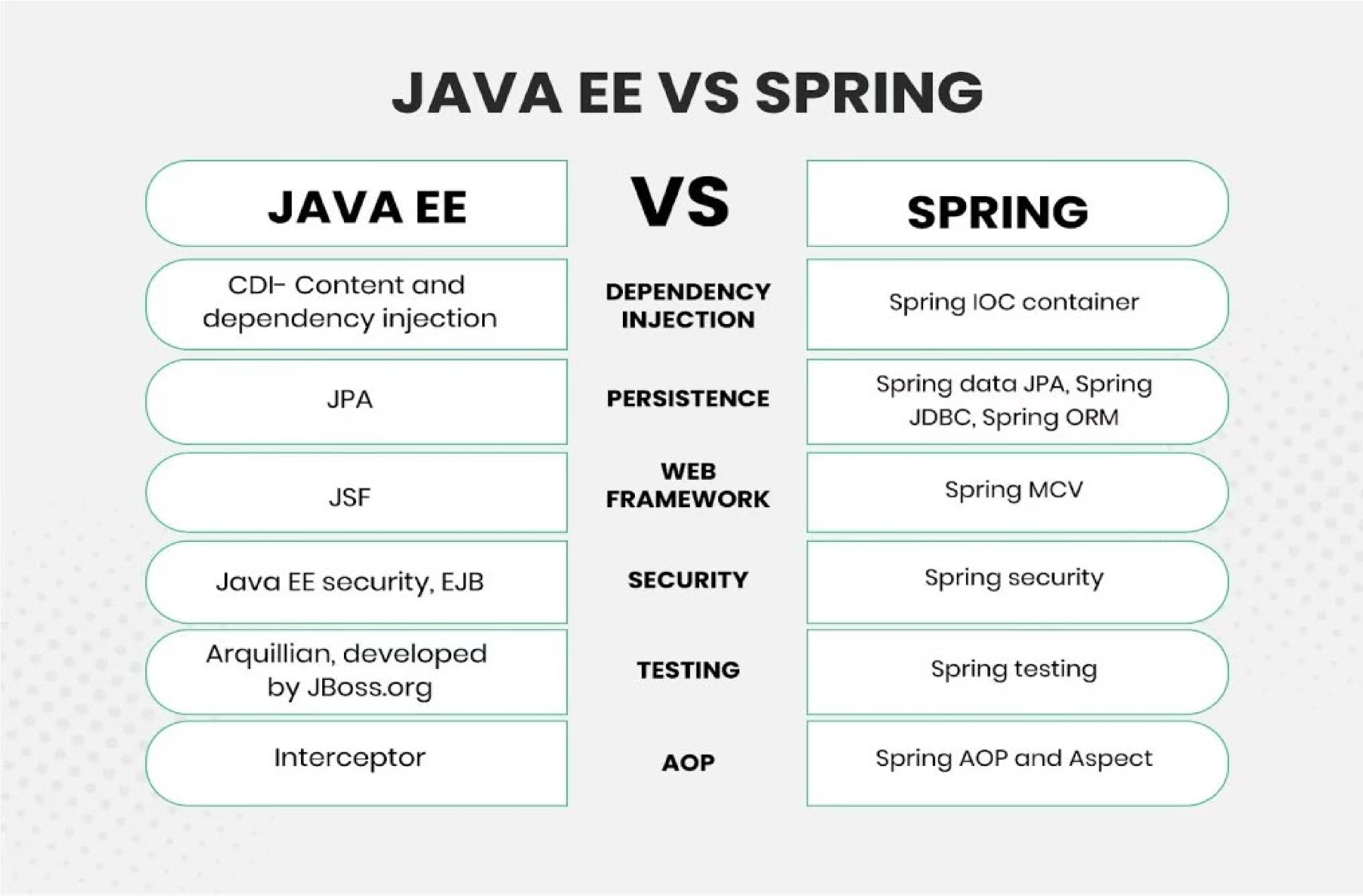
- Java EE is a platform for enterprise-level development extending Java SE.
- Provides APIs for web services, messaging, and database access (e.g., Servlets, JSP, EJB).
- Container services manage components’ security and transaction management.
- Java EE applications require application servers like GlassFish or WildFly.
- New models like CDI unify the platform with modern programming practices.

Exploring Spring Boot

- Spring Boot simplifies Spring application setup with auto-configuration.
- Supports stand-alone applications with embedded servers (Tomcat, Jetty).
- Production-ready features like health checks and metrics, ideal for microservices.
- Reduces boilerplate code with its ‘convention over configuration’ approach.

Java EE vs. Spring Boot: Key Differences

- Java EE is a specification, while Spring Boot is a project within Spring Framework.
- Java EE emphasizes flexibility and standardization; Spring Boot focuses on speed and simplicity.
- Java EE requires deployment descriptors; Spring Boot uses auto-configuration.
- Java EE uses standard APIs (e.g., @Inject); Spring Boot uses Spring-specific annotations (e.g., @Autowired).



JAVA EE VS SPRING

JAVA EE	VS	SPRING
CDI- Content and dependency injection	DEPENDENCY INJECTION	Spring IOC container
JPA	PERSISTENCE	Spring data JPA, Spring JDBC, Spring ORM
JSF	WEB FRAMEWORK	Spring MCV
Java EE security, EJB	SECURITY	Spring security
Arquillian, developed by JBoss.org	TESTING	Spring testing
Interceptor	AOP	Spring AOP and Aspect

Development and Deployment

- Java EE requires more boilerplate code and a Java EE server for deployment.
- Spring Boot simplifies setup with starter kits and uses embedded servers.
- Both frameworks support tools like Maven and Gradle for dependency management.
- Spring Boot is more convention-driven, while Java EE offers more flexibility in configuration.

Community and Support

- Java EE (now Jakarta EE) has a long history and backing by major industry players.
- Spring Boot has an active community supported by the Spring ecosystem and Pivotal Software.
- Both have strong community-driven support, resources, and professional backing.
- Spring Boot’s user-friendly nature has attracted a rapidly growing developer base.

Performance and Scalability

- Java EE leverages powerful servers (e.g., WildFly, GlassFish) with built-in scalability.
- Spring Boot’s lightweight nature excels in microservices architecture.
- Java EE offers clustering and fail-over out-of-the-box with its servers.
- Spring Boot simplifies horizontal scaling for dynamic service deployment.

Pros of Java EE and Spring Boot

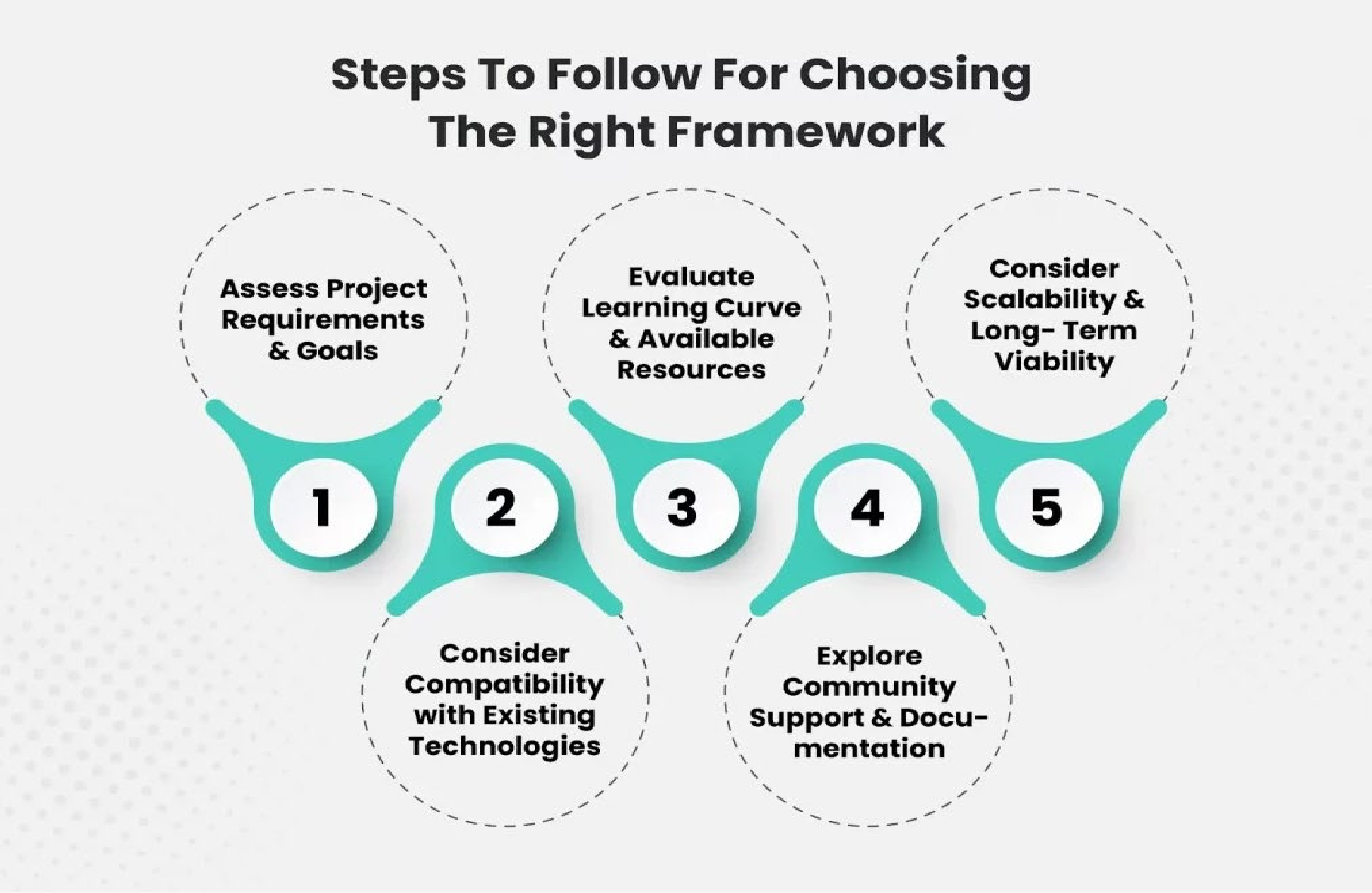
- Java EE is standardized and portable across servers, ensuring wide compatibility.
- Spring Boot simplifies development, offering rapid application setup.
- Java EE integrates various technologies like JMS and JPA for largescale systems.
- Spring Boot is microservices-ready with built-in support for packaging into executable JARs.

Cons of Java EE and Spring Boot

- Java EE is seen as overly complex due to its extensive APIs (JMS, EJB, etc.).
- Spring Boot’s auto-configuration may lead to higher memory consumption.
- Java EE’s heavyweight nature slows deployment and increases resource use.
- Spring Boot’s opinionated defaults can restrict full control over application setup.
- Java EE is slower to adopt new trends compared to Spring Boot.

Tips for Choosing the Right Framework

- Java EE is best for large-scale, enterprise-level applications requiring extensive APIs.
- Spring Boot is ideal for rapid development, smaller apps, or microservices architecture.
- Consider project size, complexity, and scalability needs when choosing.
- Spring Boot offers faster setup and development speed with minimal configuration.
- Java EE’s flexibility makes it suited for complex applications requiring high customization.



Steps To Follow For Choosing The Right Framework

1. Assess Project Requirements & Goals
2. Consider Compatibility with Existing Technologies
3. Evaluate Learning Curve & Available Resources
4. Explore Community Support & Documentation
5. Consider Scalability & Long- Term Viability

Questions?