

# PRÁCTICA 2

**SEBASTIÁN  
RODRIGUEZ VENTEO**

## Creamos la imagen con Payara Server

```
D: > DespliegueDeAplicaciones > practica2 > Dockerfile > ...
1 FROM openjdk:11
2 ENV USUARIO="srv"
3 ENV ADMIN="admin"
4 ENV PASSWORD="srv"
5 ENV INSTALL_PATH="/home/srv"
6 RUN useradd -d /home/srv srv
7 RUN echo "srv:srv" | chpasswd
8 RUN apt-get update && apt-get install -y openssh-server
9 RUN apt clean
10 RUN echo mkdir /home/srv/payara5
11 COPY payara5 /home/srv/payara5
12 #configurar el servidor
13 RUN echo 'AS_ADMIN_PASSWORD=\n\
14 AS_ADMIN_NEWPASSWORD='${PASSWORD}'\n\
15 EOF\n'\
16 >> ${INSTALL_PATH}/tmpfile
17 RUN echo 'AS_ADMIN_PASSWORD='${PASSWORD}'\n\
18 EOF\n'\
19 >> ${INSTALL_PATH}/pwdfile
20 RUN ${INSTALL_PATH}/payara5/bin/asadmin start-domain \
21 && ${INSTALL_PATH}/payara5/bin/asadmin --user $ADMIN --passwordfile=${INSTALL_PATH}/tmpfile change-admin-password \
22 && ${INSTALL_PATH}/payara5/bin/asadmin --user $ADMIN --passwordfile=${INSTALL_PATH}/pwdfile enable-secure-admin \
23 && ${INSTALL_PATH}/payara5/bin/asadmin restart-domain
24 #se borran los temporales
25 | RUN rm ${INSTALL_PATH}/pwdfile ${INSTALL_PATH}/tmpfile
26 VOLUME /home/srv
27 RUN /etc/init.d/ssh start
28 EXPOSE 22
29 EXPOSE 4848
30 EXPOSE 8080
31 ENTRYPOINT service ssh restart && bash && /home/srv/payara5/bin/asadmin start-domain
```

Y ejecutamos para ver si es correcto

```
PS D:\DespliegueDeAplicaciones\practica2> docker build --no-cache ./ -t srv_payara
[+] Building 67.3s (18/18) FINISHED
=> [internal] load build definition from Dockerfile 0.1s
=> => transferring dockerfile: 1.13kB 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/openjdk:11 1.8s
=> [auth] library/openjdk:pull token for registry-1.docker.io 0.0s
=> CACHED [ 1/12] FROM docker.io/library/openjdk:11@sha256:99bac5bf83633e3c7399aed725c8415e7b569b54e03e4599e580f 0.0s
=> [internal] load build context 11.0s
=> => transferring context: 474.17kB 10.9s
=> [ 2/12] RUN useradd -d /home/srv srv 0.6s
=> [ 3/12] RUN echo "srv:srv" | chpasswd 0.5s
=> [ 4/12] RUN apt-get update && apt-get install -y openssh-server 12.8s
=> [ 5/12] RUN apt clean 0.5s
=> [ 6/12] RUN echo mkdir /home/srv/payara5 0.6s
=> [ 7/12] COPY payara5 /home/srv/payara5 3.7s
=> [ 8/12] RUN echo 'AS_ADMIN_PASSWORD=\nAS_ADMIN_NEWPASSWORD='srv'\nEOF\n'>> /home/srv/tmpfile 0.4s
=> [ 9/12] RUN echo 'AS_ADMIN_PASSWORD='srv'\nEOF\n'>> /home/srv/pwdfile 0.8s
=> [10/12] RUN /home/srv/payara5/bin/asadmin start-domain && /home/srv/payara5/bin/asadmin --user admin --passw 22.4s
=> [11/12] RUN rm /home/srv/pwdfile /home/srv/tmpfile 0.7s
=> [12/12] RUN /etc/init.d/ssh start 0.5s
=> exporting to image 1.3s
=> => exporting layers 1.2s
=> => writing image sha256:b53035812931fdab5333644bc6cc3d25d7e4b35b9600f3722e9838d443c15d11 0.0s
=> => naming to docker.io/library/srv_payara 0.0s
```

Ahora vamos a crear el contenedor y mapeamos los puertos.

```
PS D:\DespliegueDeAplicaciones\practica2> docker run -d -t -p 8080:8080 -p 2222:22 -p 4848:4848 --name srv_payara srv_payara
65fcee5af18b8a54f0757d6e82c40a1e3acd20acb4f9ac819cb0df56db98acb0
```

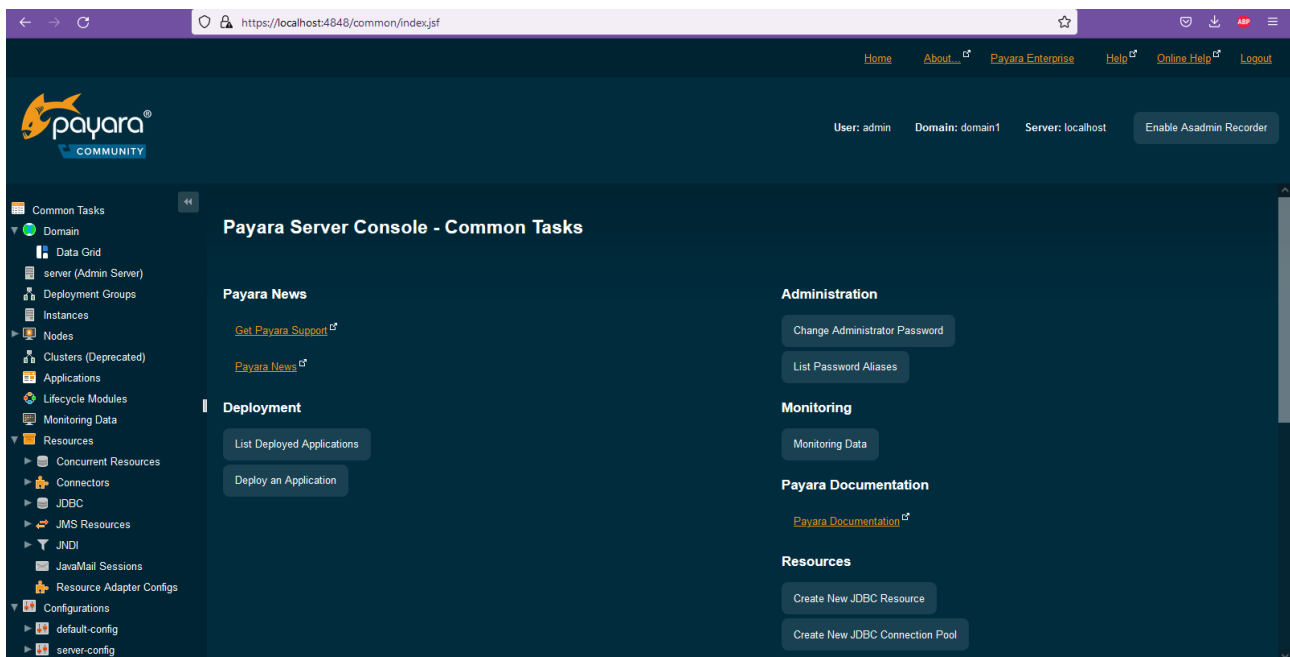
Comprobar si los path son correctos

```
root@98821294ec7a:/home/srv/payara5/bin# ls
asadmin asadmin.bat letsencrypt.py
```

Probar el ssh.

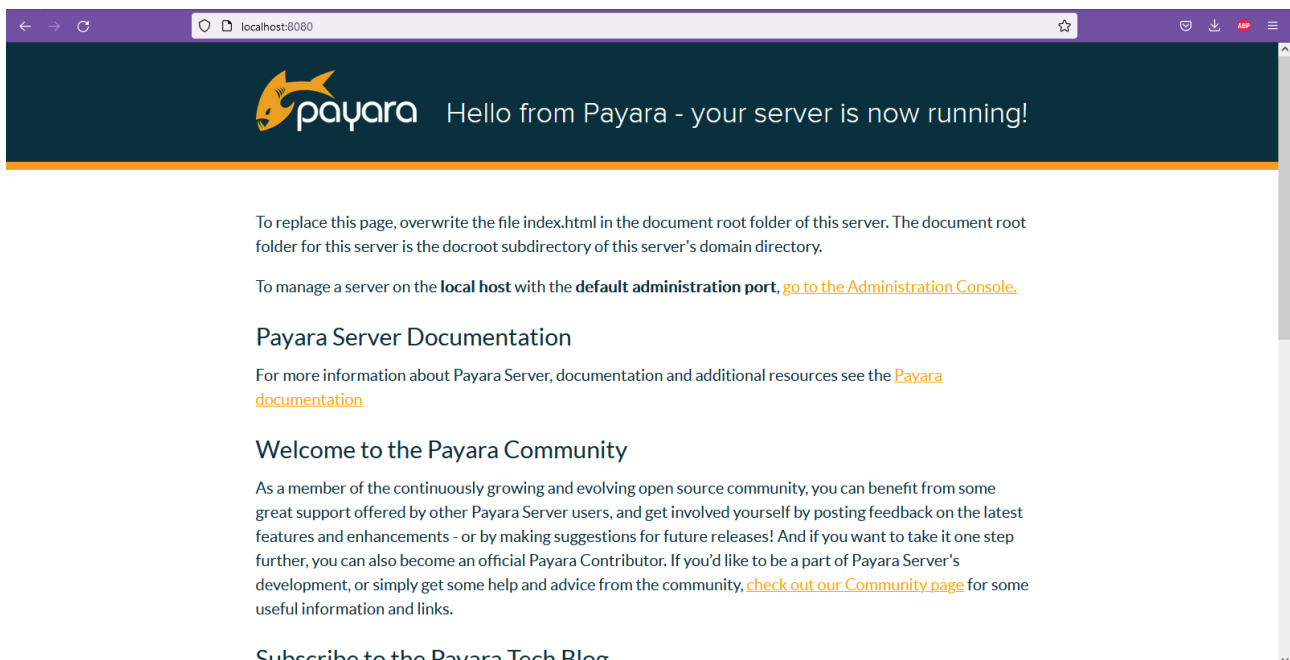
```
root@98821294ec7a:/home/srv/payara5/bin# ssh
usage: ssh [-46AaCfGgKkMNnqsTtVvXxYy] [-B bind_interface]
          [-b bind_address] [-c cipher_spec] [-D [bind_address:]port]
          [-E log_file] [-e escape_char] [-F configfile] [-I pkcs11]
          [-i identity_file] [-J [user@]host[:port]] [-L address]
          [-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p port]
          [-Q query_option] [-R address] [-S ctl_path] [-W host:port]
          [-w local_tun[:remote_tun]] destination [command]
```

Probar el puerto 4848 ¿qué sucede?



(Se abre la ventana de administración)

**Probar el puerto 8080.**



**Ver las aplicaciones instaladas desde el panel de administrador**

Common Tasks

Domain

- Data Grid
- server (Admin Server)
- Deployment Groups
- Instances

Nodes

- Clusters (Deprecated)

Applications

Lifecycle Modules

- Monitoring Data

Resources

- Concurrent Resources
- Connectors
- JDBC
- JMS Resources
- JNDI
- JavaMail Sessions
- Resource Adapter Configs

Configurations

- default-config
- server-config

HomeAboutPayara EnterpriseHelpOnline HelpLogout

User: adminDomain: domain1Server: localhostEnable Asadmin Recorder

Applications

Applications can be enterprise or web applications, or various kinds of modules. Restart an application or module by clicking on the reload link, this action will apply only to the targets that the application or module is enabled on.

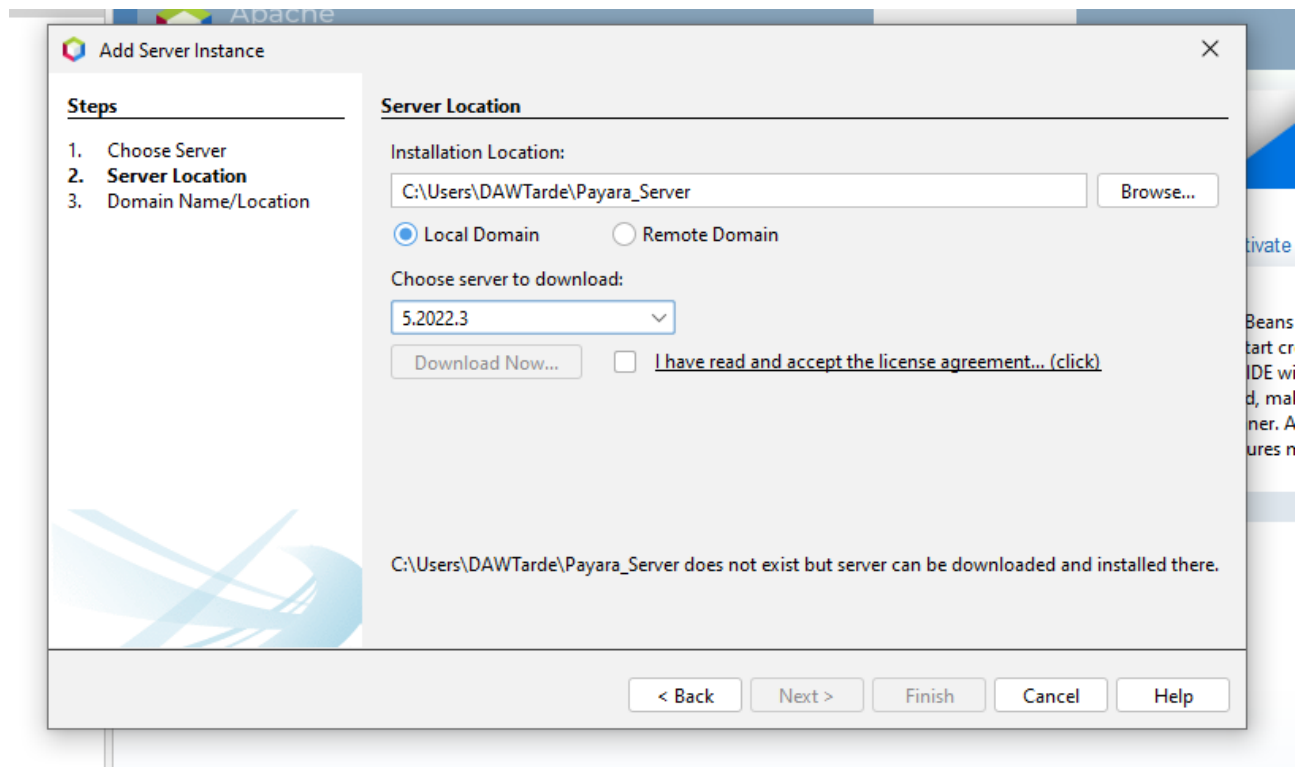
Deployed Applications (0)

Deploy...UndeployEnableDisableFilter:

| Select          | Name | Deployment Order | Time Taken To Deploy (Milliseconds) | Date/Time When Application was deployed | Enabled | Engines | Action |
|-----------------|------|------------------|-------------------------------------|---|---------|---------|--------|
| No items found. |      |                  |                                     |   |         |         |        |

5

## Creando la primera aplicación con Jakarta.



**Siguiente**

**Add Server Instance**

**Steps**

1. Choose Server
2. Server Location
3. **Domain Name/Location**

**Domain Location**

Domain:

Host:

DAS Port:  HTTP Port:  ☒ Default

Target:

User Name:

Password:

☐ Docker Volume

[Note the remote administration needs to be enabled before registering the remote domain.](#)

Remote host: localhost DAS: 4848 HTTP: 8080

< Back Next > **Finish** Cancel Help

**Y ya tenemos nuestro servidor vinculado al NetBeans**

Ahora hacemos un proyecto en Maven de tipo Web Application con nuestras iniciales

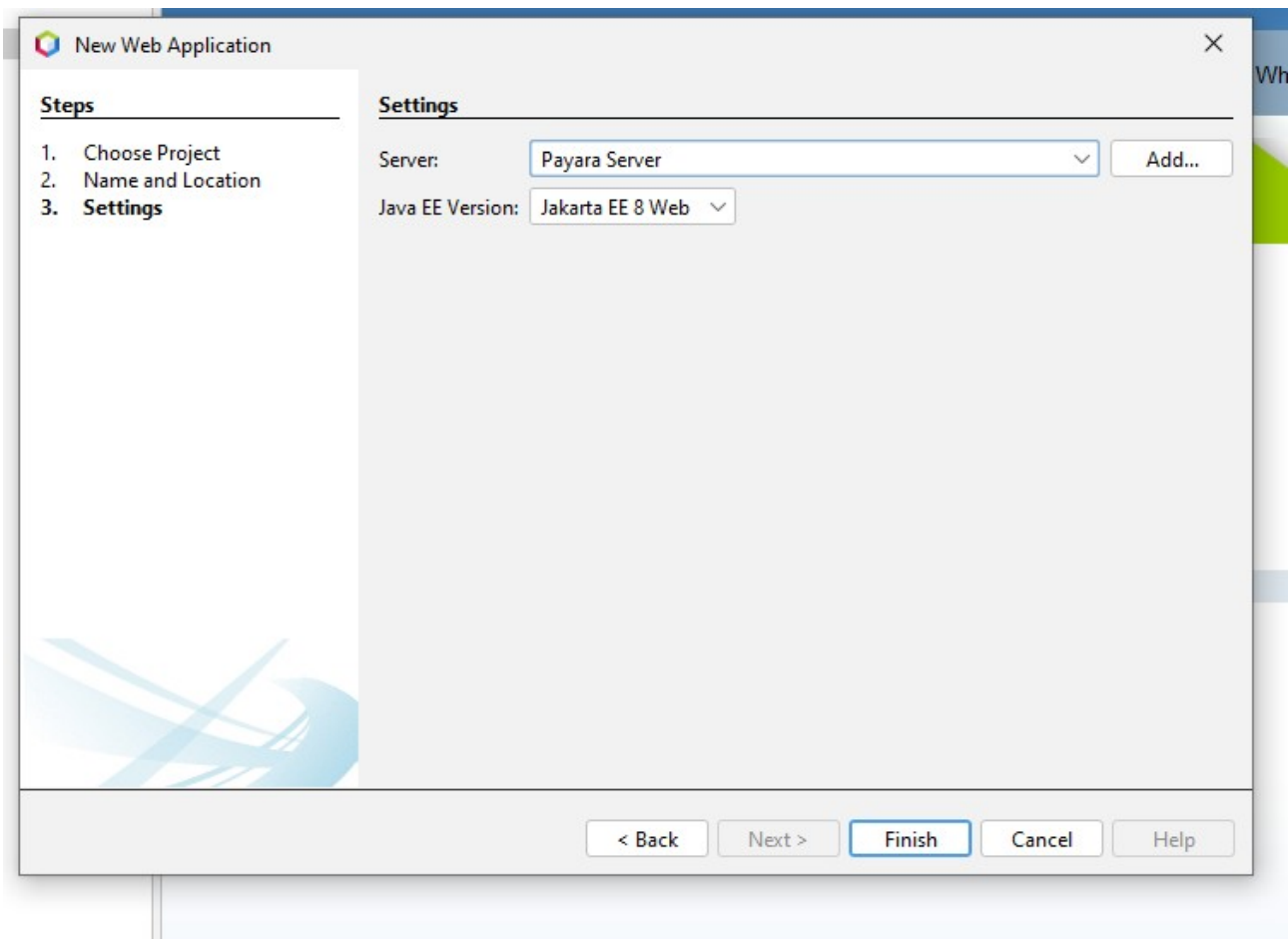
The screenshot shows the 'New Web Application' dialog box in NetBeans IDE. The 'Steps' panel on the left indicates the current step is '2. Name and Location'. The 'Name and Location' tab is selected, displaying the following fields:

- Project Name: Practica2srv
- Project Location: C:\Users\DAWTarde\Documents\NetBeansProjects (with a 'Browse...' button)
- Project Folder: ers\DAWTarde\Documents\NetBeansProjects\Practica2srv
- Artifact Id: Practica2srv
- Group Id: com.mycompany
- Version: 1.0-SNAPSHOT
- Package: com.mycompany.practica2srv (Optional)

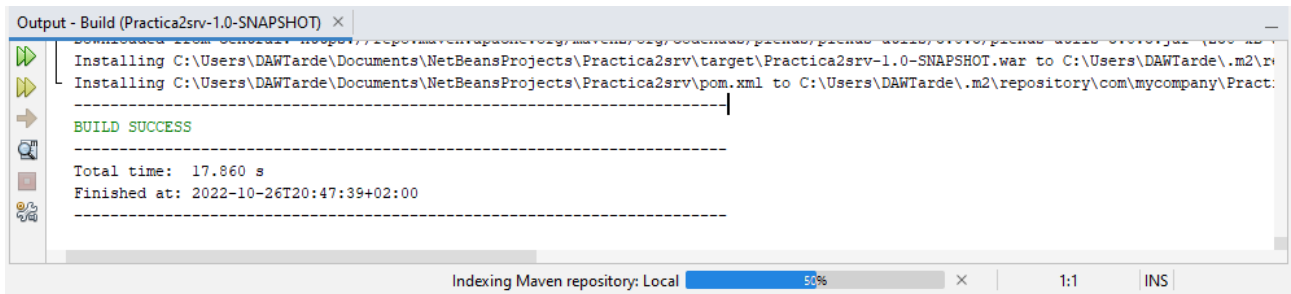
At the bottom, there are five buttons: '< Back', 'Next >' (highlighted), 'Finish', 'Cancel', and 'Help'.



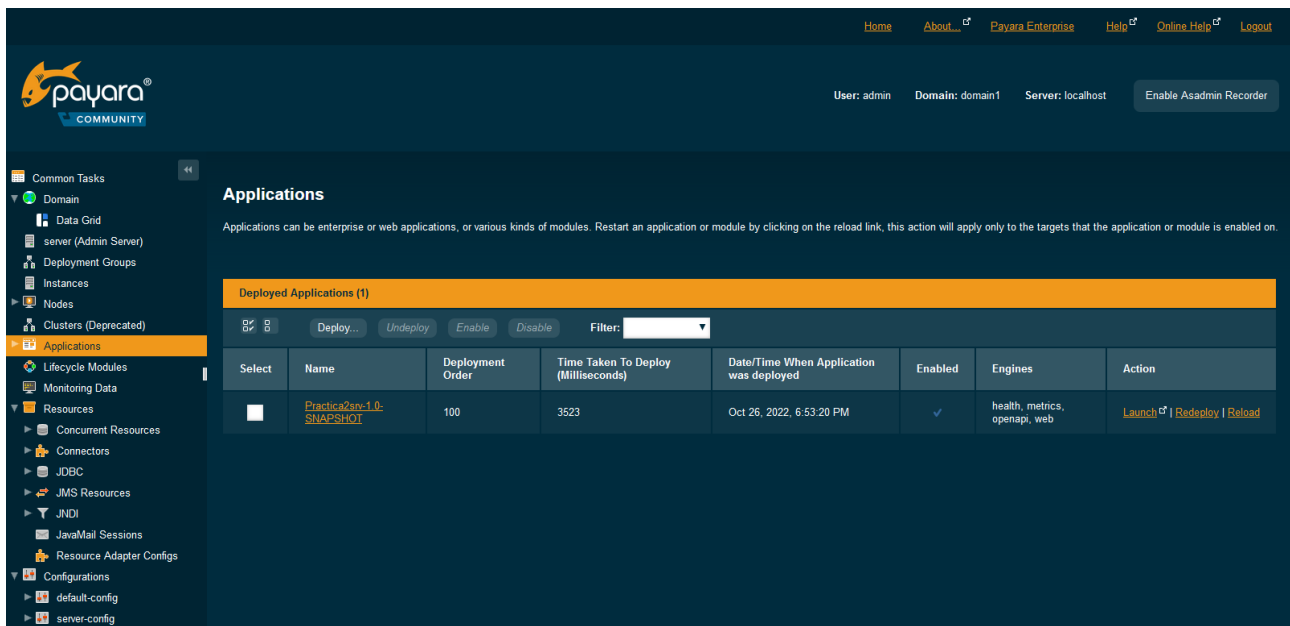
E indicamos que vamos a utilizar el payara server



## Compilamos nuestro proyecto y vemos que funciona



Ir a la página web de administración del servidor a la opción Applications, pulsar deploy y subir el fichero war generado. ¿Qué sucede?. Investigar como entrar en la aplicación web.



Y vemos correr la aplicación correctamente

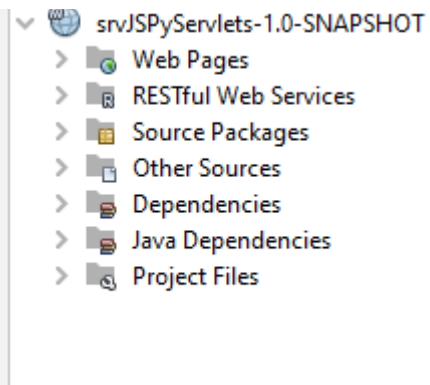
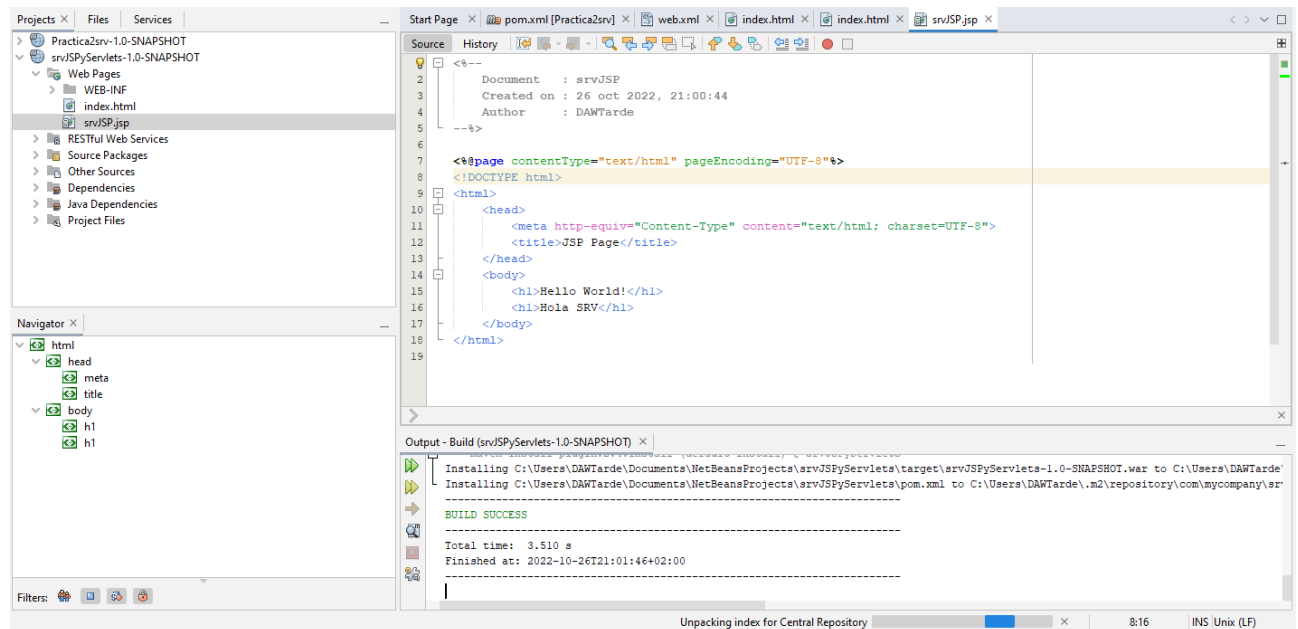


Hello World!

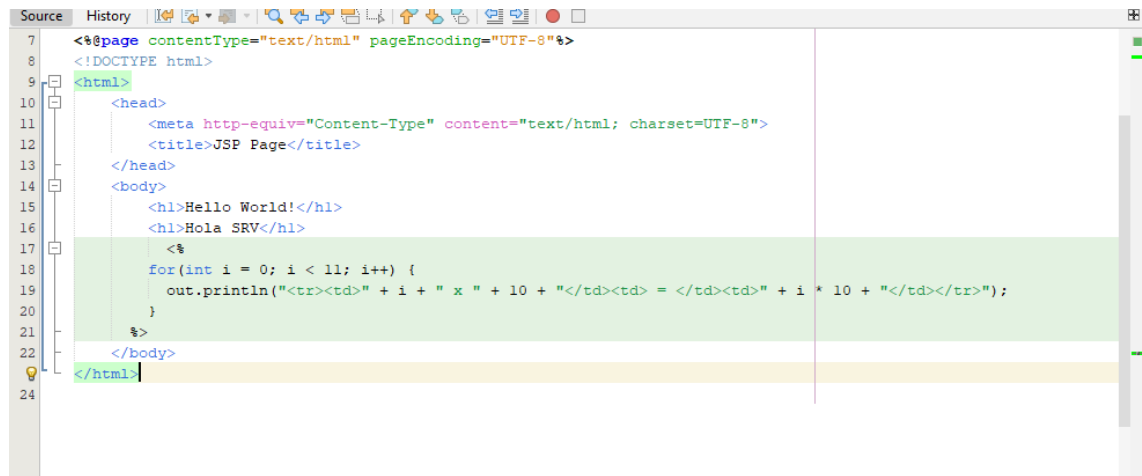
Eliminar del servidor la aplicación desplegada.

The screenshot shows the Payara Community Administration Console. The left sidebar contains a navigation menu with options like 'Common Tasks', 'Domain', 'Data Grid', 'server (Admin Server)', 'Deployment Groups', 'Instances', 'Nodes', 'Clusters (Deprecated)', 'Applications' (selected), 'Lifecycle Modules', 'Monitoring Data', 'Resources', 'Connectors', 'JDBC', 'JMS Resources', 'JNDI', 'JavaMail Sessions', 'Resource Adapter Configs', 'Configurations', 'default-config', and 'server-config'. The main area is titled 'Applications' and contains a table of 'Deployed Applications (1)'. A red arrow points to the 'Undeploy' button in the table's header.

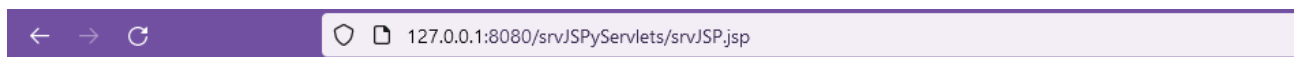
| Select                              | Name                      | Deployment Order | Time Taken To Deploy (Milliseconds) | Date/Time When Application was deployed | Enabled                             | Engines                       | Action   |
|-------------------------------------|---------------------------|------------------|-------------------------------------|---|-------------------------------------|-------------------------------|--|
| <input checked="" type="checkbox"/> | Practica2srv-1.0-SNAPSHOT | 100              | 3523                                | Oct 26, 2022, 6:53:20 PM                | <input checked="" type="checkbox"/> | health, metrics, openapi, web | <a href="#">Launch</a> <a href="#">Redeploy</a> <a href="#">Reload</a> |

**Tecnologías básicas JSP.****1. Crear otra aplicación del mismo tipo llamada TUSINICIALESJSPYSERVLETS.****2. Añade una página JSP de nombre tus iniciales.****Hello World!****Hola SRV**

3. En el jsp anterior, crear un h1 con tu nombre y hacer que se genere la tabla de multiplica de 10 en el servidor y se muestre en el cliente (investigar cómo hacerlo).



```
7 <%@page contentType="text/html" pageEncoding="UTF-8"%>
8 <!DOCTYPE html>
9 <html>
10 <head>
11 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
12 <title>JSP Page</title>
13 </head>
14 <body>
15 <h1>Hello World!</h1>
16 <h1>Hola SRV</h1>
17 <%
18     for(int i = 0; i < 11; i++) {
19         out.println("<tr><td>" + i + " x " + 10 + "</td><td> = </td><td>" + i * 10 + "</td></tr>");
20     }
21 %>
22 </body>
23 </html>
```



**Hello World!**

**Hola SRV**

0 x 10 = 0  
1 x 10 = 10  
2 x 10 = 20  
3 x 10 = 30  
4 x 10 = 40  
5 x 10 = 50  
6 x 10 = 60  
7 x 10 = 70  
8 x 10 = 80  
9 x 10 = 90  
10 x 10 = 100

#### 4. Ejecutar en el contenedor la aplicación.

```
</html>
root@98821294ec7a: /home/srv/payara5/bin# curl localhost:8080/srvJSPServlets/srvJSP.jsp

<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    <h1>Hola SRV</h1>
    <tr><td>0 x 10</td><td> = </td><td>0</td></tr><br/>
    <tr><td>1 x 10</td><td> = </td><td>10</td></tr><br/>
    <tr><td>2 x 10</td><td> = </td><td>20</td></tr><br/>
    <tr><td>3 x 10</td><td> = </td><td>30</td></tr><br/>
    <tr><td>4 x 10</td><td> = </td><td>40</td></tr><br/>
    <tr><td>5 x 10</td><td> = </td><td>50</td></tr><br/>
    <tr><td>6 x 10</td><td> = </td><td>60</td></tr><br/>
    <tr><td>7 x 10</td><td> = </td><td>70</td></tr><br/>
    <tr><td>8 x 10</td><td> = </td><td>80</td></tr><br/>
    <tr><td>9 x 10</td><td> = </td><td>90</td></tr><br/>
    <tr><td>10 x 10</td><td> = </td><td>100</td></tr><br/>
  </body>
</html>
root@98821294ec7a: /home/srv/payara5/bin#
```

#### 5. Mostrar la página y comparar el HTML con el contenido del JSP ¿Es diferente? ¿Qué ha sucedido

```
<!DOCTYPE html>
<html>
  <head> ... </head>
  <body>
    <h1>Hello World!</h1>
    <h1>Hola SRV</h1>
    0 x 10 = 0
    <br>
    1 x 10 = 10
    <br>
    2 x 10 = 20
    <br>
    3 x 10 = 30
    <br>
    4 x 10 = 40
```

Se ha convertido directamente en un HTML limpio.

**Cuestiones.**

- 1. ¿Qué es el “Web Profile” de Jakarta? ¿Y el “MicroProfile”? Indicar al menos 3 elementos que se encuentren en el perfil web y no en el perfil micro.**

El Perfil Web de Jakarta EE define un perfil de la Plataforma Jakarta EE específicamente dirigido a las aplicaciones web.

MicroProfile es una especificación impulsada por la comunidad diseñada para proporcionar una definición de plataforma de referencia que optimiza Enterprise Java para la arquitectura de microservicios y ofrece portabilidad de aplicaciones a través de múltiples tiempos de ejecución de MicroProfile, incluyendo Open Liberty.

- 2. Buscar los servidores de aplicaciones que soportan y se han certificado Jakarta EE 9 “Full”.**

Ha sido eliminado de todas las plataformas

- 3. Buscar ahora los servidores de aplicaciones certificados para Jakarta EE 9 perfil web.**

Apusic AAS, Eclipse GlassFish, FUJITSU Software Enterprise Application Platform, IBM WebSphere Liberty, Jboss Enterprise Application Platform, JEUS, Open Liberty, Payara Server.

- 4. ¿A qué piensas que se debe la existencia de los diferentes perfiles “Full”, “Web” “Core” y “Micro”? (Este último no es un perfil estándar).**

Se debe a la utilidad del servidor y los recursos que queramos según nuestras necesidades

- 5. En la siguiente lista se detallan los elementos de Jakarta WebProfile 10:**

- Jakarta Annotations 2.1\*
- Jakarta Authentication 3.0\*
- Jakarta Bean Validation 3.0
- Jakarta Concurrency 3.0\*
- Jakarta Contexts and Dependency Injection 4.0\*
- Jakarta Debugging Support for Other Languages 2.0
- Jakarta Dependency Injection 2.0
- Jakarta Enterprise Beans 4.0 Lite
- Jakarta Expression Language 5.0\*
- Jakarta Interceptors 2.1\*
- Jakarta JSON Binding 3.0\*
- Jakarta JSON Processing 2.1\*
- Jakarta Persistence 3.1\*
- Jakarta RESTful Web Services 3.1\*

- Jakarta Security 3.0\*
- Jakarta Server Faces 4.0\*
- Jakarta Server Pages 3.1\*
  - Jakarta Servlet 6.0\*
- Jakarta Standard Tag Library 3.0\*
  - Jakarta Transactions 2.0
  - Jakarta WebSocket 2.1\*

**Buscar información y explicar las función de los siguientes componentes:**

- **Jakarta Server Faces.**

Jakarta Faces, antes Jakarta Server Faces y JavaServer Faces (JSF) es una especificación de Java para la construcción de interfaces de usuario basadas en componentes para aplicaciones web y fue formalizada como estándar a través del Java Community Process formando parte de la Java Platform, Enterprise Edition.

- **Jakartaa WebSocket.**

Jakarta WebSocket™ especifica la API que los desarrolladores de Java pueden utilizar cuando quieren integrar WebSockets en sus aplicaciones - tanto en el lado del servidor como en el lado del cliente Java.

- **Jakarta Persistente.**

Java Persistence API, más conocida por sus siglas JPA, es una API de persistencia desarrollada para la plataforma Java EE. Maneja datos relacionales en aplicaciones usando la Plataforma Java en sus ediciones Standard y Enterprise

- **Jakarta Servlet.**

Un servlet de Jakarta (antes servlet de Java) es un componente de software de Java que amplía las capacidades de un servidor. Aunque los servlets pueden responder a muchos tipos de peticiones, lo más habitual es que implementen contenedores web para alojar aplicaciones web en servidores web y, por tanto, se califican como un servidor