

1. Una aplicación web puede ser desplegada en diferentes servidores web manteniendo su funcionalidad y sin ningún tipo de modificación en su código debido a la especificación servlet 2.2 , ¿cual es la estructura de directorios que debe tener?

La especificación Servlet 2.2 establece una estructura estándar para las aplicaciones web, conocida como estructura WAR (Web Application Archive). Esta estructura garantiza que las aplicaciones sean portables entre diferentes servidores compatibles con Java EE.

```

/<nombre-aplicación>/
├── /WEB-INF/
│   ├── web.xml
│   ├── /classes/
│   └── /lib/
└── Archivos públicos (HTML, CSS, JS, imágenes, etc.)

```

Detalles de cada elemento:

Directorio raíz (/<nombre-aplicación>/):

Es el contenedor principal de la aplicación web. Contiene todos los archivos y directorios de la aplicación. Este directorio es empaquetado como un archivo .war al desplegarse en el servidor de aplicaciones.

Directorio WEB-INF/:

Es un directorio reservado y no accesible directamente desde el cliente. Contiene la configuración y los recursos internos de la aplicación. Incluye:

web.xml:

Es el descriptor de despliegue de la aplicación. Contiene la configuración necesaria para el servidor web, como mapeos de servlets, filtros, parámetros de inicialización, etc.

/classes/:

Contiene los archivos .class que representan las clases Java compiladas de la aplicación. Estas clases incluyen servlets, controladores, y cualquier lógica de negocio.

/lib/:

Este directorio contiene archivos .jar con bibliotecas adicionales que la aplicación necesita para su funcionamiento.

Archivos y directorios públicos:

Estos archivos se colocan directamente en el directorio raíz y son accesibles desde el cliente a través de solicitudes HTTP. Por ejemplo:

Archivos HTML: Páginas estáticas o plantillas base de la aplicación.

CSS y JavaScript: Archivos para el diseño y la funcionalidad del frontend.

Imágenes y recursos multimedia: Archivos como .png, .jpg, .gif, etc.

Ejemplo práctico de estructura

Supongamos una aplicación llamada MiAplicación con las siguientes características:

Contiene una página principal ([index.html](#)).

Utiliza un servlet ([MiServlet.class](#)).

Requiere una biblioteca ([miBiblioteca.jar](#)).

La estructura de directorios sería:

```

/MiAplicación/
├── index.html
├── styles.css
├── script.js
├── /images/
│   └── logo.png

```

```
|      └─ fondo.jpg
|─ /WEB-INF/
|      └─ web.xml
|      └─ /classes/
|          └─ MiServlet.class
|      └─ /lib/
|          └─ miBiblioteca.jar
```

Por qué es importante esta estructura

Portabilidad:

Al seguir la especificación, la aplicación puede desplegarse en cualquier servidor web compatible con Java **EE** sin cambios en el código.

Seguridad:

Los recursos críticos de la aplicación (como las clases y bibliotecas) están protegidos dentro del directorio WEB-INF, inaccesible para los usuarios.

Mantenibilidad:

Una organización clara de los recursos facilita el desarrollo, mantenimiento y escalabilidad de la aplicación.

2. Another Neat Tool se basa en ficheros XML, normalmente configuramos el trabajo a hacer con nuestra aplicación en un fichero llamado [build.xml](#). Indica algunas de las etiquetas con las que podemos formar el contenido de este archivo.

El archivo [build.xml](#) en Apache Ant es el fichero principal que define las tareas (tasks) que se realizarán durante la construcción y despliegue de una aplicación. Se utiliza un formato XML estructurado para configurar estas tareas.

Principales etiquetas utilizadas en [build.xml](#)

<project>:

Es la etiqueta raíz del archivo.

Define el proyecto y sus propiedades generales.

Atributos comunes:

name: Nombre del proyecto.

default: Tarea que se ejecutará por defecto.

basedir: Directorio base del proyecto.

Ejemplo:

```
<project name="MiProyecto" default="compilar" basedir=".">
</project>
```

<property>:

Define propiedades (variables) que pueden ser utilizadas a lo largo del archivo.

Atributos comunes:

name: Nombre de la propiedad.

value: Valor de la propiedad.

Ejemplo:

```
<property name="src" value="src"/>
<property name="build" value="build"/>
```

<target>:

Define una tarea o conjunto de tareas.

Atributos comunes:

name: Nombre del objetivo (target).

depends: Especifica dependencias (otros targets que deben ejecutarse antes).

Ejemplo:

```
<target name="limpiar">
  <delete dir="${build}"/>
</target>
```

<mkdir>:

Crea un directorio.

Atributos comunes:

dir: Ruta del directorio a crear.

Ejemplo:

```
<delete dir="${build}"/>
```

<copy>:

Copia archivos o directorios.

Atributos comunes:

file: Archivo fuente a copiar.

tofile: Archivo destino.

todir: Directorio destino.

Ejemplo:

```
<copy file="README.md" todir="${build}"/>
```

<javac>:

Compila código fuente Java.

Atributos comunes:

srcdir: Directorio fuente.

destdir: Directorio destino de los archivos compilados.

Ejemplo:

```
<javac srcdir="${src}" destdir="${build}"/>
```

<jar>:

Genera un archivo JAR.

Atributos comunes:

destfile: Ruta y nombre del archivo JAR generado.

Ejemplo:

```
<jar destfile="MiAplicacion.jar" basedir="${build}"/>
```

Ejemplo completo de archivo `build.xml`:

```
<project name="MiProyecto" default="compilar" basedir=". ">
  <!-- Definición de propiedades -->
  <property name="src" value="src"/>
  <property name="build" value="build"/>
  <property name="jar" value="dist/MiAplicacion.jar"/>

  <!-- Tarea para limpiar -->
  <target name="limpiar">
    <delete dir="${build}"/>
  </target>

  <!-- Tarea para compilar -->
  <target name="compilar" depends="limpiar">
    <mkdir dir="${build}"/>
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <!-- Tarea para empaquetar en un JAR -->
  <target name="empaquetar" depends="compilar">
    <mkdir dir="dist"/>
    <jar destfile="${jar}" basedir="${build}"/>
  </target>
</project>
```

Notas:


La estructura modular de Ant permite definir tareas independientes y conectarlas mediante dependencias usando el atributo `depends`.

Ant es muy flexible, por lo que se pueden agregar muchas otras etiquetas para personalizar el flujo de trabajo, como `<echo>` para mensajes en la consola, o `<zip>` para crear archivos comprimidos.

3.1 Instalar el JDK 8.

Usare comando:

```
apt install openjdk-8-jre-headless
```



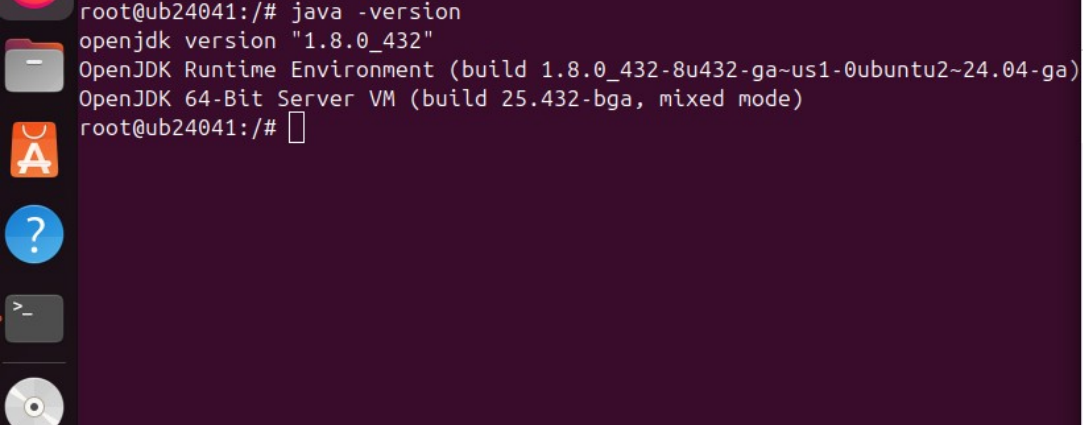
Terminal window output:

```
root@ub24041:/# java -v
No se ha encontrado la orden «java», pero se puede instalar con:
apt install openjdk-17-jre-headless # version 17.0.12+7-1ubuntu2~24.04, or
apt install openjdk-21-jre-headless # version 21.0.4+7-1ubuntu2~24.04
apt install default-jre             # version 2:1.17-75
apt install openjdk-11-jre-headless # version 11.0.24+8-1ubuntu3~24.04.1
apt install openjdk-8-jre-headless  # version 8u422-b05-1~24.04
apt install openjdk-19-jre-headless # version 19.0.2+7-4
apt install openjdk-20-jre-headless # version 20.0.2+9-1
apt install openjdk-22-jre-headless # version 22~22ea-1
root@ub24041:/# apt install openjdk-8-jre-headless
```

Browser window showing a profile for Alexey Suzdalenko.

Java instalado, compruebo con el comando:

```
java -version
```



Terminal window output:

```
root@ub24041:/# java -version
openjdk version "1.8.0_432"
OpenJDK Runtime Environment (build 1.8.0_432-8u432-ga-us1-0ubuntu2~24.04-ga)
OpenJDK 64-Bit Server VM (build 25.432-bga, mixed mode)
root@ub24041:/#
```

Browser window showing a profile for Alexey Suzdalenko.

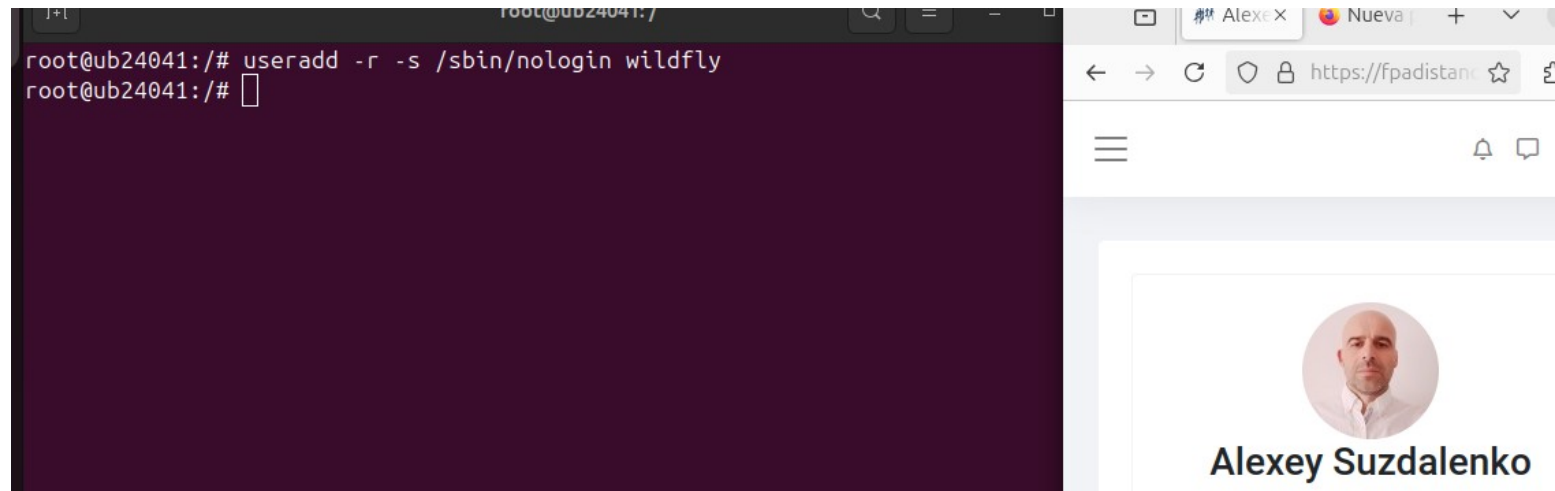
3.2 Crear usuario para WildFly

Crear un usuario para wildFly con este comando:

```
sudo useradd -r -s /sbin/nologin wildfly
```

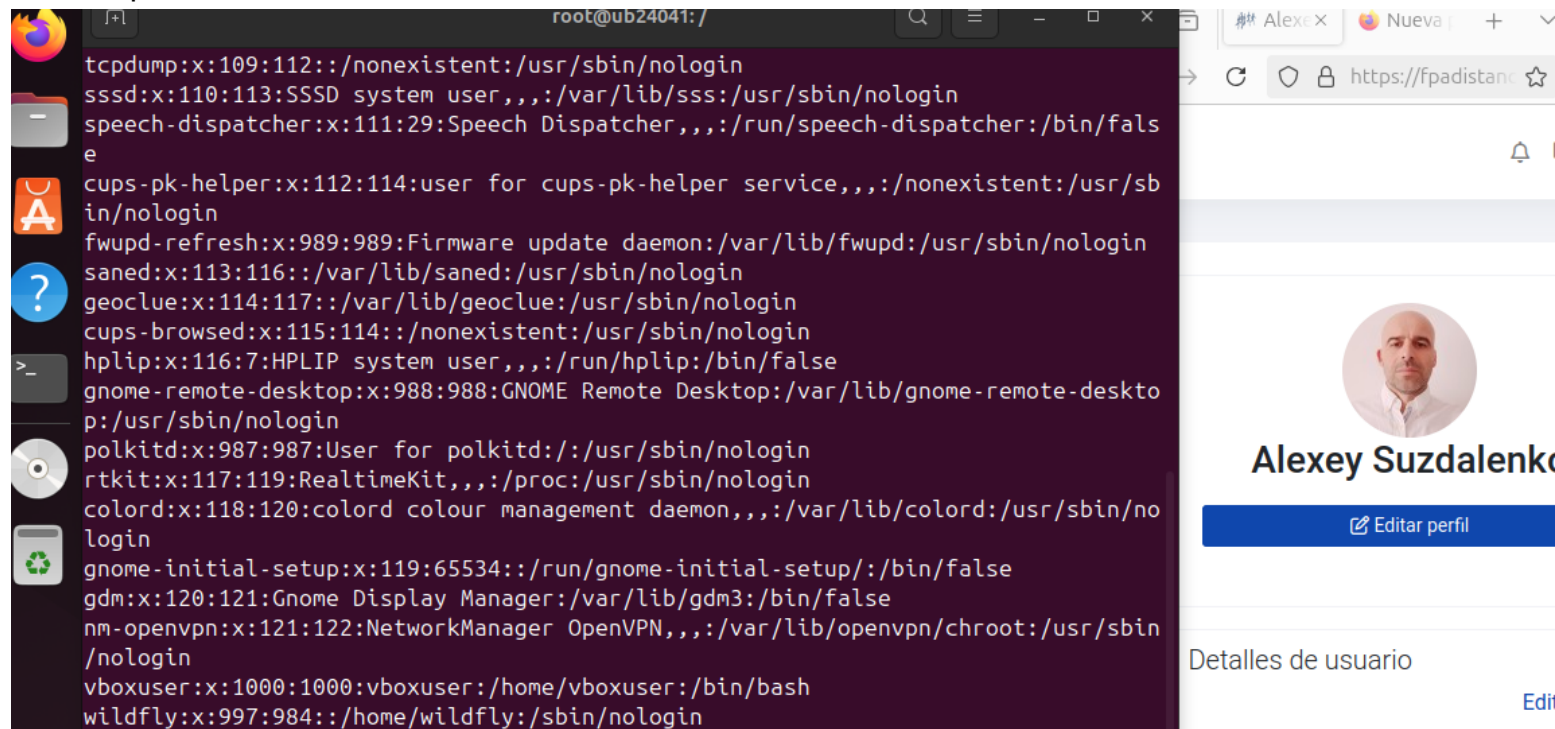
-r (opcion para un usuario del sistema)

-s (shell de inicio de sesion, que el usuario no puede iniciar sesion)



compruebo que el usuario se ha creado, es el ultimo:

```
cat /etc/passwd
```

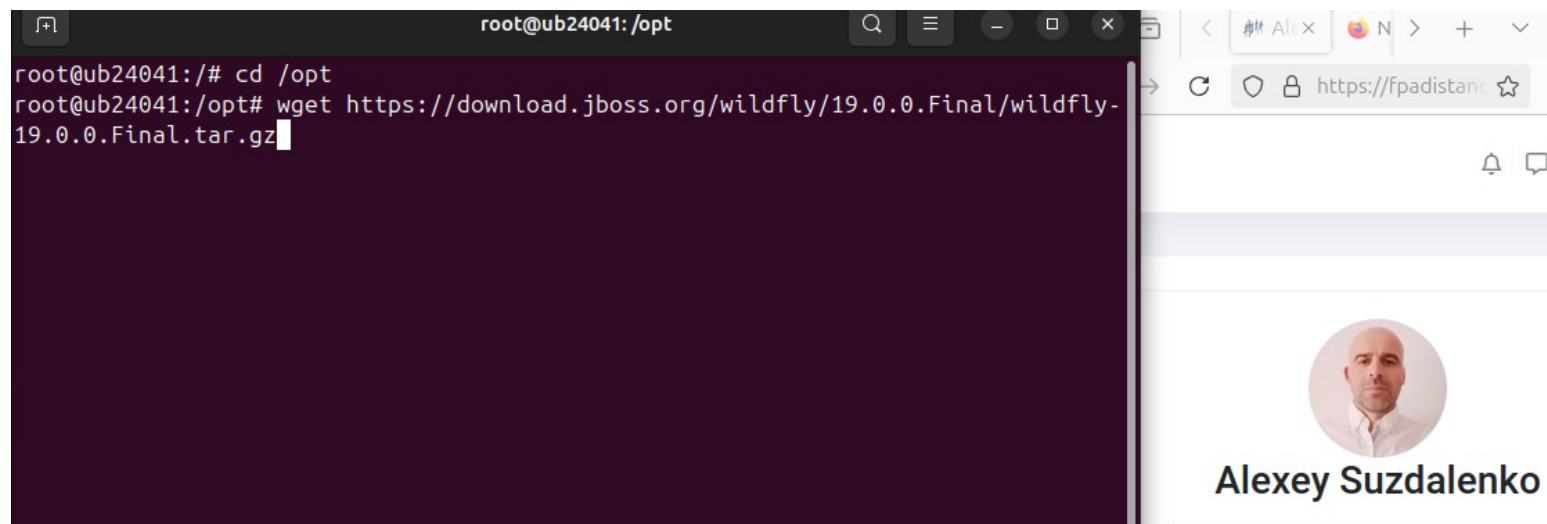


3.3 Descargar el WildFly 19.0.0 Final

Usare el comando:

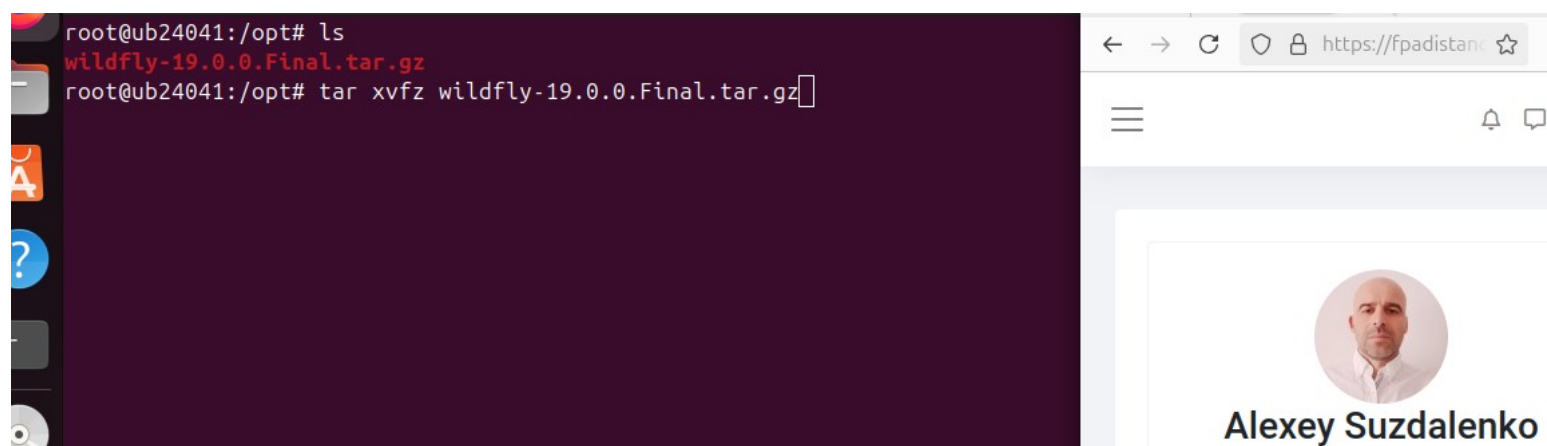
```
cd /opt
```

```
wget https://download.jboss.org/wildfly/19.0.0.Final/wildfly-19.0.0.Final.tar.gz
```



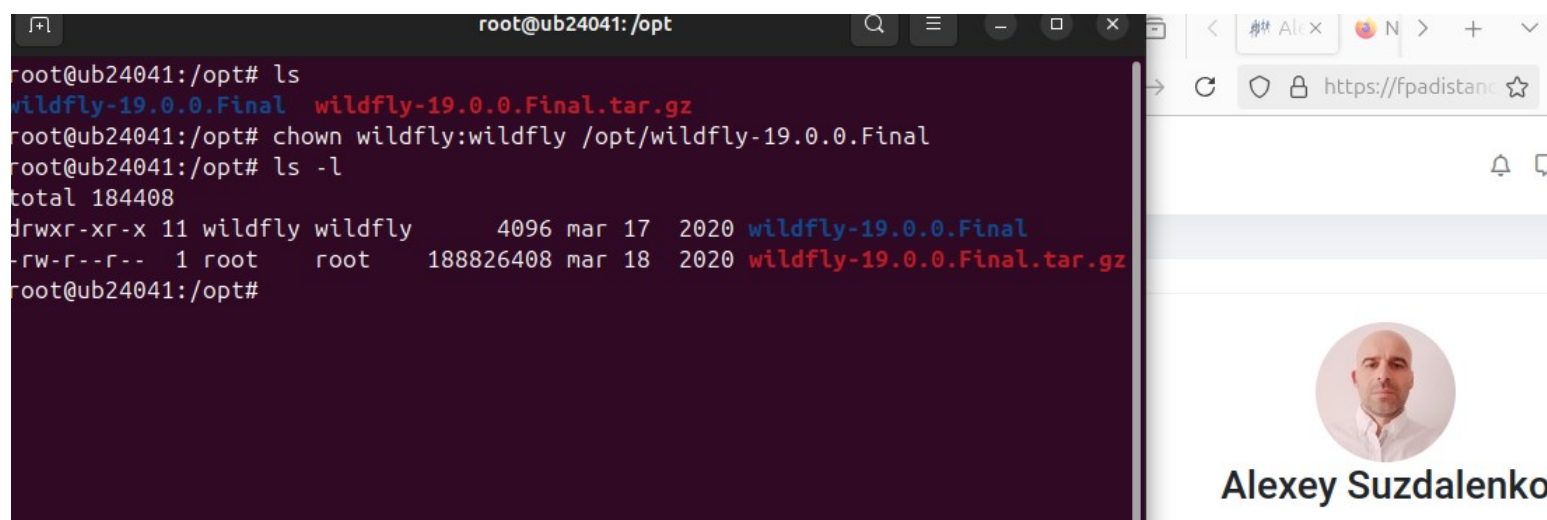
descomprimo el archivo descargado:

```
tar xvfz wildfly-19.0.0.Final.tar.gz
```



cambio la propiedad del archivo descomprimido a wildfly:

```
chown wildfly:wildfly /opt/wildfly-19.0.0.Final
```

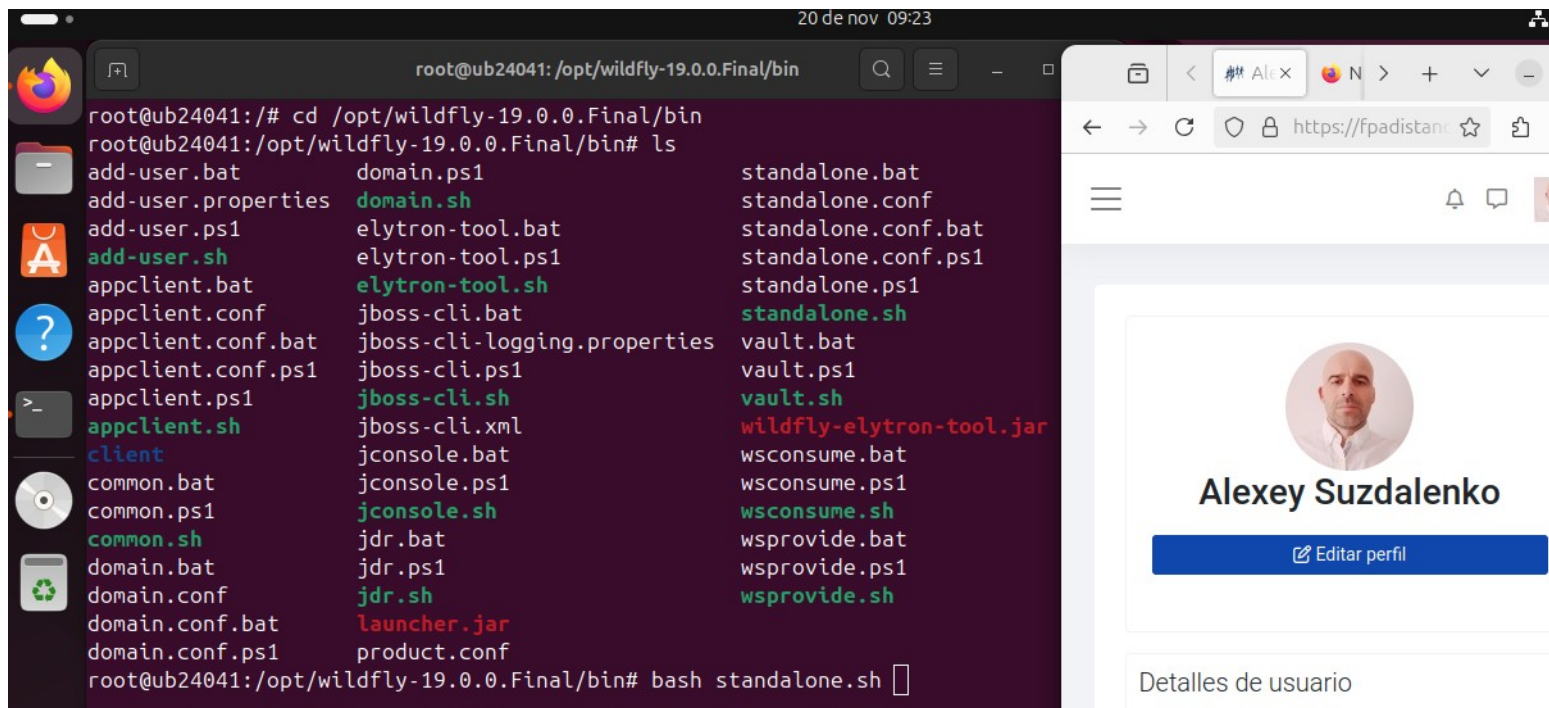


inicio el servidor WildFly:

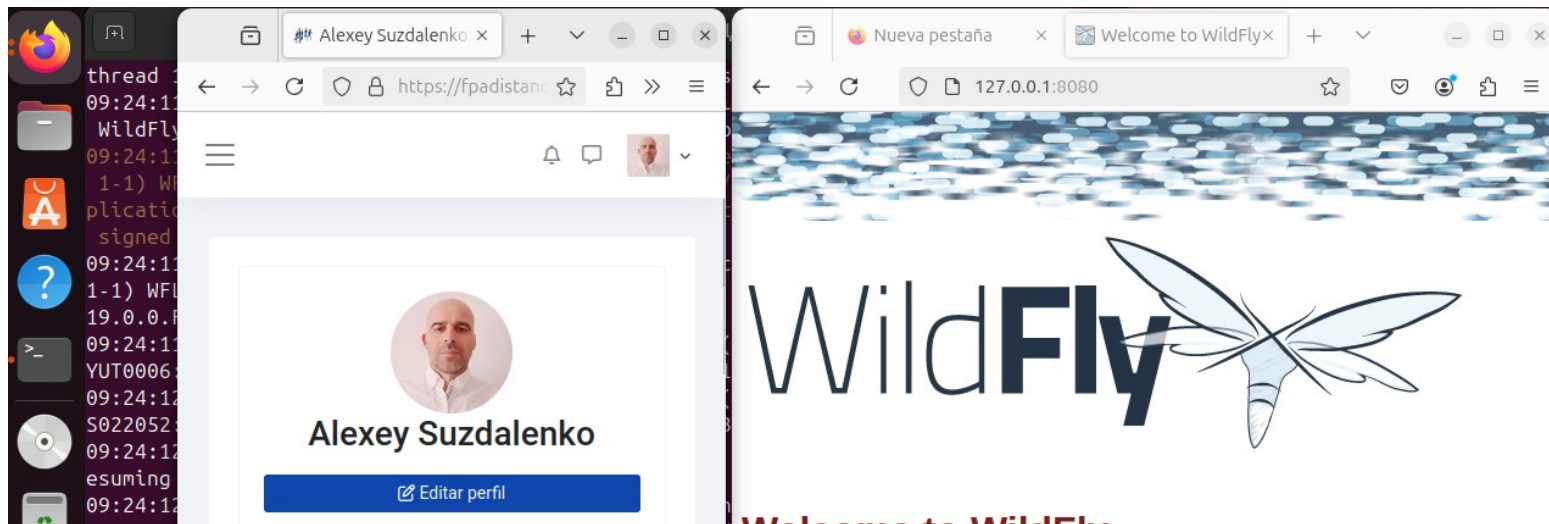
```
cd /opt/wildfly-19.0.0.Final/bin
```

```
ls
```

```
bash standalone.sh
```

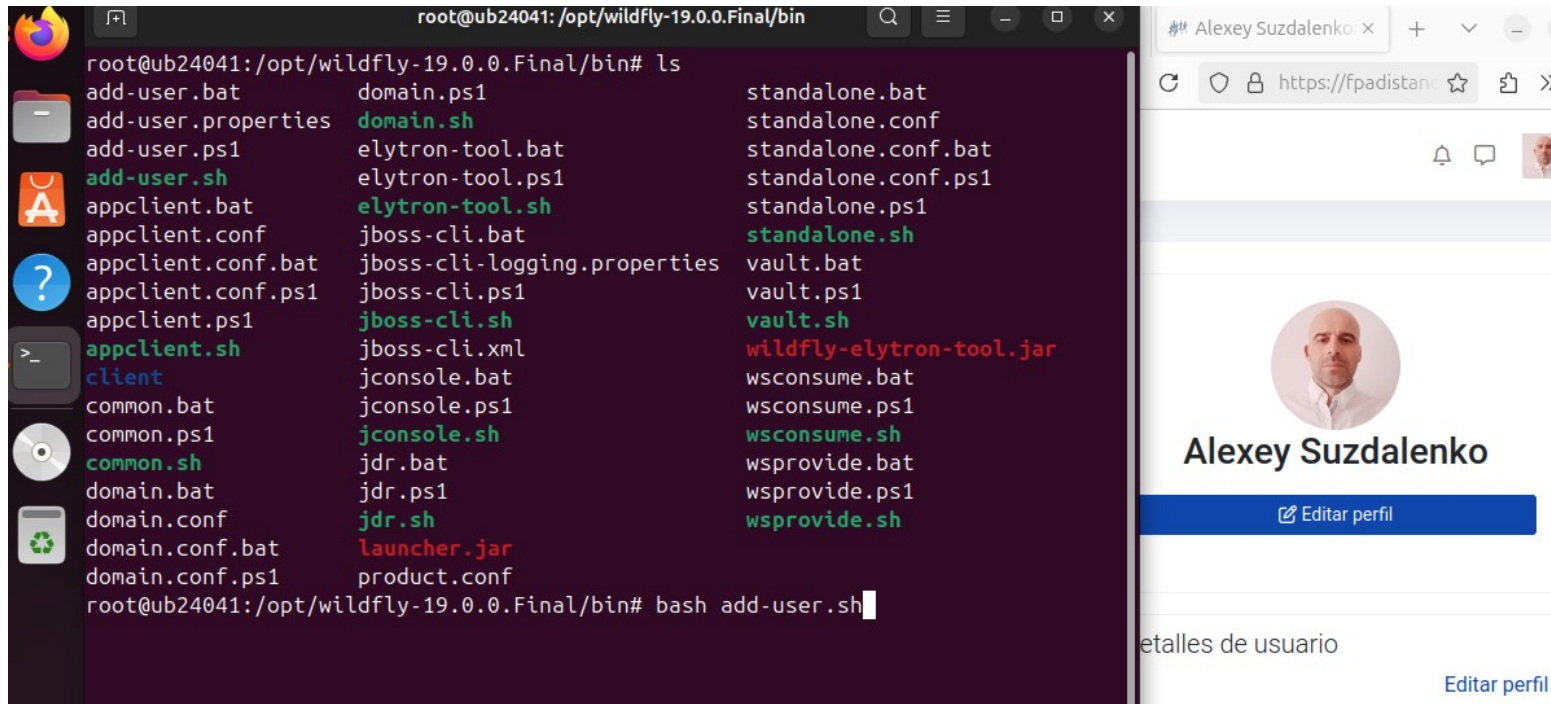
servidor wildFly trabajando, puede acceder a el en `http://127.0.0.1`



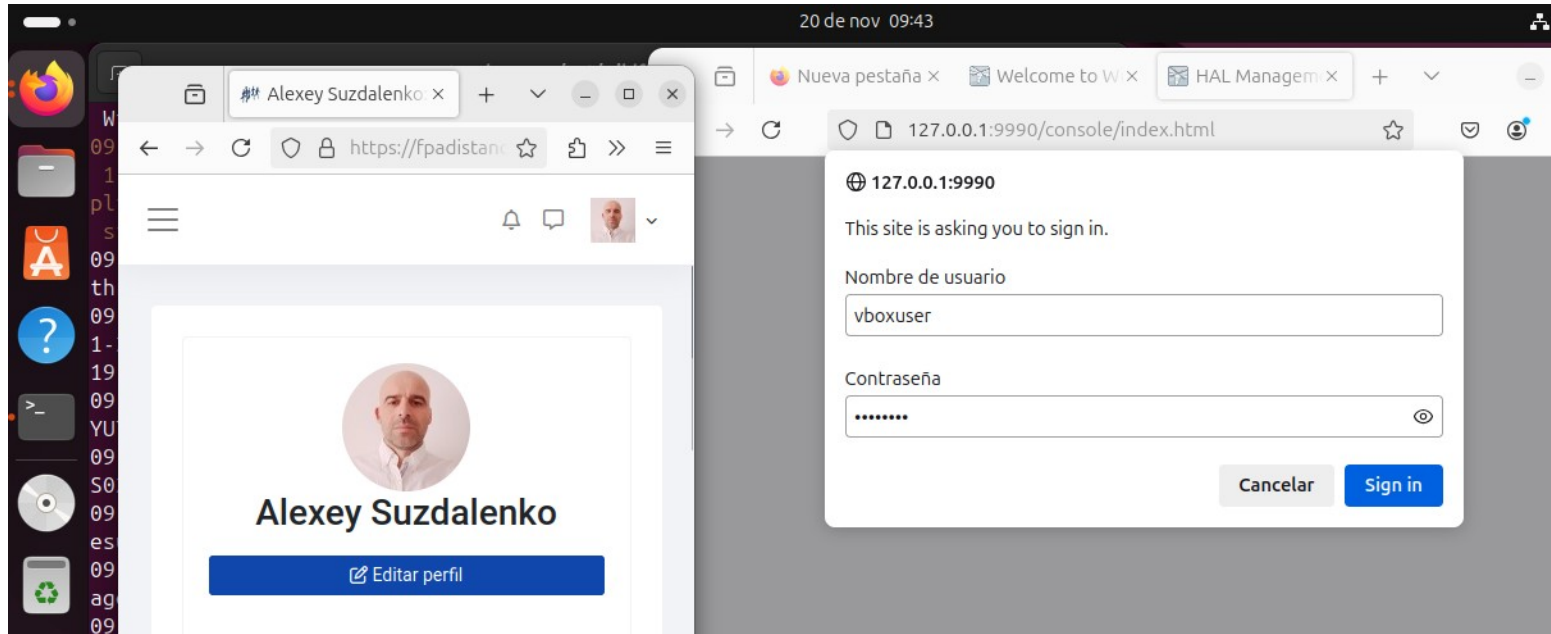
3.5 Configurar la authentication de WildFly

Estando en /opt/wildfly-19.0.0.Final/bin ejecuto comando:

bash add-user.sh



elegimos la accion a) management user
despues metemos el nombre de usuario y la contraseña,
despues de reiniciar y acceder por el navegador, nos pide el usuario y la contraseña:



y entro en la consola de wildFly:

