

Paso a Paso - Keycloak con Docker para el TPI

Este instructivo guía la instalación de Docker en distintos sistemas operativos y la posterior creación y configuración de un contenedor de **Keycloak**, para utilizarlo como **Authentication e Identity Provider** del Trabajo Práctico Integrador.

Incluye la creación de roles y usuarios de ejemplo para comenzar a integrarlo con el backend.

1. Instalación de Docker

¿Por qué Docker?

Docker es una plataforma que permite empaquetar aplicaciones junto con todas sus dependencias en contenedores. Esto simplifica enormemente la instalación, despliegue y ejecución de software, asegurando que funcione igual en cualquier entorno (desarrollo, testing o producción).

Para este trabajo práctico integrador, Docker es clave porque nos permitirá levantar servicios como **Keycloak** de manera rápida y aislada, sin necesidad de instalarlo directamente en el sistema operativo. También facilitará la integración entre microservicios a futuro, y permite que los distintos equipos trabajen con entornos equivalentes.

Entre sus funcionalidades principales se destacan:

- **Ejecución de contenedores livianos y portables:** permite aislar procesos sin la sobrecarga de una máquina virtual completa, ocupando pocos recursos y arrancando rápidamente.
- **Aislamiento entre aplicaciones:** cada contenedor corre en su propio entorno, evitando conflictos de dependencias o configuraciones entre servicios.
- **Definición y configuración por archivos de texto:** Docker permite definir el comportamiento de servicios y entornos completos mediante archivos de texto como **Dockerfile** y **docker-compose.yml**, lo que facilita la reproducción exacta de configuraciones y mejora la trazabilidad en equipos de trabajo.
- **Reutilización de imágenes y componentes:** se pueden descargar imágenes oficiales o personalizadas desde Docker Hub y combinarlas, reduciendo el tiempo de configuración.
- **Despliegue reproducible con docker-compose:** al definir múltiples servicios y sus relaciones en un solo archivo, se pueden levantar entornos completos de forma automática y coherente en cualquier equipo.

A continuación, se detallan los pasos para instalar Docker en cada sistema operativo.

1.1 En Windows 10/11

1. Descargar Docker Desktop desde: <https://www.docker.com/products/docker-desktop/>
2. Ejecutar el instalador y seguir los pasos.
3. Al finalizar, reiniciar la PC si es necesario.
4. Verificar desde terminal (PowerShell o CMD):

```
docker --version  
docker compose version
```

Si Docker está correctamente instalado, los comandos anteriores deberían devolver algo como:

- Docker version 28.3.x, build abc1234
- Docker Compose version v2.38.x

 Requiere tener habilitado **WSL2** (Subsistema de Windows para Linux versión 2), que permite ejecutar un entorno Linux directamente sobre Windows. Se recomienda instalar **Ubuntu** como distro por su compatibilidad, soporte extendido y facilidad de uso. Docker Desktop guía automáticamente en la instalación y configuración inicial si aún no está configurado.

1.2 En Linux (Ubuntu/Debian)

```
sudo apt update  
sudo apt install docker.io docker-compose -y  
sudo systemctl enable docker  
sudo systemctl start docker  
sudo usermod -aG docker $USER
```

 Reiniciar sesión para que se aplique el grupo docker

Verificar:

```
docker --version  
docker compose version
```

1.3 En macOS

1. Descargar Docker Desktop desde: <https://www.docker.com/products/docker-desktop/>
2. Abrir el archivo .dmg y arrastrar Docker a Aplicaciones.
3. Ejecutar Docker Desktop y completar configuración.

Verificar:

```
docker --version  
docker compose version
```

2. Lanzar un contenedor de Keycloak

¿Qué es Keycloak y por qué lo usamos?

Keycloak es una solución de código abierto para la gestión de identidades y accesos (IAM - Identity and Access Management). Permite centralizar el control de autenticación de usuarios, la administración de roles y la emisión de tokens compatibles con OAuth2 y OpenID Connect.

Para nuestro Trabajo Práctico Integrador, cumple la función de **Authentication Provider** (quién valida la identidad) y **Identity Provider** (quién emite la información sobre la identidad del usuario, como nombre, email, roles, etc.).

Esto permite desacoplar la seguridad del backend, delegando en Keycloak el ingreso de usuarios y la asignación de permisos, y brindando mayor flexibilidad, escalabilidad y estándares modernos de autenticación para nuestros microservicios.

Crear el contenedor Docker para soportar una instancia de Keycloak

Creamos una carpeta de trabajo y dentro de ella un archivo `docker-compose.yml` con el siguiente contenido:

```
version: '3.1'

services:
  keycloak:
    image: quay.io/keycloak/keycloak:24.0.3
    container_name: keycloak
    command: start-dev
    ports:
      - "8081:8080"
    environment:
      - KEYCLOAK_ADMIN=admin
      - KEYCLOAK_ADMIN_PASSWORD=admin123
    volumes:
      - keycloak_data:/opt/keycloak/data

volumes:
  keycloak_data:
```

 Este contenedor deja Keycloak expuesto en <http://localhost:8081/>

Iniciar:

```
docker compose up -d
```

 Explicación del comando:

- `docker compose`: utiliza el archivo `docker-compose.yml` para levantar servicios definidos allí.
- `up`: inicia y crea los contenedores especificados si no existen.
- `-d`: ejecuta los contenedores en modo "detached" (en segundo plano), permitiendo seguir usando la terminal.

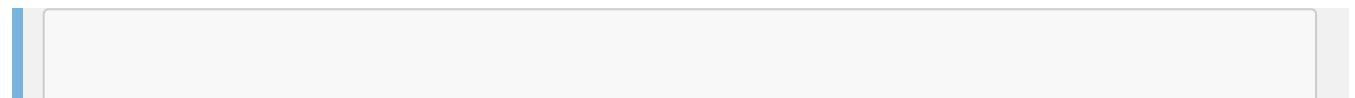
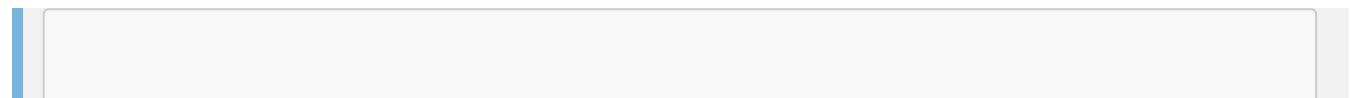
Si el contenedor se levanta correctamente, se puede comprobar con:

```
docker ps
```

Y debería aparecer un contenedor con el nombre **keycloak**, expuesto en el puerto 8081. También se puede acceder con el navegador a <http://localhost:8081/> para verificar la interfaz de administración.

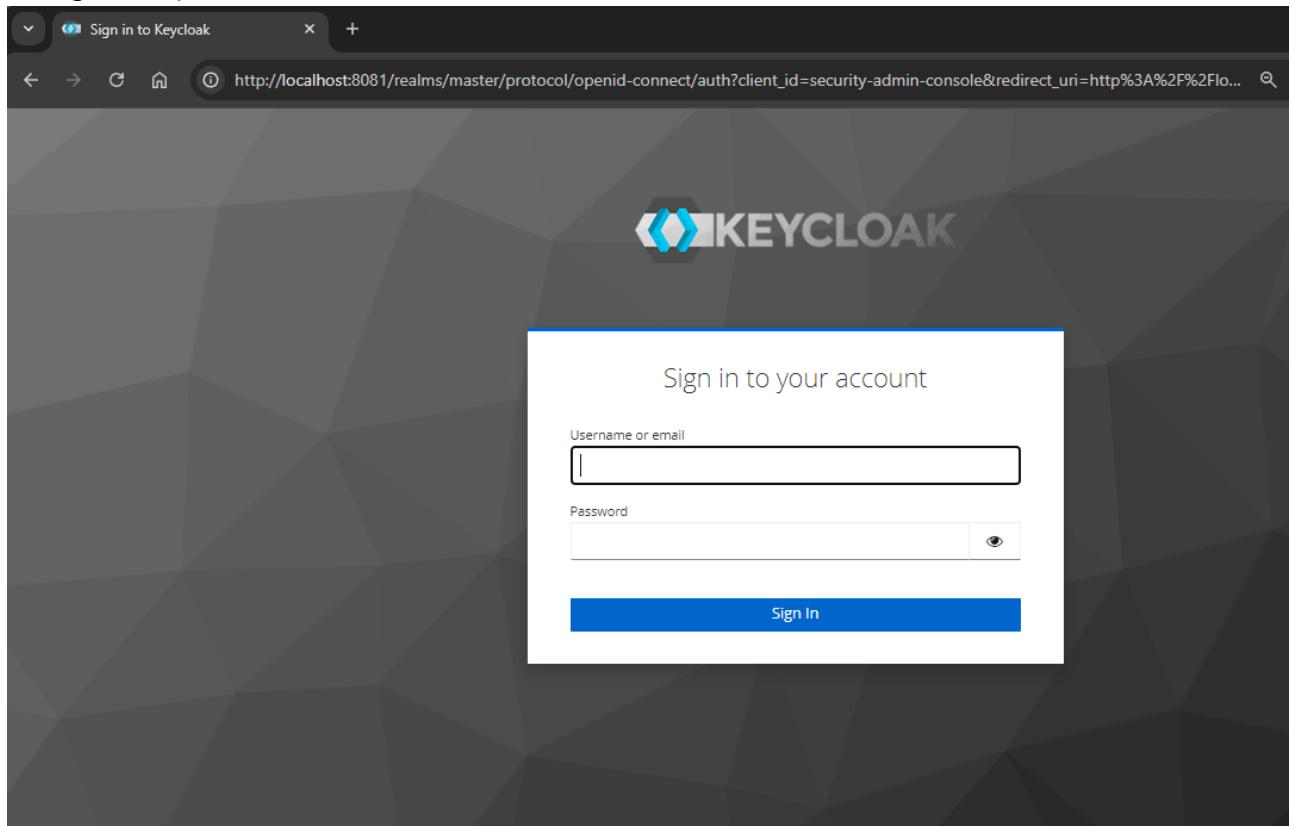
Parar:

```
docker compose down
```



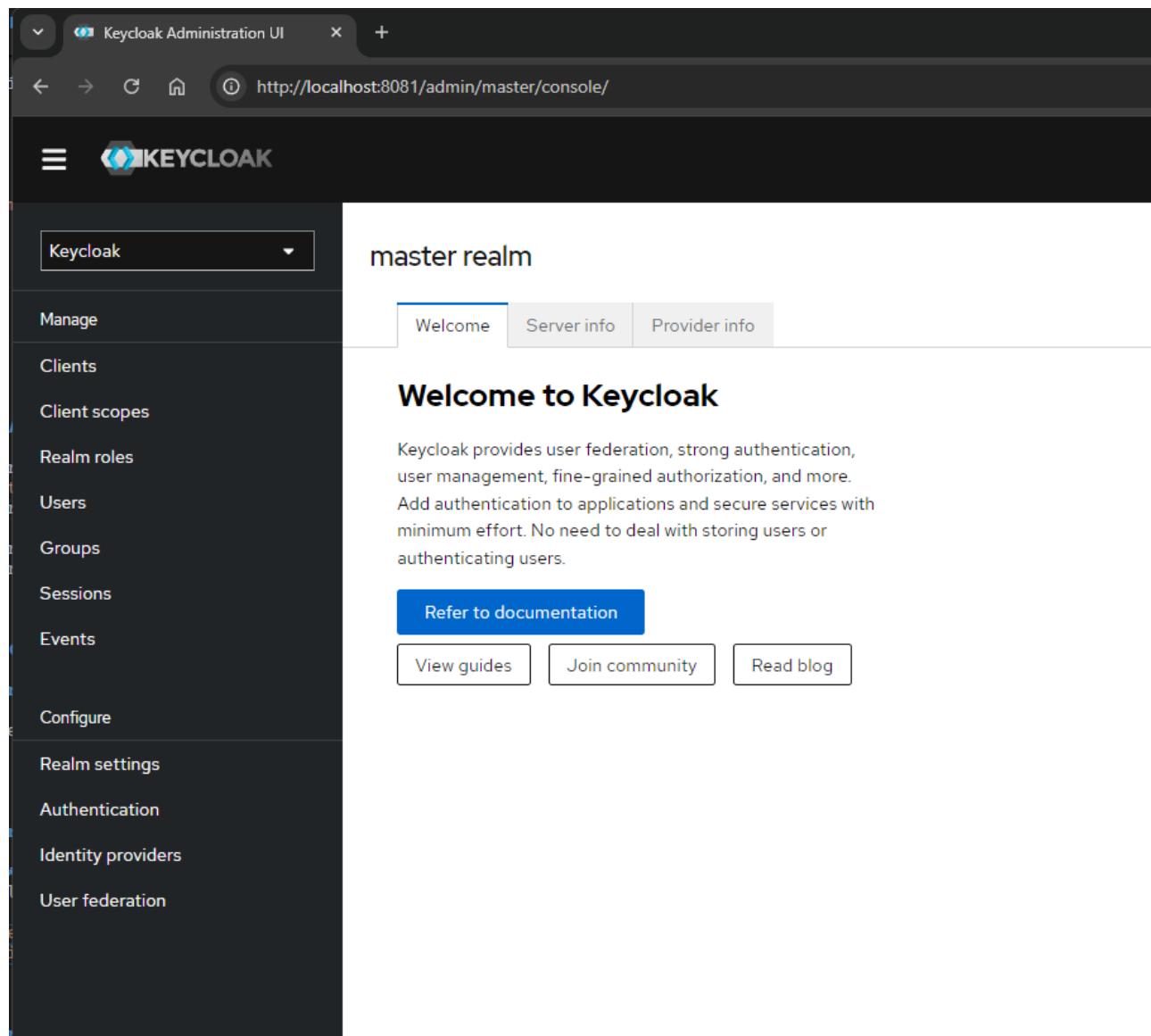
🔑 3. Acceder a Keycloak

1. Navegar a <http://localhost:8081/>



2. Iniciar sesión con:

- Usuario: **admin**
- Contraseña: **admin123**



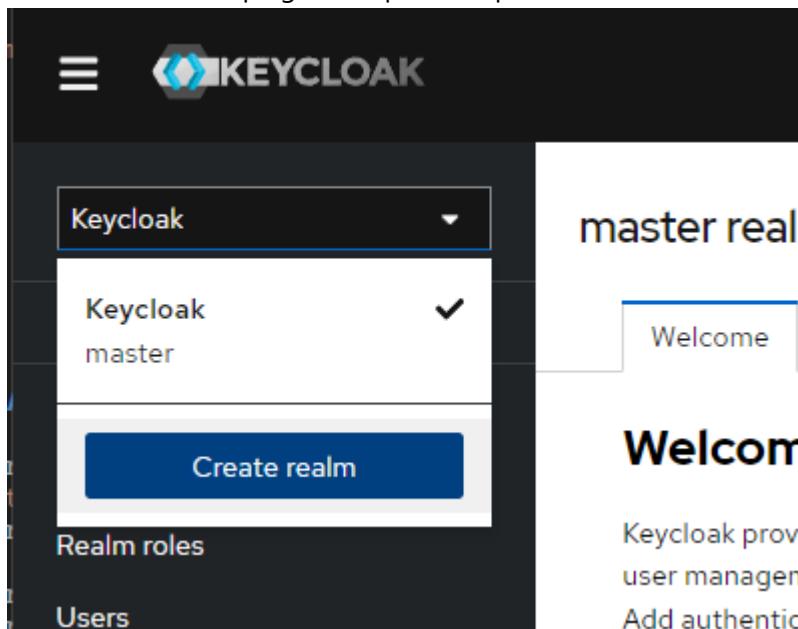
4. Crear Realm, Roles y Usuarios

4.1 Crear un Realm nuevo

Un *Realm* en Keycloak es una partición lógica del servidor que permite gestionar de manera independiente usuarios, roles, clientes (aplicaciones) y configuraciones de seguridad. Cada realm funciona como un espacio aislado dentro del mismo servidor Keycloak.

Esto permite, por ejemplo, tener diferentes entornos (producción, desarrollo, test) o aplicaciones separadas sin interferencias entre sí.

1. Desde el menú desplegable superior izquierdo, seleccionar **Create Realm**.



2. Nombre sugerido: **tpi-backend**

A screenshot of the 'Create realm' form in the Keycloak Administration UI. The URL in the browser is <http://localhost:8081/admin/master/console/#/tpi-backend/add-realm>. The form fields include:

- Resource file: A placeholder for a JSON file, with a 'Browse...' and 'Clear' button.
- Realm name *: The value 'tpi-backend' is entered into this field.
- Enabled: A toggle switch is set to 'On'.

At the bottom are 'Create' and 'Cancel' buttons.

4.2 Crear Roles

1. Ir a **Realm Roles > Create role**

If you want to leave this page and manage this realm, please click the corresponding menu items in the left navigation bar.

2. Crear los siguientes roles:

- o **cliente**
- o **admin**

Role name	Composite	Description
admin	False	-
cliente	False	-
default-roles-tpi-backend	True	\${role_default-roles}
offline_access	False	\${role_offline-access}
uma_authorization	False	\${role_uma_authorization}

4.3 Crear Usuarios

1. Ir a **Users > Create new user**
2. Completar los datos mínimos (username, nombre, correo)
3. En el perfil del usuario, ir a **Credentials**:
 - o Definir contraseña (ej: **clave123**) y marcar **Set as temporary: OFF**
4. Ir a **Role Mappings**:
 - o Asignar uno de los roles creados

Deberíamos crear al menos un usuario **cliente** y un usuario **admin** para pruebas posteriores

A modo de ejemplo, se puede utilizar la siguiente tabla de usuarios sugeridos para probar distintos roles y comportamientos en el sistema:

Username	Nombre	Email	Rol asignado	Contraseña
cliente01	Carla Gómez	carla@example.com	cliente	Clave123
cliente02	Juan Pérez	juan@example.com	cliente	Clave123
cliente03	Lucía Fernández	lucia@example.com	cliente	Clave123
cliente04	Diego Luna	diego@example.com	cliente	Clave123
cliente05	Florencia Ramos	flor@example.com	cliente	Clave123
admin01	Marcos Salas	marcos@example.com	admin	Clave123
admin02	Natalia Quinteros	natalia@example.com	admin	Clave123
admin03	Tomás Acosta	tomas@example.com	admin	Clave123
tester01	Emiliano Testa	emiliano@example.com	cliente	Clave123
tester02	Belén Rivas	belen@example.com	cliente	Clave123

Username	Email	Last name	First name
cliente01	carla@example.com	Gomez	Carla
cliente02	juan@example.com	Perez	Juan
cliente03	lucia@example.com		
cliente04	diego@example.com		
cliente05	flor@example.com		
admin01	marcos@example.com		
admin02	natalia@example.com		
admin03	tomas@example.com		
tester01	emiliano@example.com	Testa	Emiliano
tester02	belen@example.com		

5. Verificar tokens

Una vez configurado el Realm y creados los usuarios, podemos probar el proceso de autenticación y obtener un token JWT válido. Esto se realiza contra los endpoints estándar de OpenID Connect que expone Keycloak.

5.1 Crear cliente público para pruebas

Antes de comenzar, es necesario crear en Keycloak un cliente público para permitir el flujo de autorización con redirección.

1. Ingresar al panel de administración de Keycloak (<http://localhost:8081/admin/>).

2. Seleccionar el realm **tpi-backend** y navegar a la sección **Clients**.

3. Hacer clic en **Create client**.

4. Configurar:

- Client ID: **tpi-backend-client**
- Client type: **Public**
- Name: **TPI Backend Client**
- Root URL: **http://localhost:8080**
- Click en **Next**

5. En la pantalla de configuración:

- Activar **Standard Flow Enabled**
- Desactivar Client Authentication
- En **Valid redirect URIs**, colocar:
 - **http://localhost:8080/api/login/oauth2/code/keycloak**
 - O un comodín como **http://localhost:8080/***
- Click en **Save**

5.2 Probar autenticación (obtener token)

Alternativamente al flujo directo **password**, también se puede utilizar el flujo estándar de autenticación con formulario de Keycloak y luego intercambiar el código de autorización por un token válido.

1. Navegar con el navegador a la URL del Authorization Endpoint:

```
http://localhost:8081/realms/tpi-backend/protocol/openid-connect/auth  
?client_id=tpi-backend-client  
&response_type=code  
&redirect_uri=http://localhost:8080/api/login/oauth2/code/keycloak
```

2. Iniciar sesión con un usuario válido, por ejemplo:

- usuario: **cliente01**
- contraseña: **clave123**

3. Una vez autenticado, Keycloak intentará redirigir a la **redirect_uri** especificada con un parámetro **code** en la URL.

4. Luego, se debe realizar una petición **POST** al token endpoint para intercambiar el **code** por un token:

```
### Obtener token vía código de autorización  
POST http://localhost:8081/realms/tpi-backend/protocol/openid-connect/token  
Content-Type: application/x-www-form-urlencoded  
  
grant_type=authorization_code  
code=<el_code_recibido>
```

```
client_id=tpi-backend-client
redirect_uri=http://localhost:8080/api/login/oauth2/code/keycloak
```

 Este flujo es el más adecuado para aplicaciones web seguras y refleja mejor un escenario real de uso en producción.

Una vez obtenido el `access_token`, se puede inspeccionar su contenido en <https://jwt.io> para verificar:

- Expiración (`exp`)
- Identidad del usuario (`preferred_username`)
- Roles disponibles (`realm_access.roles`)

Esto permite comprobar que el token es válido, confiable y contiene los claims esperados antes de integrarlo a una aplicación real. en <https://jwt.io> y utilizarlo para consumir recursos protegidos.

Ejemplo de intercambio automático de código a token con `RestClient`

Para ilustrar cómo este proceso puede integrarse en un backend Spring moderno, aquí un ejemplo funcional utilizando `RestClient` de Spring Framework 6+ con codificación correcta del cuerpo `application/x-www-form-urlencoded`:

Creación de Servicio spring web básico para recibir la redirección del código de aplicación y obtener el token. Debe crear un microservicio básico con spring web y agregar el siguiente endpoint.

```
@GetMapping("/api/login/oauth2/code/keycloak")
public String intercambiarCode(@RequestParam String code) throws
UnsupportedEncodingException {
    RestClient restClient = RestClient.create();

    String formData = "grant_type=authorization_code" +
        "&code=" + URLEncoder.encode(code, StandardCharsets.UTF_8) +
        "&client_id=tpi-backend-client" +
        "&redirect_uri=" +
    URLEncoder.encode("http://localhost:8080/api/login/oauth2/code/keycloak",
    StandardCharsets.UTF_8);

    String token = restClient.post()
        .uri("http://localhost:8081/realm/tpi-backend/protocol/openid-
connect/token")
        .contentType(MediaType.APPLICATION_FORM_URLENCODED)
        .body(formData)
        .retrieve()
        .body(String.class);

    log.info("🔐 Token recibido desde Keycloak:{}", token);
    return "☑ Token recibido y logueado en consola";
}
```

💡 Este código realiza el intercambio del `code` directamente desde el backend y permite capturar el `access_token` para depuración o análisis posterior. Ideal para entornos de prueba o enseñanza. en <https://jwt.io> y utilizarlo para consumir recursos protegidos. Luego volver a probar y se podrá encontrar el token en la consola del servidor.