TFC

Desarrollo de una Plataforma Web con PHP, MySQL, Plugin de Minecraft Java, Docker y Kubernetes

Realizado por: Adrián Gómez Morales

Tutor del proyecto: D. Nicolás José del Pozo Madroñal

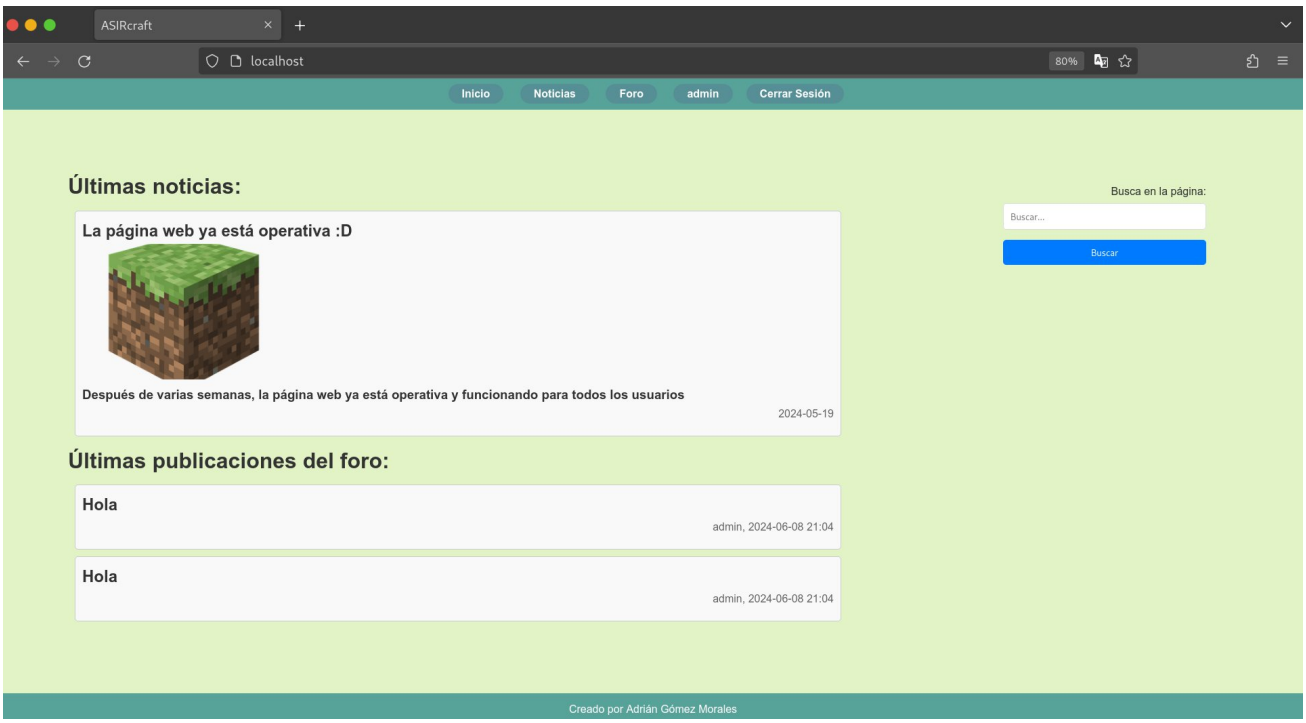
Centro: I.E.S Triana

1. Página web creada con PHP

Es una página creada desde cero, sin usar ningún CMS como WordPress (El código está comentado con las explicaciones)

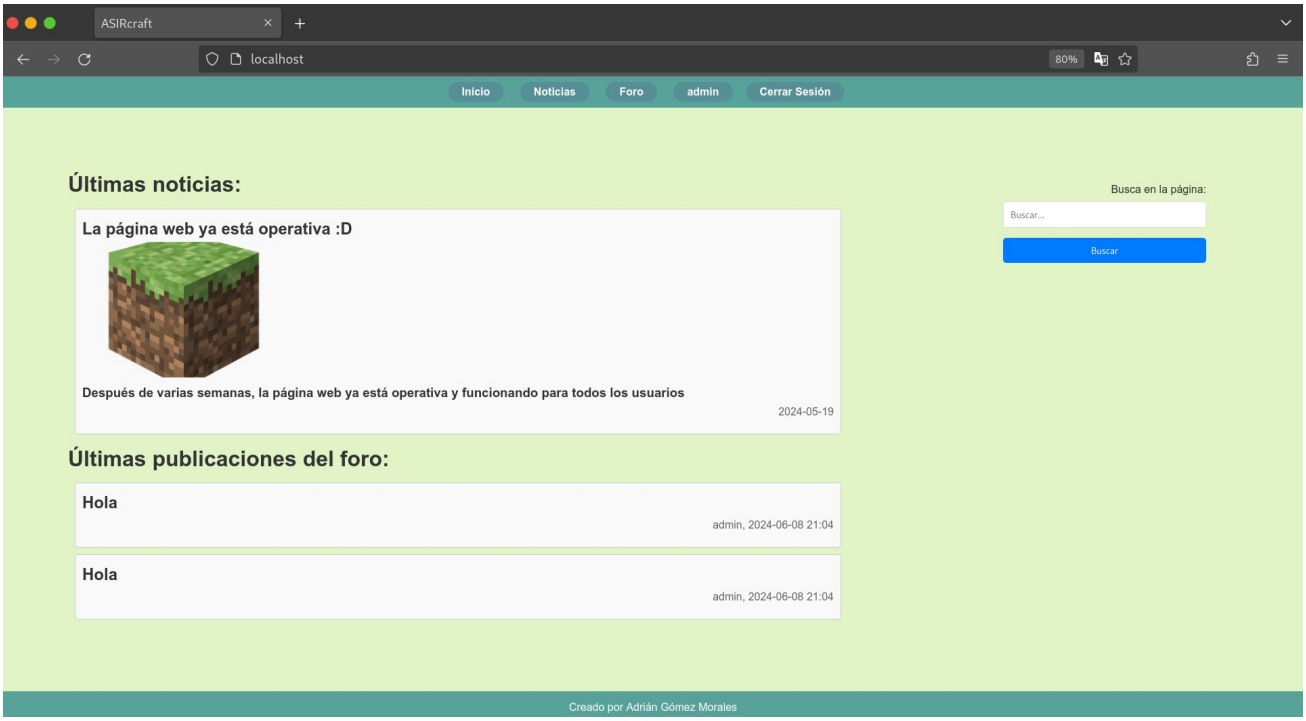
Inicio

En el inicio del sitio, aparecerán las dos últimas noticias y los dos últimos posts del foro, además de la posibilidad de buscar noticias, además de forma interna, detecta si existe un usuario con rol admin y en caso de que no, crea un usuario llamado admin con contraseña changeme

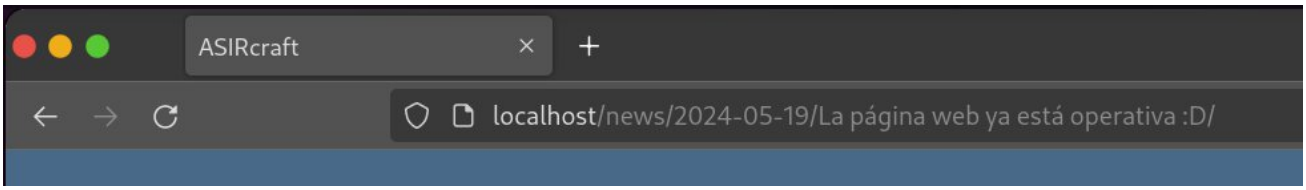


Noticias

En la página de noticias, podemos ver todas las dos últimas noticias que se han publicado en la página Web



Además, tenemos la opción de buscar las noticias si indicamos un texto dentro del formulario de la derecha

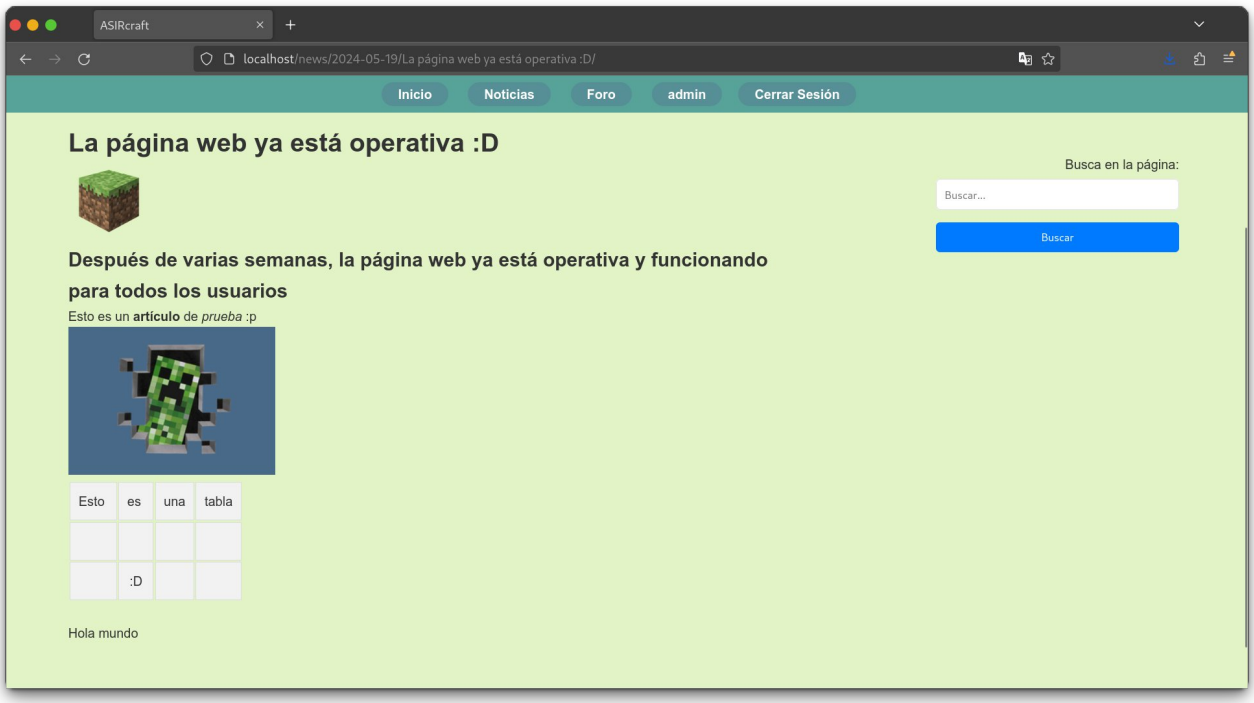


Las noticias se guardan en un archivo json ubicado en /news, al momento de publicar la noticia, creará una carpeta con la fecha en formato YYYY-MM-DD y dentro de esta carpeta, creará otra carpeta con el titular de la noticia y un archivo index.php con la noticia en sí

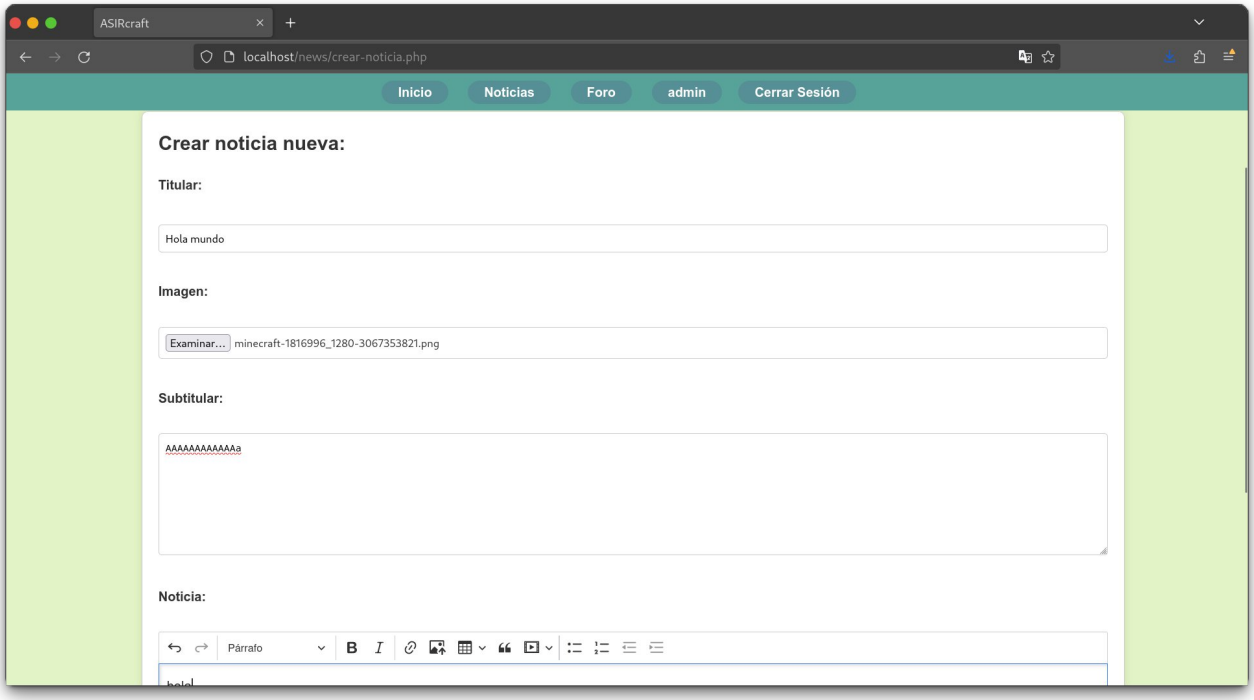
Noticias.json:

Aunque esto no guarda en sí la noticia, es usado para crear el enlace a la noticiay para filtrar usando el buscador de noticias

Si damos click en cualquier parte del rectángulo de la noticia, podemos entrar a verla:



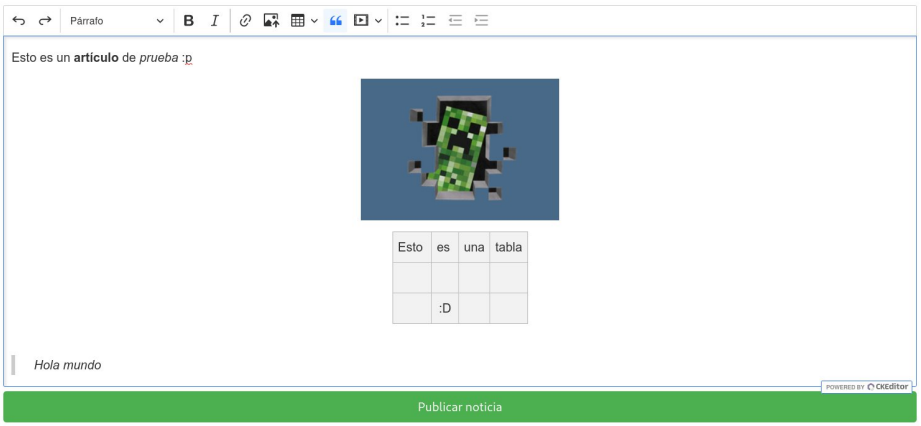
Solo los usuarios administradores pueden crear noticias dentro de la página web mediante un formulario, que pregunta el titular, el subtitular, una imagen destacada y el texto o el cuerpo de la noticia mediante CKEditor



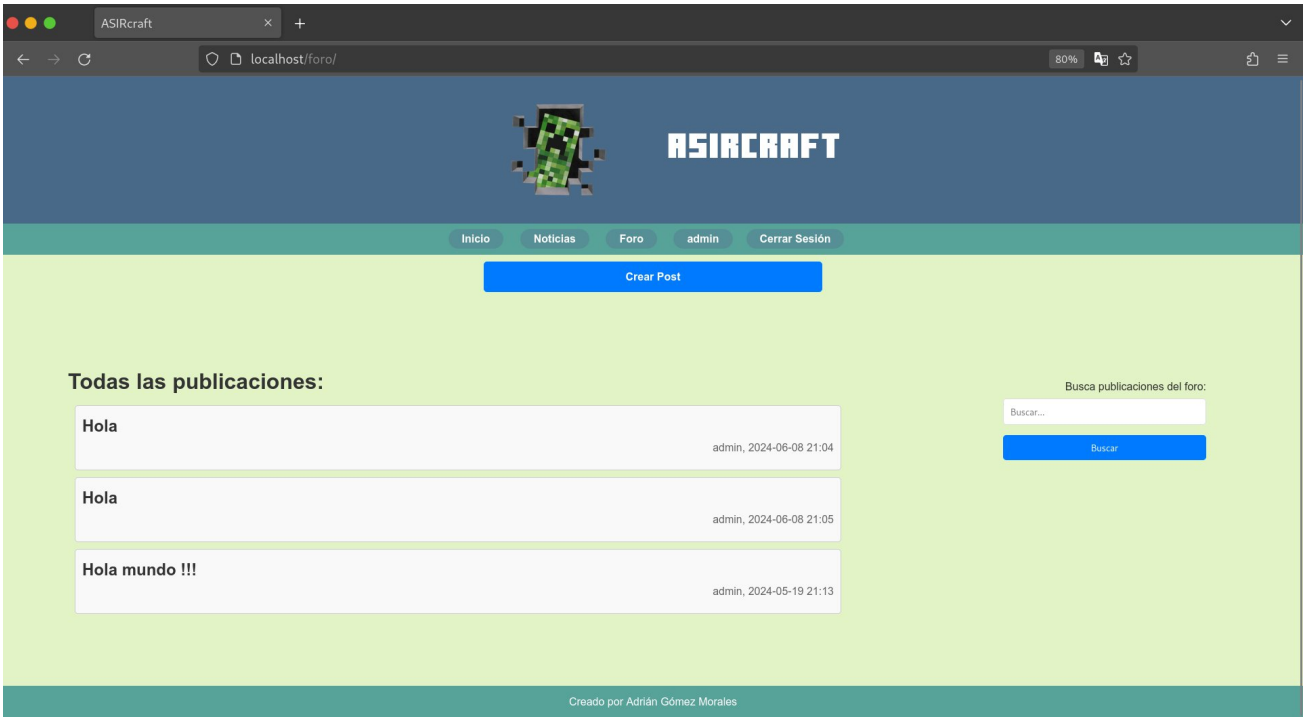
CKEditor y CKFinder

CKEditor es un editor de texto enriquecido que se utiliza comúnmente en aplicaciones web para permitir a los usuarios formatear texto de manera similar a un procesador de texto. Proporciona una interfaz de usuario intuitiva que incluye opciones para cambiar el estilo del texto, insertar imágenes, crear listas, etc... [https://ckeditor.com/ckeditor-5/]

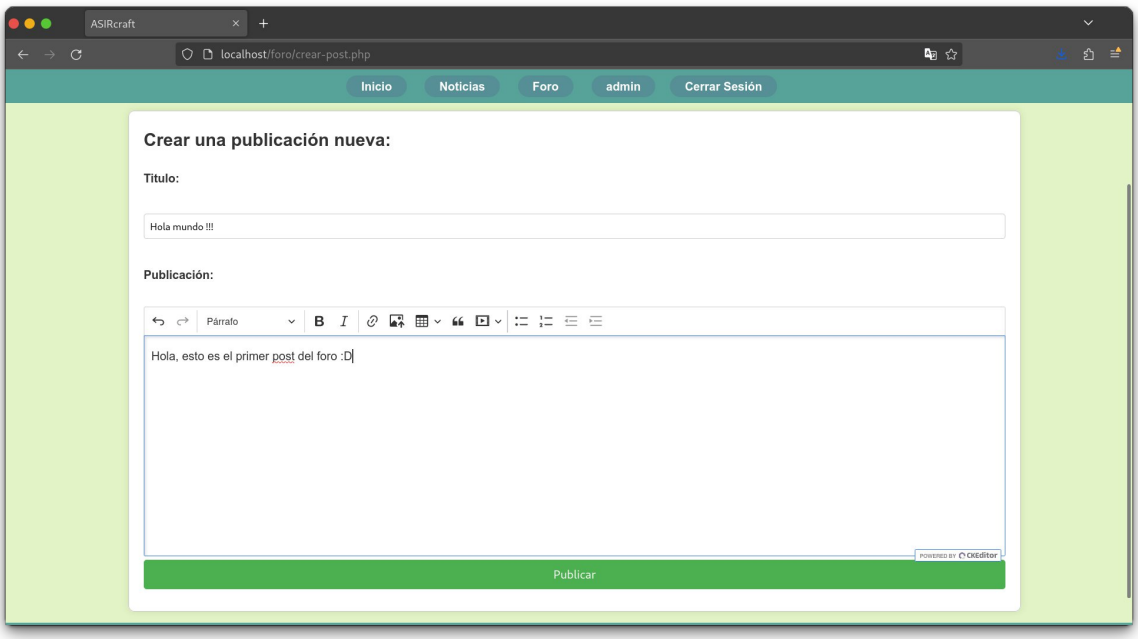
CKFinder es una herramienta complementaria de CKEditor que se utiliza para la gestión de archivos en aplicaciones web. Permite a los usuarios subir, organizar y gestionar archivos de forma sencilla a través de una interfaz intuitiva. [https://ckeditor.com/ckfinder/]



Foro

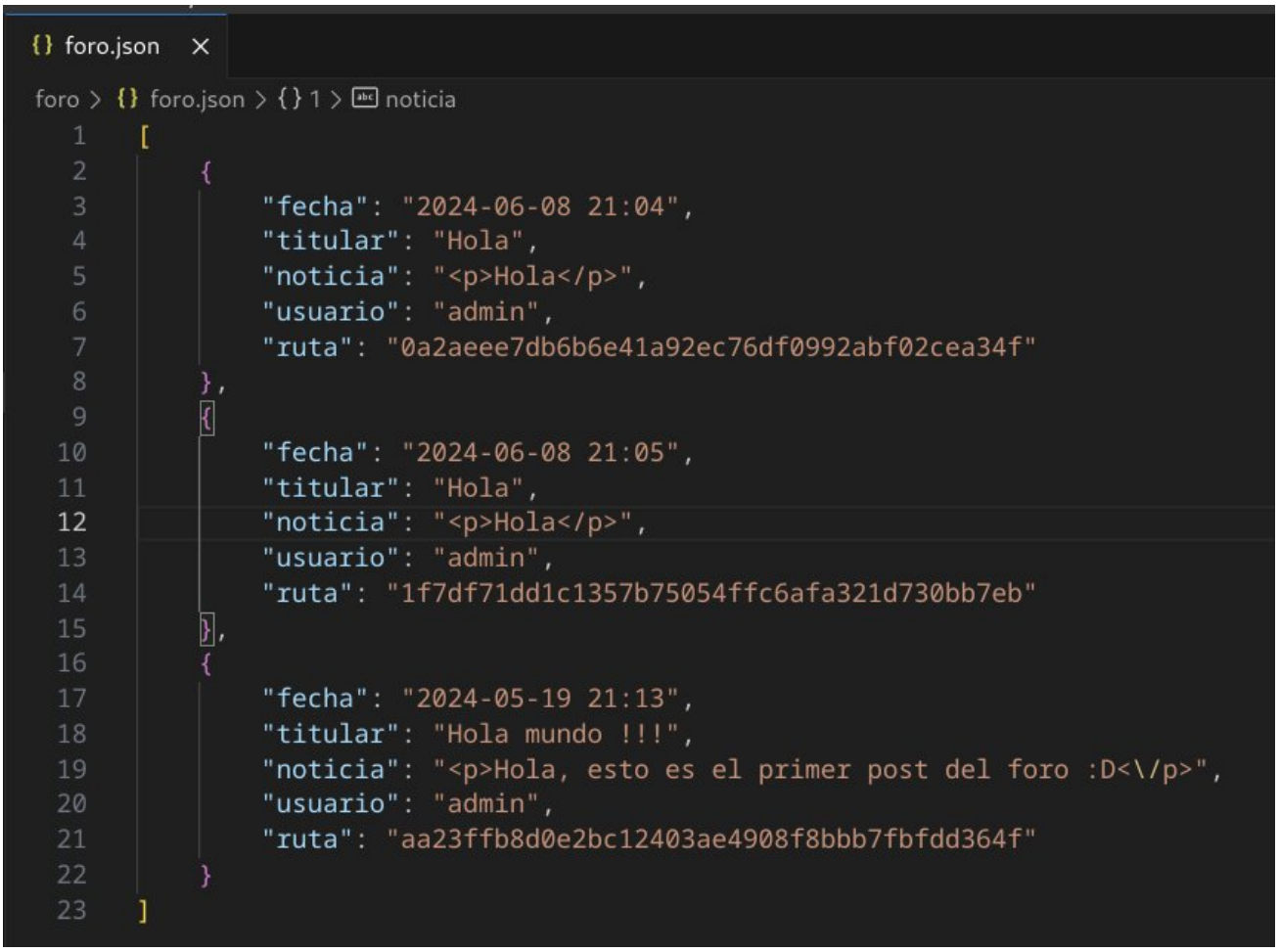


El foro es donde los usuarios idéntificados pueden crear Posts para que estos sean comentados por otros usuarios, la idea sería en un supuesto de que si por ejemplo un usuario encuentra algo en el juego o le ocurre cualquier tipo de problema técnico, pueda crear un post para pedir ayuda a otros usuarios. Para abrir el formulario de crear el post, tenemos que dar click en el “botón” de arriba de “Crear Post”

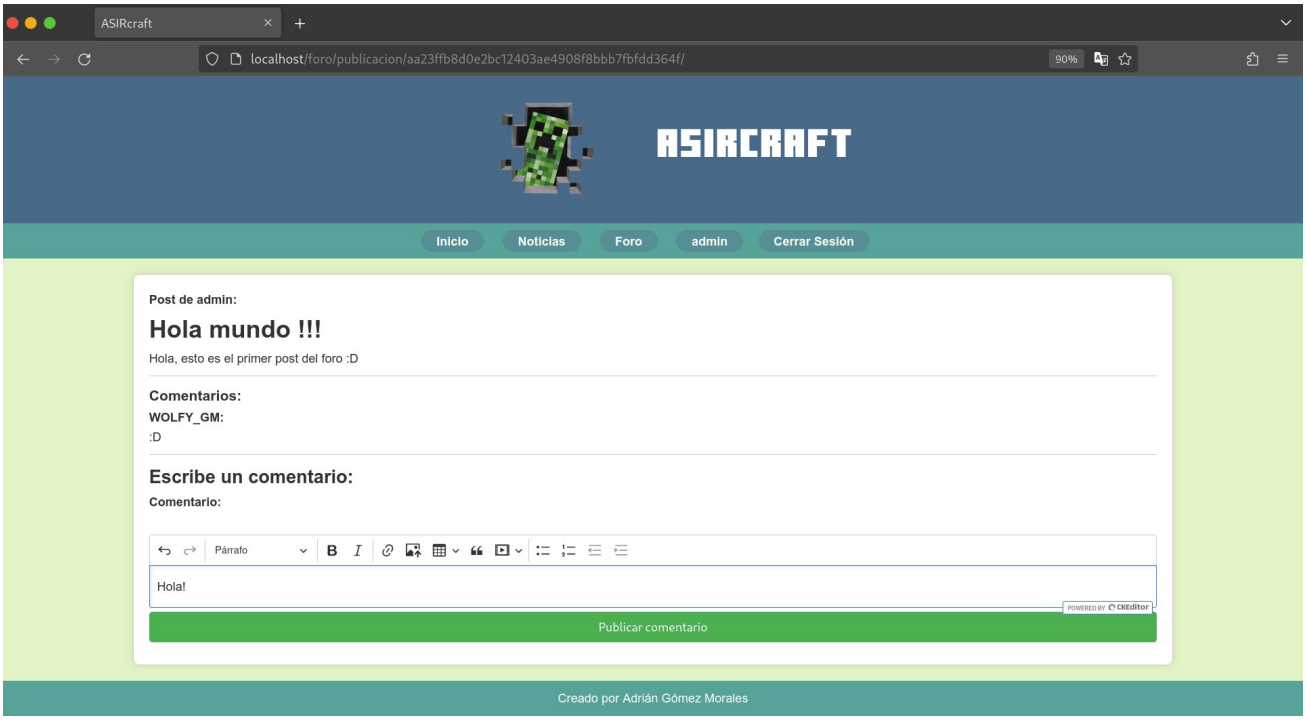


Al igual que con las noticias, CkEditor es usado como procesador de texto

Los posts se crean dentro del directorio publicacion con un hash que es la union de la fecha, titular y usuario cifrado en sha1, para evitar conflictos de varias carpetas con el mismo nombre en la misma ruta

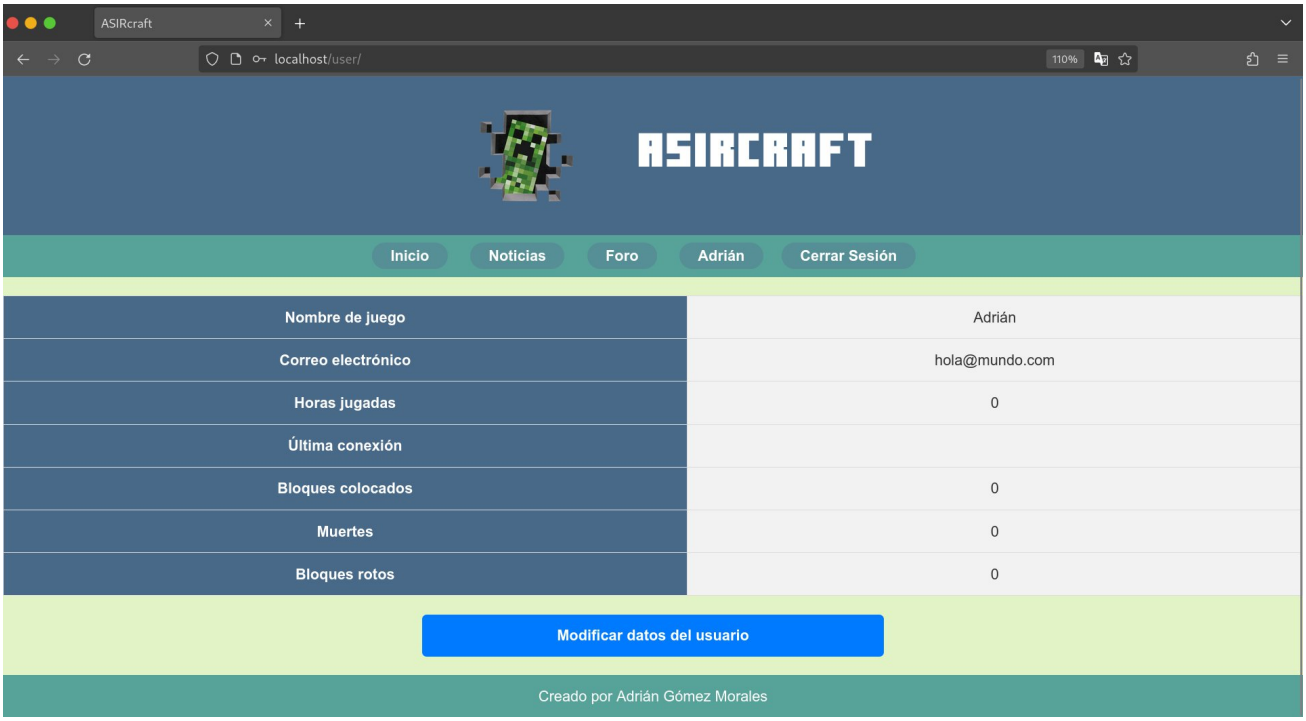


Este sería el json con la información del foro, aunque esto no guarda en sí el post del foro, es usado para crear el enlace al post y para filtrar usando el buscador de post, la página del post se crea usando una plantilla predefinida

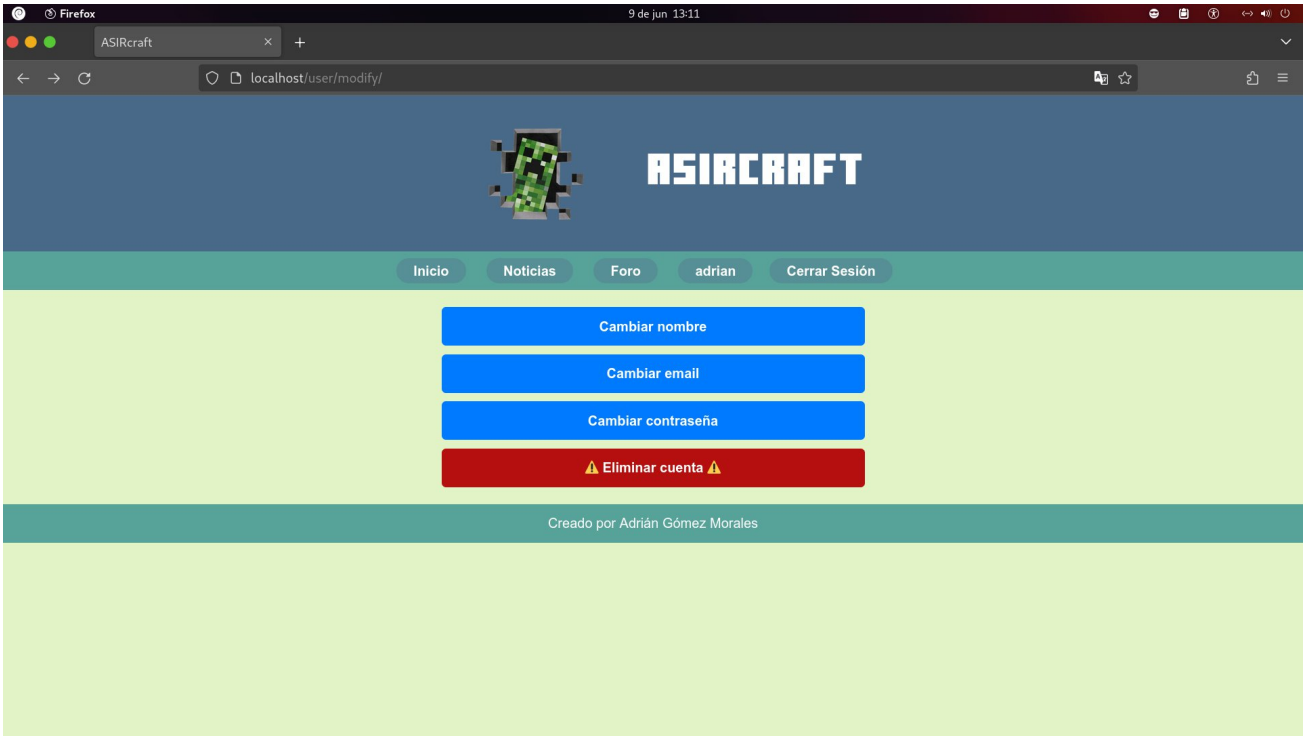


Si estás autenticado, podrás enviar comentarios al post, usando CkEditor, los comentarios se guardarán en un archivo json dentro del directorio de la publicación

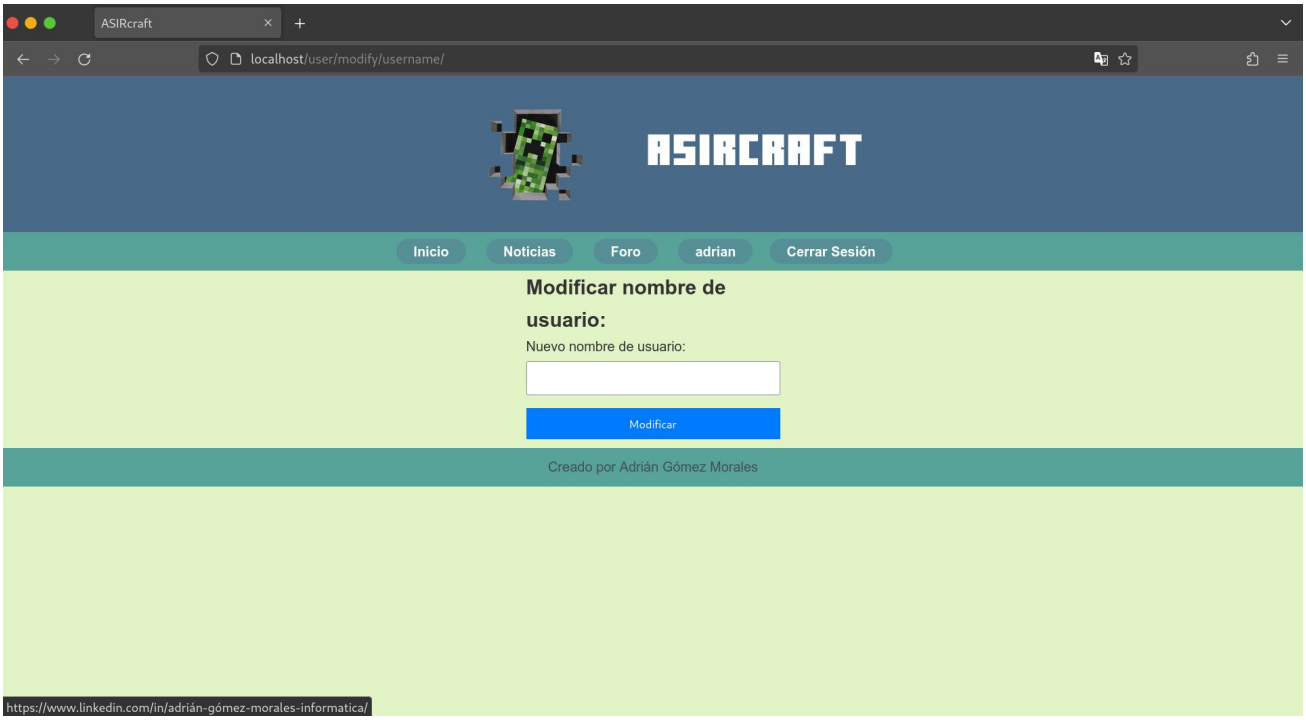
Área de Usuarios



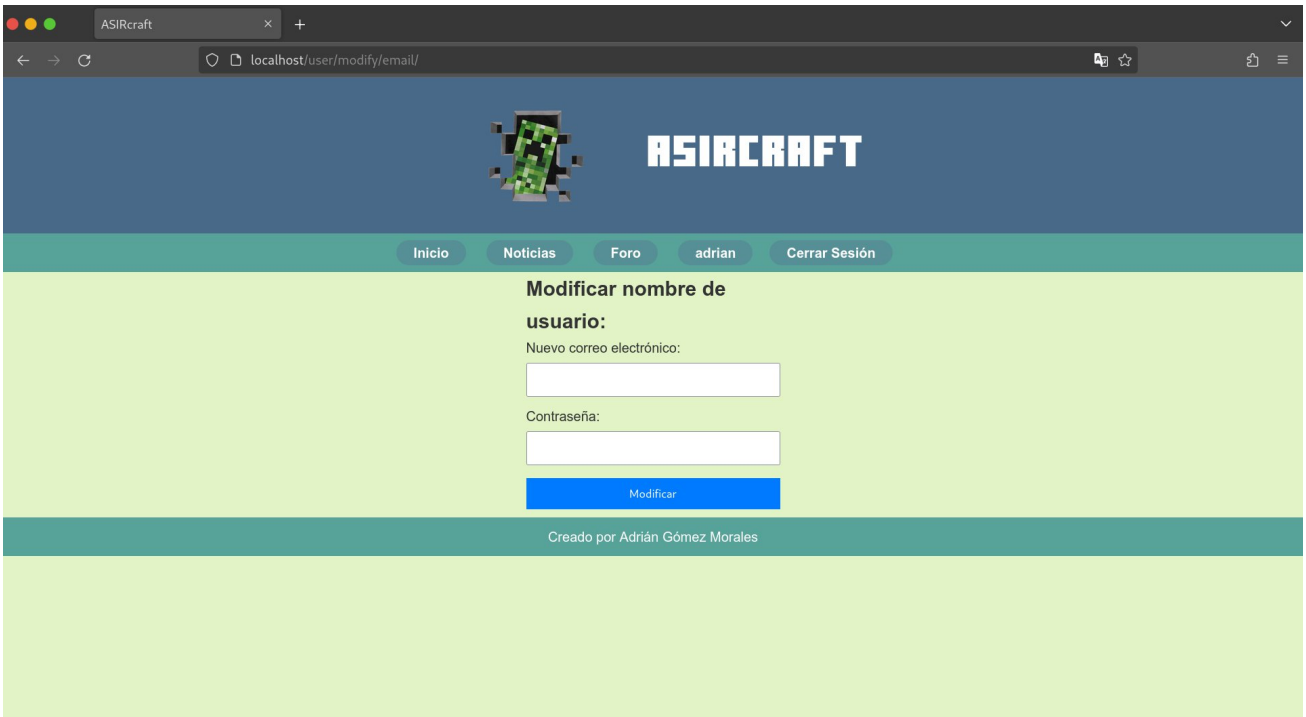
Este es el área de usuarios, en esta página, los usuarios podrán ver las estádisticas de su juego y además podrán acceder a modificar los datos de su propio usuario



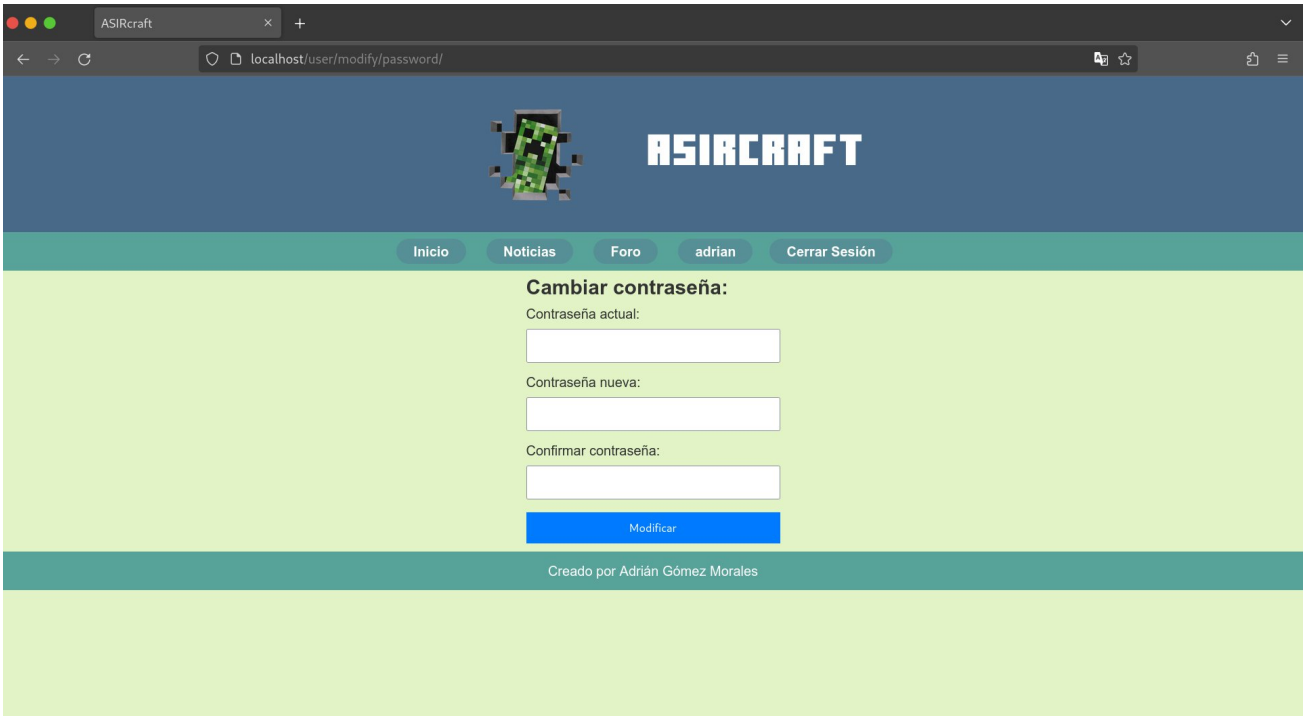
Si damos click en “Modificar datos del usuario” entraremos a un menú para cambiar los datos como el nombre, email o la contraseña, también podremos eliminar nuestra cuenta



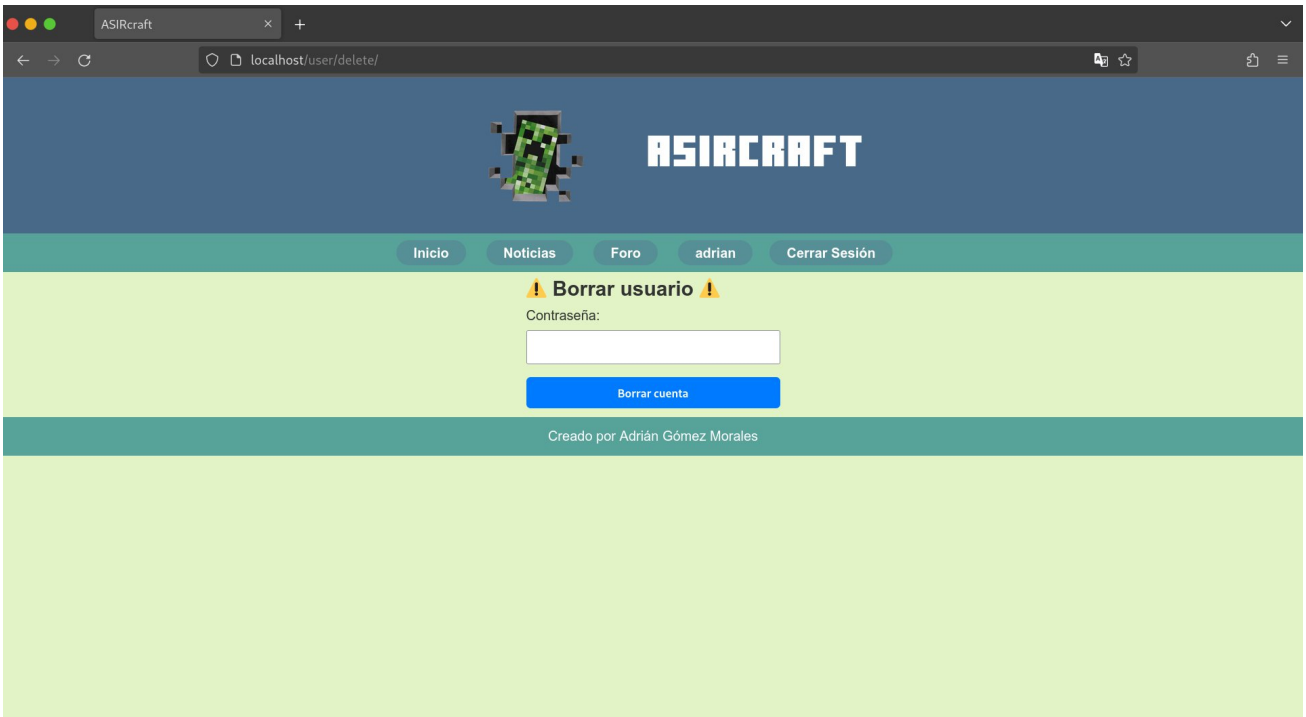
Si entramos en “Cambiar nombre”, entramos en una página con un formulario para actualizar el nuevo nombre_juego de la BBDD, antes de realizar el UPDATE en la BBDD, comprobamos que ese usuario no exista



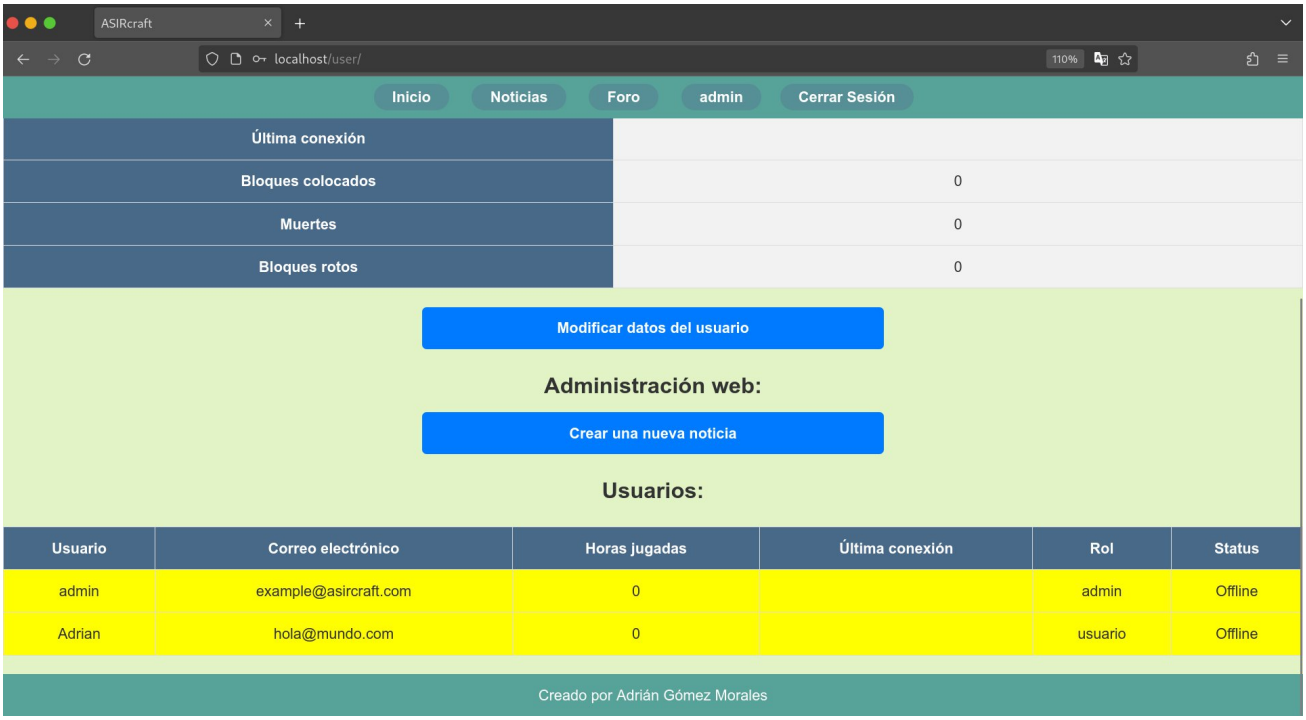
Si entramos a “Cambiar email”, accedemos a un formulario para cambiar nuestro correo, nos pedirá el nuevo correo y la contraseña que está usando el usuario, comprueba que la contraseña coincida con la almacenada en la BBDD y si es así, actualiza el correo



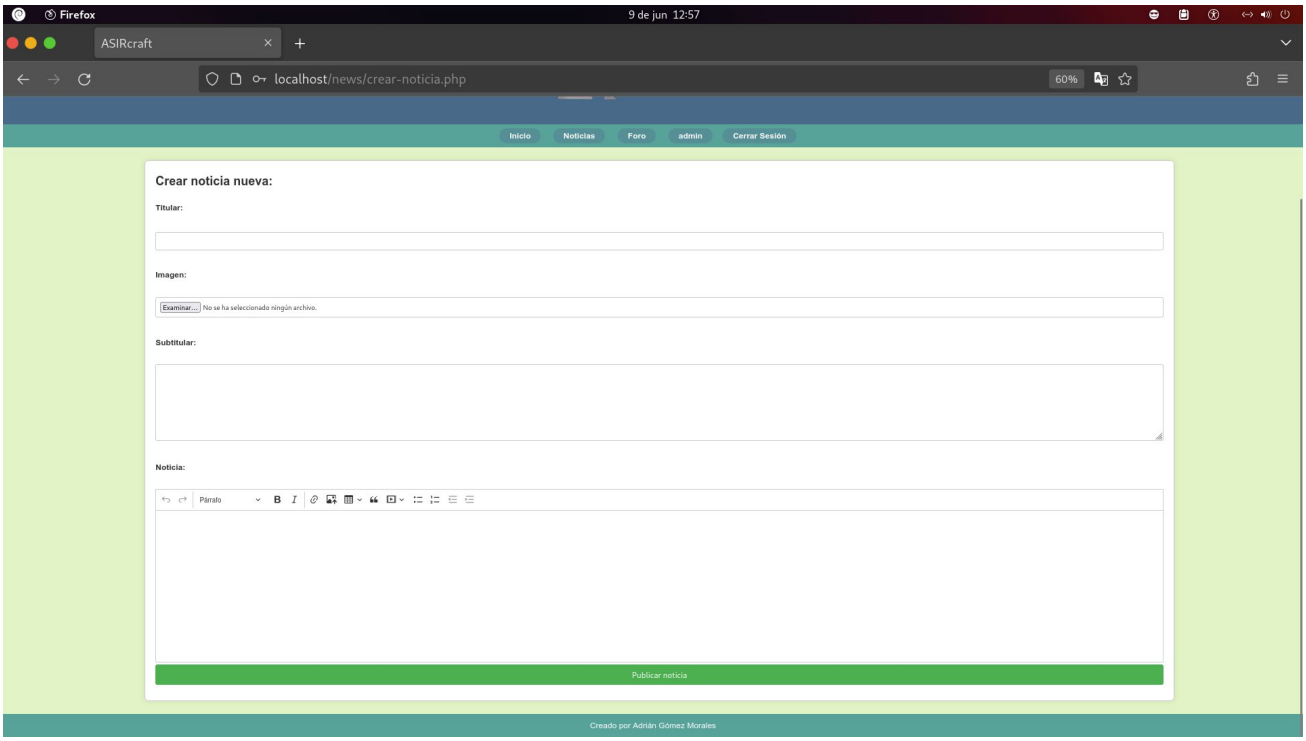
Si entramos al botón de “Cambiar contraseña”, accedemos a otro formulario donde nos pedirá la contraseña actual y dos veces la nueva contraseña, por detrás, comprobará que las dos contraseñas actuales coincidan y si es así, actualizará la contraseña en la BBDD



Si entramos a “Eliminar cuenta” entramos en un formulario que nos pedirá la contraseña actual y en caso de que coincida con la de la BBDD, el usurio se eliminará de ambas tablas

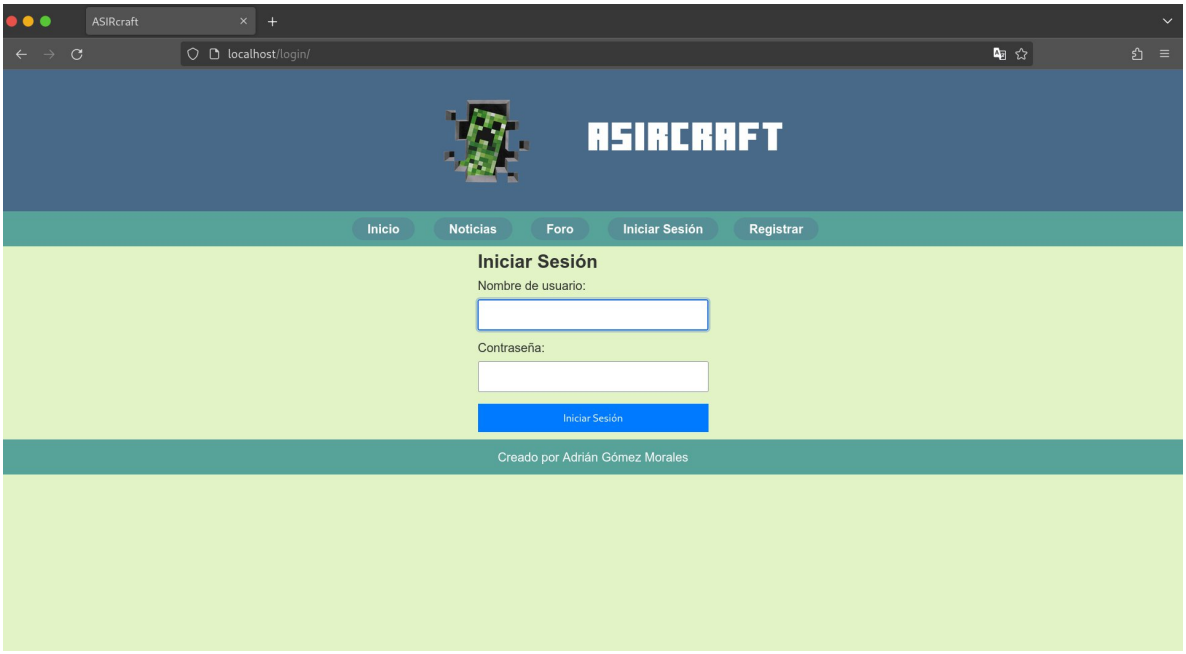


Volviendo al panel del usuario, si entramos al mismo panel pero como un usuario con rol de admin, podemos ver una lista de usuarios registrados, su correo, el número de horas jugadas, así como su última conexión rol y si están conectados o no, además tendremos acceso a crear noticias



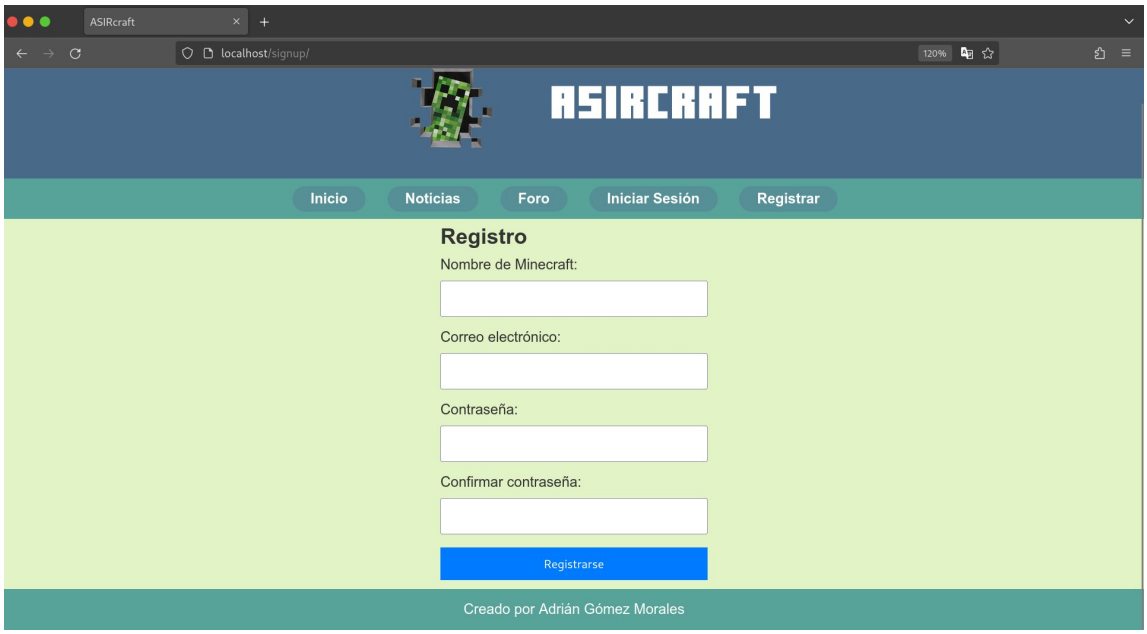
Si damos click en “Crear nueva noticia”, nos redigirá a una página con un formulariopara poder crear noticias, solo los usuarios administradores pueden entrar

Inciar Sesión



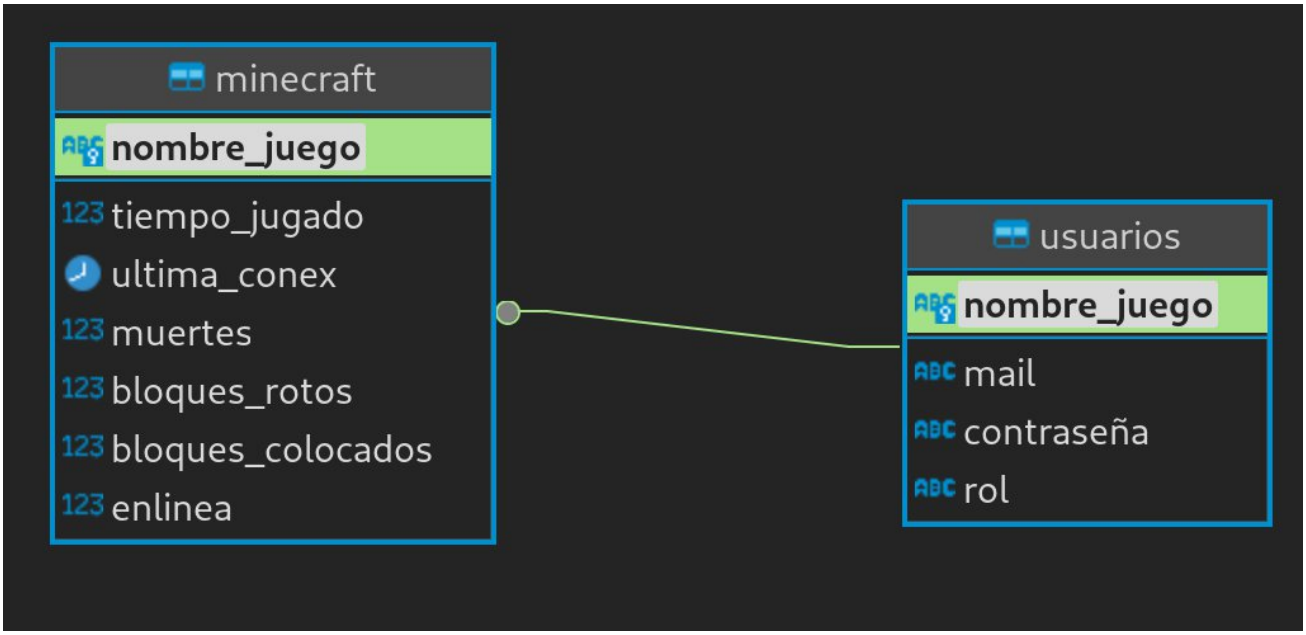
Para ello debemos de entrar en la página /login, donde nos pedirá el usuario y la contraseña en un formulario y en caso de que el usuario y la contraseña coincida con la de la BBDD, podremos iniciar sesión

Registrar Usuario



La página de registro se encuentra en /signup, nos pedirá un nombre de usuario (debe ser distinto a los que están guardados en la BBDD), un email, una contraseña y la confirmación de la contraseña, por defecto los usuarios que se registren mediante este formulario, se crearán con el rol de usuario

2. Base de datos MySQL



proyecto.sql
~/Escritorio

```sql
CREATE DATABASE IF NOT EXISTS proyecto;

USE proyecto;

CREATE TABLE IF NOT EXISTS usuarios (
    nombre_juego VARCHAR(255),
    mail VARCHAR(255),
    contrasea VARCHAR(255),
    rol VARCHAR(50),
    PRIMARY KEY (nombre_juego)
);

CREATE TABLE IF NOT EXISTS minecraft (
    nombre_juego VARCHAR(255) NOT NULL,
    tiempo_jugado INT DEFAULT 0,
    ultima_conex DATE DEFAULT NULL,
    muertes INT DEFAULT 0,
    bloques_rotos INT DEFAULT 0,
    bloques_colocados INT DEFAULT 0,
    enlinea SMALLINT NOT NULL DEFAULT 0,
    PRIMARY KEY (nombre_juego),
    FOREIGN KEY (nombre_juego) REFERENCES usuarios(nombre_juego)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);


CREATE USER IF NOT EXISTS 'user_proyect'@'%' IDENTIFIED WITH mysql_native_password BY 'H4-D14s*fV¡.4_P35k4R';

GRANT ALL PRIVILEGES ON proyecto.* TO 'user_proyect'@'%';

FLUSH PRIVILEGES;
```

Script de la base de datos, creará una base de datos si no existe llamada proyecto, que tendrá dos tablas, tabla web:

Tabla web

Esta tabla almacena información de los usuarios de la web con los siguientes campos:

nombre_juego (VARCHAR(255))

- Es el identificador único para cada usuario. Representa el nombre que el usuario utiliza en el juego
- Restricción: Clave primaria (PRIMARY KEY), lo que significa que cada valor debe ser único y not null

mail (VARCHAR(255))

- Almacena la dirección de correo electrónico del usuario

contrasea (VARCHAR(255))

- Contraseña que será usada para iniciar sesión por parte del usuario

  Nota: *No se puede usar la 'ñ', da problemas luego para kubernetes*

rol (VARCHAR(50))

- Indica el rol que tendrá el usuario (usuario o admin)

Tabla Minecraft

Esta tabla almacena la información relacionada con el juego en sí

nombre_juego (VARCHAR(255))

- Es el identificador único para cada usuario. Representa el nombre que el usuario utiliza en el juego
- Restricción: Clave primaria (PRIMARY KEY), lo que significa que cada valor debe ser único y not null y además es clave foránea de la tabla usuarios.

tiempo_jugado (INT)

- Usada para guardar el tiempo que pasa el jugador online, es de tipo númerico que por defecto asignará el valor de 0

Ultima_conex (DATE)

- Usada para guardar la última fecha a la que se conectó el usuario al juego, es de tipo fecha y por defecto es NULL

Bloques_rotos (INT)

- Usada para guardar el número de bloques que rompe el jugador, es de tipo númerico que por defecto asignará el valor de 0

Bloques_colocados (INT)

- Usada para guardar el número de bloques que coloca el jugador, es de tipo númerico que por defecto asignará el valor de 0

enlinea (SMALLINT)

- Usada para guardar el estado dell jugador, 1 representa online y 0 representa que está offline, es de tipo númerico que por defecto asignará el valor de 0

ON DELETE/UPDATE CASCADE

La cláusula ON DELETE y ON UPDATE con la opción CASCADE en bases de datos SQL se utiliza para mantener la integridad referencial entre las tablas. La integridad referencial asegura que los valores en una tabla (la tab hija) siempre sean válidos en relación con los valores en otra tabla (la tab padre). Cuando se realiza una operación de eliminación o actualización en la tabla padre, estas opciones permiten que automáticamente se realicen cambios correspondientes en la tabla hija.

Usuario de la Base de datos

Creamos un usuario que permita la conexión desde cualquier IP (“%”), esto se debe a que no sabemos que IP tendrá los distintos servicios que usarán la BBDD. Además asignamos todos los privilegios al usuario para todas las tablas de esa BBDD

3. Plugin para Minecraft Server usando Java

Importación de librerías

Verysimplesqllogin.java

```java
package es.adriangomezmorales.verysimplesqllogin;

import org.bukkit.command.Command;
import org.bukkit.command.CommandSender;
import org.bukkit.command.CommandExecutor;
import org.bukkit.entity.Player;
import org.bukkit.event.EventHandler;
import org.bukkit.event.Listener;
import org.bukkit.event.entity.PlayerDeathEvent;
import org.bukkit.event.player.PlayerMoveEvent;
import org.bukkit.event.player.PlayerJoinEvent;
import org.bukkit.event.player.PlayerQuitEvent;
import org.bukkit.plugin.java.JavaPlugin;
import org.bukkit.event.block.BlockBreakEvent;

import java.sql.*;
import java.util.HashMap;
import java.util.Map;
import java.util.UUID;
```

Se importa las librerías necesarias para trabajar con Bukkit API (La API de minecraft server) y JDBC para las conexiones a la base de datos

Clase principal

```java
public final class Verysimplesqllogin extends JavaPlugin implements CommandExecutor, Listener {  no usages
    private Map<UUID, Boolean> sesionesJugador = new HashMap<>();  7 usages
    private Connection conexion;  12 usages
```

Define la clase principal Verysimplesqllogin que extiende JavaPlugin, implementando CommandExecutor y Listener para manejar comandos y eventos de Minecraft. Declara un mapa sesionesJugador para rastrear las sesiones de los jugadores y una conexión conexion a la base de datos. El resto del código está dentro de esta clase principal

Método onEnable

```
@Override  no usages
public void onEnable() {
    getLogger().info( msg: "Inicio del plugin de login SQL simple por Adrian Gomez Morales");
    getServer().getPluginManager().registerEvents( listener: this,  plugin: this);
    getCommand( name: "login").setExecutor(this);

    // Conexión a la base de datos
    try {
        conexion = DriverManager.getConnection( url: "jdbc:mysql://mysql:3306/proyecto",  user: "user_proyect",  password: "H4-D14s*fV¡.4_P35k4R");
    } catch (SQLException e) {
        getLogger().severe( msg: "Error al conectar a la base de datos: " + e.getMessage());
    }
}
```

El método onEnable() de los plugins de Minecraft Server hace referencia al momento de cargarse el plugin dentro del servidor, que por norma general es cuando se está iniciando el propio servidor de juego.

getLogger().info(“Hola”);

Sirve para registrar a “Hola” dentro del log de Minecraft

getServer().getPluginManager().registerEvents(this, this);

Indica que el plugin es un listener de eventos. El método registerEvents del PluginManager se utiliza para que este plugin (representado por this) pueda manejar eventos que ocurren en el servidor. El primer this se refiere al listener que maneja los eventos, y el segundo this se refiere al propio plugin.

getCommand("login").setExecutor(this);

Establece que el comando "login" del plugin será gestionado por este mismo objeto. setExecutor(this) significa que la clase actual implementa la interfaz commandExecutor, lo que permite manejar lo que ocurre cuando un jugador usa el comando "/login" en el juego.

try{

conexion = DriverManager.getConnection("jdbc:mysql://mysql:3306/proyecto", "user_proyect", "H4-D14s*fV¡.4_P35k4R");

}

Intenta establecer una conexión a MySQL. Usa la clase DriverManager para obtener una conexión (getConnection) a la base de datos ubicada en mysql (Nombre de DNS del servicio mysql de Kubernetes que crearemos posteriormente) en el puerto 3306, usando el usuario y su contraseña. La conexión establecida se asigna a la variable conexion.

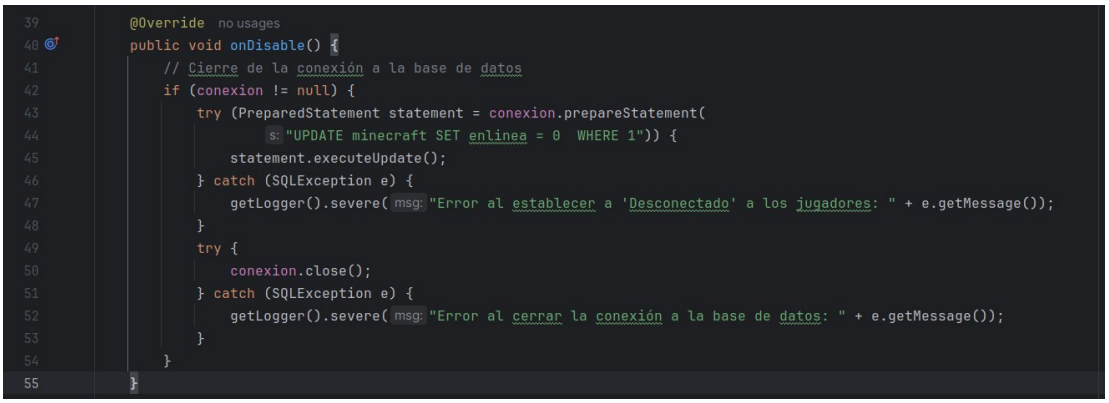
catch (SQLException e) {

getLogger().severe("Error al conectar a la base de datos: " + e.getMessage());

}

Si ocurre una excepción, se guarda y se registra en el log del servidor con getLogger().severe.

Método onDisable



onDisable() hace referencia a cuando el plugin se desactiva del servidor que por norma general es cuando el servidor se apaga o reinicia

if (conexion != null) {

Comprueba que la conexion a la BBDD se estableció anteriormente y en el caso contrario no hará nada. Si la condicion conexion != null, es true, entonces ejecutará lo siguiente (En todas los try hay catch que recogen el error y lo almacenan en el log si algo va mal durante la ejecución):

try (PreparedStatement statement = conexion.prepareStatement(

"UPDATE minecraft SET enlinea = 0 WHERE 1")) {

statement.executeUpdate();

Como es cuando por norma general se apaga, debemos de hacer que todos los usuarios que estaban marcados como “Online” deben de pasar a “Offline” ya que si el servidor se apaga o se reinicia, los jugadores se desconectarán del servidor

try {
conexion.close();
}

Esta línea cierra la conexión a la BBBDD

Manejo del evento PlayerJoinEvent

Este evento hace referencia a cuando un jugador entra al servidor

Player jugador = event.getPlayer();

Este código obtiene el objeto Player que representa al jugador que se acaba de unir al servidor.

sesionesJugador.put(jugador.getUniqueId(), false);

Agrega la UUID (Identificador Único Universal) del jugador al mapa sesionesJugador con un valor de false, indicando que el jugador no ha iniciado sesión todavía.

jugador.sendMessage("Usa /login [contraseña] para iniciar sesión");

El jugador recibe por el chat del juego "Usa /login [contraseña] para iniciar sesión" y solo lo recibe ese jugador en concreto

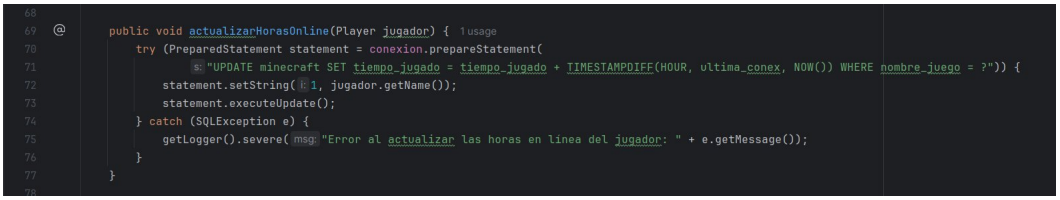
if (!estaJugadorRegistrado(jugador)) {

jugador.kickPlayer("¡Por favor, regístrate en nuestra web!");

}

Aquí se verifica si el jugador está registrado en la base de datos utilizando el método estaJugadorRegistrado que se definirá más adelante. Si el jugador no está registrado, es expulsado del servidor.

Método actualizarHorasOnline



Este método se usa para actualizar el tiempo de juego de un jugador en la base de datos.

El Parametro jugador es una instancia de la clase Player, que representa al jugador cuyo tiempo de juego se desea actualizar.

try (PreparedStatement statement = conexion.prepareStatement(

"UPDATE minecraft SET tiempo_jugado = tiempo_jugado + TIMESTAMPDIFF(HOUR, ultima_conex, NOW()) WHERE nombre_juego = ?")) {

Aquí se prepara una consulta SQL para actualizar la columna tiempo_jugado en la tabla minecraft. La consulta utiliza la función TIMESTAMPDIFF para calcular la diferencia en horas (HOUR) entre ultima_conex (la última vez que el jugador se conectó) y NOW() (el momento actual).

statement.setString(1, jugador.getName());

Aquí se establece el valor del parámetro nombre_juego en la consulta SQL. El nombre del jugador se obtiene utilizando jugador.getName()

statement.executeUpdate();

Esta línea ejecuta la consulta

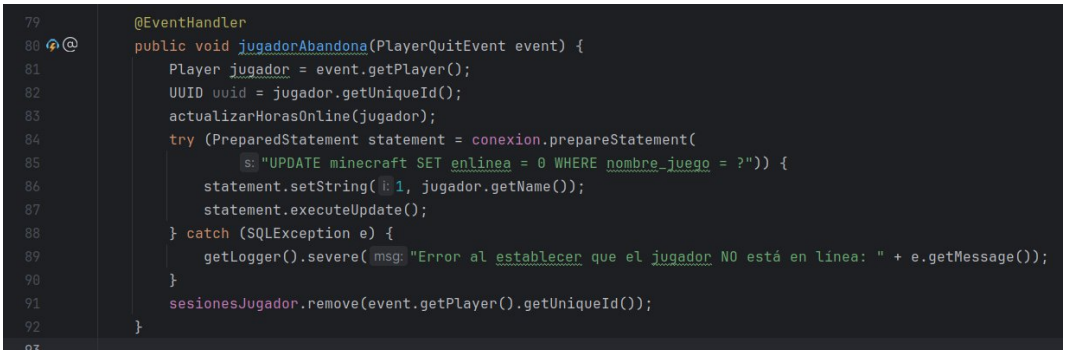
} catch (SQLException e) {

getLogger().severe("Error al actualizar las horas en línea del jugador: " + e.getMessage());

}

Si ocurre una excepción SQL se registra y se guarda en el log

Manejo del evento PlayerQuitEvent



```
@EventHandler
public void jugadorAbandona(PlayerQuitEvent event) {
    Player jugador = event.getPlayer();
    UUID uuid = jugador.getUniqueId();
    actualizarHorasOnline(jugador);
    try (PreparedStatement statement = conexion.prepareStatement(
            s: "UPDATE minecraft SET enlinea = 0 WHERE nombre_juego = ?")) {
        statement.setString( i: 1, jugador.getName());
        statement.executeUpdate();
    } catch (SQLException e) {
        getLogger().severe( msg: "Error al establecer que el jugador NO está en línea: " + e.getMessage());
    }
    sesionesJugador.remove(event.getPlayer().getUniqueId());
}
```

Este método es un manejador de eventos (event handler) que se ejecuta cada vez que un jugador se desconecta del servidor

Player jugador = event.getPlayer();

UUID uuid = jugador.getUniqueId();

Obtiene el objeto Player que representa al jugador que se está desconectando y su UUID.

actualizarHorasOnline(jugador);

Llama al método actualizarHorasOnline con el parámetro jugador

```
try (PreparedStatement statement = conexion.prepareStatement(
        "UPDATE minecraft SET enlinea = 0 WHERE nombre_juego = ?")) {
    statement.setString(1, jugador.getName());
    statement.executeUpdate();
} catch (SQLException e) {
    getLogger().severe("Error al establecer que el jugador NO está en línea: " +
e.getMessage());
}
```

En esta sección, se intentará ejecutar un update en la tabla minecraft para poner enlinea con valor 0, que significa offline al jugador que deja el servidor y en caso de que falle, registrará la excepción y la guardará en el log

sesionesJugador.remove(event.getPlayer().getUniqueId());

Esta línea elimina la sesión del jugador que abandona

Manejo del evento PlayerMoveEvent

```
@EventHandler
public void jugadorSeMueve(PlayerMoveEvent event) {
    Player jugador = event.getPlayer();
    UUID uuid = jugador.getUniqueId();
    if (sesionesJugador.containsKey(uuid) && !sesionesJugador.get(uuid)) {
        event.setCancelled(true); // Cancela el movimiento mientras no haya iniciado sesión
    }
}
```

Este bloque está destinado a cancelar el movimiento del jugador en caso de que no haya iniciado sesión

Player jugador = event.getPlayer();

UUID uuid = jugador.getUniqueId();

Obtiene el objeto Player que representa al jugador que se está intentando mover y su UUID.

if (sesionesJugador.containsKey(uuid) && !sesionesJugador.get(uuid)) {
event.setCancelled(true);
}

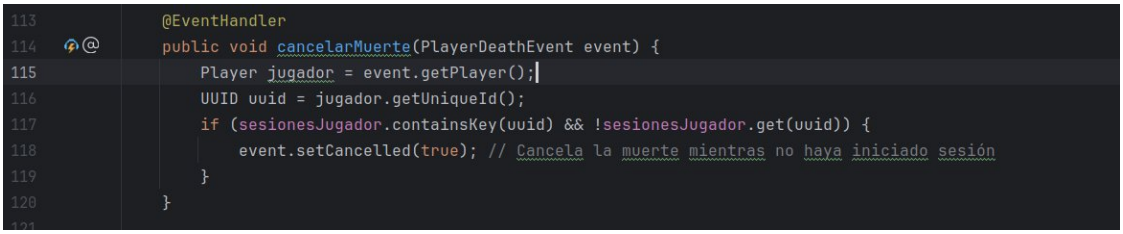
Comprueba si el uuid del jugador está presente en el mapa sesionesJugador y en caso de que no sea así, cancelará el evento, es decir, no se podrá mover y por lo tanto no podrá jugar

Manejo del evento PlayerDropItemEvent

```
@EventHandler
public void soltarItem(PlayerDropItemEvent event) {
    Player jugador = event.getPlayer();
    UUID uuid = jugador.getUniqueId();
    if (sesionesJugador.containsKey(uuid) && !sesionesJugador.get(uuid)) {
        event.setCancelled(true); // Cancela el dropeo de items mientras no haya iniciado sesión
    }
}
```

Prácticamente hace lo mismo que el anterior, solo que en lugar de cancelar el evento de moverse, cancela el evento de tirar un objeto del inventario del jugador, para evitar que otra persona pueda intentar conectarse como el jugador original y tirar su inventario

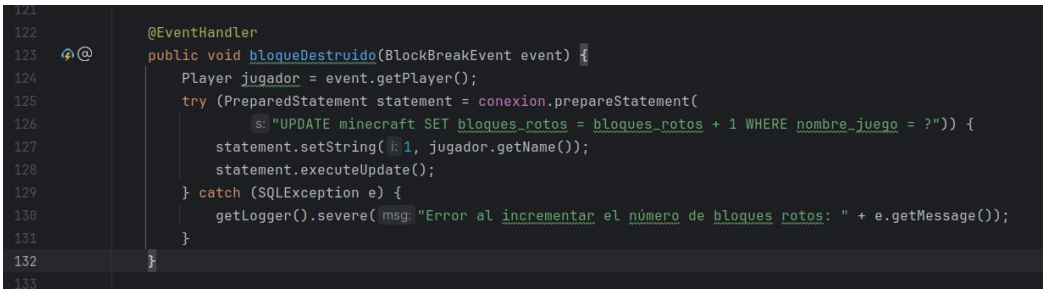
Manejo del evento PlayerDeathEvent



```
@EventHandler
public void cancelarMuerte(PlayerDeathEvent event) {
    Player jugador = event.getPlayer();
    UUID uuid = jugador.getUniqueId();
    if (sesionesJugador.containsKey(uuid) && !sesionesJugador.get(uuid)) {
        event.setCancelled(true); // Cancela la muerte mientras no haya iniciado sesión
    }
}
```

Más o menos lo mismo que los otros dos, pero esta vez cancela el evento de muerte del jugador, evitando que pueda morir mientras aún no ha iniciado sesión

Manejo del evento BlockBreakEvent



```
@EventHandler
public void bloqueDestruido(BlockBreakEvent event) {
    Player jugador = event.getPlayer();
    try (PreparedStatement statement = conexion.prepareStatement(
            s: "UPDATE minecraft SET bloques_rotos = bloques_rotos + 1 WHERE nombre_juego = ?")) {
        statement.setString( i: 1, jugador.getName());
        statement.executeUpdate();
    } catch (SQLException e) {
        getLogger().severe( msg: "Error al incrementar el número de bloques rotos: " + e.getMessage());
    }
}
```

Esta sección se dedica a actualizar la BBDD en la tabla minecraft para contar cuantos bloques destruye el jugador

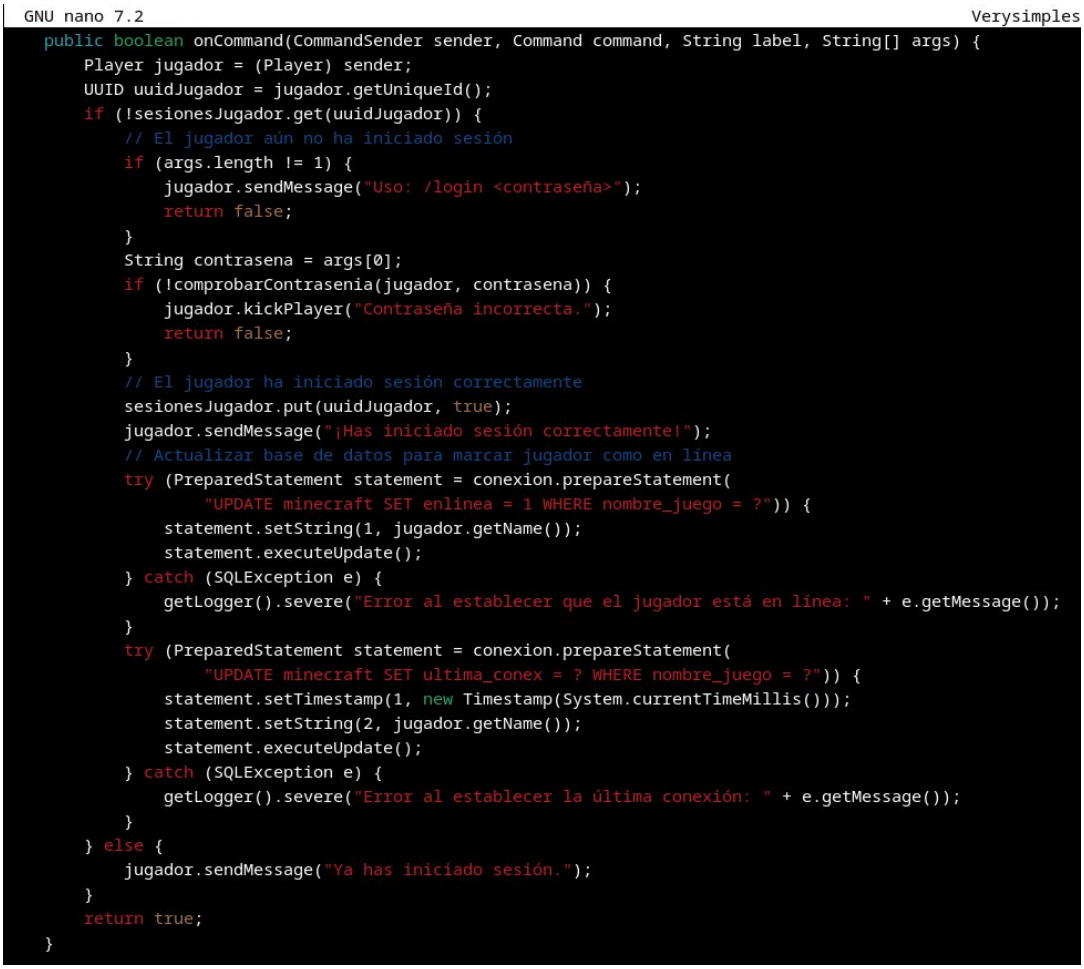
Player jugador = event.getPlayer();

Obtiene el jugador que activa el evento

```
try (PreparedStatement statement = conexion.prepareStatement(
      "UPDATE minecraft SET bloques_rotos = bloques_rotos + 1 WHERE nombre_juego = ?")) {
   statement.setString(1, jugador.getName());
   statement.executeUpdate();
} catch (SQLException e) {
   getLogger().severe("Error al incrementar el número de bloques rotos: " + e.getMessage());
}
```

Esta sección intentará actualizar en la BBDD la tabla minecraft sumando 1 a los bloques_rotos donde el nombre_juego es el jugador que ha activado el evento, en caso de excepción, se guardará en el log del servidor

Método onCommand



Maneja el comando /login ejecutado por el jugador, comprueba la contraseña y también maneja las posibles excepciones y las registra en el log

public boolean onCommand(CommandSender sender, Command command, String label, String[] args) {

public boolean onCommand: Define un método que devuelve un booleano.

CommandSender sender: El objeto que envió el comando. Puede ser un jugador u otro tipo de emisor de comandos.

Command command: El comando que se ejecuta.

String label: La etiqueta del comando.

String[] args: Los argumentos del comando.

Player jugador = (Player) sender;

Convierte el sender a un objeto Player, asumiendo que el comando solo puede ser ejecutado por un jugador.

UUID uuidJugador = jugador.getUniqueId();

Obtiene el UUID único del jugador.

if (!sesionesJugador.get(uuidJugador)) {

Primero se verifica que la sesión no haya sido creada

if (args.length != 1) {

jugador.sendMessage("Uso: /login <contraseña>");

return false;

}

Verifica que el argumento si es distinto a uno, significa que o no ha enviado la contraseña o que ha envíado como mínimo 2, así que se entiende que el jugador ha cometido un error y se explica como se usa el comando

String contrasena = args[0];

Guardamos el primer argumento del comando en un string contrasena

if (!comprobarContrasenia(jugador, contrasena)) {

jugador.kickPlayer("Contraseña incorrecta.");

return false;

}

Llama a un método para verificar la contraseña. Si es incorrecta, expulsa al jugador con un mensaje de que su contraseña es incorrecta y devuelve false

sesionesJugador.put(uuidJugador, true);

Marca al jugador como que ha iniciado sesión

jugador.sendMessage("¡Has iniciado sesión correctamente!");

Envía al jugador que ha iniciado sesión correctamente

```
try (PreparedStatement statement = conexion.prepareStatement(
        "UPDATE minecraft SET enlinea = 1 WHERE nombre_juego = ?")) {
    statement.setString(1, jugador.getName());
    statement.executeUpdate();
} catch (SQLException e) {
    getLogger().severe("Error al establecer que el jugador está en línea: " +
e.getMessage());
}
```

Después de marcar al jugador como que ha iniciado sesión, hará un update en la base de datos para marcar que está en línea, si hay un excepción, se enviará al log

```
try (PreparedStatement statement = conexion.prepareStatement(
        "UPDATE minecraft SET ultima_conex = ? WHERE nombre_juego = ?")) {
    statement.setTimestamp(1, new Timestamp(System.currentTimeMillis()));
    statement.setString(2, jugador.getName());
    statement.executeUpdate();
} catch (SQLException e) {
    getLogger().severe("Error al establecer la última conexión: " +
e.getMessage());
}
```

Este bloque de código registra la última vez que un jugador se conectó al servidor haciendo un UPDATE en la base de datos, también, en caso de excepción, se envía al log del servidor

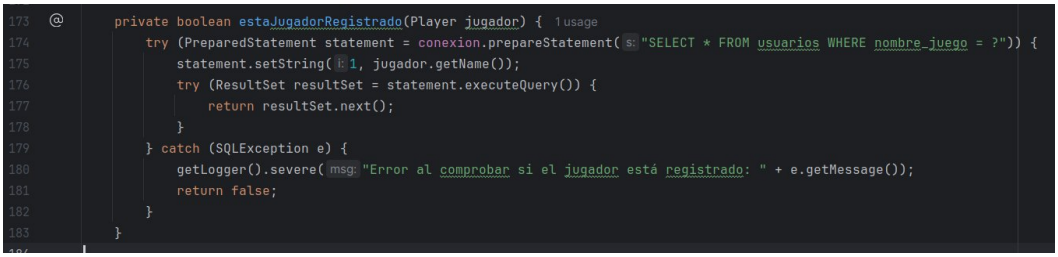
```
} else {
    jugador.sendMessage("Ya has iniciado sesión.");
}
```

Si el jugador vuelve a usar /login <contraseña> después de haber iniciado sesión, recibirá el mensaje de que ya había iniciado sesión

```
return true;

}
```

Devuelve true para indicar que el comando se ejecutó correctamente

Método estaJugadorRegistrado



Este método privado comprueba de que el jugador exista en la base de datos y por ende esté registrado

private boolean estaJugadorRegistrado(Player jugador) {

Método privado que se le indica un objeto (player) como parámetro (jugador) y devuelve true or false

try (PreparedStatement statement = conexion.prepareStatement("SELECT * FROM usuarios WHERE nombre_juego = ?")) {

En esta línea se prepara la consulta SQL que seleccionará todos los datos de la tabla usuarios donde el nombre_juego será

statement.setString(1, jugador.getName());

El nombre del jugador que hemos pasado antes como parámetro

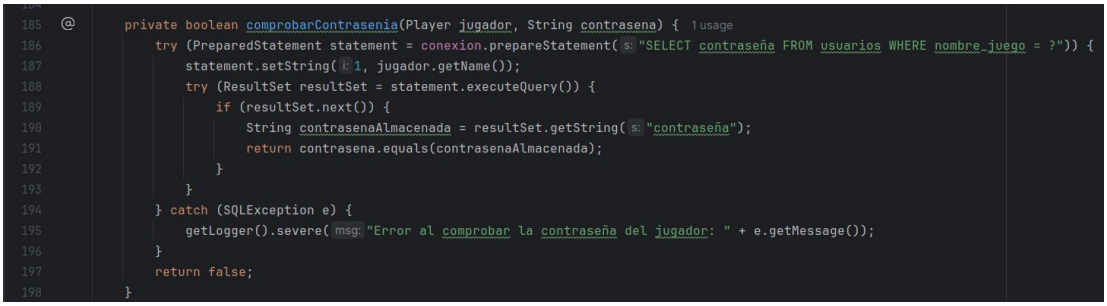
```
try (ResultSet resultSet = statement.executeQuery()) {
   return resultSet.next();
}
```

Este fragmento ejecuta la consulta SQL y verifica si el resultado contiene al menos una fila para mirar si un jugador está registrado en la base de datos

```
} catch (SQLException e) {

        getLogger().severe("Error al comprobar si el jugador está registrado: " + e.getMessage());

        return false;

    }

}
```

En caso de excepción, se guarda en el log y devuelve false

Método comprobarContrasenia



```
185  @    private boolean comprobarContrasenia(Player jugador, String contrasena) {  1 usage
186           try (PreparedStatement statement = conexion.prepareStatement( s: "SELECT contraseña FROM usuarios WHERE nombre_juego = ?")) {
187               statement.setString( i: 1, jugador.getName());
188               try (ResultSet resultSet = statement.executeQuery()) {
189                   if (resultSet.next()) {
190                       String contrasenaAlmacenada = resultSet.getString( s: "contraseña");
191                       return contrasena.equals(contrasenaAlmacenada);
192                   }
193               }
194           } catch (SQLException e) {
195               getLogger().severe( msg: "Error al comprobar la contraseña del jugador: " + e.getMessage());
196           }
197           return false;
198       }
```

Este método que es llamado en el método comando, para verificar si la contraseña proporcionada por el jugador coincide con la almacenada en la base de datos.

private boolean comprobarContrasenia(Player jugador, String contrasena) {

Método privado que devuelve true o false. Toma como parámetros un objeto Player, que es el jugador que ejecutará el comando y un string contrasena, que es el parámetro 0 del comando

try (PreparedStatement statement = conexion.prepareStatement("SELECT contraseña FROM usuarios WHERE nombre_juego = ?")) {
statement.setString(1, jugador.getName());

Prepara para la consulta SQL que seleccionará la contraseña de la tabla usuarios donde la columna nombre_juego coincide con el nombre del jugador.

try (ResultSet resultSet = statement.executeQuery()) {

Ejecuta la consulta y guarda el resultado en un ResultSet

if (resultSet.next()) {
String contrasenaAlmacenada = resultSet.getString("contraseña");
return contrasena.equals(contrasenaAlmacenada);
}
}

Con resultSet.next() verifica si hay al menos una fila en el resultado de la consulta y si hay al menos una fila, guarda la contraseña en un string llamado contrasenaAlmacenada

return contrasena.equals(contrasenaAlmacenada): Compara la contraseña proporcionada (contrasena) con la contraseña de la BBDD (contrasenaAlmacenada). Devuelve true si son iguales, de lo contrario false.

```
    } catch (SQLException e) {
        getLogger().severe("Error al comprobar la contraseña del jugador: " +
e.getMessage());
    }
```

En caso de excepción SQL, se guardará en el log del servidor

```
    return false;
}
```

Si la contraseña no coincide o hay excepción SQL, se devuelve false

Manejo del evento PlayerDeathEvent

```
@EventHandler
public void muerteJugador(PlayerDeathEvent event) {
    Player jugador = event.getEntity();
    try (PreparedStatement statement = conexion.prepareStatement(
            s: "UPDATE minecraft SET muertes = muertes + 1 WHERE nombre_juego = ?")) {
        statement.setString( i: 1, jugador.getName());
        statement.executeUpdate();
    } catch (SQLException e) {
        getLogger().severe( msg: "Error al incrementar el número de muertes del jugador: " + e.getMessage());
    }
}
```

Método público que se ejecuta cuando ocurre un evento de tipo PlayerDeathEvent y no devuelve nada ya que es un public void

Player jugador = event.getEntity();

Obtiene el jugador que muere

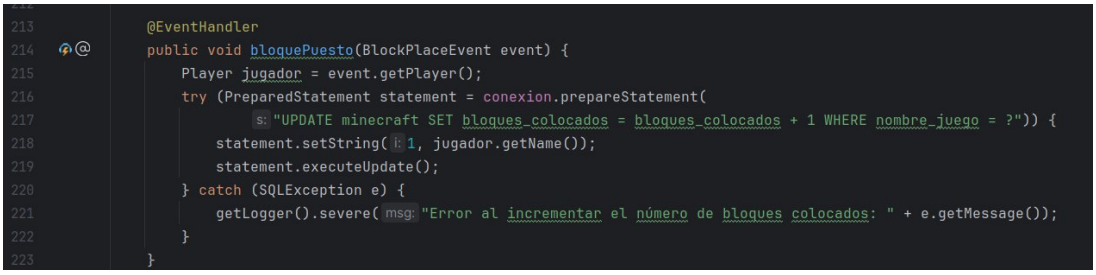
```
try (PreparedStatement statement = conexion.prepareStatement(
        "UPDATE minecraft SET muertes = muertes + 1 WHERE nombre_juego
= ?")) {
    statement.setString(1, jugador.getName());
    statement.executeUpdate();
}
```

Intentará ejecutar la sentencia SQL que sumará +1 al número que ya tenga la BBDD en la columna muertes de la tabla minecraft donde el nombre_juego es el nombre del jugador que ha muerto

```
catch (SQLException e) {
            getLogger().severe("Error al incrementar el número de muertes del
jugador: " + e.getMessage());
        }
    }
}
```

En caso de excepción SQL, se registrará en el log del servidor

Manejo del evento BlockPlaceEvent



```
@EventHandler
public void bloquePuesto(BlockPlaceEvent event) {
    Player jugador = event.getPlayer();
    try (PreparedStatement statement = conexion.prepareStatement(
            s: "UPDATE minecraft SET bloques_colocados = bloques_colocados + 1 WHERE nombre_juego = ?")) {
        statement.setString( i: 1, jugador.getName());
        statement.executeUpdate();
    } catch (SQLException e) {
        getLogger().severe( msg: "Error al incrementar el número de bloques colocados: " + e.getMessage());
    }
}
```

Este bloque de código actualiza el número de bloques que ha colocado el jugador cada vez que ocurra el evento

Player jugador = event.getPlayer();

Obtiene el jugador que colocó el bloque

```
try (PreparedStatement statement = conexion.prepareStatement(
      "UPDATE minecraft SET bloques_colocados = bloques_colocados + 1
WHERE nombre_juego = ?")) {
   statement.setString(1, jugador.getName());
   statement.executeUpdate();
}
```

Esta sección intentará hacer el UPDATE en la BBDD al jugador en concreto que ejecutó el evento

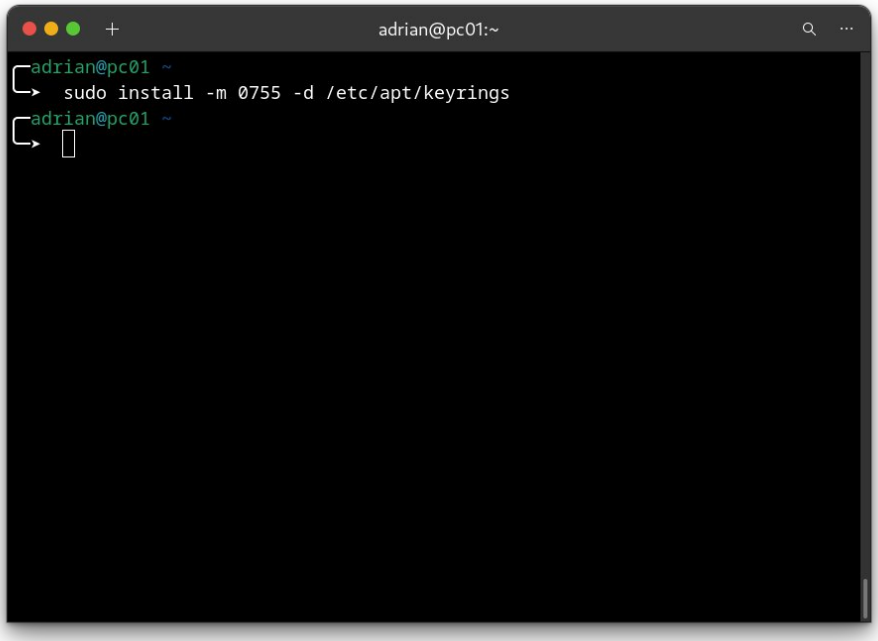
```
catch (SQLException e) {
      getLogger().severe("Error al incrementar el número de bloques colocados:
" + e.getMessage());
   }
}
```

} → Cierre de la clase principal

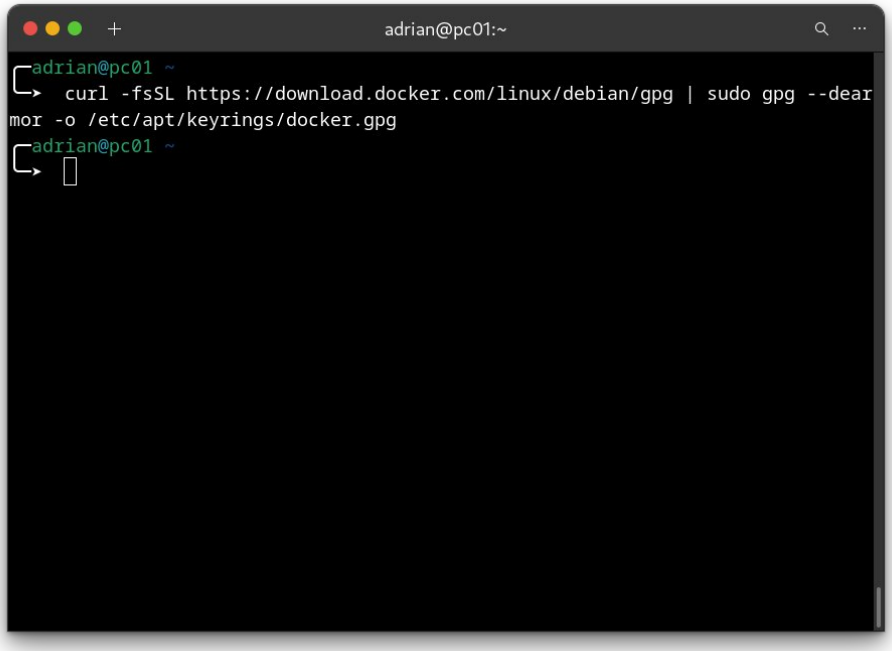
Si ocurre una excepción SQL, se registrará en el log

4. Docker

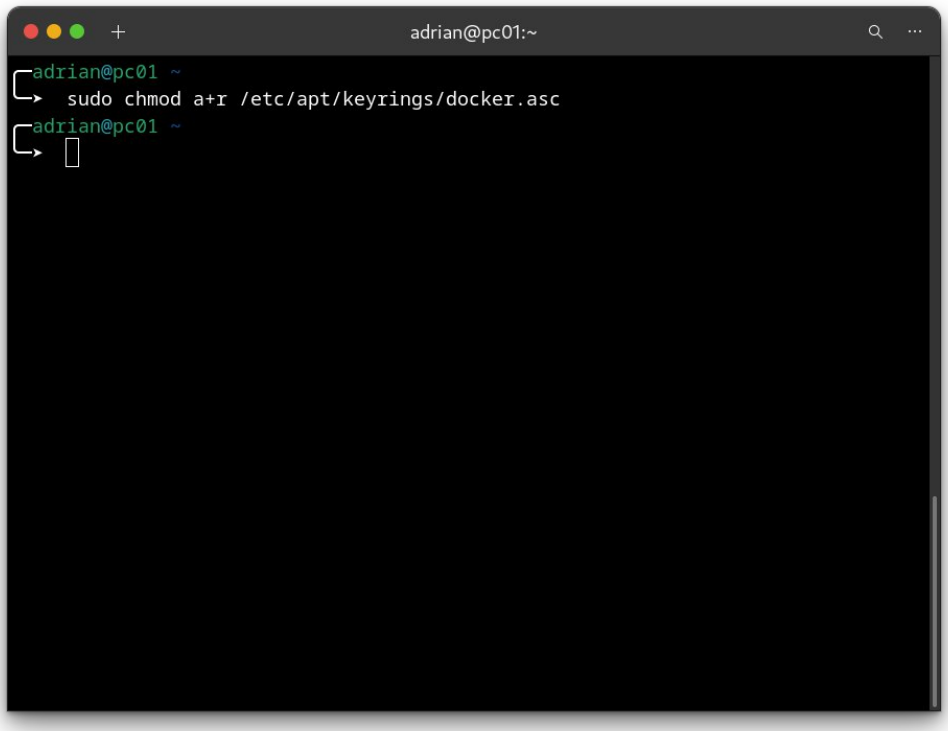
Instalación de docker



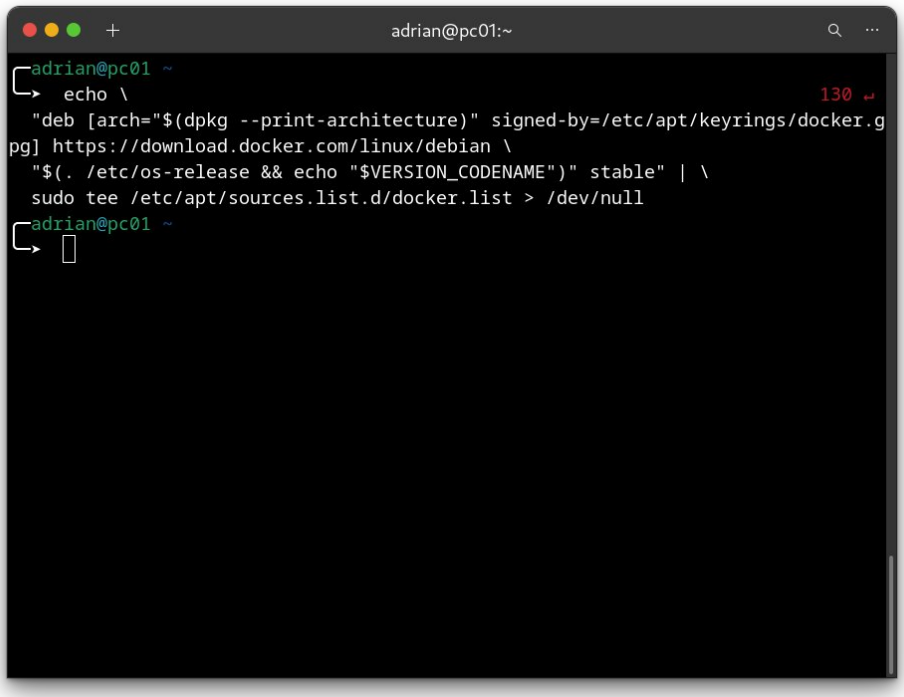
Creamos el directorio keyrings con permisos 755 en /etc/apt



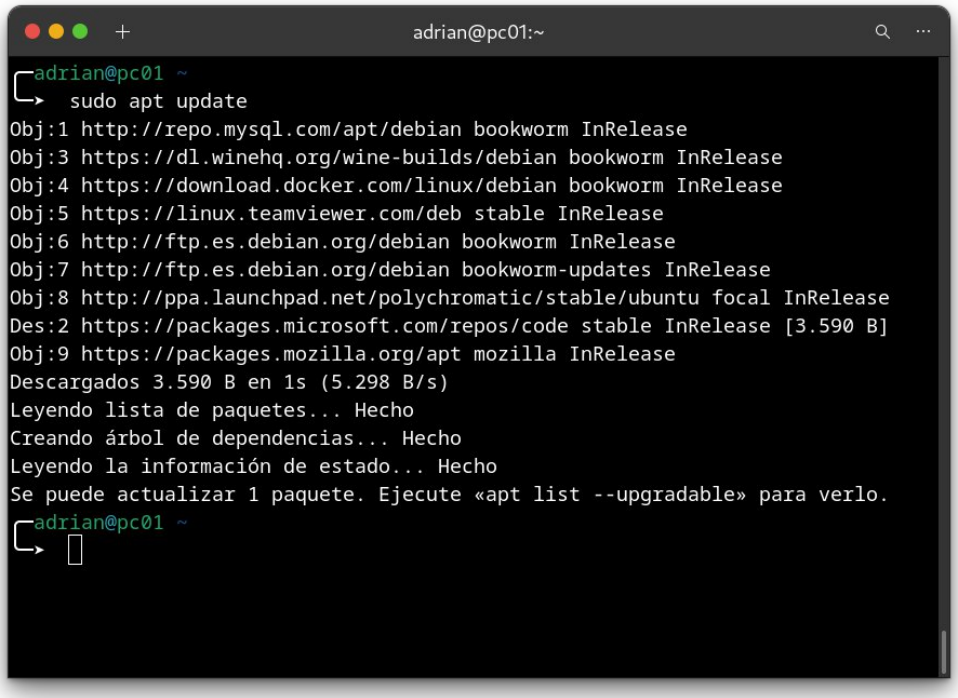
Ahora necesitamos descargar y guardar la clave GPG desde el sitio web oficial de Docker



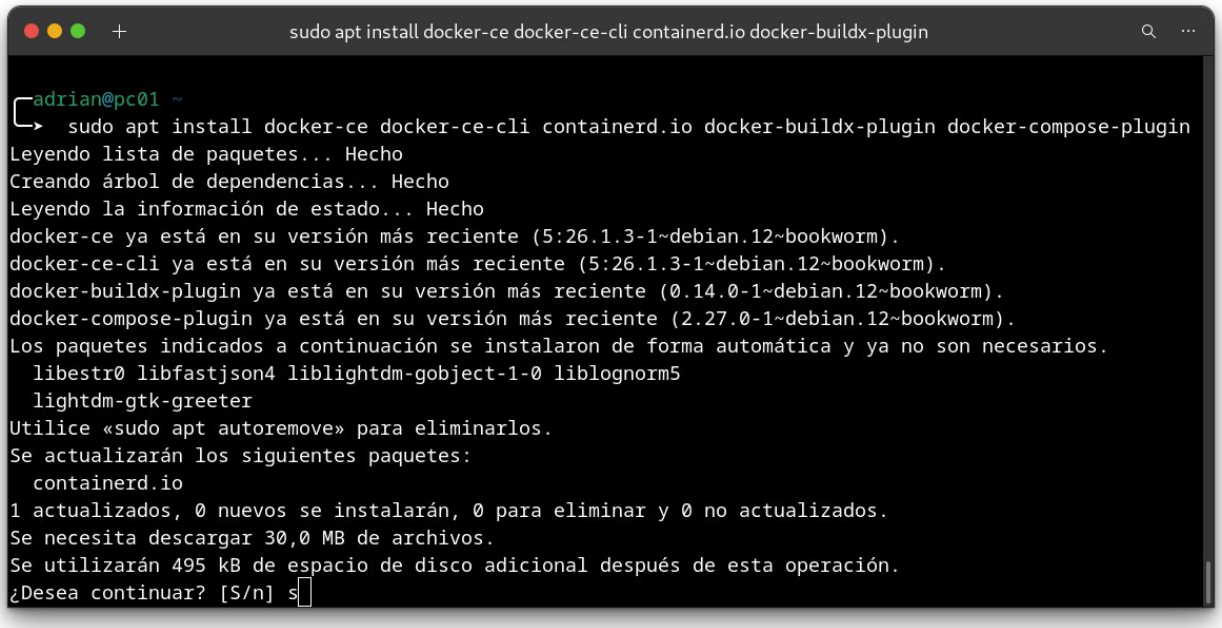
Asignamos a todo el mundo permisos de lectura a la clave GPG



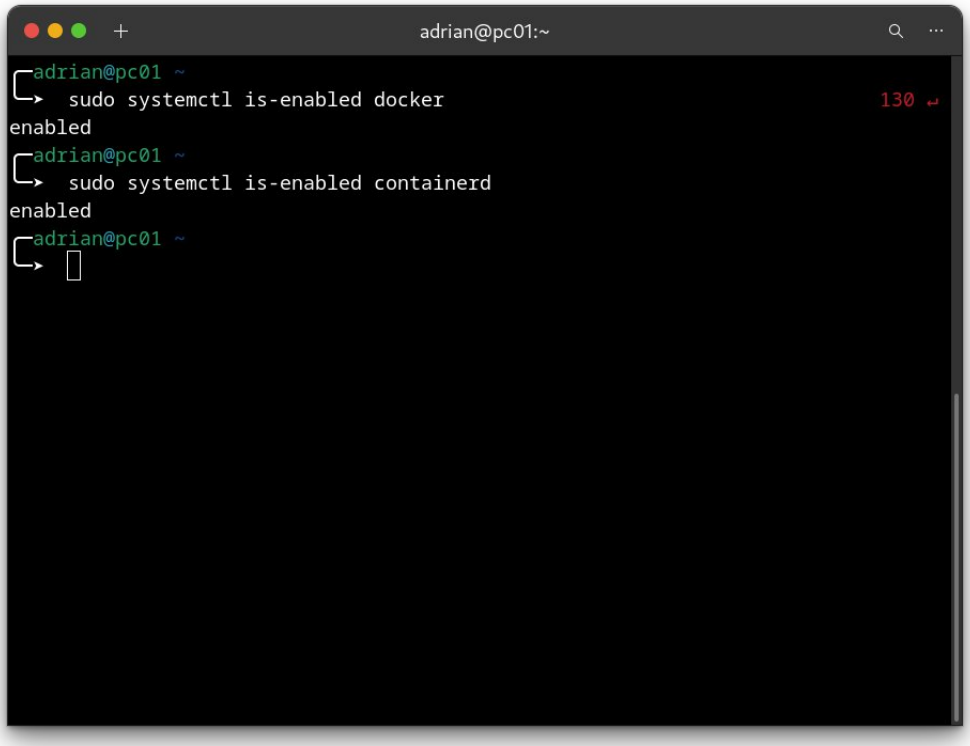
Agregamos el repositorio al source list



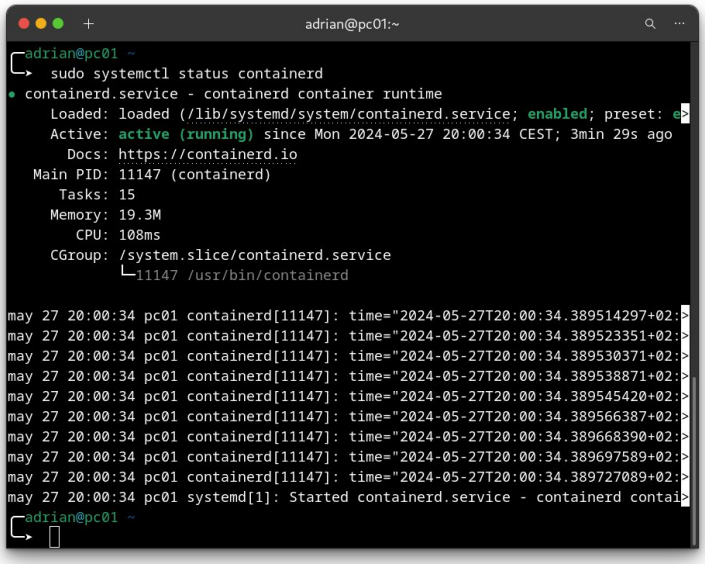
Actualizamos la lista de paquetes

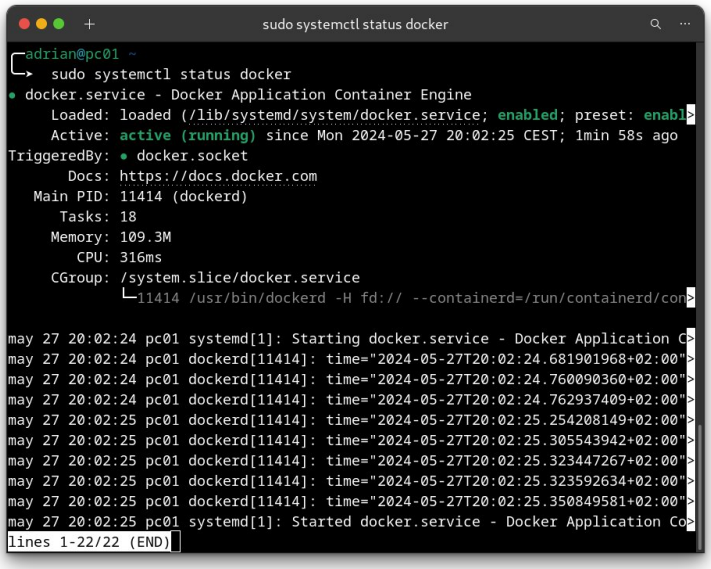


Instalamos los paquetes necesarios

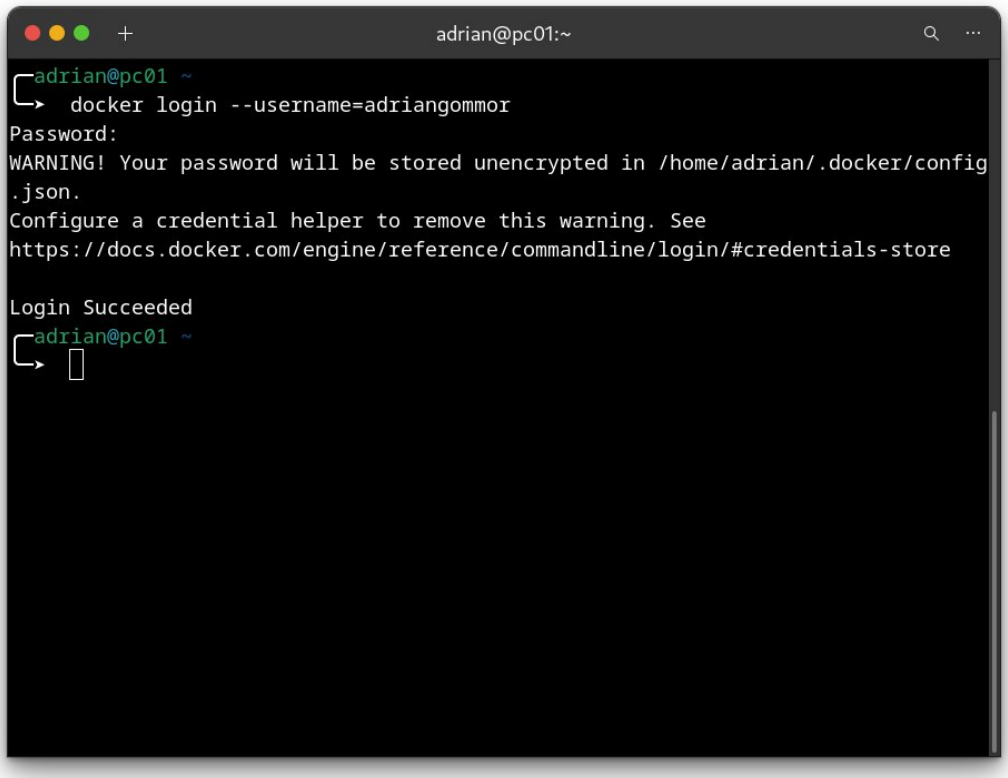


Comprobamos que los servicios estan habilitados

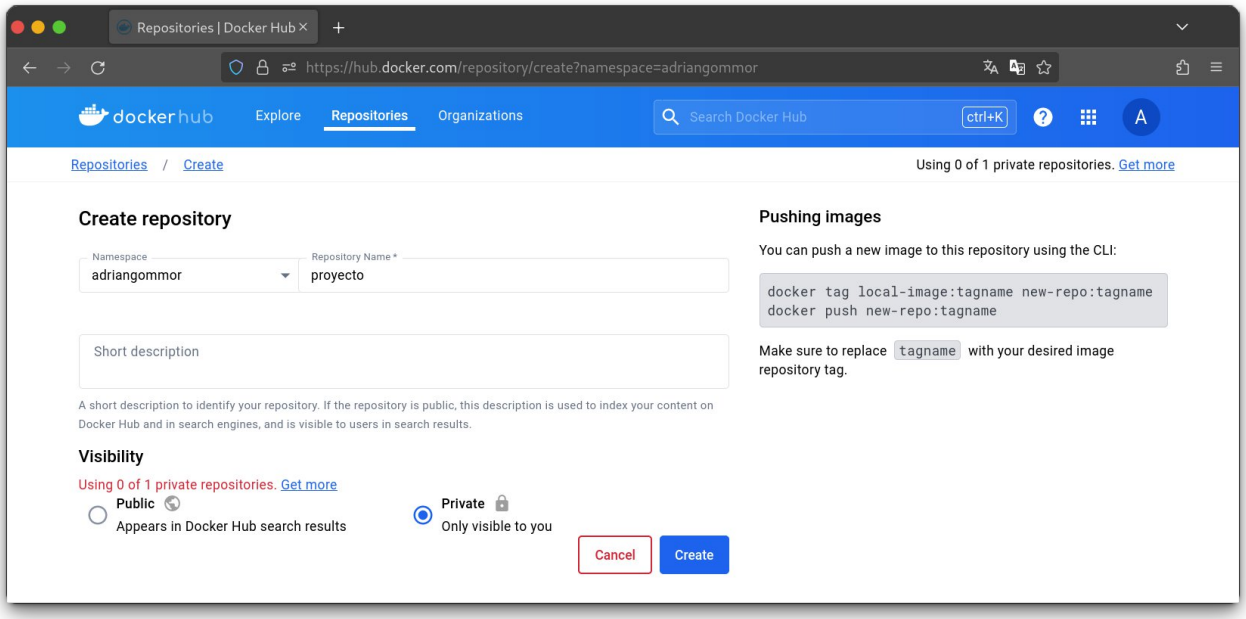




Comprobamos que ambos servicios se están ejecutando



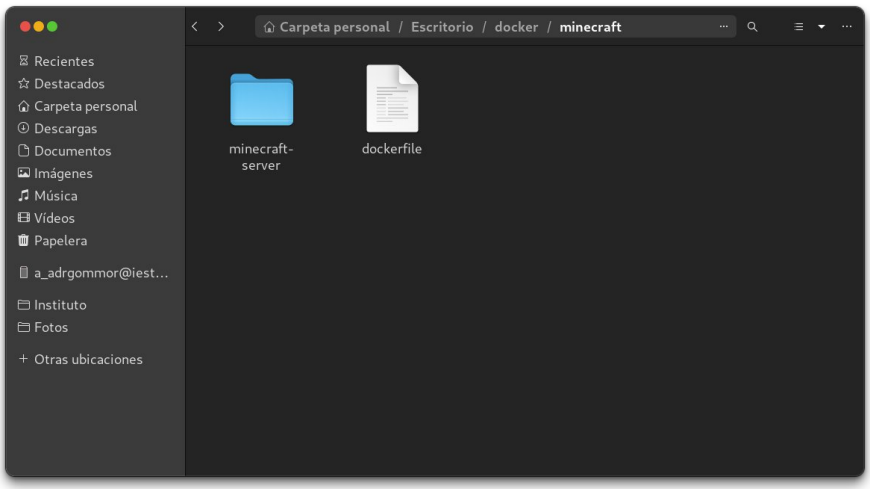
Para en un futuro poder subir nuestras imágenes a un repositorio, debemos iniciar sesión con las credenciales de docker hub (https://hub.docker.com/)



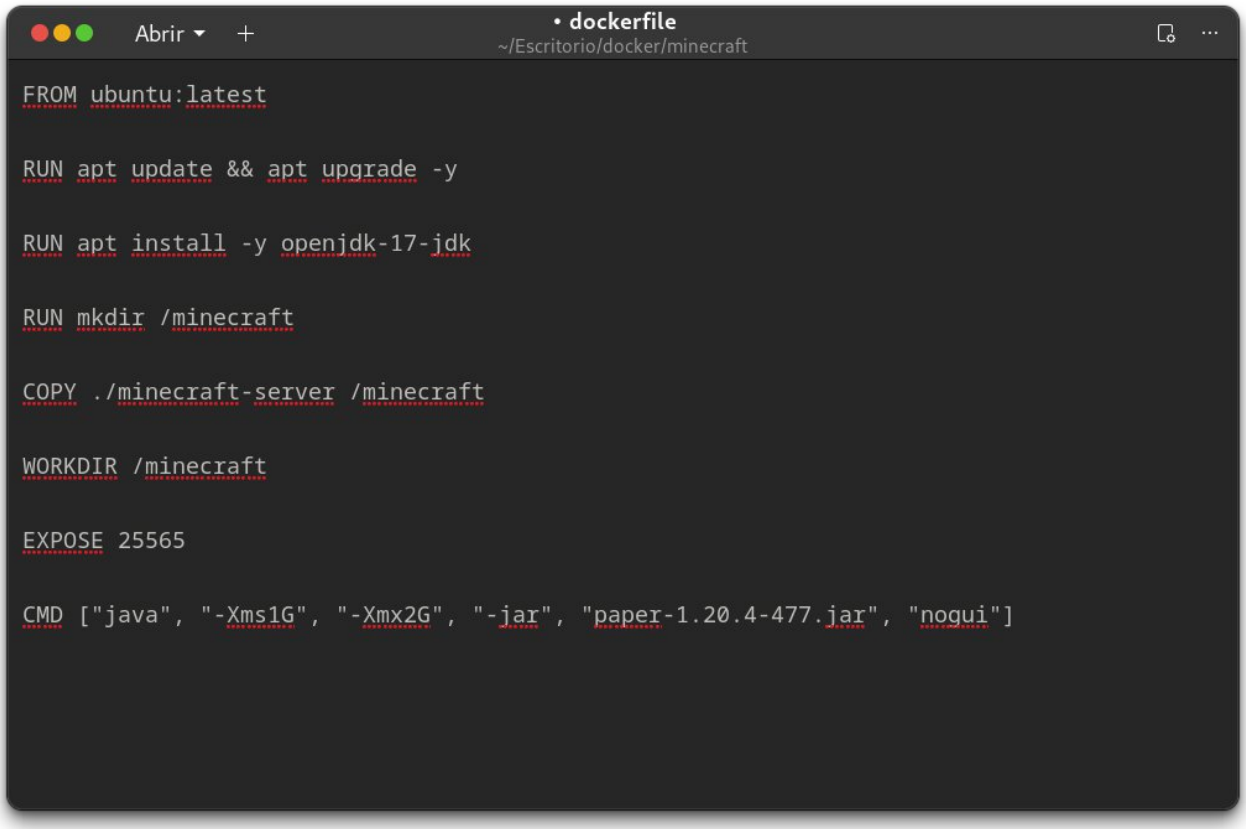
En docker hub, creamos un repositorio privado, donde subiremos las imágenes

Imágenes personalizadas

Minecraft

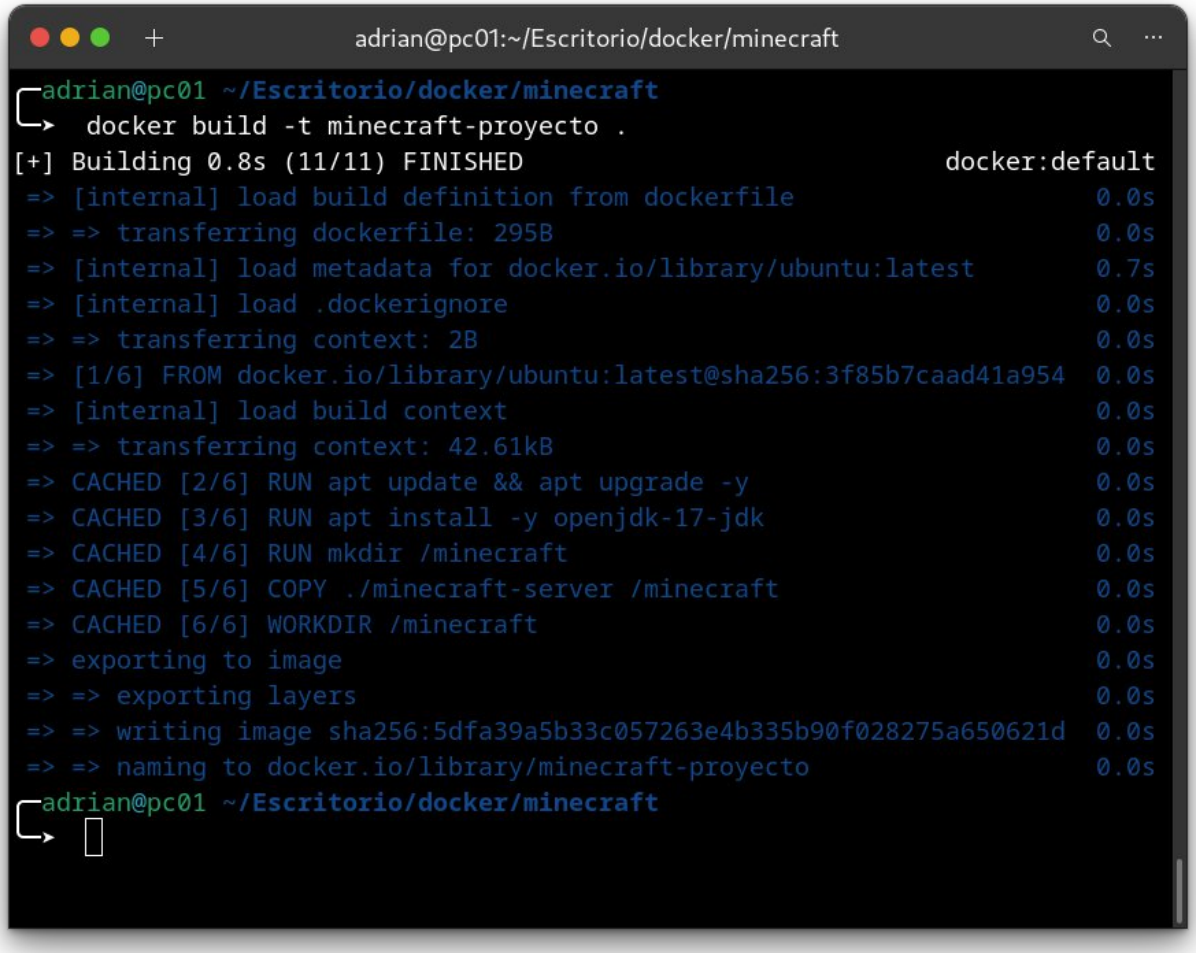


En el directorio de trabajo, donde tenemos los archivos del servidor de Minecraft, creamos un archivo llamado “dockerfile”

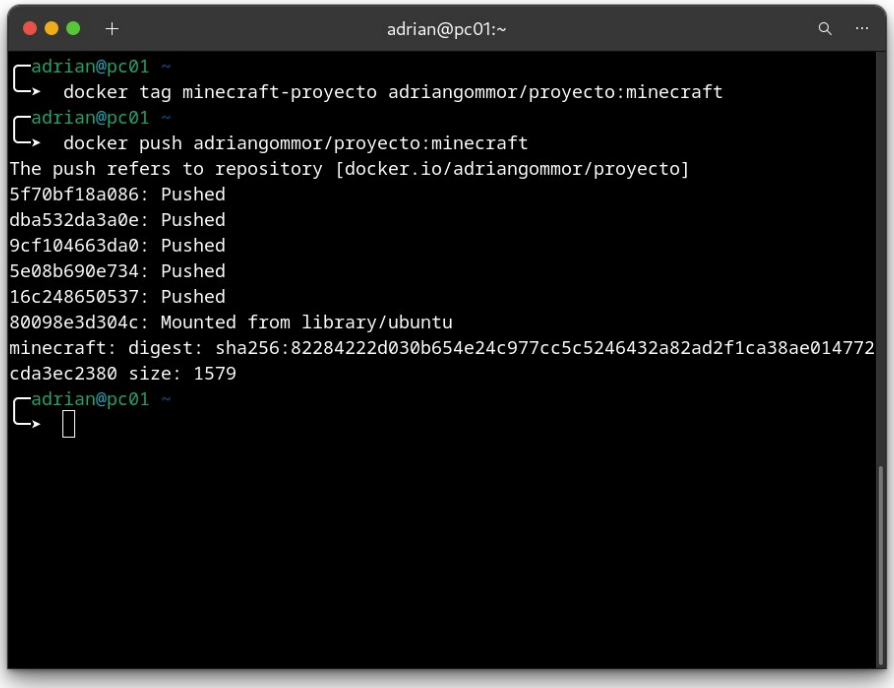


Establecemos que vamos a partir de la imagen ubuntu:latest (la última LTS) vamos a copiar nuestra carpeta que contiene el servidor a /minecraft del contenedor (previamente hay que aceptar el eula), establecemos el directorio de trabajo en /minecraft y establecemos que se ejecute automáticamente paper-1.20.4-477.jar usando java, mínimo usará 1GB de RAM y como máximo

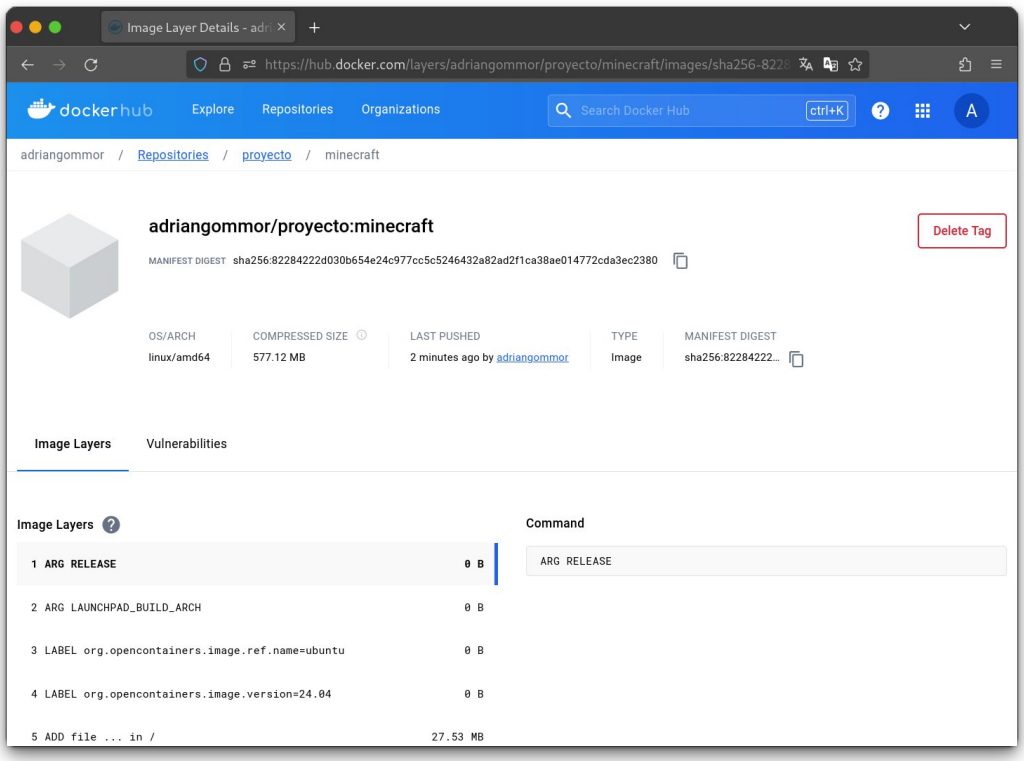
usará 2GB de RAM, además se lanzará con el parametro nogui para que no se ejecute la interfaz gráfica y solo se ejecute la versión de terminal



Creamos la imagen con **docker build -t minecraft-proyecto .**

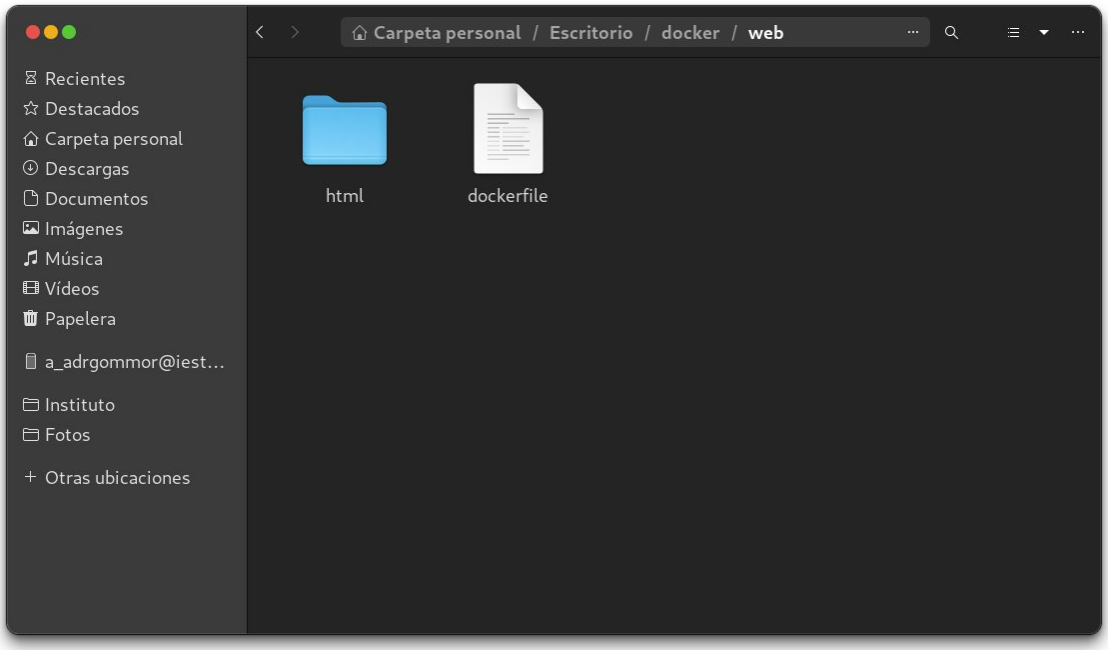


Etiquetamos la imagen con **docker tag minecraft-proyecto adriangommor/proyecto:minecraft** y subimos la imagen a nuestro repositorio privado con **docker push adriangommor/proyecto:minecraft**

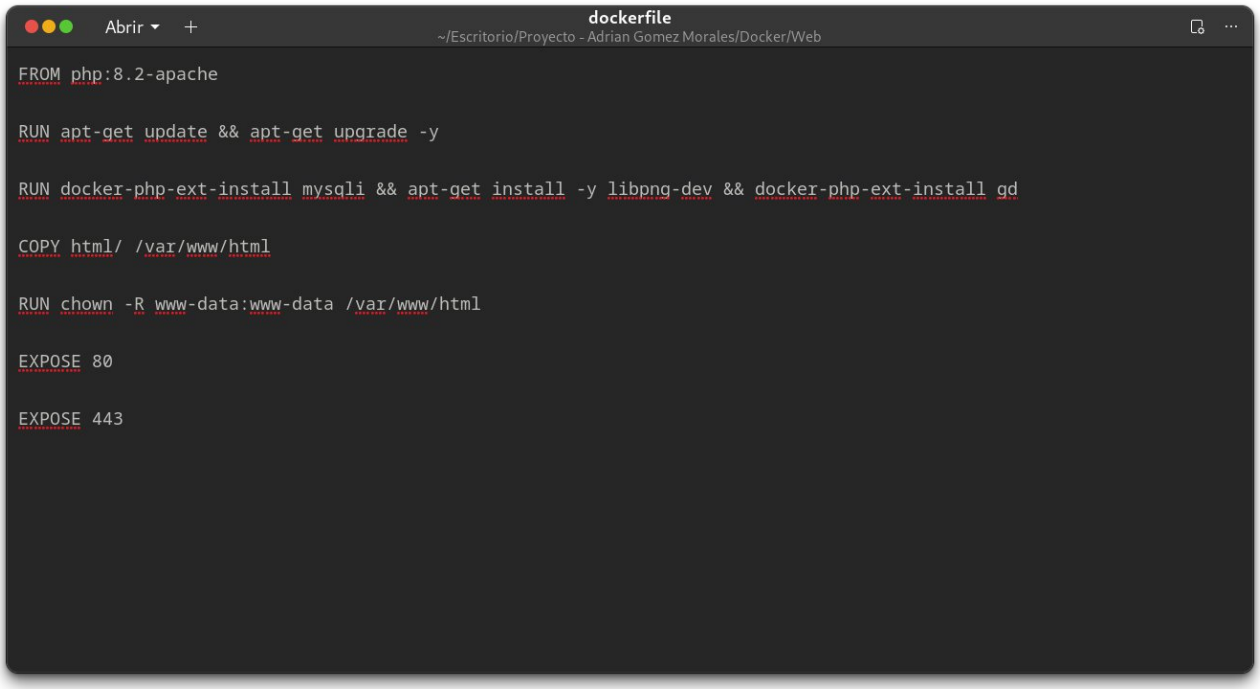


Y podemos ver la imagen subida correctamente correctamente

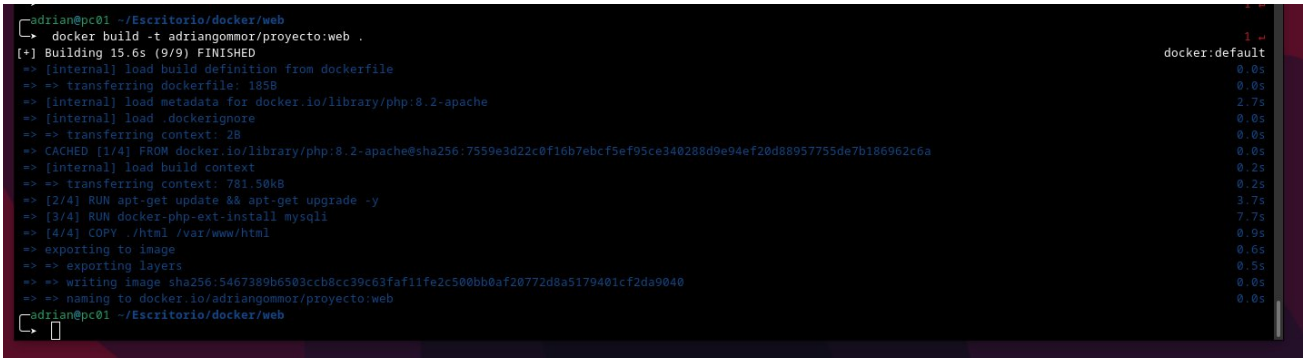
Web



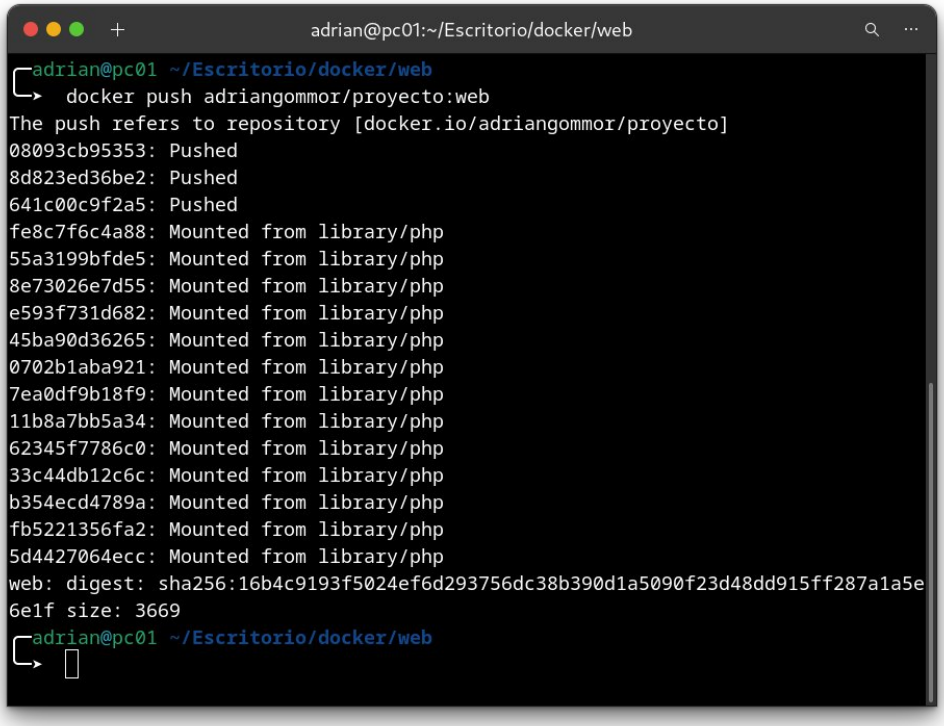
Ahora cambiamos de directorio de trabajo, vamos a crear la imagen para nuestra página web, para ello, en el mismo directorio, dejamos nuestra web y hacemos un dockerfile



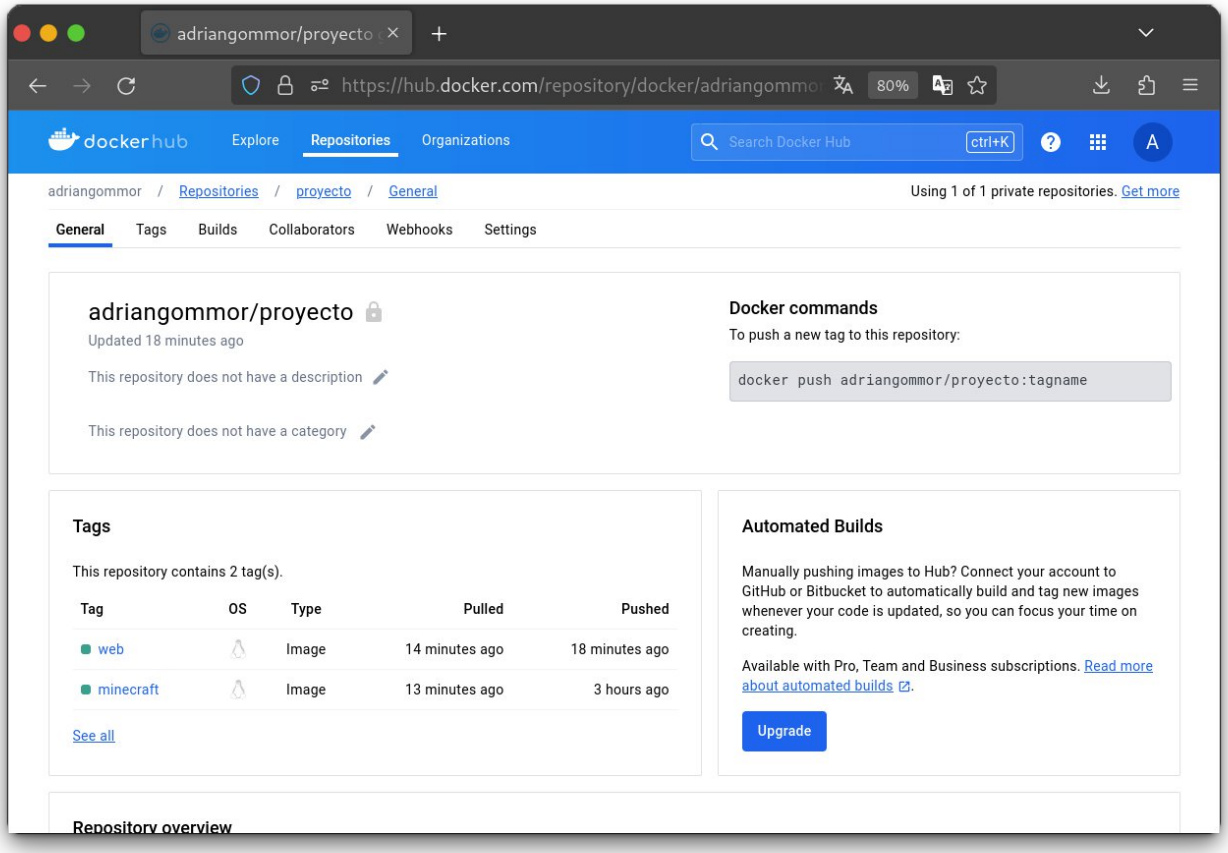
Añadimos las dependencias de mysqli para la conexión a MySQL y gd que es necesario para CkEditor



Creamos la imagen con **docker build -t adriangommor/proyecto:web .** además ya se crea etiquetada

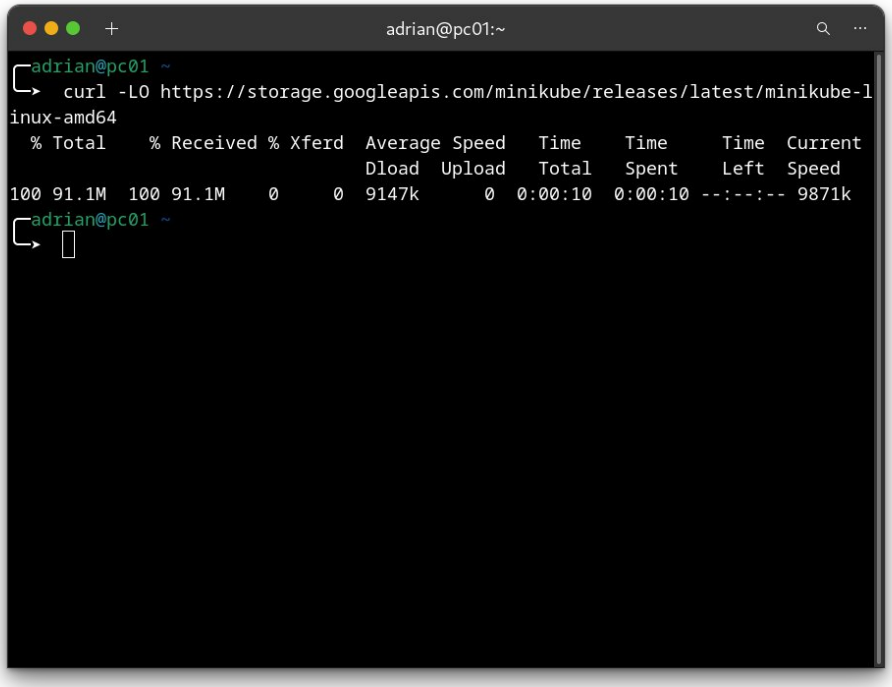


Subimos la imagen a nuestro repositorio privado de docker hub

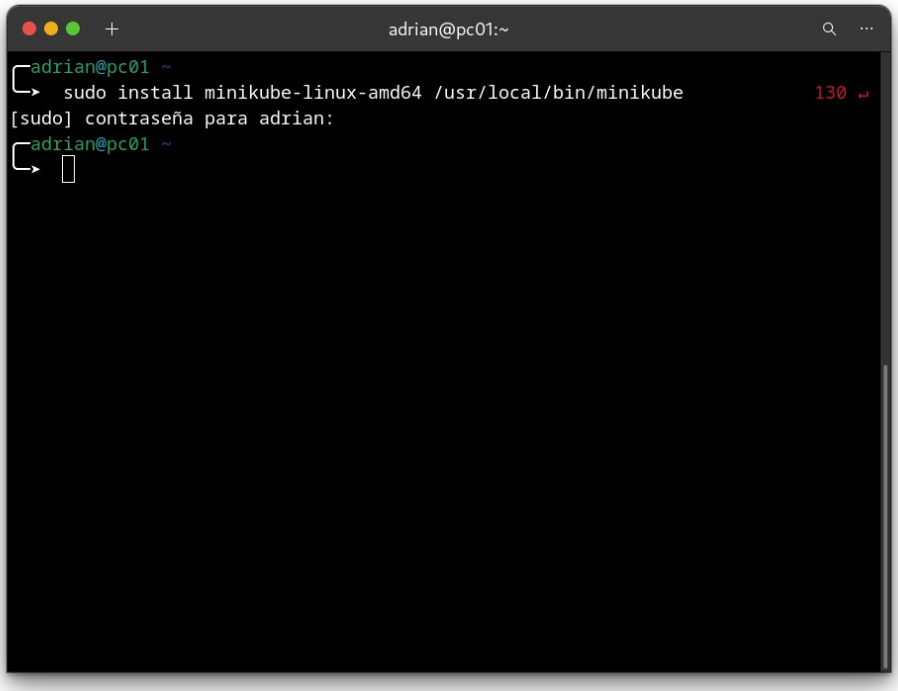


Así quedaría nuestro repositorio con las 2 imágenes

5. Kubernetes

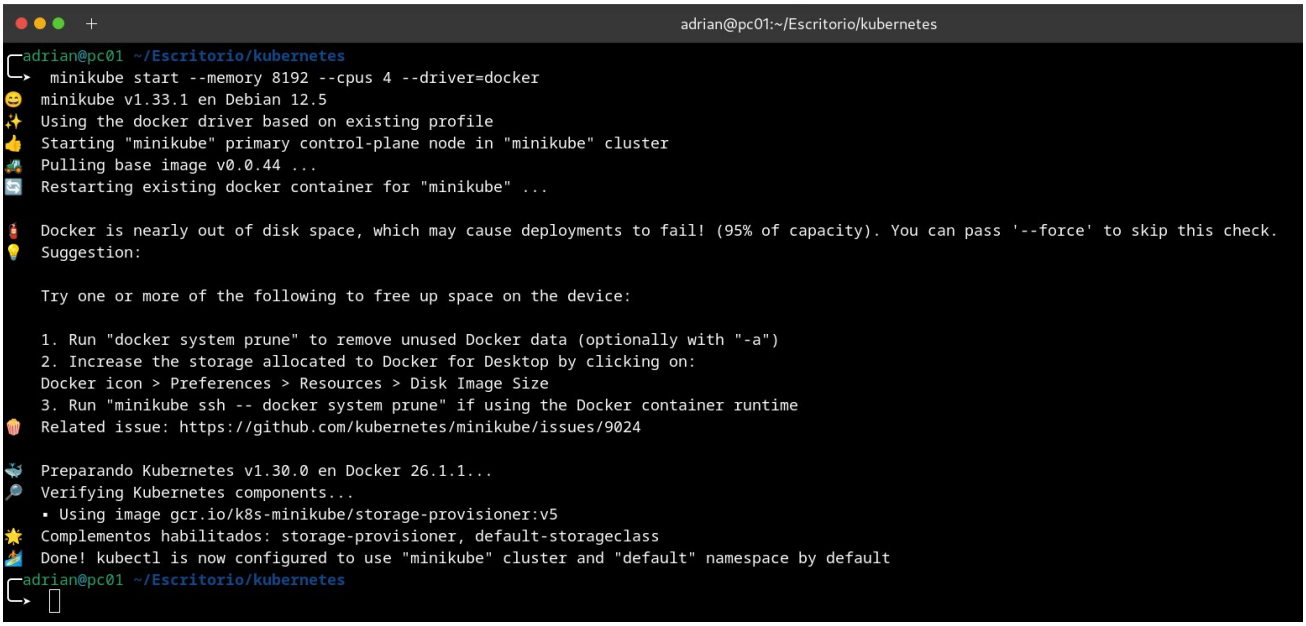


Descargamos Minikube usando curl



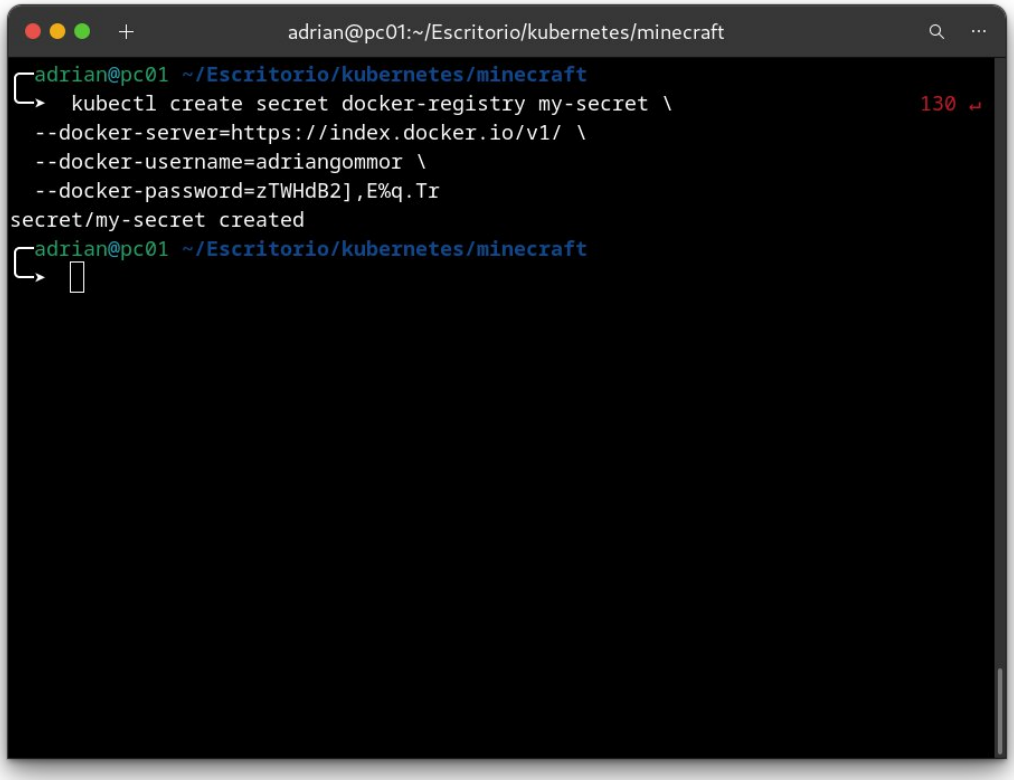
Instalamos Minikube

Ahora instalamos kubectl desde los repositorios de Debian



Iniciamos Minikube usando como driver docker y asignando 8GB de memoria RAM y 4 núcleos de mi procesador para que tenga los suficientes recursos

Secret para acceder al repositorio privado de Docker Hub



Creamos el secret

Minecraft en Kubernetes

Este yaml crea un despliegue de un servidor Minecraft utilizando una imagen propia de Docker Hub. Define un PersistentVolumeClaim de 3GiB para almacenar los datos del mundo de Minecraft, para que sean persistentes. Monta el volumen persistente en /minecraft/world. Además, expone el servicio a través de un NodePort en el puerto 30000, permitiendo el acceso externo al servidor Minecraft. Se utiliza un secret para el pull de la imagen

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

  name: pvc-minecraft

spec:

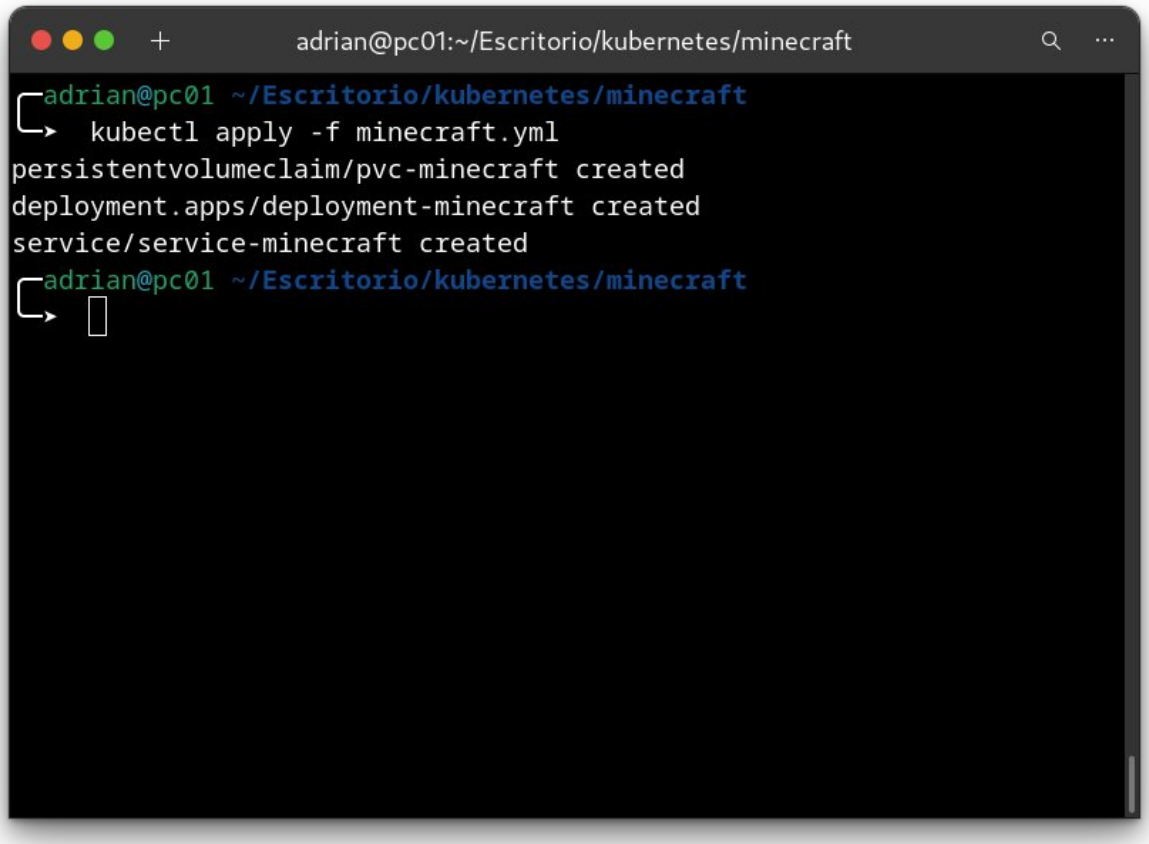
  accessModes:

    - ReadWriteOnce

  resources:

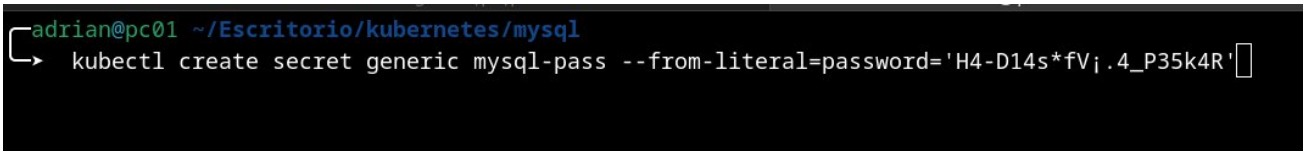
```
    requests:
      storage: 3Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-minecraft
  labels:
    app: minecraft
    service: game
spec:
  replicas: 1
  selector:
    matchLabels:
      app: minecraft
      service: game
  template:
    metadata:
      labels:
        app: minecraft
        service: game
    spec:
      volumes:
        - name: volumen-minecraft
          persistentVolumeClaim:
            claimName: pvc-minecraft
      containers:
        - name: contenedor-minecraft
```

```
        image: adriangommor/proyecto:minecraft
        volumeMounts:
          - mountPath: "/minecraft/world"
            name: volumen-minecraft
      imagePullSecrets:
       - name: my-secret
---
apiVersion: v1
kind: Service
metadata:
  name: service-minecraft
  labels:
    app: minecraft
    service: game
spec:
  type: NodePort
  selector:
    app: minecraft
    service: game
  ports:
    - name: minecraft
      port: 25565
      targetPort: 25565
      nodePort: 30000
```

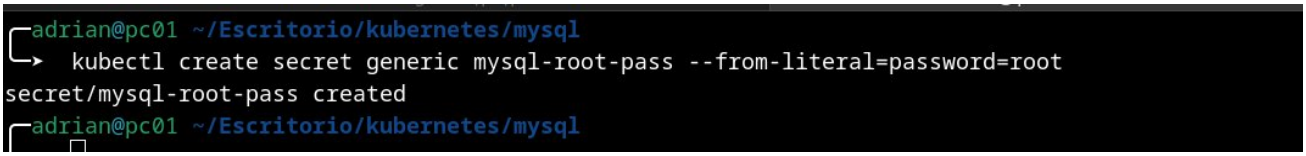


Aplicamos el yaml con “kubectl apply -f minecraft.yml”

Base de datos MySQL en Kubernetes



Creamos el secret para nuestra contraseña de la BBDD



Creamos el secret para la contraseña de root

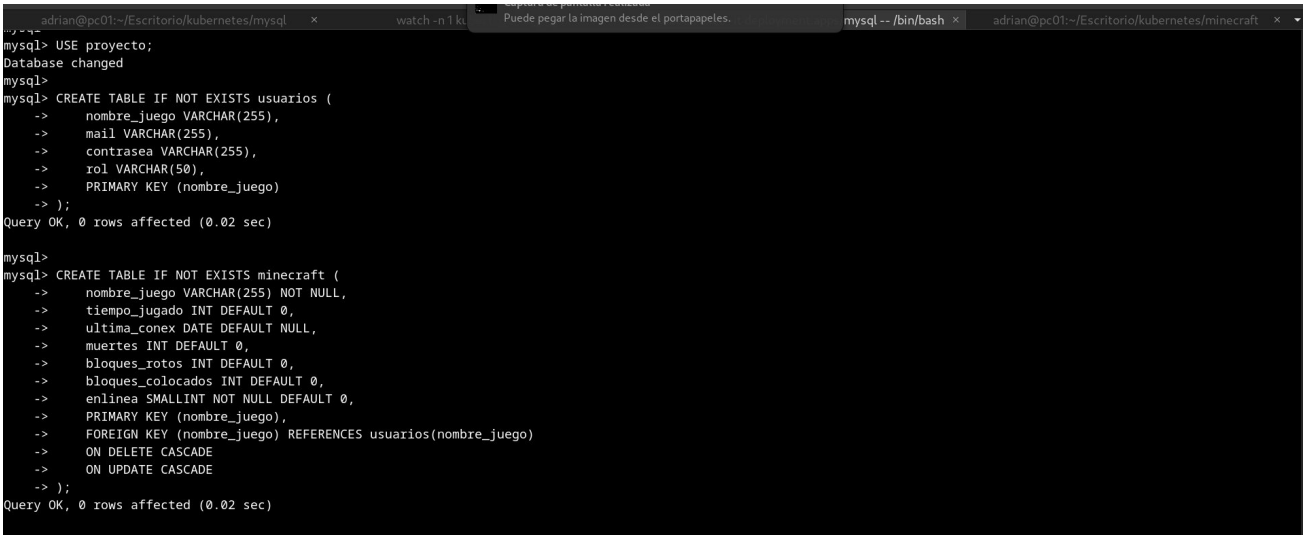
Archivo yaml

Este yaml configura el despliegue de una BBDD MySQL usando la imagen de Docker Hub. Define un volumen de 1GiB para almacenar los datos de MySQL, para que los datos sean persistentes. Además, expone el servicio MySQL a través de un Service ClusterIP para que pueda ser accedido por otros pods en el clúster usando la resolución DNS

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
  labels:
    app: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
```

```
    containers:
    - name: mysql
      image: mysql:latest
      env:
      - name: MYSQL_ROOT_PASSWORD
        valueFrom:
          secretKeyRef:
            name: mysql-root-pass
            key: password
      - name: MYSQL_PASSWORD
        valueFrom:
          secretKeyRef:
            name: mysql-pass
            key: password
      - name: MYSQL_DATABASE
        value: proyecto
      - name: MYSQL_USER
        value: user_proyect
      ports:
      - containerPort: 3306
      volumeMounts:
      - name: mysql-persistent-storage
        mountPath: /var/lib/mysql
    volumes:
    - name: mysql-persistent-storage
      persistentVolumeClaim:
        claimName: mysql-pv-claim
---
```

```
apiVersion: v1
kind: Service
metadata:
  name: mysql
spec:
  selector:
    app: mysql
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```



Entramos dentro de una bash del deployment y cargamos las tablas de la base de datos

Web en Kubernetes

Con este yaml creamos un despliegue de mi página usando mi imagen propia de docker hub, creamos dos vólumenes (de 1GiB cada uno), uno para guardar la web (/var/www/html) en sí y otro para guardar /tmp, ya que en /tmp se guardan las sesiones de PHP y si vamos a tener varias replicas, necesitamos que usen las mismas sesiones, además creamos un servicio de tipo NodePort para exponer nuestra web. El comando ejecutado en el contenedor de inicialización copia la web al volumen persistente si está vacío, asegurando que la web no se borre por culpa del volumen.

apiVersion: v1

kind: PersistentVolume

metadata:

  name: pv-web

spec:

  capacity:

    storage: 1Gi

  accessModes:

    - ReadWriteOnce

  hostPath:

    path: "/mnt/data"

  persistentVolumeReclaimPolicy: Retain

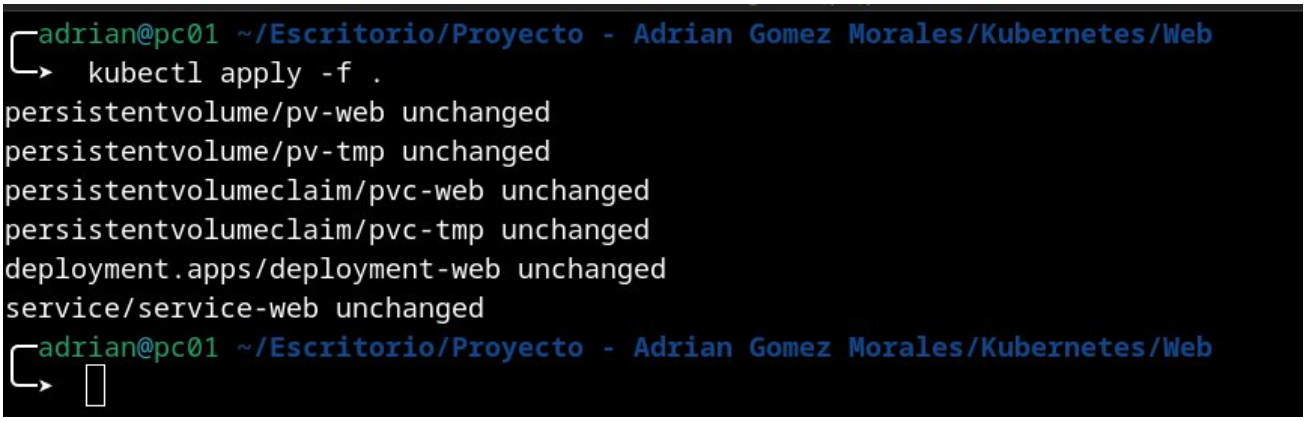
  storageClassName: manual

```
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-tmp
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/tmp"
  persistentVolumeReclaimPolicy: Retain
  storageClassName: manual
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-web
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: manual
---
apiVersion: v1
```

```
kind: PersistentVolumeClaim
metadata:
  name: pvc-tmp
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: manual
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-web
  labels:
    app: web
    service: http
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web
      service: http
  template:
    metadata:
      labels:
        app: web
```

```
    service: http
  spec:
    volumes:
      - name: volumen-web
        persistentVolumeClaim:
          claimName: pvc-web
      - name: volumen-tmp
        persistentVolumeClaim:
          claimName: pvc-tmp
    initContainers:
      - name: init-web
        image: adriangommor/proyecto:web
        command: ['sh', '-c', 'if [ -z "$(ls -A /mnt/html)" ]; then cp -r /var/www/html/* /mnt/html; fi']
        volumeMounts:
          - name: volumen-web
            mountPath: /mnt/html
    containers:
      - name: contenedor-web
        image: adriangommor/proyecto:web
        ports:
          - containerPort: 80
          - containerPort: 443
        volumeMounts:
          - mountPath: "/var/www/html"
            name: volumen-web
          - mountPath: "/tmp"
            name: volumen-tmp
    imagePullSecrets:
```

```
        - name: my-secret
---
apiVersion: v1
kind: Service
metadata:
  name: service-web
  labels:
    app: web
    service: http
spec:
  type: NodePort
  selector:
    app: web
    service: http
  ports:
    - name: http
      port: 80
      targetPort: 80
      nodePort: 30001
    - name: https
      port: 443
      targetPort: 443
      nodePort: 30002
```

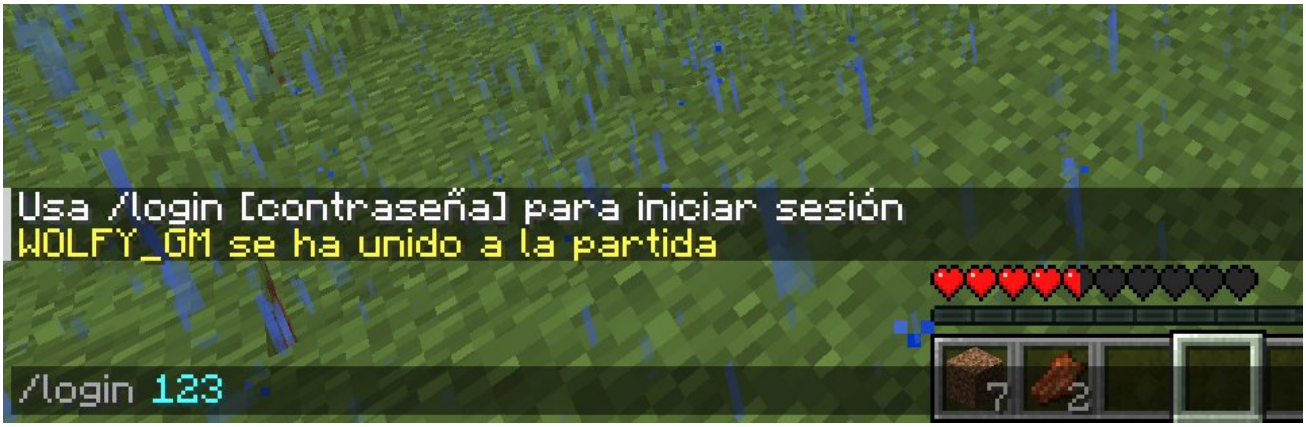


Aplicamos el archivo yaml

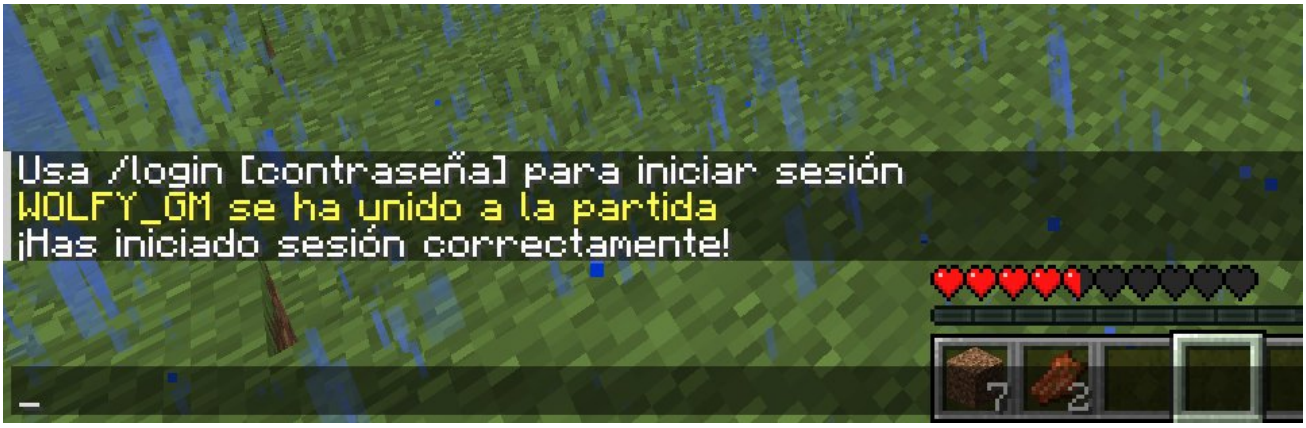
Nota Importante:

Es importante lanzar antes el deployment de MySQL antes que el de Minecraft ya que si no funciona la conexión a la BBDD, el plugin no funcionará de forma correcta, también por primera vez tenemos que cargar la BBDD usando *kubectl exec -it [deployment_mysql] -- /bin/bash* y de la misma forma, en el deployment de la web, hay que establecer /tmp con los permisos 777 y hay que cambiar el propieterio de todos los archivos de /var/www/html a www-data de lo contrario no se podrá subir nada a la web

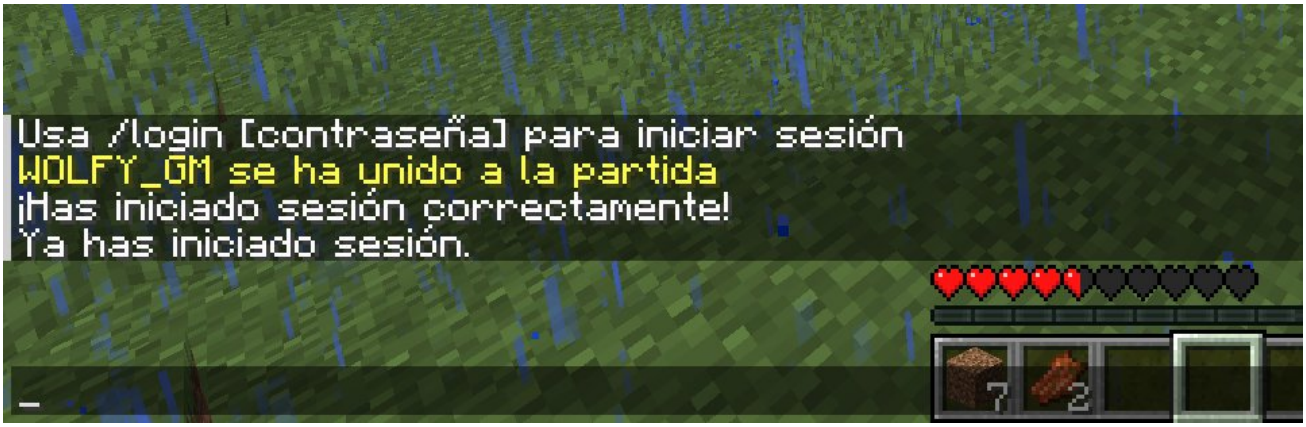
6. Cliente Minecraft



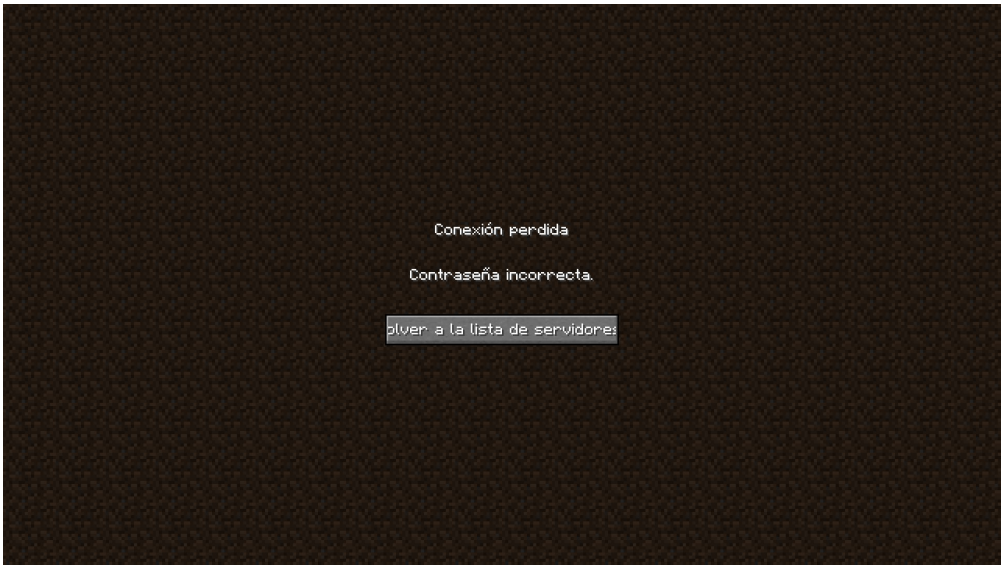
Cuando un jugador se une, se le solicitará la contraseña



Si es correcta saldrá el mensaje de aviso de que es correcta



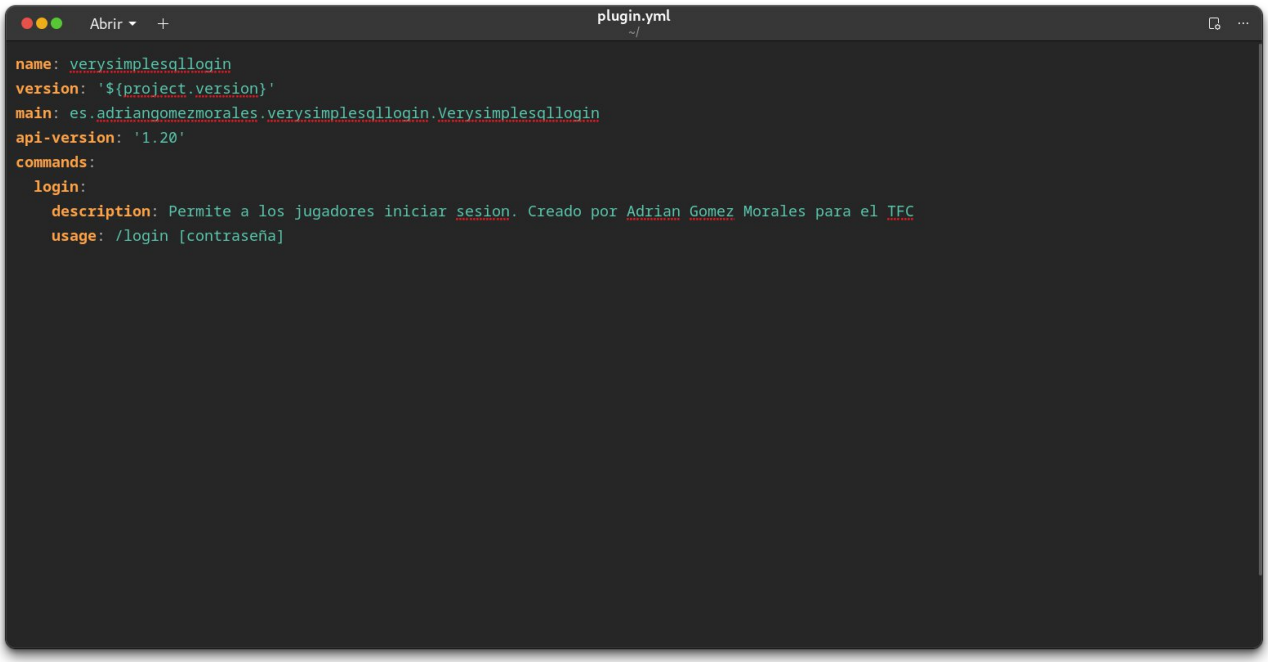
Si ya había iniciado sesión, se lo notificará al usuario



En caso de que sea incorrecta, se expulsará al jugador del servidor

Si no estamos registrados, ni si quiera podríamos entrar al servidor

Si hacemos un /help login muestra los detalles del comando, debido a que en nuestro plugin.yml del proyecto es



7. Vídeo demostrativo

Para ver el vídeo:

[Click aquí] o copie este enlace https://www.youtube.com/watch?v=e2ZDbtxC61U