***Diseñar un servidor Web TCP a nivel de capa de transporte (con sockets), sencillo, que almacene un diccionario, que cada clave devuelve una página, el servidor lo tiene que leer de disco, por ejemplo de un json o kassandra***

***Hacerlo tanto con json como kassandra***

***Que el cliente le pida a un servidor a través de socket las páginas de un diccionario (le envía una página el cliente, el servidor me devuelve 1, si el cliente envía 2, el servidor me devuelve 2, y así continuamente etc)***

***El servidor debe guardar las páginas en disco, no en memoria(backend, oracle misma mente)***

* ***Usando json***
* ***Usando kassandra***

## **1. Implementación con JSON**

### **Paso 1: Crear el archivo JSON**

Primero, crearemos un archivo JSON que contendrá nuestras páginas. Este archivo se llamará pages.json y tendrá la siguiente estructura:

**Contenido del archivo pages.json:**

{

"1": "Página 1: Bienvenido a la página 1.",

"2": "Página 2: Bienvenido a la página 2.",

"3": "Página 3: Bienvenido a la página 3."

}

### **Paso 2: Servidor TCP usando JSON**

Este servidor leerá las páginas desde el archivo JSON y responderá a las solicitudes del cliente.

#### **Código del servidor (server\_json.py):**

import json # Importa el módulo JSON para manejar archivos JSON

import socket # Importa el módulo socket para la comunicación de red

# Función para cargar las páginas desde un archivo JSON

def load\_pages(filename):

with open(filename, 'r') as file:

return json.load(file) # Carga y devuelve el contenido del archivo JSON como un diccionario

# Función para guardar una nueva página en el archivo JSON

def save\_page(filename, key, content):

with open(filename, 'r+') as file: # Abre el archivo en modo lectura y escritura

data = json.load(file) # Carga el contenido actual

data[key] = content # Agrega o actualiza la página

file.seek(0) # Mueve el puntero al inicio del archivo

json.dump(data, file) # Guarda el nuevo contenido

file.truncate() # Elimina cualquier contenido sobrante

# Configuración del servidor

HOST = '10.1.2.108' # Dirección IP del servidor

PORT = 12000 # Puerto en el que el servidor escuchará las conexiones

# Cargar el diccionario de páginas

pages = load\_pages('pages.json') # Llama a la función para cargar las páginas desde el archivo JSON

# Crear el socket TCP

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) # Crea un socket TCP

server\_socket.bind((HOST, PORT)) # Vincula el socket a la dirección IP y puerto

server\_socket.listen(1) # Escucha por una conexión entrante

print(f"Servidor escuchando en {HOST}:{PORT}...") # Mensaje de inicio

while True:

conn, addr = server\_socket.accept() # Acepta una conexión entrante

print(f"Conexión establecida con: {addr}") # Muestra la dirección del cliente conectado

while True:

data = conn.recv(1024).decode() # Recibe datos del cliente

if not data: # Si no hay datos, salir del bucle

break

# Devolver la página solicitada

if data in pages: # Verifica si la clave existe en el diccionario

response = pages[data] # Obtiene la respuesta correspondiente

else:

response = "Página no encontrada." # Mensaje de error si la clave no existe

conn.send(response.encode()) # Envía la respuesta al cliente

conn.close() # Cierra la conexión con el cliente

server\_socket.close() # Cierra el socket del servidor

### **Explicación del servidor:**

1. **Importaciones**: Importamos los módulos necesarios para trabajar con JSON y crear un socket.
2. **Funciones**:
   * **load\_pages(filename):** Carga las páginas desde un archivo JSON y devuelve un diccionario.
   * **save\_page(filename, key, content):** Guarda o actualiza una página en el archivo JSON. (Aunque no se utiliza en este ejemplo, está disponible si se quiere añadir nuevas páginas).
3. **Configuración del servidor**: Definimos la dirección IP y el puerto en el que el servidor escuchará las conexiones.
4. **Creación del socket:** Creamos un socket TCP, lo vinculamos a la dirección y puerto, y comenzamos a escuchar conexiones.
5. **Bucle principal**: Aceptamos conexiones entrantes. Por cada conexión, entramos en un bucle donde recibimos datos del cliente, buscamos la página correspondiente en el diccionario, y enviamos la respuesta al cliente.

### **Paso 3: Cliente TCP**

El cliente enviará solicitudes al servidor para obtener las páginas.

#### **Código del cliente (client.py):**

import socket

# Configuración del cliente

HOST = '10.1.2.108' # Dirección IP del servidor

PORT = 12000 # Puerto del servidor

# Crear el socket TCP

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((HOST, PORT))

while True:

page\_number = input("Ingrese el número de página (o 'exit' para salir): ") # Solicita el número de página

if page\_number.lower() == 'exit':

break

client\_socket.send(page\_number.encode()) # Envía el número de página al servidor

response = client\_socket.recv(1024).decode() # Recibe la respuesta

print(f"Respuesta del servidor: {response}")

client\_socket.close() # Cierra la conexión del cliente

### **Explicación del cliente:**

1. **Configuración**: Define la dirección IP y el puerto del servidor.
2. **Creación del socket**: Crea un socket TCP y se conecta al servidor.
3. **Bucle de entrada:** Pide al usuario que ingrese un número de página. Si se ingresa "exit", el bucle se interrumpe. De lo contrario, envía el número al servidor y muestra la respuesta.

## **2. Implementación con Cassandra**

### **Paso 1: Configuración de Cassandra**

Asegúrate de tener Cassandra instalado y en ejecución. Para ello usaremos el comando:

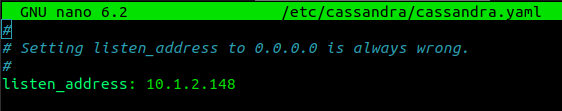
* sudo service cassandra status

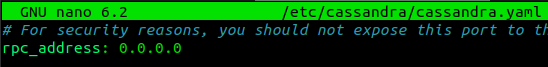
Si vemos un botón verde significa que está en ejecución y podemos proseguir con la configuración

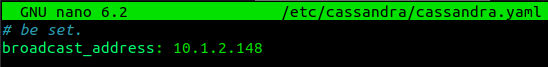
Una vez comprobado abriremos el archivo de configuración cassandra.yaml, que generalmente se encuentra en /etc/cassandra/cassandra.yaml

* sudo nano /etc/cassandra/cassandra.yaml

1. **Modificar las siguientes configuraciones**
   * **listen\_address:** Cambia esto a tu dirección IP. En mi caso, 10.1.2.148
   * **rpc\_address:** Configúralo como 0.0.0.0 para permitir conexiones desde cualquier dirección IP.
   * **broadcast\_address:** Configúralo también a 10.1.2.148 si tienes un entorno de red distribuido.







1. **Puerto JMX**

Asegúrate de que jmx\_port esté configurado correctamente:

* JMX\_PORT="7199"

1. **Permitir Puertos en el Firewall**

Asegúrate de que el firewall permita el tráfico en los siguientes puertos:

* 9042: Para el protocolo CQL. *sudo ufw allow 9042*
* 7000: Para la comunicación entre nodos: *sudo ufw allow 7000*
* 7199: Para JMX: *sudo ufw allow 7199*

1. **Crear un Keyspace y Tabla**

Conéctate a Cassandra utilizando cqlsh:

* cqlsh 10.1.2.148 9042

Crea un keyspace y una tabla para almacenar las paginas, como por ejemplo:

CREATE KEYSPACE IF NOT EXISTS paginas\_keyspace

WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication\_factor' : 1 };

USE paginas\_keyspace;

CREATE TABLE IF NOT EXISTS paginas (

pagina\_id text PRIMARY KEY,

content text );

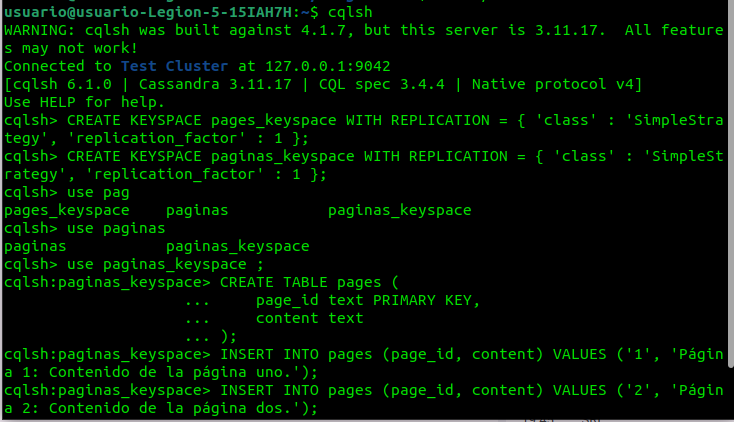
1. **Insertar datos en la tabla**

Insertar algunas página de ejemplo como pide el ejercicio:

INSERT INTO paginas (pagina\_id, content) VALUES ('1', 'Página 1: Contenido de la página uno.');

INSERT INTO paginas (pagina\_id, content) VALUES ('2', 'Página 2: Contenido de la página dos.');

INSERT INTO paginas (pagina\_id, content) VALUES ('3', 'Página 3: Contenido de la página tres.');



1. **Configuración del Servidor en Python**

import socket

from cassandra.cluster import Cluster

HOST = '10.1.2.148'

PORT = 12000

# Conectar a Cassandra

cluster = Cluster(['127.0.0.1'])

session = cluster.connect('pages\_keyspace')

# Crear socket

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((HOST, PORT))

server\_socket.listen(5)

print(f"Servidor escuchando en {HOST}:{PORT}")

while True:

client\_socket, addr = server\_socket.accept() # Aceptar conexión del cliente

print(f"Conexión desde: {addr}")

page\_number = client\_socket.recv(1024).decode()

print(f"Solicitando página: {page\_number}")

# Consultar la base de datos

query = "SELECT content FROM pages WHERE page\_id = %s"

row = session.execute(query, (page\_number,)).one()

# Enviar respuesta al cliente

if row:

response = row.content

else:

response = "Página no encontrada."

client\_socket.send(response.encode())

client\_socket.close() # Cerrar la conexión con el cliente

Este código implementa un servidor TCP que espera conexiones de clientes. Cuando un cliente se conecta, el servidor recibe el número de página solicitado, busca esa página en una base de datos Cassandra y le envía el contenido de la página al cliente. Si la página no existe, el servidor responde con "Página no encontrada". Después de enviar la respuesta, cierra la conexión con el cliente.

Este proceso se repite indefinidamente, escuchando nuevas conexiones de clientes en el bucle while True.

1. **Configuración del Cliente en Python**

import socket

# Configuración del cliente

HOST = '10.1.2.148' # Dirección IP del servidor

PORT = 12000 # Puerto del servidor

# Crear el socket TCP

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((HOST, PORT))

while True:

page\_number = input("Ingrese el número de página (o 'exit' para salir): ") # Solicita el número de página

if page\_number.lower() == 'exit':

break

client\_socket.send(page\_number.encode())

response = client\_socket.recv(1024).decode() # Recibe la respuesta

print(f"Respuesta del servidor: {response}")

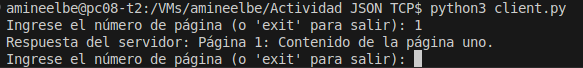
client\_socket.close() # Cierra la conexión del cliente

Esta configuración deberá ser relacionada con el propio servidor de Cassandra, para ello ponemos la dirección ‘10.1.2.148’ en HOST, ya que esta es la dirección y es la que debe apuntar el cliente para la conexión.

1. **Conexion Servidor/Cliente**

Por último crearemos la conexión entre los dos equipos siguiendo los siguientes pasos:

* Iniciaremos el código Python mediante la terminal, debemos tener en cuenta donde hemos guardado el archivo, ya que necesitaremos la ruta completa para llegar
* Mientras el servidor busca la conexión, el cliente iniciará mediante comandos su archivo, si todo va bien, el cliente debería de ver el siguiente mensaje: ‘Ingrese el número de la página (o “exit” para salir):’
* Una vez aquí el cliente pulsa cualquier tecla (1,2 o 3) para recibir la respuesta del servidor



* El cliente recibirá la respuesta del servidor mediante la conexión. En el podemos ver la solicitud que ha realizado el cliente.

