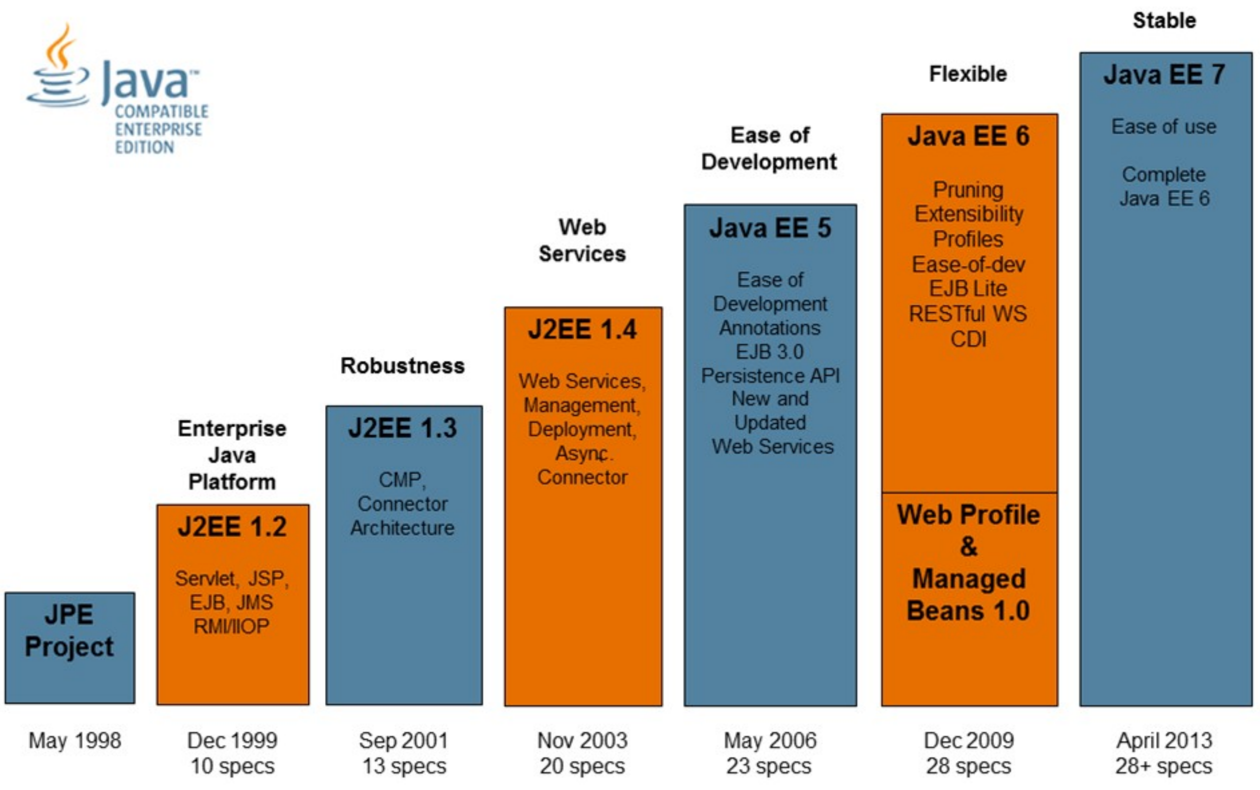
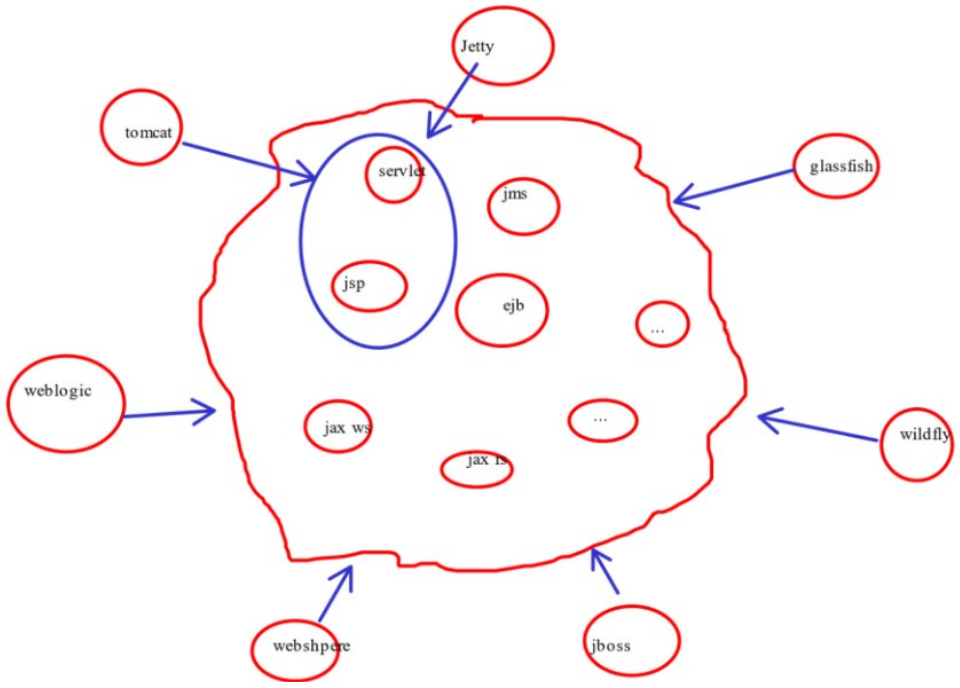
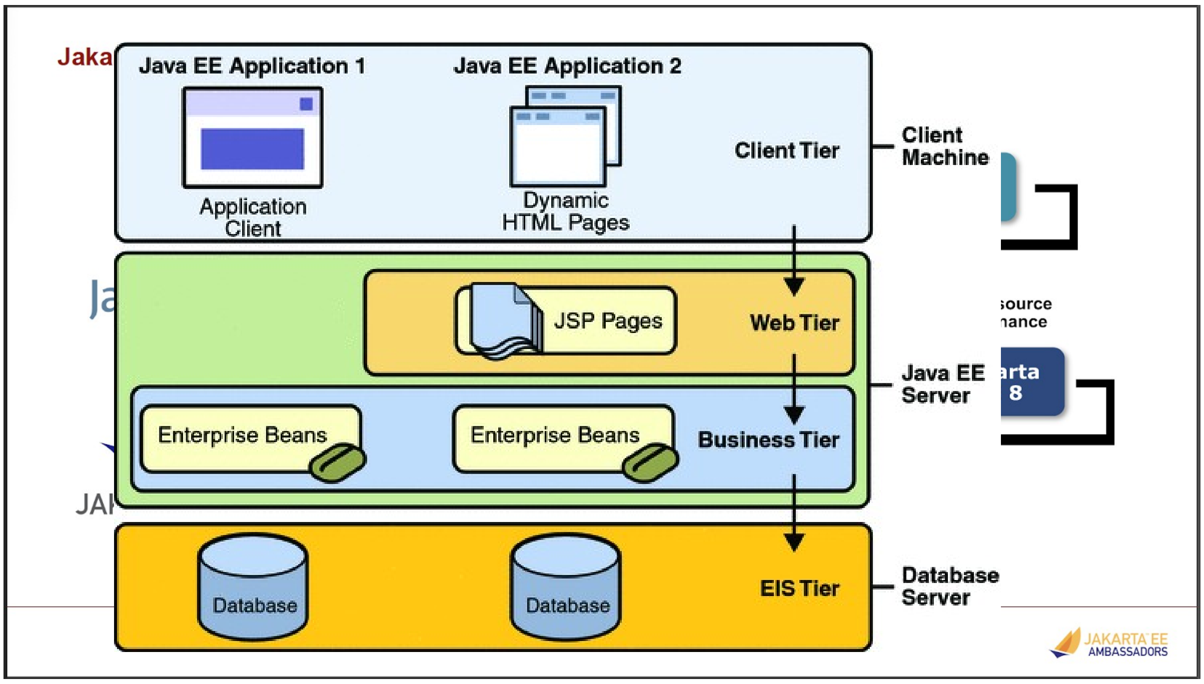
🔷 Java EE (Jakarta EE) Ecosystem Overview

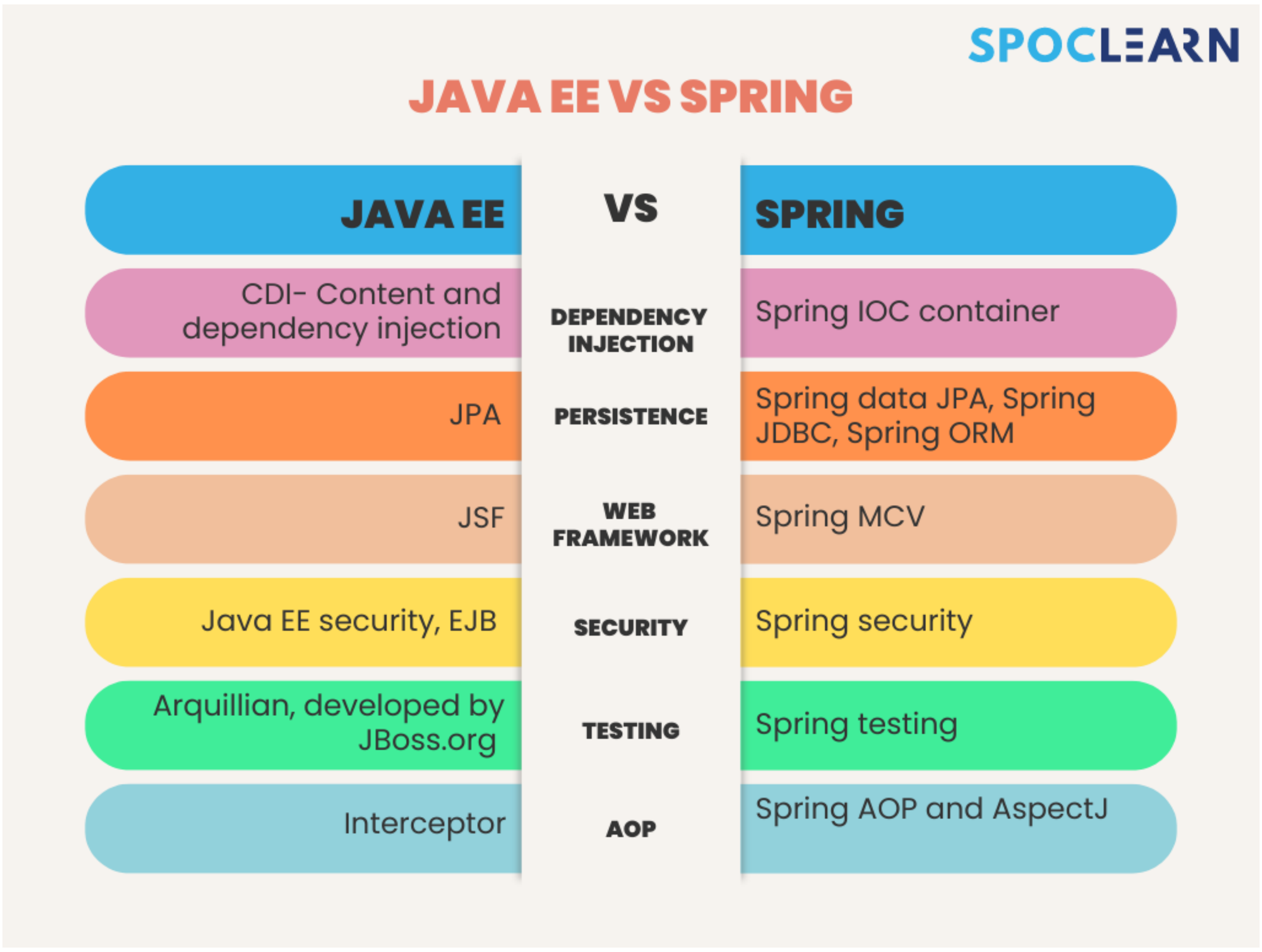
Java EE (now **Jakarta EE**) is a set of **specifications** for building **enterprise-level applications** using Java.



J2EE is group of specification to create dynamic distributed application



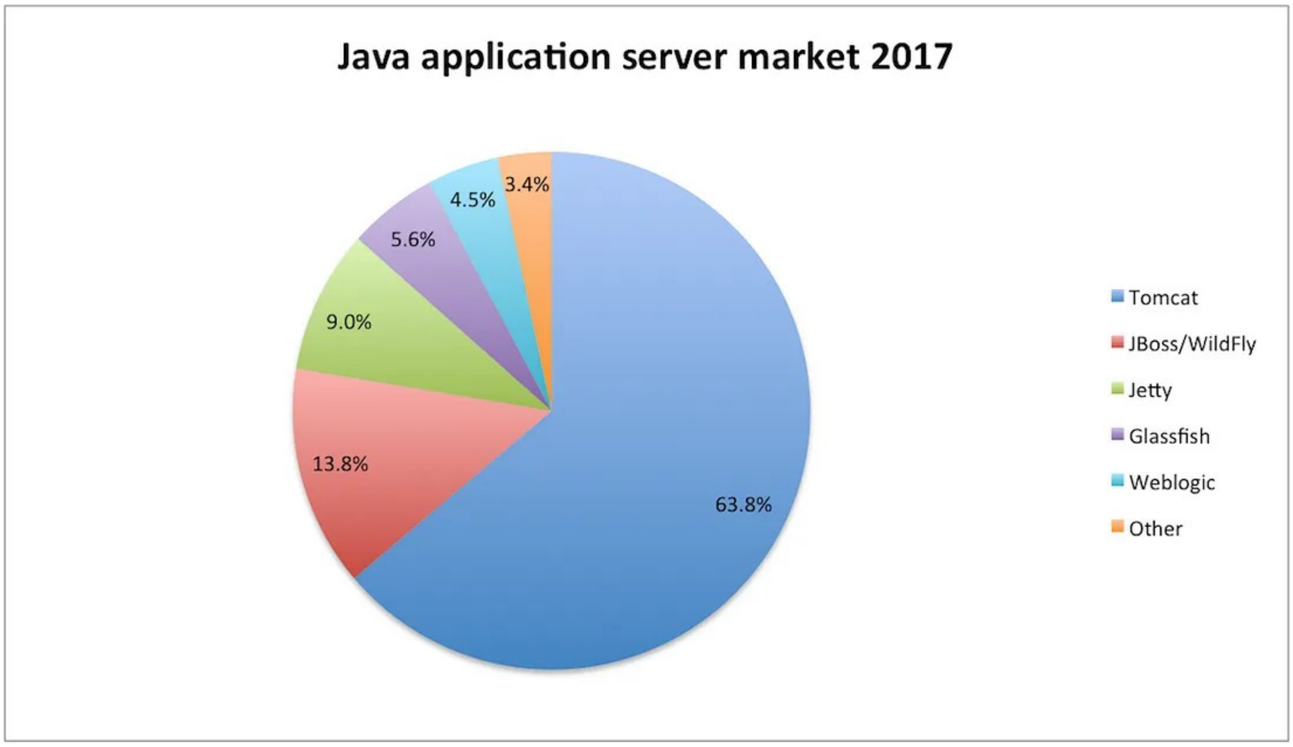

Java EE Application 1
Java EE Application 2
Application Client
Dynamic HTML Pages
Client Tier
Client Machine
JSP Pages
Web Tier
Enterprise Beans
Enterprise Beans
Business Tier
Java EE Server
Database
Database
EIS Tier
Database Server



SPOCLEARN

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 AspectJ



🔹 Core Components of Java EE

Component	Description
Servlet	Handles HTTP requests and responses; backbone of web apps.
JSP (JavaServer Pages)	Allows embedding Java in HTML for dynamic web pages.
EJB (Enterprise Java Beans)	For building scalable, transactional business logic.
JPA (Java Persistence API)	ORM layer for database interaction using entities.
JSF (JavaServer Faces)	UI framework for building component-based web interfaces.
JAX-RS / JAX-WS	Build RESTful (JAX-RS) and SOAP (JAX-WS) web services.
CDI (Contexts and Dependency Injection)	Dependency injection and lifecycle management.
JTA (Java Transaction API)	Manages distributed transactions.
JMS (Java Messaging Service)	Enables asynchronous communication via messaging.

🔹 Popular Application Servers

- **Oracle WebLogic**
- **Red Hat WildFly (JBoss)**
- **IBM WebSphere**
- **Apache TomEE**
- **Payara Server / GlassFish**

🔹 Build Tools and Frameworks

- **Maven / Gradle** – Build and dependency management
- **Hibernate** – Implementation of JPA
- **Spring Framework** – Often used as an alternative to EJB/CDI
- **Jakarta EE 10+** – Newer versions under Eclipse Foundation

---

🔹 Modern Trends

- **MicroProfile** – Lightweight Java EE for microservices
- **Jakarta EE** – Official name after Oracle donated Java EE to Eclipse
- **Cloud-native** – Integration with Docker, Kubernetes
- **Spring Boot vs Java EE** – Spring is more flexible and popular for microservices

🔷 Brief Introduction to WebLogic Application Server

Oracle WebLogic Server is a powerful, enterprise-grade **Java EE application server** used to **deploy, run, and manage** Java-based web and enterprise applications.

It provides a complete environment for developing and hosting **scalable**, **secure**, and **transactional** Java applications.

🔹 Key Features

- **Full Java EE support** (Servlets, JSP, EJB, JPA, JMS, JAX-RS/WS)
- High performance and scalability for large applications
- Built-in **clustering**, **load balancing**, and **failover**
- Integrated with **Oracle Database** and other Oracle middleware
- Supports **JMX-based monitoring**, **logging**, and **performance tuning**
- Web-based **Administration Console** and **WLST (WebLogic Scripting Tool)** for automation

---

🔹 Core Components

Component	Description
Domain	A logical group of WebLogic resources (servers, clusters)
Admin Server	Central controller that manages the domain
Managed Server	Hosts business applications, managed by the Admin Server
Node Manager	Starts, stops, and monitors servers remotely
Deployment Descriptors	XML files like `web.xml`, `weblogic.xml` to configure deployments

---

🔹 Typical Use Cases

- Hosting **Java EE web apps** and **enterprise apps**
- Running **SOAP/REST services**
- Backend for **Oracle Fusion Middleware** and **SOA Suite**
- Integrating with legacy Oracle systems

🔹 Simple Deployment Process

1. Package your application as `.war` or `.ear`

2. Log in to **WebLogic Admin Console**
3. Upload and deploy via GUI or use `WLST`/CLI

Lab setup:

1. Download and install jdk 8

https://www.oracle.com/in/java/technologies/javase/javase8-archive-downloads.html#license-lightbox

2. Eclipse download

https://www.eclipse.org/downloads/packages/release/oxygen/r/eclipse-ide-java-ee-developers

3. Weblogic download

https://www.oracle.com/middleware/technologies/weblogic-server-installers-downloads.html

https://www.oracle.com/middleware/technologies/weblogic-server-installers-downloads.html#license-lightbox

Step 1: Install JDK and setup environment variable
Step 2: Setup eclipse and run some example using tomcat server
Step 3: Install weblogic server

Step 3.1
"C:\Program Files\Java\jdk1.8.0_202\bin\java.exe" -jar fmw_12.2.1.2.0_wls_quick.jar ORACLE_HOME="C:\Weblogic"

Creating a domain in weblogic 12c

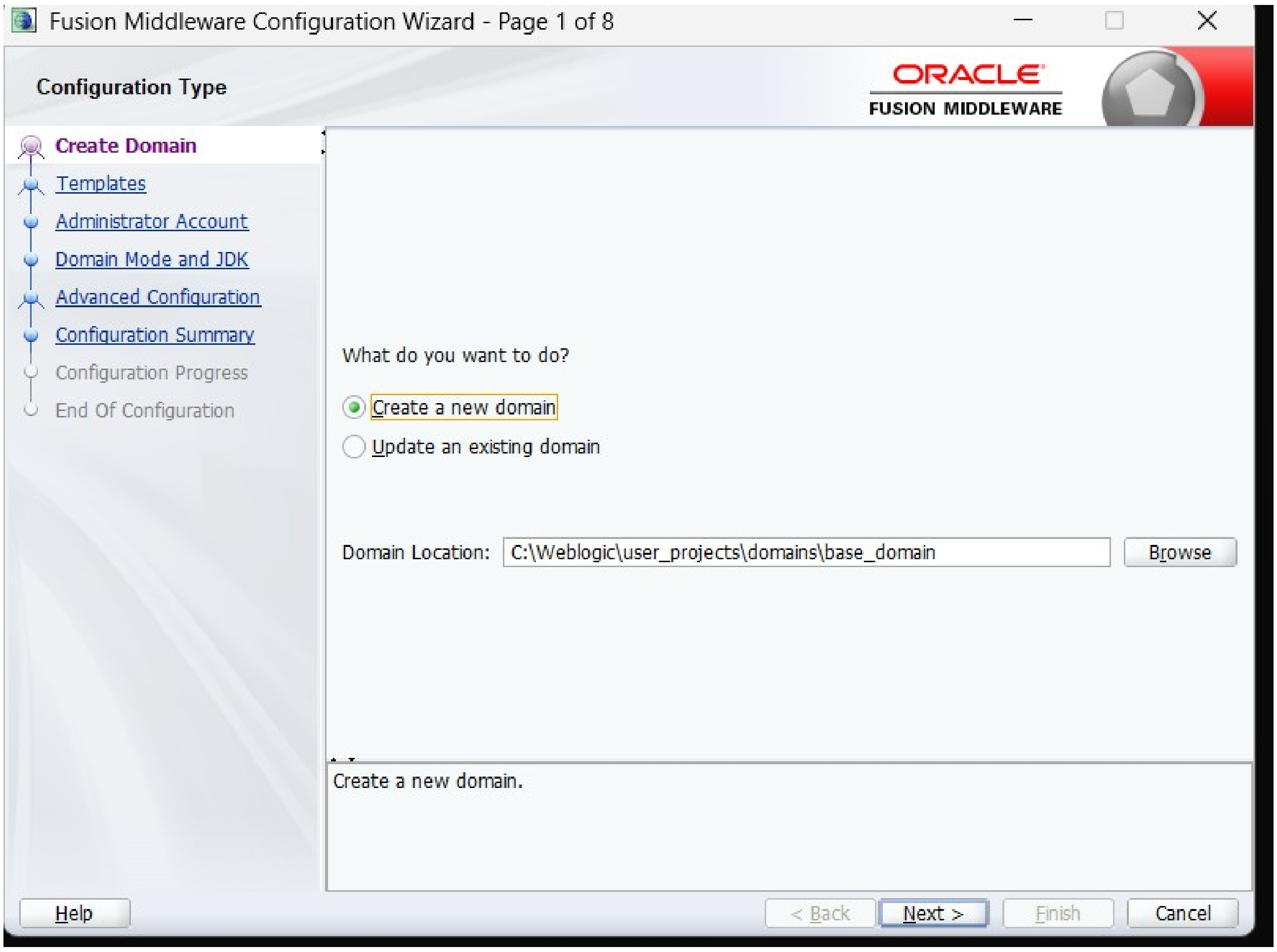
Step 1.1 : go to the location

C:\Weblogic\oracle_common\common\bin

and run config.cmd

An configuration wizard will come up

step 1.2:



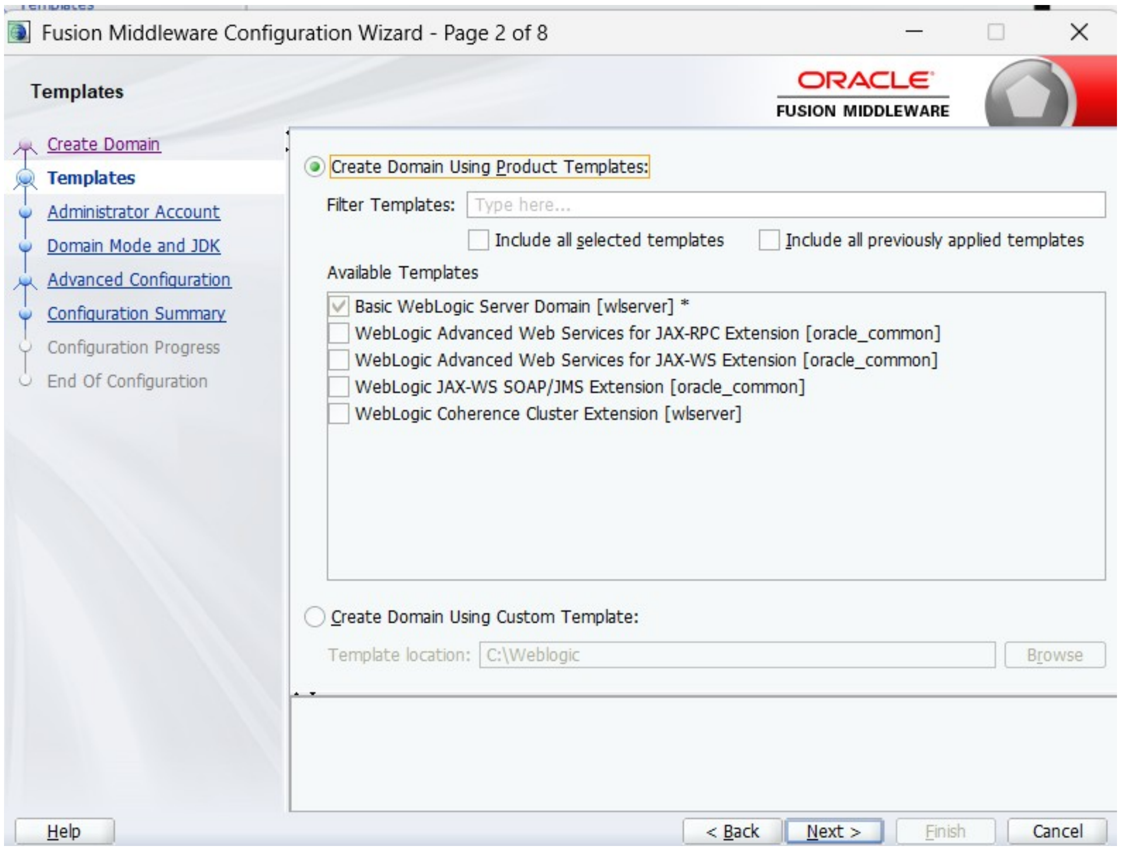
What do you want to do?

(●) Create a new domain

( ) Update an existing domain

Domain Location: C:\Weblogic\user_projects\domains\base_domain

Create a new domain.


Fusion Middleware Configuration Wizard - Page 2 of 8

Templates

[✓ Create Domain Using Product Templates:]

Filter Templates: Type here...

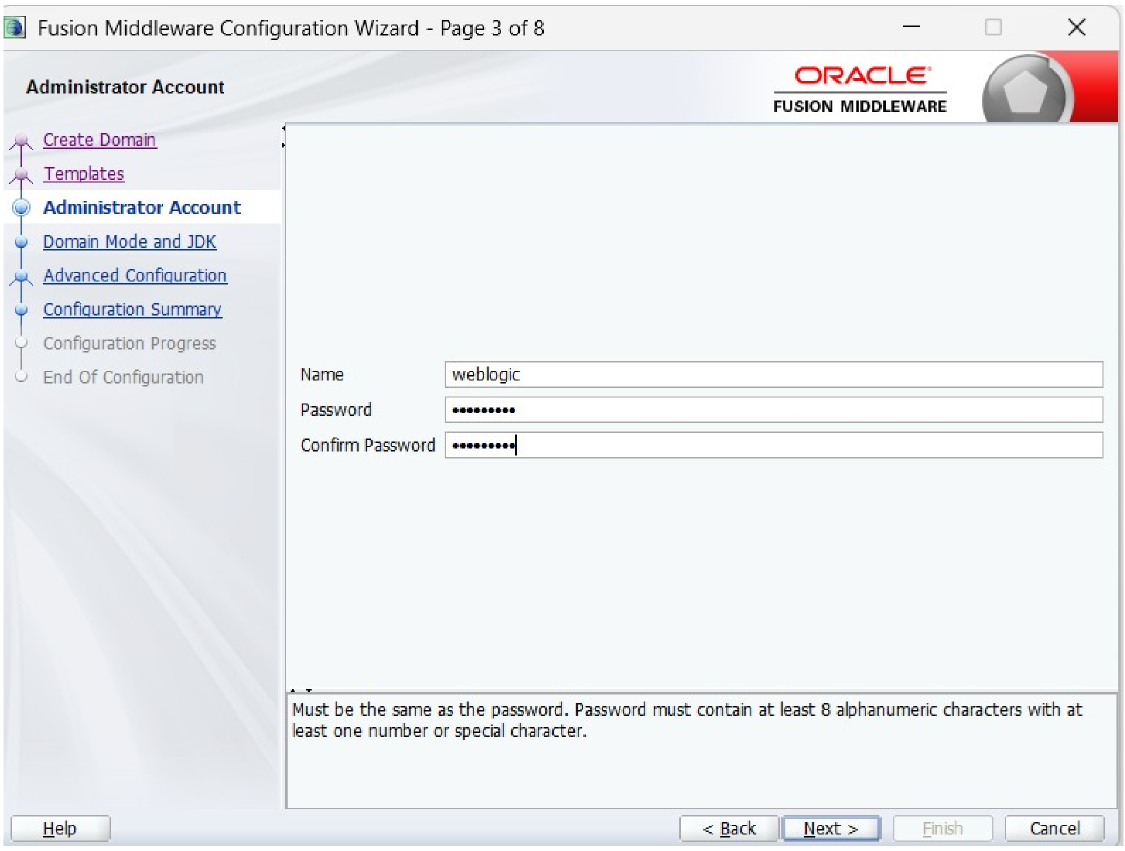
☐ Include all selected templates ☐ Include all previously applied templates

Available Templates

- ✓ Basic WebLogic Server Domain [wlserver] *
- ☐ WebLogic Advanced Web Services for JAX-RPC Extension [oracle_common]
- ☐ WebLogic Advanced Web Services for JAX-WS Extension [oracle_common]
- ☐ WebLogic JAX-WS SOAP/JMS Extension [oracle_common]
- ☐ WebLogic Coherence Cluster Extension [wlserver]

Create Domain Using Custom Template:

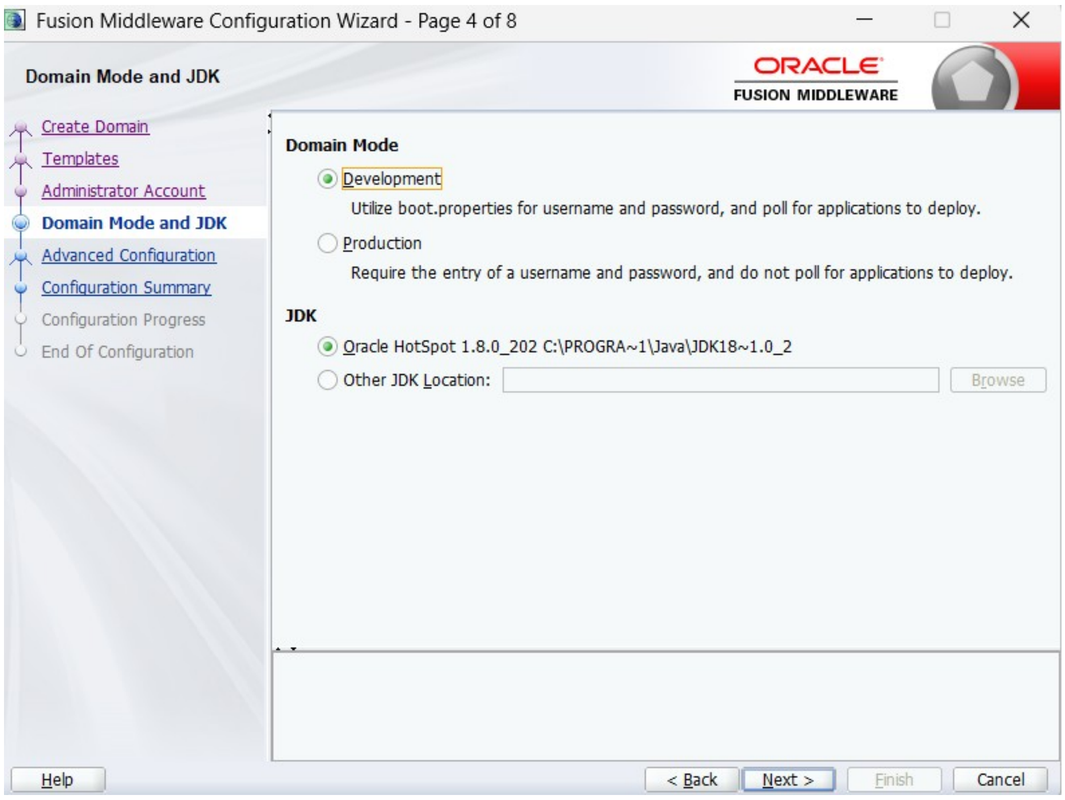
Template location: C:\Weblogic Browse


Fusion Middleware Configuration Wizard - Page 3 of 8

Administrator Account

Name	weblogic
Password	•••••••••
Confirm Password	•••••••••

Must be the same as the password. Password must contain at least 8 alphanumeric characters with at least one number or special character.


Fusion Middleware Configuration Wizard - Page 4 of 8

Domain Mode and JDK

Domain Mode

[✓ Development]

Utilize boot.properties for username and password, and poll for applications to deploy.

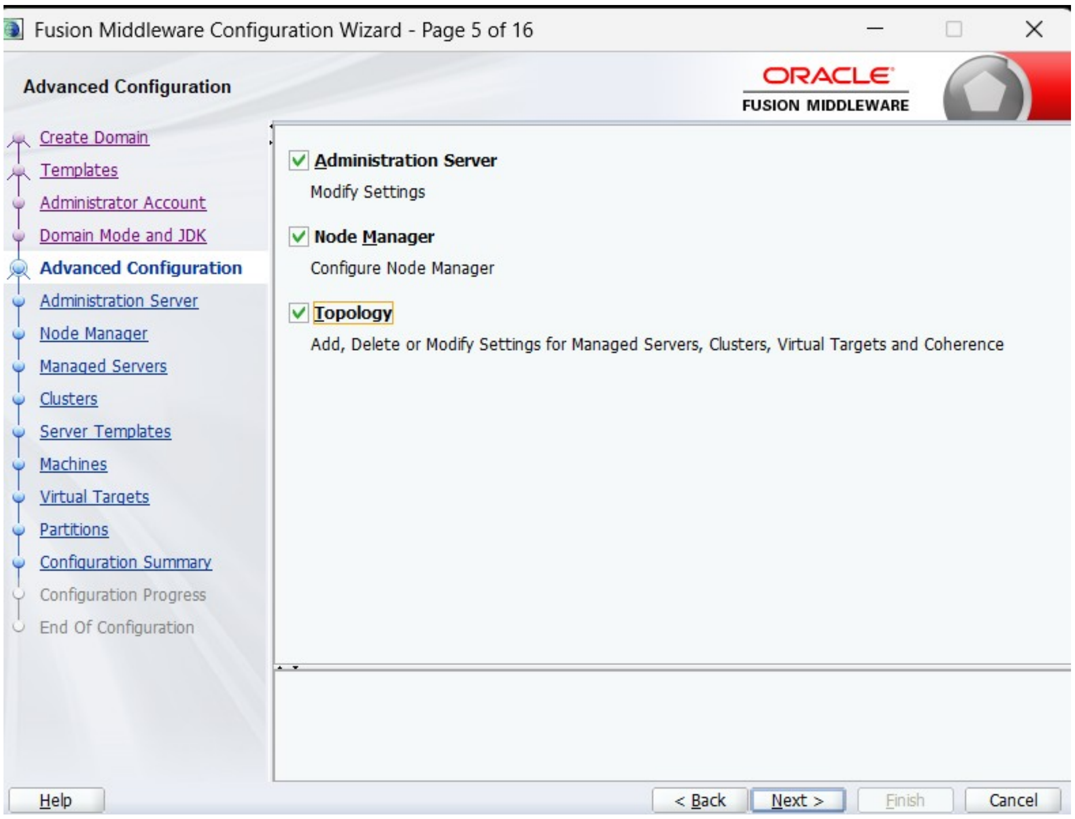
Production

Require the entry of a username and password, and do not poll for applications to deploy.

JDK

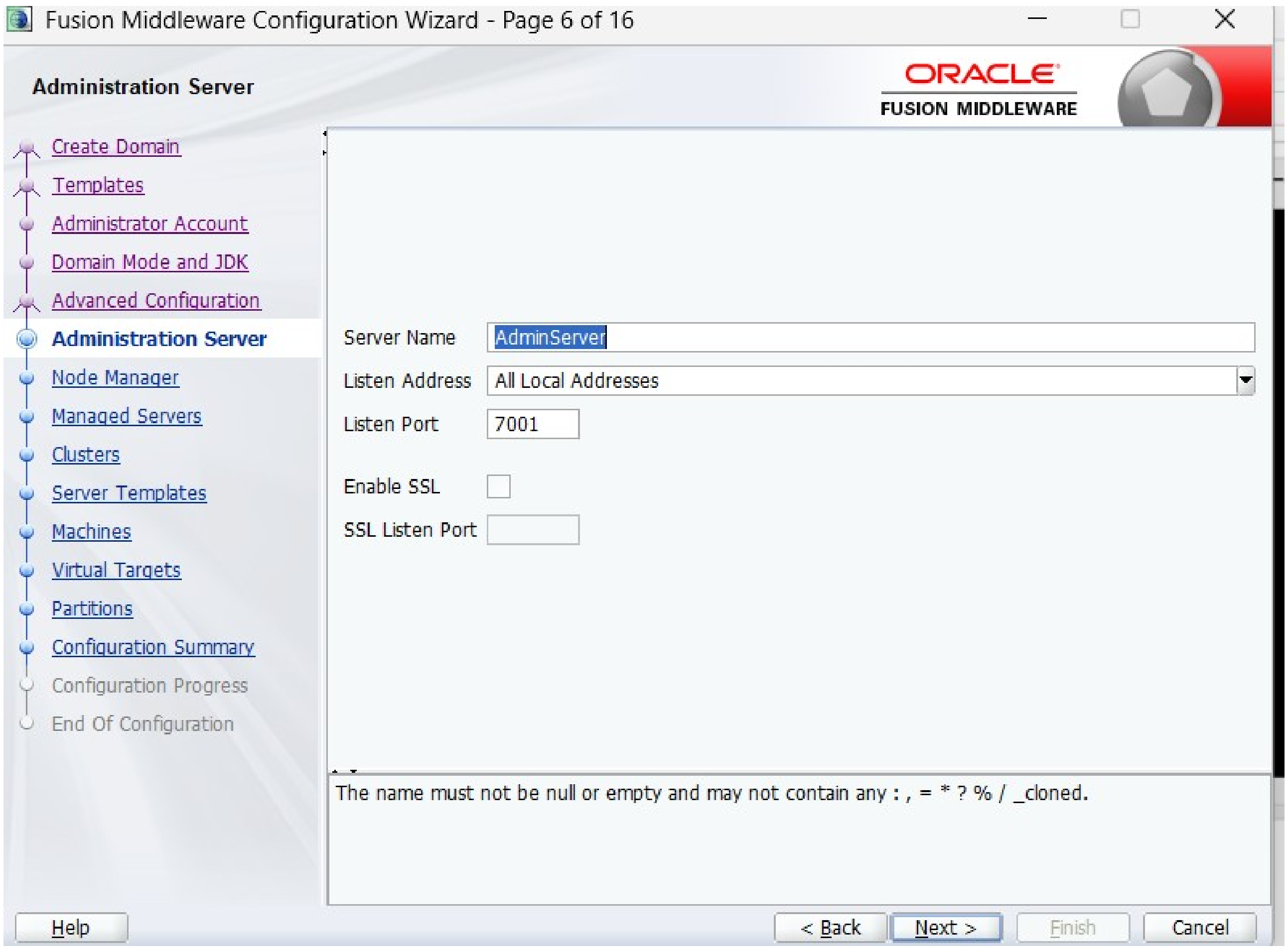
[✓ Oracle HotSpot 1.8.0_202 C:\PROGRA~1\Java\JDK18~1.0_2]

Other JDK Location:


Fusion Middleware Configuration Wizard - Page 5 of 16

Advanced Configuration

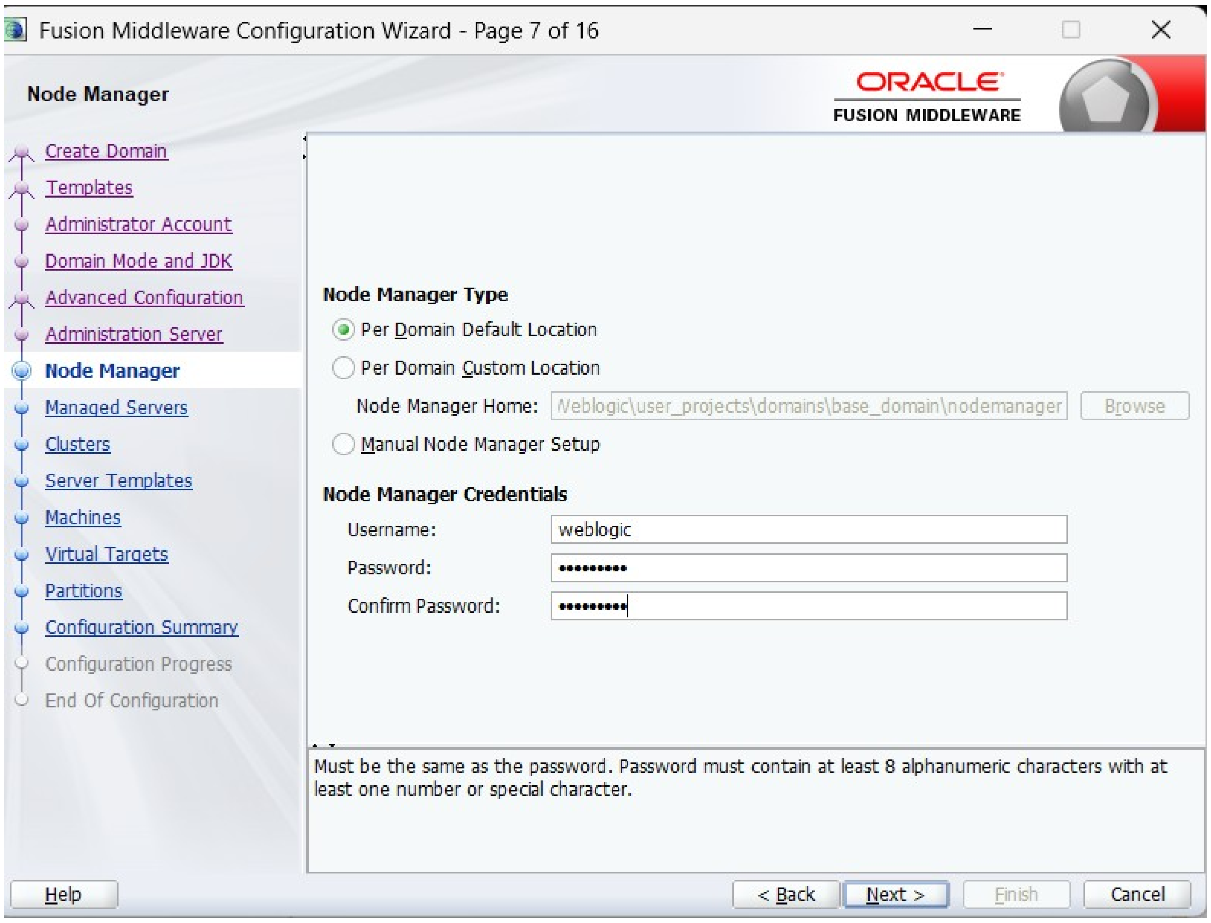
- [✓] Administration Server
  Modify Settings
- [✓] Node Manager
  Configure Node Manager
- [✓] Topology
  Add, Delete or Modify Settings for Managed Servers, Clusters, Virtual Targets and Coherence


Fusion Middleware Configuration Wizard - Page 6 of 16

Administration Server

Server Name	AdminServer
Listen Address	All Local Addresses
Listen Port	7001
Enable SSL	☐
SSL Listen Port	

The name must not be null or empty and may not contain any : , = * ? % / _cloned.


Fusion Middleware Configuration Wizard - Page 7 of 16

Node Manager

Node Manager Type

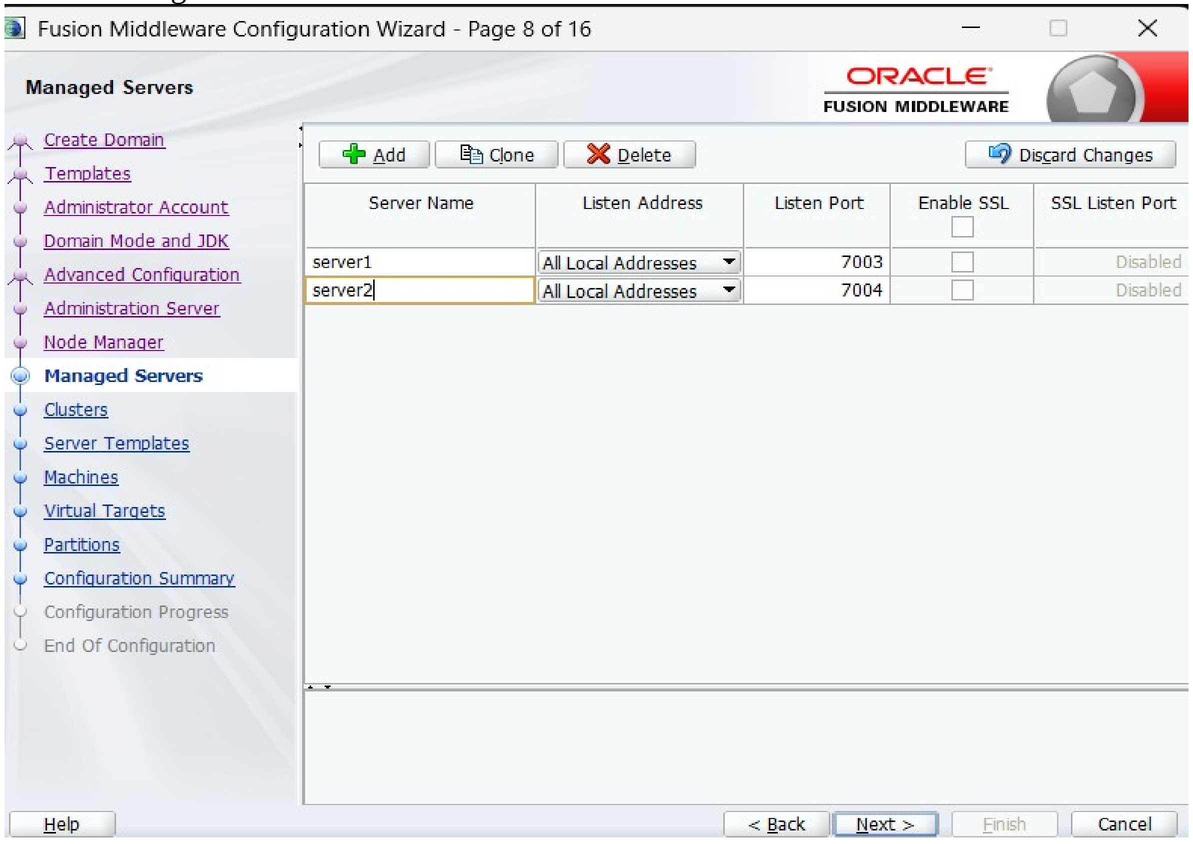
- [✓ Per Domain Default Location]
- Per Domain Custom Location
  - Node Manager Home: \Weblogic\user_projects\domains\base_domain\nodemanager
- Manual Node Manager Setup

Node Manager Credentials

Username:	weblogic
Password:	••••••••••
Confirm Password:	••••••••••

Must be the same as the password. Password must contain at least 8 alphanumeric characters with at least one number or special character.

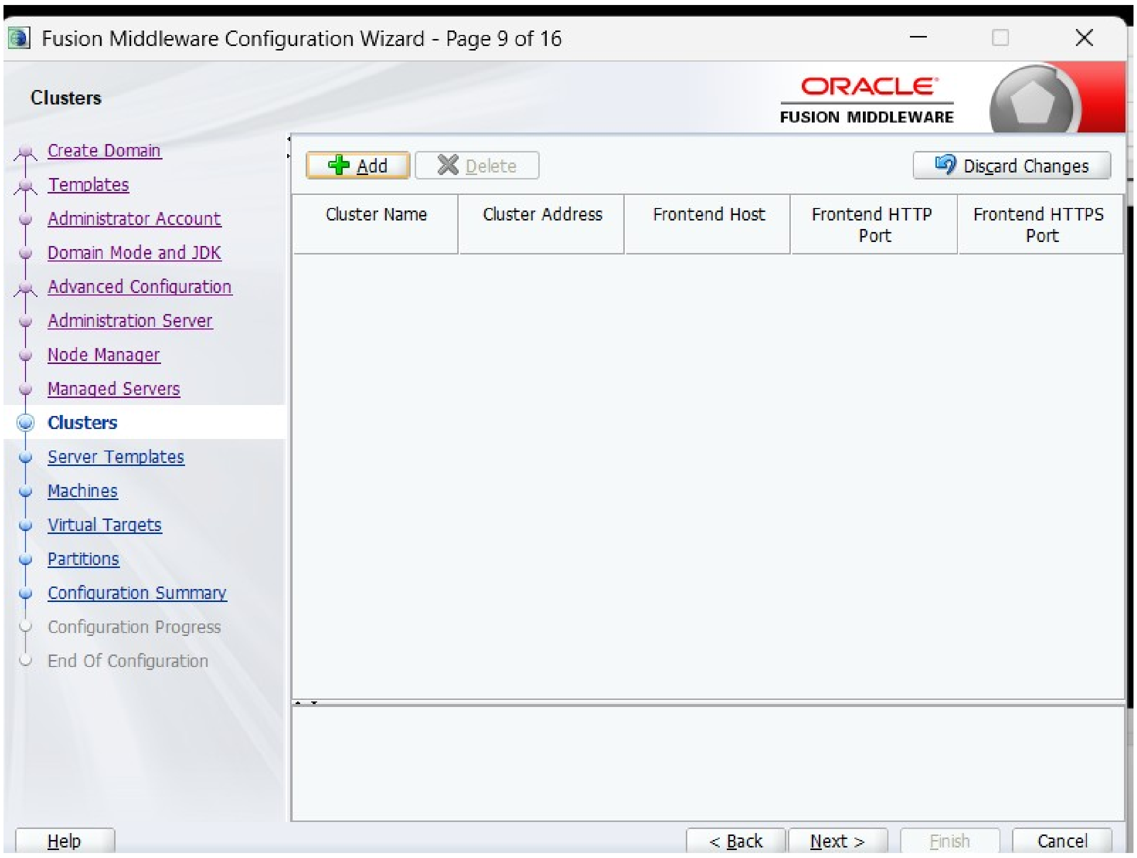
Create two managed server as shown


Fusion Middleware Configuration Wizard - Page 8 of 16

Managed Servers

Server Name	Listen Address	Listen Port	Enable SSL	SSL Listen Port
server1	All Local Addresses	7003	☐	Disabled
server2	All Local Addresses	7004	☐	Disabled

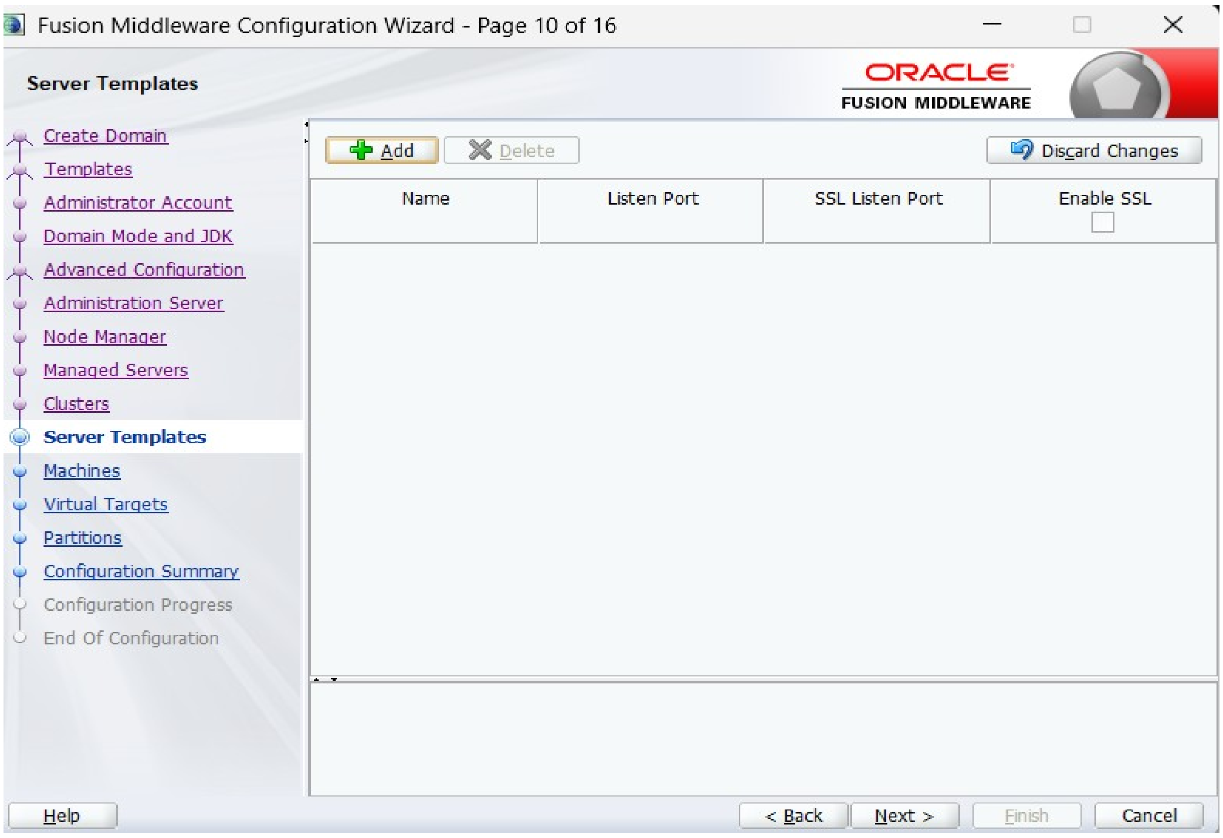
I dont want to configure any cluster


Fusion Middleware Configuration Wizard - Page 9 of 16

Clusters

Cluster Name	Cluster Address	Frontend Host	Frontend HTTP Port	Frontend HTTPS Port

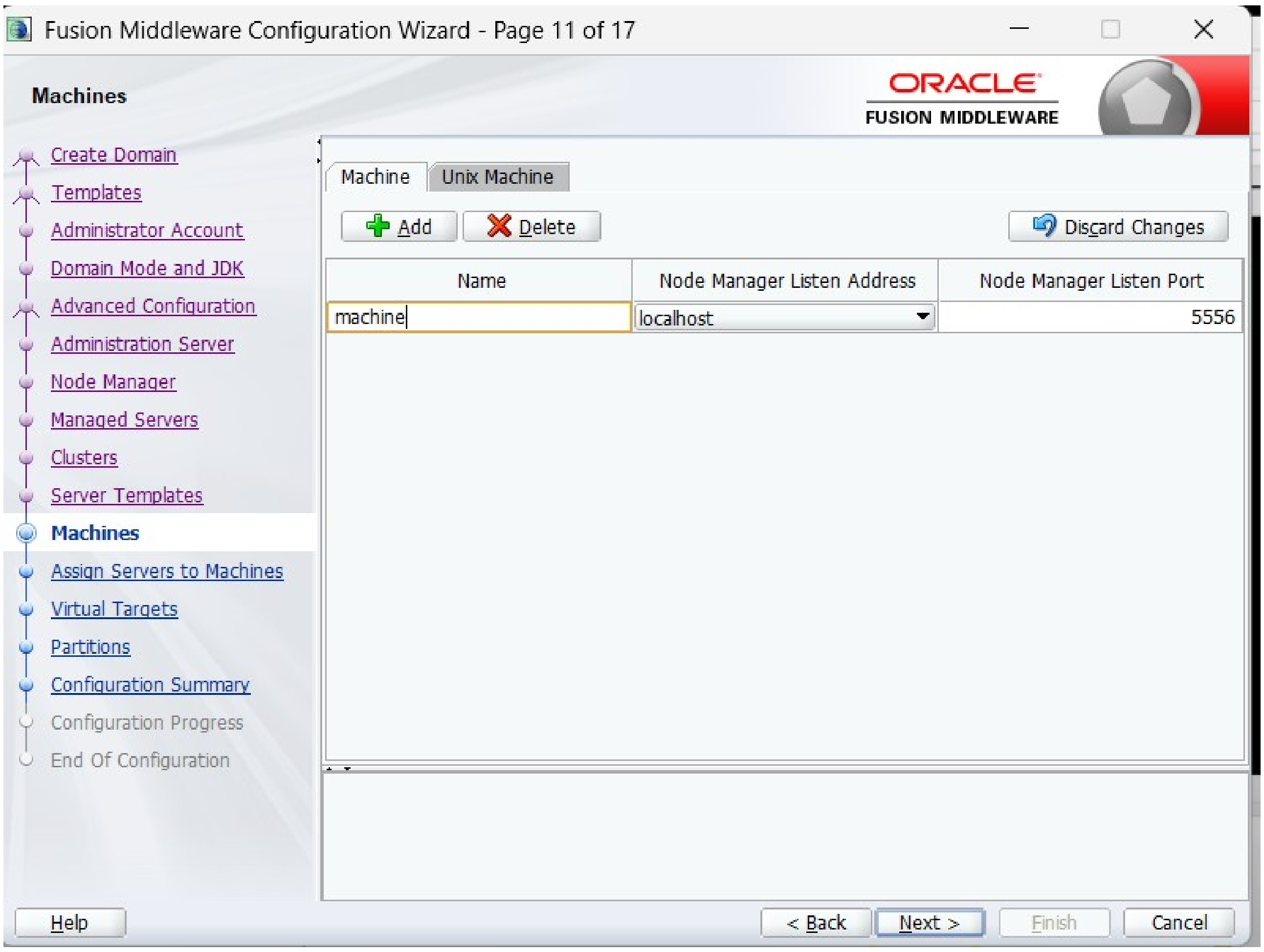
I dont want to configure any server template


Fusion Middleware Configuration Wizard - Page 10 of 16

Server Templates

Name	Listen Port	SSL Listen Port	Enable SSL ☐

Create machine as shown below

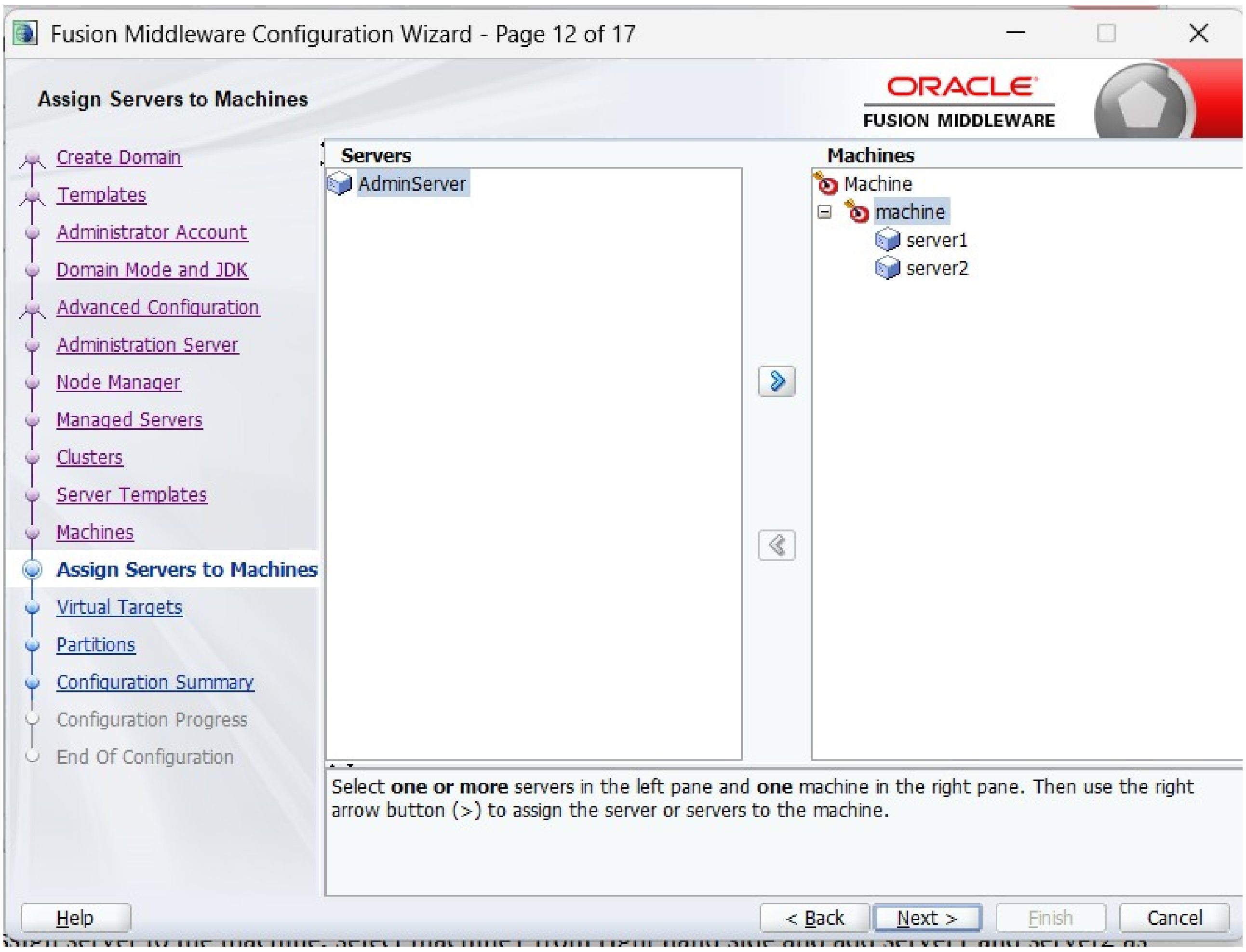


Fusion Middleware Configuration Wizard - Page 11 of 17

Machines

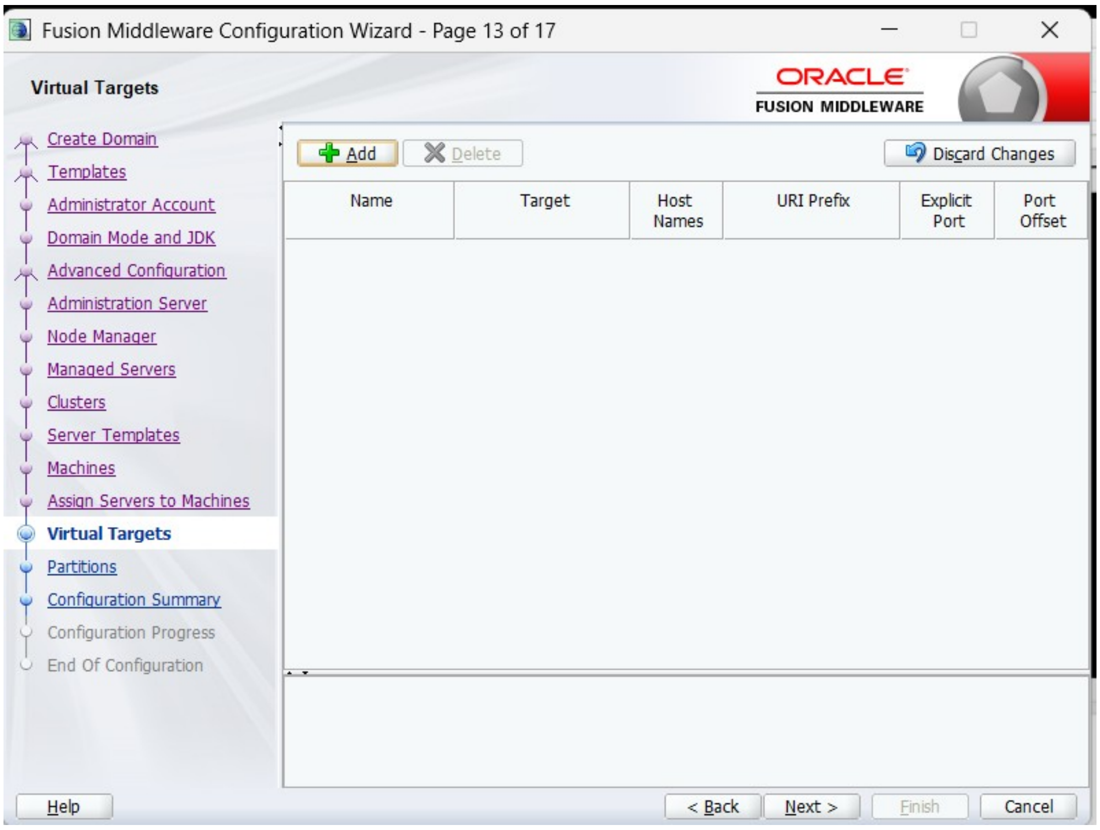
Name	Node Manager Listen Address	Node Manager Listen Port
machine	localhost	5556

Assign server to the machine, select machine1 from right hand side and add server1 and server2 as shown



Select **one or more** servers in the left pane and **one** machine in the right pane. Then use the right arrow button (>) to assign the server or servers to the machine.

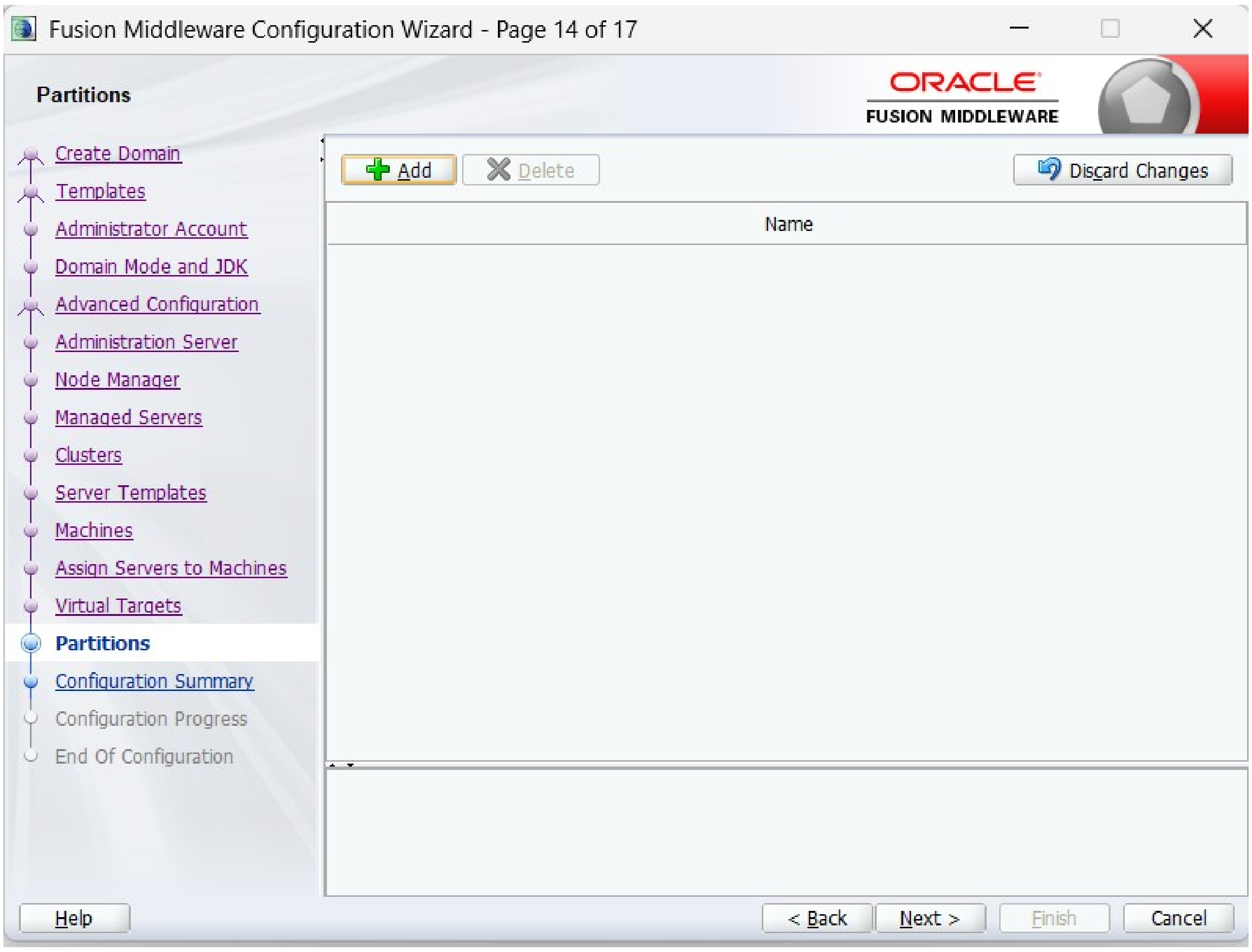
Dont do anything in next screen



Fusion Middleware Configuration Wizard - Page 13 of 17

Virtual Targets

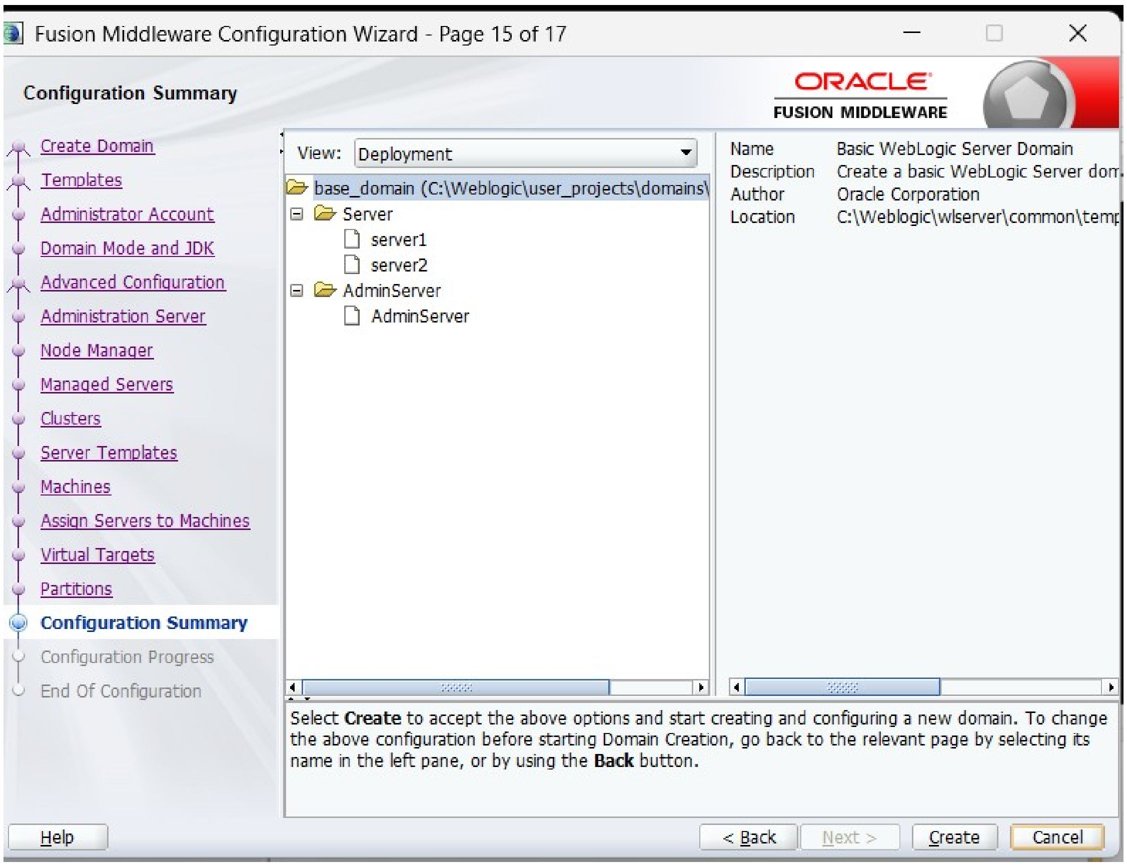
Name	Target	Host Names	URI Prefix	Explicit Port	Port Offset



Fusion Middleware Configuration Wizard - Page 14 of 17

Partitions

Name



Fusion Middleware Configuration Wizard - Page 15 of 17

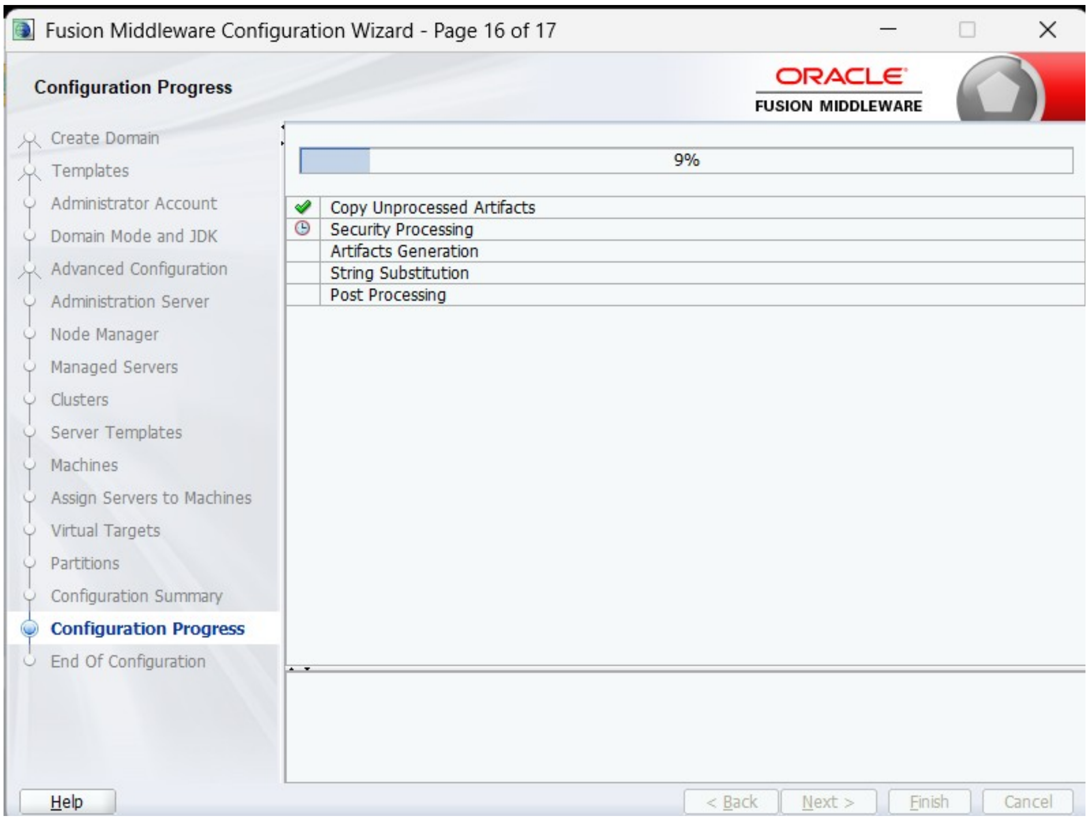
Configuration Summary

View: Deployment

```
base_domain (C:\Weblogic\user_projects\domains\
⊟ Server
    server1
    server2
⊟ AdminServer
    AdminServer
```

Name	Basic WebLogic Server Domain
Description	Create a basic WebLogic Server dom
Author	Oracle Corporation
Location	C:\Weblogic\wlserver\common\temp

Select **Create** to accept the above options and start creating and configuring a new domain. To change the above configuration before starting Domain Creation, go back to the relevant page by selecting its name in the left pane, or by using the **Back** button.

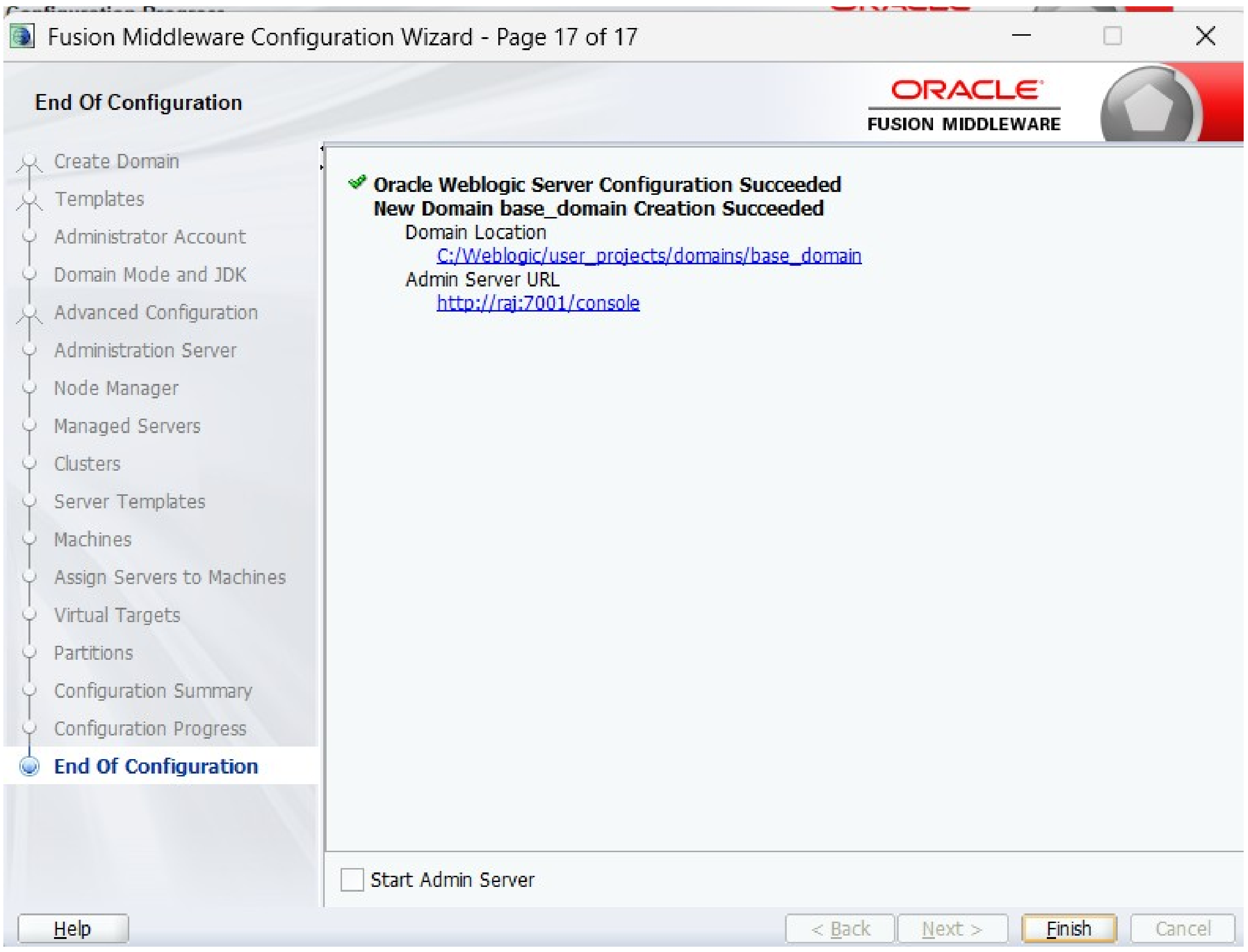


Fusion Middleware Configuration Wizard - Page 16 of 17

Configuration Progress

9%

✓	Copy Unprocessed Artifacts
◷	Security Processing
	Artifacts Generation
	String Substitution
	Post Processing

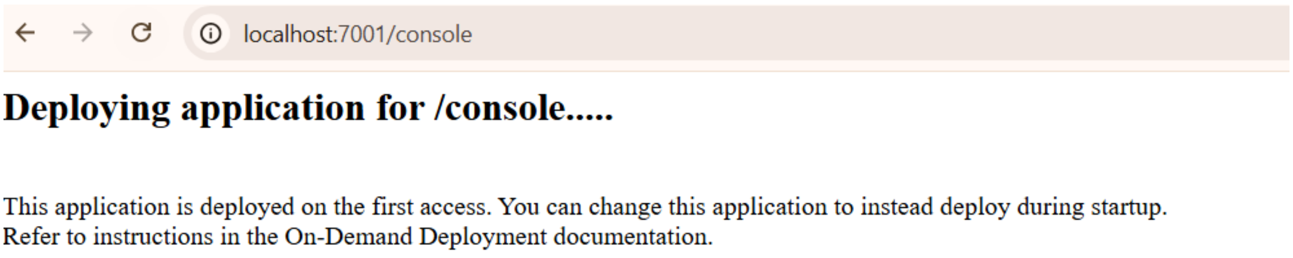


Oracle Weblogic Server Configuration Succeeded
New Domain base_domain Creation Succeeded
Domain Location
C:/Weblogic/user_projects/domains/base_domain
Admin Server URL
http://raj:7001/console

☐ Start Admin Server

Start and access webLogic12c server administration console

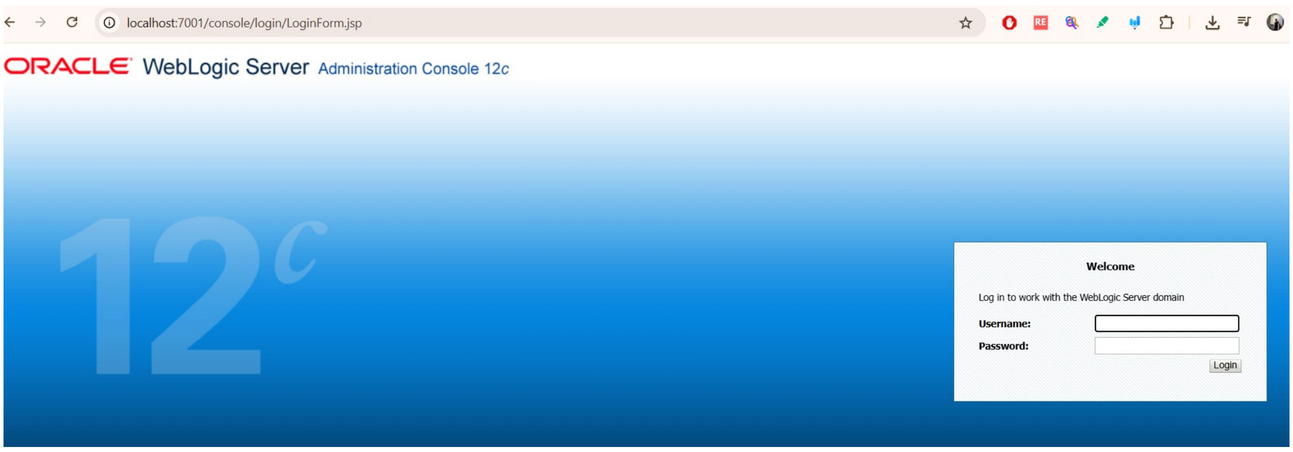
Go to the location
C:\Weblogic\user_projects\domains\base_domain
and start startWebLogic.cmd


localhost:7001/console

Deploying application for /console.....

This application is deployed on the first access. You can change this application to instead deploy during startup.
Refer to instructions in the On-Demand Deployment documentation.

After a while we will get login page, provide username : weblogic and password what you provided earlier during installation


localhost:7001/console/login/LoginForm.jsp

ORACLE WebLogic Server Administration Console 12c

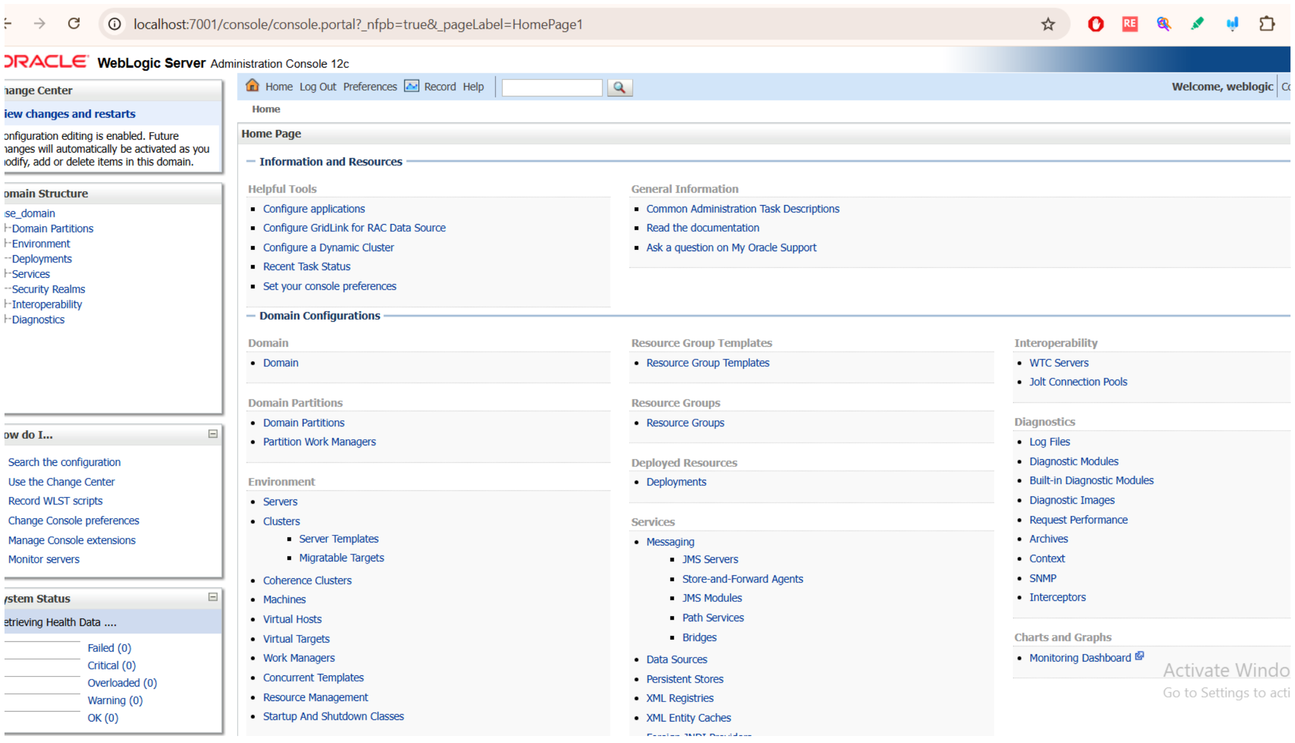
Welcome

Log in to work with the WebLogic Server domain

Username:

Password:

Finally we get logged in and get home page of weblogic server


localhost:7001/console/console.portal?_nfpb=true&_pageLabel=HomePage1

ORACLE WebLogic Server Administration Console 12c

Home Log Out Preferences Record Help

Welcome, weblogic

hange Center

iew changes and restarts

onfiguration editing is enabled. Future
hanges will automatically be activated as you
odify, add or delete items in this domain.

omain Structure

se_domain
- Domain Partitions
- Environment
- Deployments
- Services
- Security Realms
- Interoperability
- Diagnostics

ow do I...

- Search the configuration
- Use the Change Center
- Record WLST scripts
- Change Console preferences
- Manage Console extensions
- Monitor servers

ystem Status

etrieving Health Data ....

- Failed (0)
- Critical (0)
- Overloaded (0)
- Warning (0)
- OK (0)

Home

Home Page

Information and Resources

Helpful Tools
- Configure applications
- Configure GridLink for RAC Data Source
- Configure a Dynamic Cluster
- Recent Task Status
- Set your console preferences

General Information
- Common Administration Task Descriptions
- Read the documentation
- Ask a question on My Oracle Support

Domain Configurations

Domain
- Domain

Domain Partitions
- Domain Partitions
- Partition Work Managers

Environment
- Servers
- Clusters
  - Server Templates
  - Migratable Targets
- Coherence Clusters
- Machines
- Virtual Hosts
- Virtual Targets
- Work Managers
- Concurrent Templates
- Resource Management
- Startup And Shutdown Classes

Resource Group Templates
- Resource Group Templates

Resource Groups
- Resource Groups

Deployed Resources
- Deployments

Services
- Messaging
  - JMS Servers
  - Store-and-Forward Agents
  - JMS Modules
  - Path Services
  - Bridges
- Data Sources
- Persistent Stores
- XML Registries
- XML Entity Caches

Interoperability
- WTC Servers
- Jolt Connection Pools

Diagnostics
- Log Files
- Diagnostic Modules
- Built-in Diagnostic Modules
- Diagnostic Images
- Request Performance
- Archives
- Context
- SNMP
- Interceptors

Charts and Graphs
- Monitoring Dashboard

Activate Windo
Go to Settings to acti

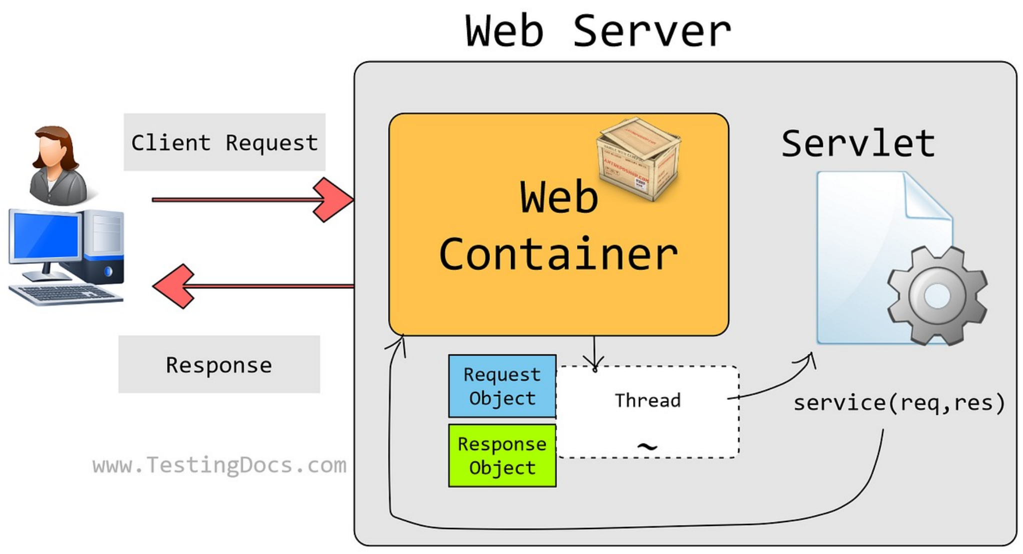
How to configure Oracle Weblogic12c server with Eclipse

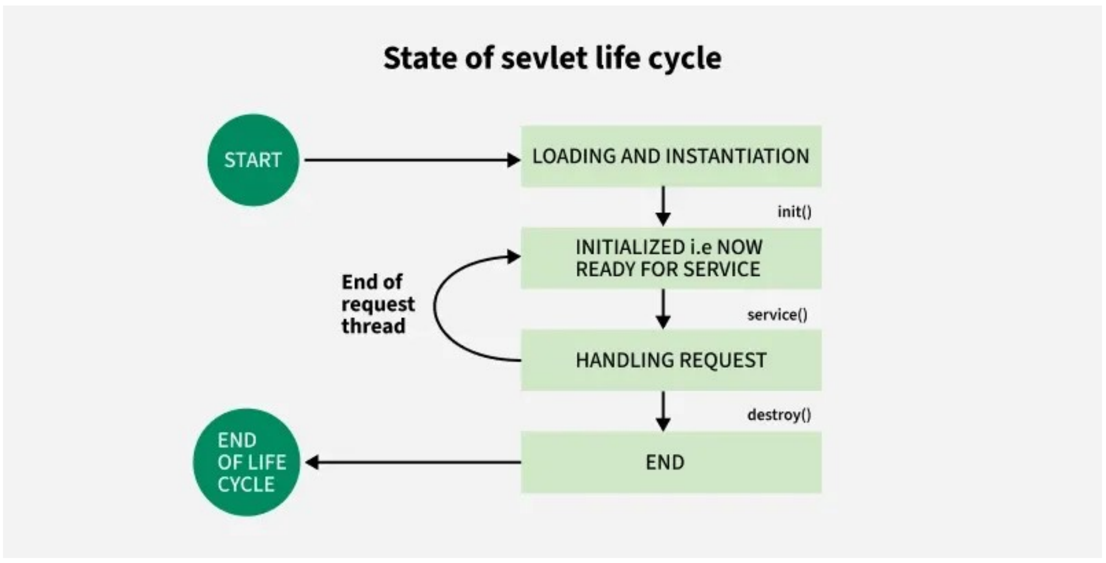
🔷 Brief Introduction to Servlet API (Java Servlet API)

The **Servlet API** is a core part of the **Java EE (Jakarta EE)** platform that allows developers to build **dynamic web applications** by handling **HTTP requests and responses** on the server side.

🔹 What is a Servlet?

A **Servlet** is a Java class that runs in a **servlet container** (like Tomcat) and responds to requests from web clients, typically browsers.




State of sevlet life cycle
START
LOADING AND INSTANTIATION
init()
INITIALIZED i.e NOW READY FOR SERVICE
service()
End of request thread
HANDLING REQUEST
destroy()
END
END OF LIFE CYCLE

◆ Key Features

- Platform-independent, component-based, server-side technology
- Handles HTTP methods like **GET**, **POST**, **PUT**, **DELETE**
- Lifecycle managed by the **servlet container**

◆ Servlet Lifecycle Methods

Defined in the `javax.servlet.Servlet` interface:

Method	Purpose
`init()`	Called once when the servlet is initialized

Method	Purpose
`service()`	Called for every request
`destroy()`	Called once before servlet is removed

🔹 Common Classes and Interfaces

- `HttpServlet` – Base class for handling HTTP requests
- `HttpServletRequest` – Represents the client's request
- `HttpServletResponse` – Represents the response sent back
- `ServletConfig`, `ServletContext` – For configuration and context info

🔹 Typical Workflow

1. Client sends a request (e.g., from a browser)
2. Servlet container forwards request to the servlet
3. Servlet processes request using `doGet()` or `doPost()`
4. Servlet sends response (HTML, JSON, etc.)

🔹 Basic Servlet Example

```
java
CopyEdit
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws IOException {
        response.getWriter().println("Hello, Servlet!");
    }
}
```

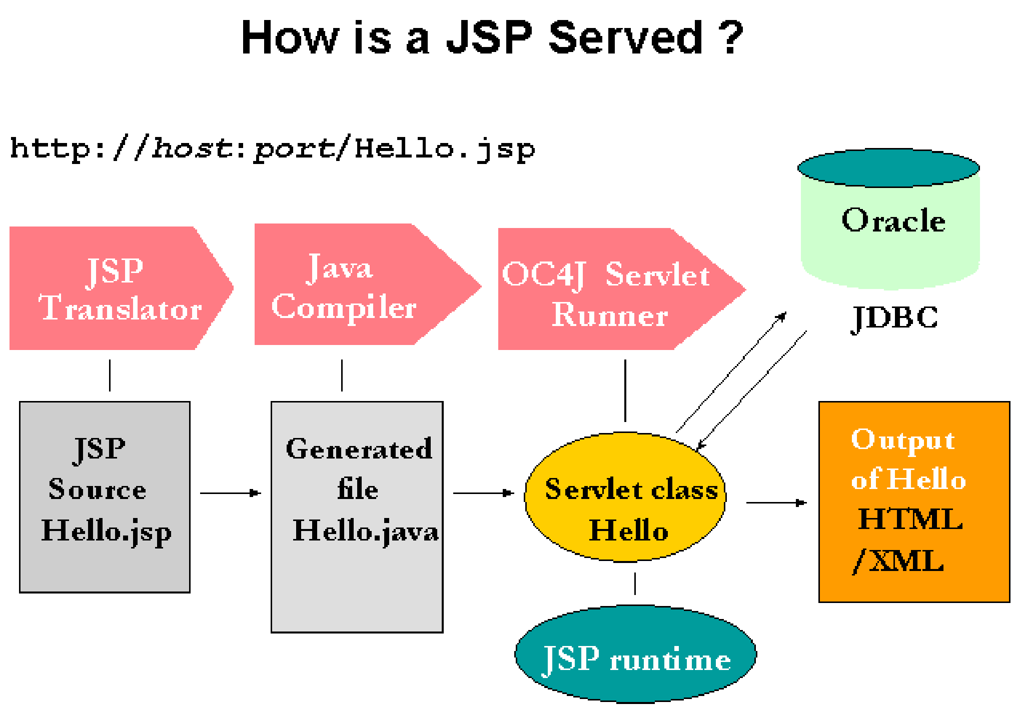
🔷 Brief Introduction to JSP (JavaServer Pages)

JavaServer Pages (JSP) is a **server-side technology** in the **Java EE (Jakarta EE)** ecosystem used to create **dynamic web content** using **HTML + Java**.

🔹 What is JSP?

JSP allows embedding **Java code** directly into **HTML pages**. It is ideal for building the **view layer** of web applications.

> Think of JSP as a way to mix HTML and Java to produce dynamic web pages.



🔹 Key Features

- Simplifies creation of dynamic pages
- Automatically compiled into **Servlets** by the container
- Supports custom tags, JSP directives, and expression language (EL)
- Can access JavaBeans, session, request, and application objects

---

🔹 JSP Lifecycle (Internally a Servlet)

1. **Translation**: JSP → Servlet
2. **Compilation**: Compiled into a `.class` file
3. **Initialization**: Servlet initialized
4. **Execution**: `service()` method processes requests
5. **Destruction**: Servlet destroyed

🔹 Common JSP Tags

Tag	Description
`<% %>`	Scriptlet: Java code
`<%= %>`	Expression: Output result
`<%! %>`	Declaration: Define methods or variables
`<%@ %>`	Directive: E.g., include files or page settings

🔹 Simple JSP Example

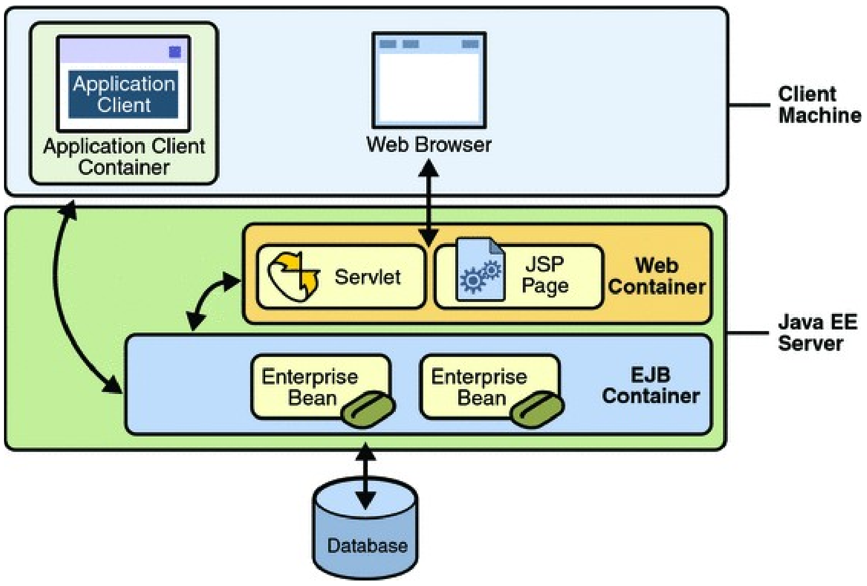
```
jsp
CopyEdit
<%@ page language="java" contentType="text/html" %>
<html>
<head><title>Hello JSP</title></head>
<body>
<%
    String name = "Rajeev";
%>
<h2>Hello, <%= name %>!</h2>
</body>
</html>
```

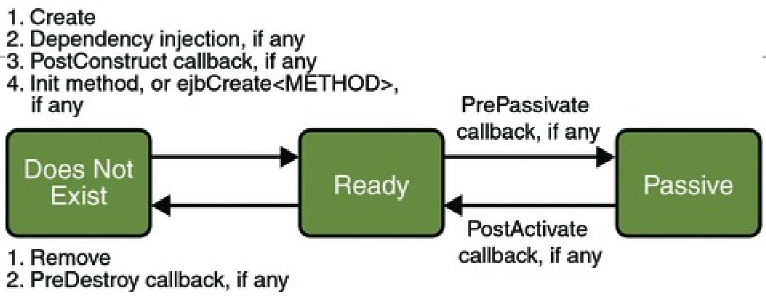
🔹 JSP vs Servlet

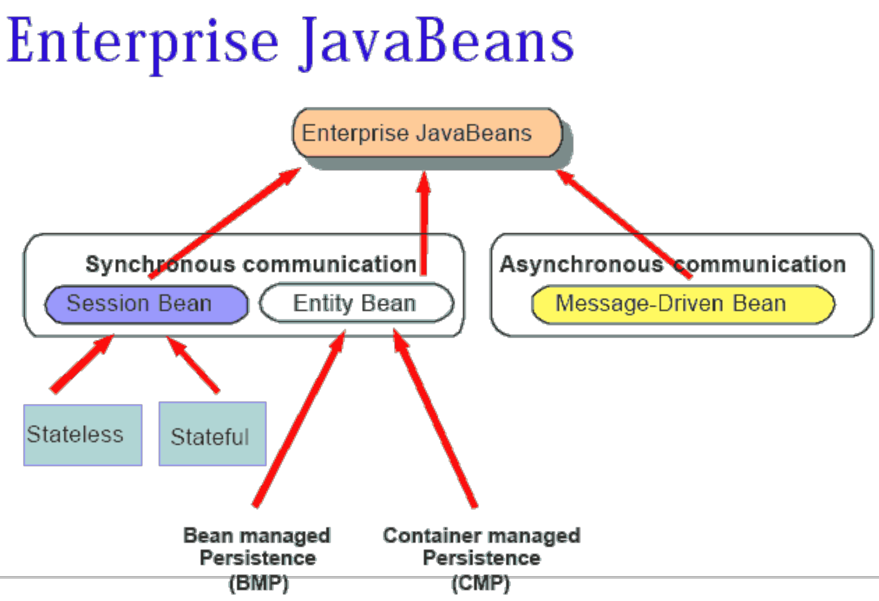
Feature	JSP	Servlet
Focus	View (UI)	Logic/Controller
Code Type	HTML with Java	Pure Java
Easier for	Designers	Developers

🔷 Brief Introduction to EJB 3 (Enterprise JavaBeans 3)

EJB 3 is a part of the **Java EE (Jakarta EE)** platform used for building **scalable, distributed, and transactional** enterprise applications.
It simplifies enterprise development with **POJO-style programming** and **annotations** (removing the heavy XML configs from earlier versions).




1. Create
2. Dependency injection, if any
3. PostConstruct callback, if any
4. Init method, or ejbCreate<METHOD>, if any
PrePassivate callback, if any
Does Not Exist
Ready
Passive
PostActivate callback, if any
1. Remove
2. PreDestroy callback, if any


Enterprise JavaBeans
Enterprise JavaBeans
Synchronous communication
Session Bean
Entity Bean
Asynchronous communication
Message-Driven Bean
Stateless
Stateful
Bean managed Persistence (BMP)
Container managed Persistence (CMP)

🔹 Key Features of EJB 3

- Declarative transactions
- Dependency injection
- Security, concurrency, and persistence support
- Remotely accessible business logic

🔹 Types of EJBs

Type	Purpose	Example Use Case
Stateless	No client-specific state	Login service
Stateful	Maintains state for one client	Shopping cart
Singleton	One shared instance for all clients	App config cache
Message-Driven	Asynchronous processing via JMS	Email processor

---

🔹 Very Simple Code Snippets

✅ 1. Stateless EJB

```
java
CopyEdit
import javax.ejb.Stateless;

@Stateless
public class CalculatorBean {
    public int add(int a, int b) {
        return a + b;
    }
}
```

✅ 2. Stateful EJB

```
java
CopyEdit
import javax.ejb.Stateful;

@Stateful
public class ShoppingCartBean {
    private List<String> items = new ArrayList<>();

    public void addItem(String item) {
        items.add(item);
    }

    public List<String> getItems() {
        return items;
    }
}
```

✅ 3. Singleton EJB

```
java
CopyEdit
import javax.ejb.Singleton;

@Singleton
public class ConfigBean {
    private String appVersion = "1.0.0";

    public String getAppVersion() {
        return appVersion;
    }
}
```

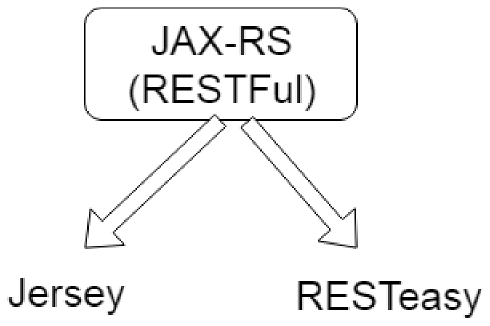
✅ 4. Message-Driven Bean (MDB)

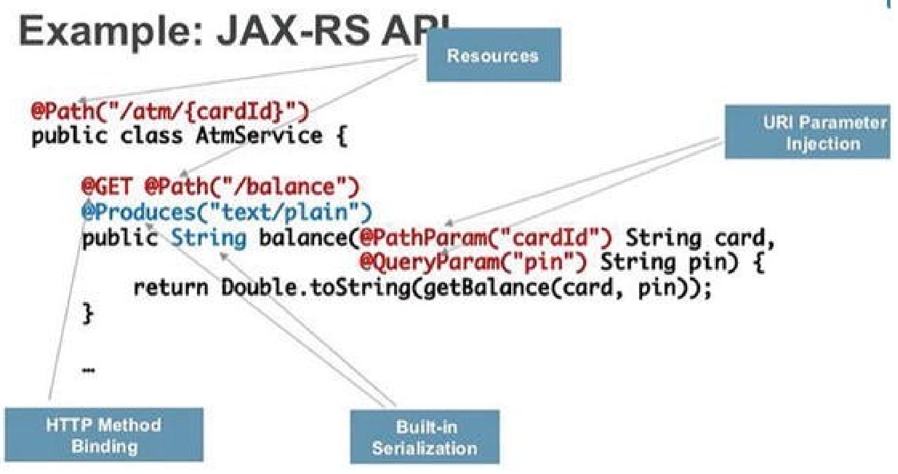
```
java
CopyEdit
import javax.ejb.MessageDriven;
import javax.jms.Message;
import javax.jms.MessageListener;

@MessageDriven
public class EmailListenerBean implements MessageListener {
    public void onMessage(Message message) {
        // Process message (e.g., send email)
    }
}
```

🔷 Brief Introduction to JAX-RS (Java API for RESTful Web Services)

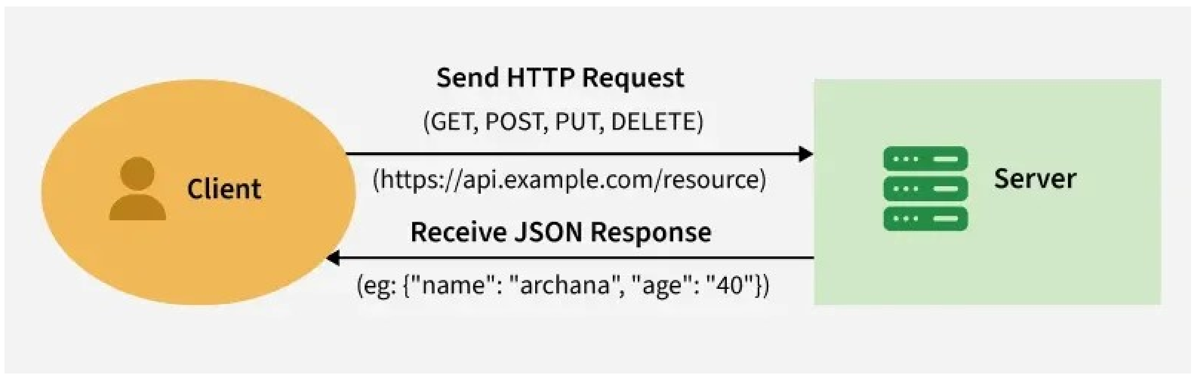
JAX-RS is a Java EE (now Jakarta EE) specification that simplifies the development of **RESTful web services** using **Java annotations**.

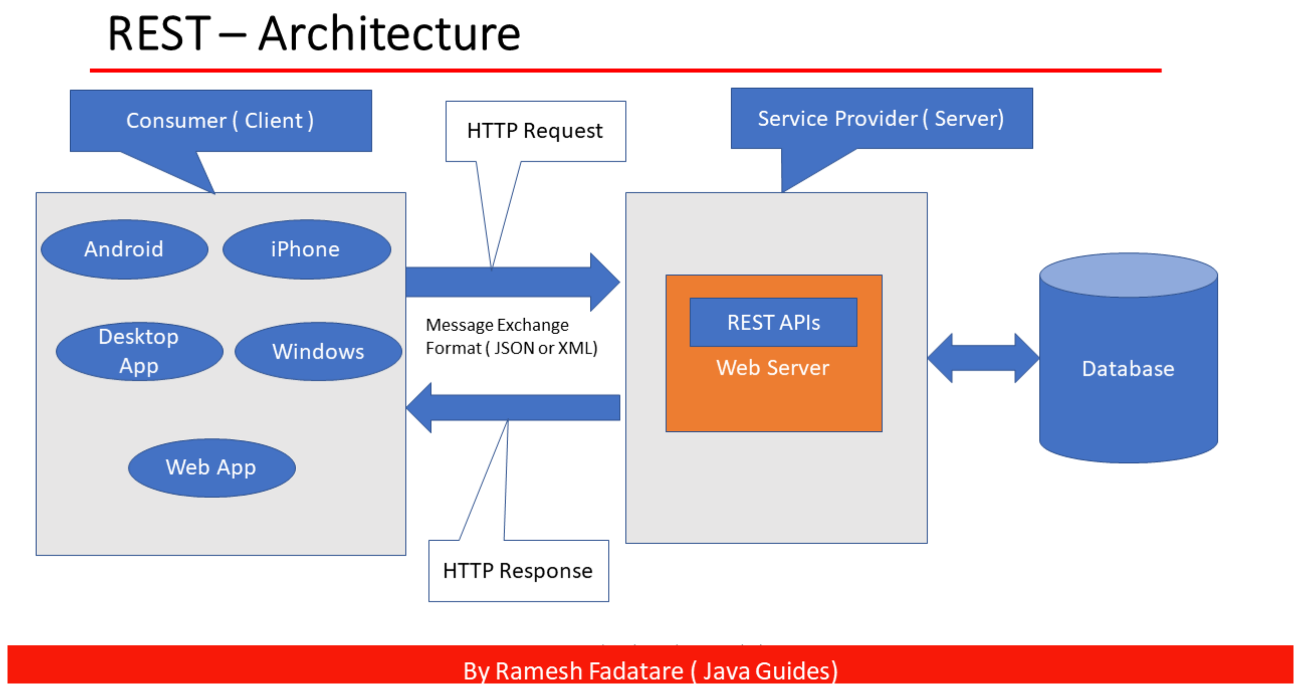
It allows developers to expose Java classes and methods as HTTP resources (GET, POST, PUT, DELETE), enabling client-server communication over HTTP using standard REST principles.





```
@Path("/atm/{cardId}")
public class AtmService {

    @GET @Path("/balance")
    @Produces("text/plain")
    public String balance(@PathParam("cardId") String card,
                          @QueryParam("pin") String pin) {
        return Double.toString(getBalance(card, pin));
    }

    …
```





Key Features of JAX-RS

- Annotation-based programming model
- Maps HTTP methods to Java methods
- Supports JSON and XML
- Can be easily integrated with frameworks like Jersey, RESTEasy

---

Common JAX-RS Annotations

Annotation	Purpose
`@Path`	Defines the URI path of the resource
`@GET` / `@POST`	Maps to HTTP GET/POST method
`@PUT` / `@DELETE`	Maps to PUT/DELETE
`@Produces`	Specifies response format (JSON/XML)
`@Consumes`	Specifies accepted request format
`@PathParam`	Extracts values from URI path
`@QueryParam`	Extracts query parameters

Simple JAX-RS Example

```
java
CopyEdit
import javax.ws.rs.*;
```

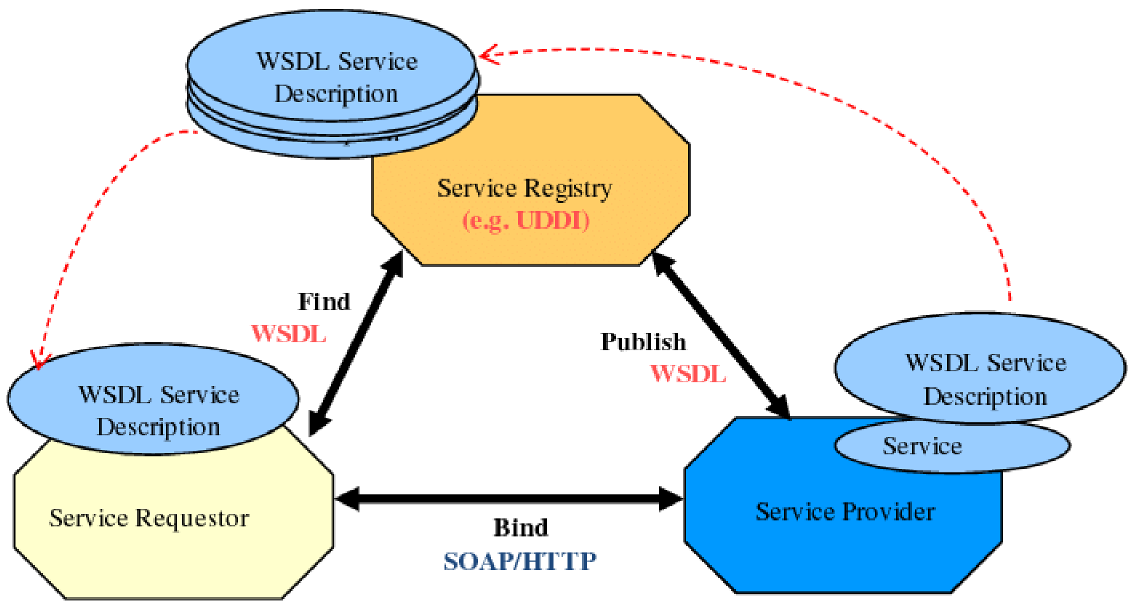
```
@Path("/hello")
public class HelloResource {

    @GET
    @Produces("text/plain")
    public String sayHello() {
        return "Hello, JAX-RS!";
    }

    @GET
    @Path("/{name}")
    @Produces("text/plain")
    public String greetUser(@PathParam("name") String name) {
        return "Hello, " + name + "!";
    }
}
```

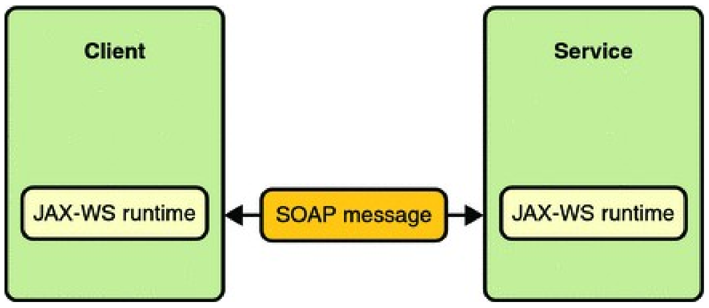
◆ How it Works

- A web server or application server (like Tomcat or GlassFish) hosts the JAX-RS app.
- Clients (browsers, mobile apps) send HTTP requests.
- JAX-RS maps those requests to appropriate Java methods.

🔷 Brief Introduction to JAX-WS (Java API for XML Web Services)

JAX-WS is a Java EE (Jakarta EE) API used to develop **SOAP-based web services**.
It enables communication between distributed applications using **XML messages** over **HTTP** (or other protocols).





🔹 Key Features of JAX-WS

- Based on **SOAP** (Simple Object Access Protocol)
- Uses **WSDL** (Web Services Description Language) to describe services
- Supports **RPC** and **document-style** messaging
- Can generate client and server code using `wsimport` and `wsgen`

🔹 Common Annotations

Annotation	Purpose
`@WebService`	Marks a class as a web service
`@WebMethod`	Marks a method to be exposed
`@WebParam`	Binds a parameter to the SOAP message
`@WebResult`	Binds return value to the SOAP response
`@SOAPBinding`	Specifies SOAP style and use

🔹 Simple JAX-WS Example

✅ Web Service Class

```
java
CopyEdit
import javax.jws.WebService;
import javax.jws.WebMethod;

@WebService
public class HelloService {

    @WebMethod
    public String sayHello(String name) {
        return "Hello, " + name + "!";
    }
}
```

✅ Publishing the Service (Standalone)

```
java
CopyEdit
import javax.xml.ws.Endpoint;

public class Publisher {
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8080/hello", new HelloService());
    }
}
```

🔹 How it Works

- The service is described via **WSDL**.
- A client can generate proxy classes using `wsimport`.
- Communication occurs over **SOAP** using **XML messages**.

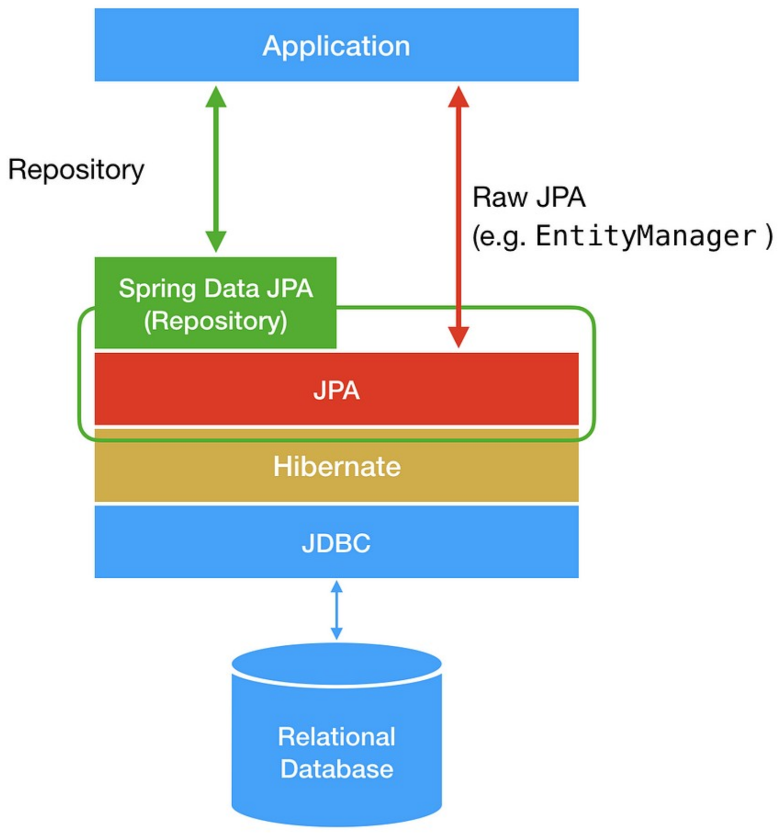
🔹 JAX-WS vs JAX-RS

Feature	JAX-WS	JAX-RS
Protocol	SOAP	REST/HTTP
Format	XML (SOAP)	JSON, XML, Plain Text
Use Case	Enterprise integration, legacy systems	Lightweight web APIs

Feature	**JAX-WS**	**JAX-RS**
WSDL	Required	Not required

🔷 Brief Introduction to JPA (Java Persistence API)

JPA (Java Persistence API) is a Java EE (now Jakarta EE) specification for **object-relational mapping (ORM)**.
It allows Java developers to **map Java objects (entities) to relational database tables** and manage persistent data using standard APIs.



🔹 Key Features of JPA

- Maps Java classes to database tables using annotations
- Handles CRUD operations through **EntityManager**
- Supports **JPQL (Java Persistence Query Language)** for querying
- Enables relationships: One-to-One, One-to-Many, etc.

🔹 Core JPA Annotations

Annotation	Purpose
`@Entity`	Marks a class as a persistent entity
`@Id`	Specifies the primary key
`@GeneratedValue`	Auto-generates ID values
`@Column`	Maps a field to a DB column
`@Table`	Maps class to a specific DB table
`@ManyToOne`, `@OneToMany`	Define relationships

🔹 Simple JPA Example

✅ Entity Class

```
import javax.persistence.*;

@Entity
@Table(name = "students")
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    // Getters and setters
}
```

✅ Persisting Data with EntityManager

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("my-pu");
EntityManager em = emf.createEntityManager();

Student student = new Student();
student.setName("Rajeev");

em.getTransaction().begin();
em.persist(student);
em.getTransaction().commit();
em.close();
emf.close();
```

🔹 JPQL Query Example

```
java
CopyEdit
List<Student> list = em.createQuery("SELECT s FROM Student s", Student.class)
                       .getResultList();
```

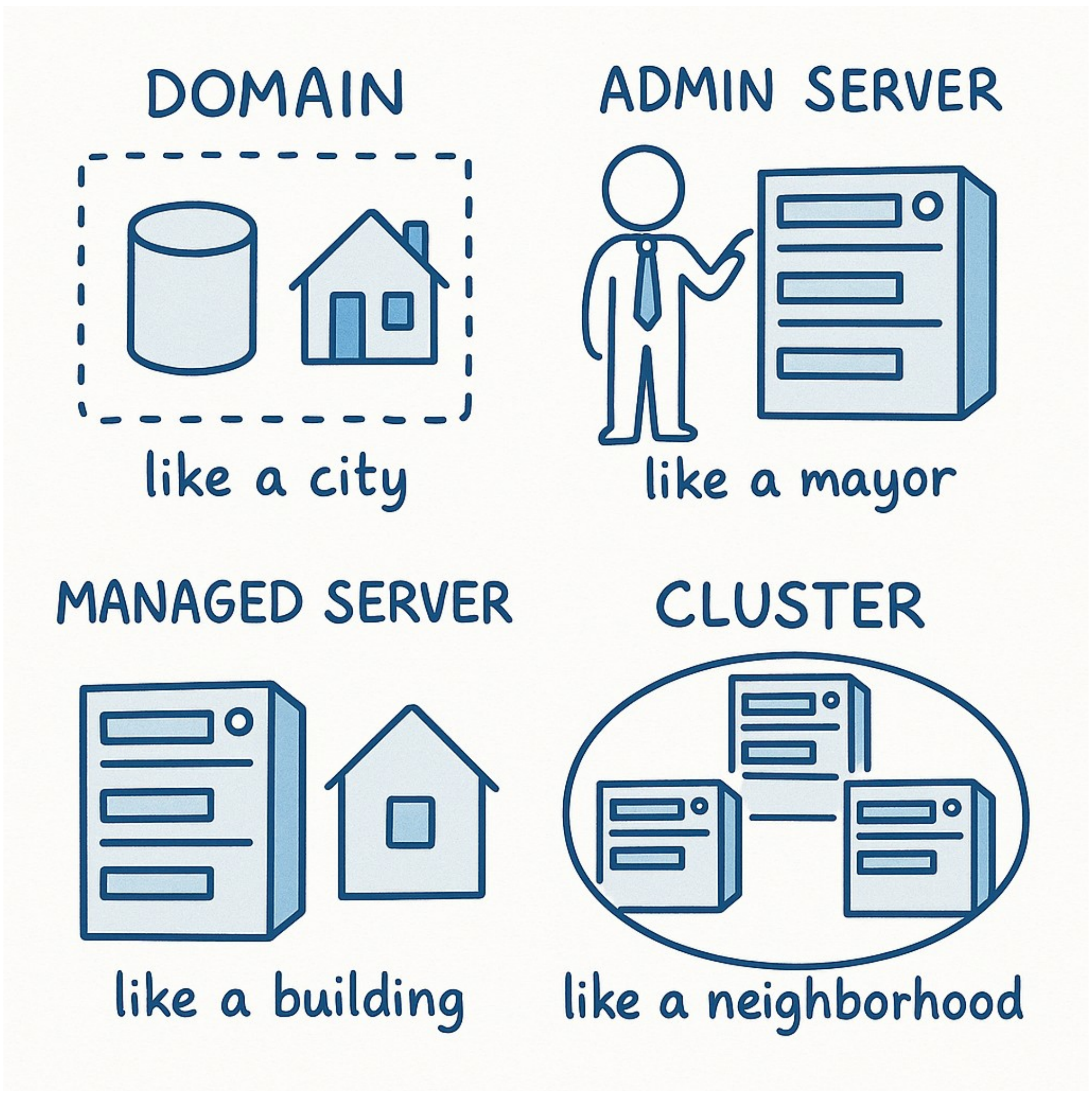
Introduction to WebLogic Server

- **Owned by**: Oracle Corporation
- **Type**: Java EE Application Server (supports Servlet, JSP, EJB, JMS, JPA, etc.)
- **Use Case**: Enterprise deployment platform for Java-based applications.

WebLogic Architecture Overview

- **Domain**: A logically related group of WebLogic Server resources.

- **Admin Server**: Central control entity of a domain (only one per domain).
- **Managed Server**: Hosts actual apps and resources.
- **Node Manager**: Utility to control server instances remotely.
- **Cluster**: Group of managed servers for load balancing & failover.
- **Machine**: A logical representation of the physical computer (used with Node Manager).



🔷 1. Domain

💡 Real-life Analogy: A Housing Society

Imagine a **gated housing society**:

- All buildings, security, gardens, electricity, and residents are part of **one society**.
- It is managed as **one logical unit**.

🔍 In WebLogic:

- A **Domain** is like that society.
- It contains everything: **Admin Server, Managed Servers, configurations, deployments**, etc.
- It's the **highest-level structure** in WebLogic.

🔷 2. Admin Server

💡 Real-life Analogy: Society Office / RWA President

In your housing society:

- The **society office** (or Resident Welfare Association head) keeps **full control**.
- Handles all administration: security, maintenance, new resident entries, etc.

🔍 In WebLogic:

- The **Admin Server** is the **control center** of the domain.
- It manages configurations, deployments, and controls all Managed Servers.
- There's always **only one Admin Server** in a domain.

🔷 3. Managed Server

💡 Real-life Analogy: Individual Residential Buildings

Each building in the society:

- Has its own rooms (flats) where people live.
- Some have gyms, some have shops.

🔍 In WebLogic:

- A **Managed Server** is where **your actual applications run** (like websites, APIs).
- You can have **many managed servers**.
- Admin Server **controls them**, but does **not run applications** itself (in most cases).

🔷 4. Node Manager

💡 Real-life Analogy: Remote Watchman with Master Keys

Say each building has a **watchman** who can **remotely open or close** any building or flat:

- Can start/stop electricity remotely.
- Keeps track of status and reports to society office.

🔍 In WebLogic:

- **Node Manager** is a utility that **starts/stops Admin or Managed Servers** remotely.
- Helps in **automated restarts**, **health monitoring**, etc.
- Installed on each **machine** (server box) to control servers on that machine.

🔷 5. Cluster

💡 Real-life Analogy: Multiple Buildings with Same Facilities

Imagine 3 buildings in the same society:

- All have gyms and shops.
- If gym in building A is full, you go to B or C.

🔍 In WebLogic:

- A **Cluster** is a **group of Managed Servers** doing the same job.
- Used for **Load Balancing** (spread users across servers).
- And for **Failover** (if one fails, others take over).

Example: If 10,000 users hit your website, the load is split across all servers in the cluster.

🔷 6. Machine

💡 Real-life Analogy: Physical Plots in the Society

Every building (Managed Server) stands on some **land/plot**.
That plot is a physical entity.

🔍 In WebLogic:

- A **Machine** represents the **physical or virtual hardware**.
- Used mostly with **Node Manager** to say: “This server runs on this machine.”

✅ Summary Table for Quick Revision

WebLogic Term	Real-Life Analogy	Role in WebLogic
Domain	Housing Society	Logical group of all WebLogic resources
Admin Server	Society Office / President	Central controller of the domain

WebLogic Term	Real-Life Analogy	Role in WebLogic
Managed Server	Individual Residential Buildings	Hosts and runs the real applications
Node Manager	Remote Watchman with Master Keys	Starts/stops servers remotely
Cluster	Similar Buildings with Same Facilities	Group of managed servers for load balancing/failover
Machine	Physical Plot or Land	Represents hardware where server runs

---

```
==================== WebLogic Architecture ====================
                    +------------------+
                    |   DOMAIN        |  <-- Like a Housing Society
                    |------------------|
                    |  Admin Server    |  <-- Like Society Office / President
                    |  (Control Panel) |
                    +--------+---------+
                             |
          -------------------------------------------
         |                                |
   +-------------------+          +-------------------+
   | Managed Server 1  |  <-- Like Building 1  |  [Runs App A]  |
   +-------------------+          +-------------------+
         |                                |
   +-------------------+          +-------------------+
   | Managed Server 2  |  <-- Like Building 2  |  [Runs App B]  |
   +-------------------+          +-------------------+
         |                                |
   +-------------------+          +-------------------+
   | Managed Server 3  |  <-- Like Building 3  |  [Runs App A]  |
   +-------------------+          +-------------------+

                    CLUSTER
         (Managed Servers grouped together for
           Load Balancing & High Availability)
------------------------------------------------------------
                   NODE MANAGER
     (Remote Watchman to Start/Stop Servers on Each Machine)

    Machine A -------------> Controls Admin + MS1
    Machine B -------------> Controls MS2
    Machine C -------------> Controls MS3

============================================================
```

WebLogic Administrator

🔷 Introduction

What is Oracle WebLogic Server 14c?

Oracle WebLogic Server 14c is a modern Java EE-compatible application server and the runtime foundation for Oracle Fusion Middleware, supporting enterprise applications, microservices, cloud-native deployments, and more.

Editions in 14c:

- **Standard Edition (SE)**: For developers and small-scale deployments.
- **Enterprise Edition (EE)**: SE + clustering + advanced diagnostics.
- **WebLogic Suite**: EE + Coherence + TopLink + enhanced scalability.

🔷 Key Concepts in WebLogic 14c

🔶 WebLogic Server Instance

- A **JVM process** that hosts your Java EE applications.
- Two types:
  - **Admin Server**: Central controller.
  - **Managed Server**: Runs business applications.

🔶 WebLogic Domain

- A **logical container** that includes:
  - 1 Admin Server
  - N Managed Servers
  - Services (JDBC, JMS, etc.)

  - Machines and optional clusters

🔸 **Admin Server**

- Special WebLogic instance for **configuration and monitoring**.
- Hosts the **Admin Console** (web-based GUI at `http://localhost:7001/console`).
- Should **not host applications in production**.

🔸 **Managed Server**

- Hosts deployed applications and their related resources (JDBC, JTA, JMS).
- Contacts Admin Server on startup to sync config.
- Can run **independently** after starting (MSI mode).

🔸 **Cluster**

- A **group of Managed Servers** with:
  - Same app deployed
  - Shared configuration
- Enables **load balancing** and **failover**.
- Often used with a **hardware or software load balancer**.

🔸 **Node Manager**

- A separate process used to **remotely start/stop servers**.
- Must be configured per machine.
- Used in combination with Admin Console or WLST.
- Two types: **Java-based** (cross-platform) and **Script-based** (Unix-only).

🔸 **Machine**

- A **logical representation of a host machine** (physical/VM).
- Required for using Node Manager and for clustering configuration.