Universidad Nacional de Moquegua

Manual de despliegue
Sistema de Chat Distribuido

Escuela Profesional de Ingeniería en Sistemas e Informática

Ciclo: V

Curso: Sistemas Distribuidos

Docente: Juan Carlos Valero Gómez

Alumno: Juan Edwin Calizaya Llanos

8 de junio de 2025

Índice

1. Visión general y motivación **2**

2. Creación de las máquinas virtuales **2**
2.1. Clonar y numerar 2
2.2. Configurar red con Netplan 3

3. Nodo storage: MinIO **3**
3.1. Instalación 3
3.2. Usuario y carpeta de datos 3
3.3. Servicio systemd 3

4. Base de datos: RethinkDB en clúster **4**
4.1. Instalación (ambos nodos) 4
4.2. Servicio maestro (192.168.122.104) 4
4.3. Servicio réplica (192.168.122.105) 4

5. Capa de aplicación: API (FastAPI + Uvicorn) **5**
5.1. Instalar Python y crear venv 5
5.2. Estructura del proyecto 5
5.3. Servicio systemd 13

6. Balanceador L7: Caddy **14**
6.1. Instalación 14
6.2. Caddyfile (sin cambios) 14

7. Pruebas de extremo a extremo **14**

1. Visión general y motivación

Este documento explica cómo implementar paso a paso un sistema de chat distribuido. Se utilizan seis máquinas virtuales Ubuntu Server 20.04 LTS (o posterior) creadas con `virt-manager` y QEMU:

Rol	Hostname	IP fija
Balanceador	`lb`	192.168.122.101
API 1	`api1`	192.168.122.102
API 2	`api2`	192.168.122.103
BD maestro	`dbmaster`	192.168.122.104
BD réplica	`replica`	192.168.122.105
Almacenamiento S3	`storage`	192.168.122.106

¿Por qué esta separación? Aislar responsabilidades simplifica la escalabilidad horizontal (p. ej. añadir más nodos API) y permite mantener o reiniciar un componente sin tumbar el resto.

Stack elegido

- **MinIO**: almacén S3 auto-contenedor, ideal para ficheros de usuario.
- **RethinkDB**: base orientada a documentos con *change feeds* en tiempo real, perfecta para WebSockets.
- **FastAPI + Uvicorn**: framework ASGI moderno, tipado y de alto rendimiento en Python.
- **Caddy**: reverse-proxy L7 con sintaxis declarativa y certificados automáticos.
- **systemd**: arranque, reinicios y logs unificados en todas las capas.

2. Creación de las máquinas virtuales

2.1. Clonar y numerar

Crea una VM base (Ubuntu Server mínima), realiza las actualizaciones de seguridad y después clónala cinco veces con `virt-manager`. Cambia nombre interno y MAC en cada clon para evitar colisiones.

Configuración de red en qemu (virt-manager):

```
<interface type="network">
  <mac address="52:54:00:e2:66:a2"/>
  <source network="default" portid="bfb2cac6-fa66-4d9e-99f1-4db4513fcca5" bridge="virbr0"/>
  <target dev="vnet1"/>
  <model type="virtio"/>
  <alias name="net0"/>
  <address type="pci" domain="0x0000" bus="0x01" slot="0x00" function="0x0"/>
</interface>
```

2.2. Configurar red con Netplan

Ejemplo en `api1` —solo cambia la IP en `addresses`:

```
network:
  version: 2
  ethernets:
    enp1s0:
      dhcp4: false
      addresses: [192.168.122.102/24]
      routes:
        - to: 0.0.0.0/0
          via: 192.168.122.1
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
```

Aplica la configuración:

```
$ sudo netplan apply
```

3. Nodo storage: MinIO

3.1. Instalación

```
$ wget https://dl.min.io/server/minio/release/linux-amd64/archive/minio_20250524170830.0.0
    _amd64.deb -O minio.deb
$ sudo dpkg -i minio.deb

# Desactivamos el servicio gen rico: lanzaremos el nuestro.
sudo systemctl disable minio.service
```

3.2. Usuario y carpeta de datos

```
$ sudo useradd -r --shell /sbin/nologin minio-user
$ sudo mkdir /var/minio-data
$ sudo chown -R minio-user:minio-user /var/minio-data
```

3.3. Servicio systemd

```
# /etc/systemd/system/minio-server.service
[Unit]
Description=MinIO S3 Object Storage Service
Wants=network-online.target
After=network-online.target

[Service]
User=minio-user
Group=minio-user
Environment=MINIO_ROOT_USER=minioadmin
Environment=MINIO_ROOT_PASSWORD=minioadmin
WorkingDirectory=/var/minio-data
ExecStart=/usr/local/bin/minio server /var/minio-data --console-address :9001
Restart=always
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
```

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable --now minio-server.service
```

Conéctate a `http://192.168.122.106:9001` y crea el `bucket chat` con política de sólo-lectura.

4. Base de datos: RethinkDB en clúster

Motivación Los *change feeds* permiten transmitir mensajes nuevos sin realizar *polling* costoso.

4.1. Instalación (ambos nodos)

```
$ wget -qO- https://download.rethinkdb.com/repository/raw/pubkey.gpg | \
  sudo gpg --dearmor -o /usr/share/keyrings/rethinkdb-archive-keyrings.gpg

$ echo "deb [signed-by=/usr/share/keyrings/rethinkdb-archive-keyrings.gpg] \
https://download.rethinkdb.com/repository/ubuntu-$(lsb_release -cs) \
$(lsb_release -cs) main" | \
  sudo tee /etc/apt/sources.list.d/rethinkdb.list

$ sudo apt update
$ sudo apt install -y rethinkdb
```

4.2. Servicio maestro (192.168.122.104)

```
# /etc/systemd/system/rethinkdb-master.service
[Unit]
Description=RethinkDB Server (Master)
After=network.target

[Service]
Type=simple
ExecStart=/usr/bin/rethinkdb --bind all
Restart=always

[Install]
WantedBy=multi-user.target
```

4.3. Servicio réplica (192.168.122.105)

```
# /etc/systemd/system/rethinkdb-replica.service
[Unit]
Description=RethinkDB Server (Replica)
After=network.target

[Service]
Type=simple
ExecStart=/usr/bin/rethinkdb --bind all \
  --join 192.168.122.104:29015
Restart=always

[Install]
WantedBy=multi-user.target
```

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable --now rethinkdb-master.service
$ sudo systemctl enable --now rethinkdb-replica.service
```

Panel web: `http://192.168.122.104:8080`

5. Capa de aplicación: API (FastAPI + Uvicorn)

5.1. Instalar Python y crear venv

```
$ sudo apt install -y python3 python3-venv python3-pip
$ python3 -m venv ~/venv
source ~/venv/bin/activate

$ pip install --upgrade pip
$ pip install fastapi "uvicorn[standard]" \
  boto3 botocore bcrypt python-jose[cryptography] rethinkdb
```

¿Por qué `venv`? Aísla dependencias y evita conflictos con otros proyectos Python que pudieran instalarse en el sistema.

5.2. Estructura del proyecto

```
/home/administrador/srv/chat-api/
           main.py        # logica del backend
           static/
               index.html # interfaz web
```

En `main.py` va la api, la cual es esta:

```
import json
from botocore.exceptions import ClientError
from fastapi.middleware.cors import CORSMiddleware
from fastapi import Depends
from fastapi import (
    FastAPI, WebSocket, UploadFile, File,
    HTTPException, Depends
)
from fastapi.staticfiles import StaticFiles
from fastapi.responses import FileResponse
from rethinkdb import RethinkDB
from rethinkdb.errors import ReqlOpFailedError
import bcrypt, jwt, datetime, os, boto3, uuid, json, asyncio
from typing import Any, Callable
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials
import jwt
TBL_ROOMS = "rooms"
auth_scheme = HTTPBearer()
# ---------- RethinkDB ----------
r = RethinkDB()
RDB_HOST = "192.168.122.104"
RDB_PORT = 28015
DB_NAME   = "chat"
TBL_USERS = "users"
TBL_MSGS  = "messages"

def _get_conn():
    """Devuelve una conexi n *nueva* a la BD."""
    return r.connect(host=RDB_HOST, port=RDB_PORT, db=DB_NAME)

def _run_sync(f: Callable[[], Any]):
    """Ejecuta bloqueante en thread pool y devuelve el resultado."""
    loop = asyncio.get_event_loop()
    return loop.run_in_executor(None, f)

app = FastAPI()

app.add_middleware(
    CORSMiddleware,
```

```
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],        # GET, POST, OPTIONS, etc.
    allow_headers=["*"],        # Authorization, Content-Type
)

@app.on_event("startup")
def _init_db():
    """Crea BD y tablas si no existen."""
    conn = r.connect(host=RDB_HOST, port=RDB_PORT)
    # BD
    if DB_NAME not in r.db_list().run(conn):
        r.db_create(DB_NAME).run(conn)

    # Tabla usuarios (PK = username)
    try:
        r.db(DB_NAME).table_create(
            TBL_USERS, primary_key="username", replicas=2
        ).run(conn)
    except ReqlOpFailedError:
        pass

    # Tabla mensajes
    try:
        r.db(DB_NAME).table_create(
            TBL_MSGS, replicas=2
        ).run(conn)
    except ReqlOpFailedError:
        pass
    try:
        r.db(DB_NAME).table_create(
            TBL_ROOMS,
            primary_key="id",
            replicas=2
        ).run(conn)
    except ReqlOpFailedError:
        pass
    conn.close()

# ---------- MinIO ----------

MINIO = boto3.client(
    "s3",
    endpoint_url="http://192.168.122.106:9000",
    aws_access_key_id="minioadmin",
    aws_secret_access_key="minioadmin",
)

# Crear bucket 'chat' si no existe
try:
    MINIO.head_bucket(Bucket="chat")
except ClientError as e:
    code = e.response["Error"]["Code"]
    if code in ("404", "NoSuchBucket"):
        MINIO.create_bucket(Bucket="chat")
    else:
        raise

# Pol tica p blica de s lo lectura para todos los objetos en el bucket
public_read_policy = {
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {"AWS": ["*"]},
        "Action": ["s3:GetObject"],
        "Resource": ["arn:aws:s3:::chat/*"]
    }]
}
```

```
MINIO.put_bucket_policy(
    Bucket="chat",
    Policy=json.dumps(public_read_policy)
)




# ---------- JWT ----------
SECRET = "super_secreto"

def create_jwt(username: str):
    payload = {
        "sub": username,
        "exp": datetime.datetime.utcnow() + datetime.timedelta(hours=3),
    }
    return jwt.encode(payload, SECRET, algorithm="HS256")


def verify_token(
    credentials: HTTPAuthorizationCredentials = Depends(auth_scheme)
) -> str:
    token = credentials.credentials
    try:
        payload = jwt.decode(token, SECRET, algorithms=["HS256"])
        return payload["sub"]
    except jwt.ExpiredSignatureError:
        raise HTTPException(status_code=401, detail="Token expirado")
    except Exception:
        raise HTTPException(status_code=401, detail="Token inv lido")


# ---------- Archivos est ticos ----------
app.mount("/static", StaticFiles(directory="static"), name="static")

@app.get("/")
def read_index():
    return FileResponse(os.path.join(os.path.dirname(__file__), "static", "index.html"))

# ---------- Endpoints ----------
@app.post("/api/register", status_code=201)
async def register(u: dict):
    hashed = bcrypt.hashpw(u["password"].encode(), bcrypt.gensalt()).decode()

    def _tx():
        conn = _get_conn()
        #   Existe ?
        if r.table(TBL_USERS).get(u["username"]).run(conn):
            conn.close()
            raise ValueError("Usuario ya existe")
        r.table(TBL_USERS).insert(
            {"username": u["username"], "password": hashed,
             "created_at": r.now()}
        ).run(conn)
        conn.close()

    try:
        await _run_sync(_tx)
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))
    return {"ok": True}

@app.post("/api/login")
async def login(u: dict):
    def _tx():
        conn = _get_conn()
        row = r.table(TBL_USERS).get(u["username"]).run(conn)
```

```
        conn.close()
        return row
    row = await _run_sync(_tx)
    if not row or not bcrypt.checkpw(u["password"].encode(),
                                     row["password"].encode()):
        raise HTTPException(status_code=401, detail="Credenciales inv lidas")
    return {"token": create_jwt(u["username"])}


@app.get("/api/rooms")
async def list_rooms(user: str = Depends(verify_token)):
    def _tx():
        conn = _get_conn()
        cursor = (
            r.table(TBL_ROOMS)
             .filter(lambda room: room["participants"].contains(user))
             .run(conn)
        )
        rooms = list(cursor)
        conn.close()
        return rooms

    rooms = await _run_sync(_tx)
    return rooms
@app.post("/api/room", status_code=201)
async def create_room(
    body: dict,
    user: str = Depends(verify_token)
):
    """
    Crea una sala con id y lista de participantes.
    body = {
      "id": "alice-bob",
      "participants": ["alice","bob"]
    }
    """
    def _tx():
        conn = _get_conn()
        # Verifica que no exista ya
        if r.table(TBL_ROOMS).get(body["id"]).run(conn):
            conn.close()
            raise ValueError("La sala ya existe")
        # Inserta la sala
        r.table(TBL_ROOMS).insert({
            "id":           body["id"],
            "participants": body["participants"],
            "created_by":   user,
            "created_at":   r.now()
        }).run(conn)
        conn.close()

    try:
        await _run_sync(_tx)
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))
    return {"ok": True}

@app.post("/api/send")
async def send(msg: dict, user: str = Depends(verify_token)):
    """
    msg = {
      "room": "sala-id",
      "content": "texto libre o URL de archivo"
    }
    """
    def _tx():
        conn = _get_conn()
        r.table(TBL_MSGS).insert({
```

```
            "from":    user,
            "room":    msg["room"],
            "content": msg["content"],
            "ts":      r.now(),
        }).run(conn)
        conn.close()
    await _run_sync(_tx)
    return {"ok": True}


@app.get("/api/history/{room}")
async def history(room: str, limit: int = 50,
                  user: str = Depends(verify_token)):
    def _tx():
        conn = _get_conn()
        cursor = (r.table(TBL_MSGS)
                    .filter({"room": room})
                    .order_by(r.desc("ts"))
                    .limit(limit)
                    .run(conn))
        msgs = list(cursor)
        conn.close()
        return msgs
    msgs = await _run_sync(_tx)
    # Convertimos a objetos JSON-serializables
    for m in msgs:
        m["ts"] = str(m["ts"])
    return msgs


@app.post("/api/upload")
async def upload_file(
    f: UploadFile = File(...),
    user: str = Depends(verify_token)
):
    """
    Sube cualquier tipo de archivo a MinIO y devuelve metadatos:
      - file: el ID generado
      - filename: nombre original
      - content_type: MIME
      - url: enlace de descarga
    """
    # Leer todo el contenido del archivo
    content = await f.read()
    # Generar un ID  nico  conservando la extensi n
    file_id = f"{uuid.uuid4()}{os.path.splitext(f.filename)[1]}"
    # Subir a MinIO en el bucket "chat"
    MINIO.put_object(
        Bucket="chat",
        Key=file_id,
        Body=content,
        ContentType=f.content_type
    )
    # Devolver metadatos completos
    return {
        "file": file_id,
        "filename": f.filename,
        "content_type": f.content_type,
        "url": f"http://192.168.122.106:9000/chat/{file_id}"
    }



# ---------- WebSocket ----------
@app.websocket("/ws/{room}")
async def websocket_chat(ws: WebSocket, room: str):
    await ws.accept()
    conn = _get_conn()
```

```
    feed = r.table(TBL_MSGS).filter({"room": room}).changes().run(conn)

    loop = asyncio.get_event_loop()

    async def next_change():
        # Ejecuta 'feed.next()' en un hilo para que no bloquee el loop
        return await loop.run_in_executor(
            FEED_EXECUTOR,
            feed.next
        )

    try:
        while True:
            change = await next_change()
            new = change.get("new_val")
            if not new:
                continue
            new["ts"] = str(new["ts"])
            await ws.send_text(json.dumps(new))

    except Exception:
        pass
    finally:
        try:
            feed.close()
        except Exception:
            pass
        conn.close()
```

En `index.html` Va la interfaz web para el chat:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Chat Distribuido</title>
  <style>
    body{margin:0;font-family:Arial,Helvetica,sans-serif;background:#111;color:#eee}
    .hidden{display:none}
    header{background:#222;padding:10px 20px;display:flex;justify-content:space-between;
        align-items:center}
    header h1{margin:0;font-size:1.2rem}
    button,input{padding:6px;font-size:1rem;border:none;border-radius:4px}
    button{background:#0066cc;color:#fff;cursor:pointer}
    button:hover{background:#005bb5}
    input{background:#222;border:1px solid #444;color:#eee}
    #main{display:flex;height:calc(100vh - 60px)}
    #sidebar{width:250px;border-right:1px solid #333;padding:10px;overflow-y:auto}
    #rooms li{padding:8px;border:1px solid #444;margin-bottom:4px;cursor:pointer;border-
        radius:4px}
    #rooms li:hover{background:#222}
    #chat{flex:1;display:flex;flex-direction:column;padding:10px}
    #messages{flex:1;overflow-y:auto;padding:8px;background:#000;border:1px solid #444;
        border-radius:4px;display:flex;flex-direction:column;gap:6px}
    .msg-wrap{display:flex;flex-direction:column;max-width:100%}
    .mine{align-items:flex-end}.other{align-items:flex-start}
    .bubble{max-width:70%;padding:6px 8px;border-radius:6px;word-wrap:break-word}
    .mine .bubble{background:#0066cc}.other .bubble{background:#444}
    .name{font-size:.7rem;color:#bbb;margin-top:2px}.mine .name{text-align:right}
    #messages img{max-height:150px;border-radius:4px}
    #composer{display:flex;gap:6px;margin-top:8px}
    #composer input[type=text]{flex:1}
    #login-card,#register-card{max-width:300px;margin:80px auto;padding:20px;border:1px
        solid #444;background:#222;border-radius:6px}
    #login-card h2,#register-card h2{text-align:center;margin:0 0 10px}
    #login-card input,#register-card input{width:100%;margin-bottom:10px}
    a{color:#4aaaff;cursor:pointer}
  </style>
```

```
</head>
<body>
  <header>
    <h1>Chat Distribuido</h1>
    <button id="btn-logout" class="hidden">Cerrar Sesi n</button>
  </header>

  <!-- LOGIN -->
  <div id="login-card">
    <h2>Iniciar Sesi n</h2>
    <input id="login-user" placeholder="Usuario" />
    <input id="login-pass" type="password" placeholder="Contrase a" />
    <button id="btn-login">Entrar</button>
    <p style="margin-top:8px;text-align:center"> No  tienes cuenta? <a id="link-register">
        Reg strate</a></p>
    <div id="login-error" style="color:#f44;text-align:center"></div>
  </div>

  <!-- REGISTER -->
  <div id="register-card" class="hidden">
    <h2>Crear Cuenta</h2>
    <input id="reg-user" placeholder="Usuario" />
    <input id="reg-pass" type="password" placeholder="Contrase a" />
    <button id="btn-register">Registrar</button>
    <p style="margin-top:8px;text-align:center"> Ya  tienes cuenta? <a id="link-login">
        Iniciar sesi n</a></p>
    <div id="reg-error" style="color:#f44;text-align:center"></div>
  </div>

  <!-- APP -->
  <div id="app" class="hidden">
    <div id="main">
      <aside id="sidebar">
        <div style="display:flex;justify-content:space-between;align-items:center;margin-
            bottom:6px">
          <span>Salas</span><button id="btn-new-room" style="padding:2px 6px">   </button>
        </div>
        <ul id="rooms"></ul>
      </aside>
      <section id="chat" class="hidden">
        <h2 id="room-title" style="margin:4px 0 8px"></h2>
        <div id="messages"></div>
        <div id="composer">
          <input id="msg-input" placeholder="Escribe un mensaje..." />
          <button id="btn-upload">    </button>
          <button id="btn-send">Enviar</button>
          <input id="file-input" type="file" class="hidden" />
        </div>
      </section>
    </div>
  </div>

<script>
const API = window.location.origin;
const STORAGE = 'http://192.168.122.106:9000/chat/';
let token='', user='', room=''; let pollRooms, pollMsgs, participantCount=0;
const $ = id => document.getElementById(id);
// obtener nombre de archivo desde URL
const getFileName = url => decodeURIComponent(url.split('/').pop());

// Panel toggles
$('link-register').onclick = () => { $('login-card').classList.add('hidden'); $('register-
    card').classList.remove('hidden'); };
$('link-login').onclick    = () => { $('register-card').classList.add('hidden'); $('login-
    card').classList.remove('hidden'); };

// Register
$('btn-register').onclick = async () => {
```

```
  const u = $('reg-user').value.trim(), p = $('reg-pass').value.trim();
  if (!u || !p) return;
  let r = await fetch(API + '/api/register', {method:'POST',headers:{'Content-Type':'
      application/json'},body:JSON.stringify({username:u,password:p})});
  if (r.ok) { $('login-user').value = u; $('login-pass').value = p; $('link-login').click();
       }
  else $('reg-error').textContent = 'Usuario ya existe';
};

// Login
$('btn-login').onclick = async () => {
  const u = $('login-user').value.trim(), p = $('login-pass').value.trim();
  if (!u || !p) return;
  let r = await fetch(API + '/api/login', {method:'POST',headers:{'Content-Type':'
      application/json'},body:JSON.stringify({username:u,password:p})});
  if (!r.ok) { $('login-error').textContent = 'Credenciales inv lidas'; return; }
  token = (await r.json()).token; user = u; localStorage.setItem('chat_jwt', token);
      enterApp();
};

// Auto-login
window.addEventListener('DOMContentLoaded', () => { const saved = localStorage.getItem('
    chat_jwt'); if (saved) { token = saved; try { user = JSON.parse(atob(saved.split('.')
    [1])).sub; } catch{} enterApp(); }});

function enterApp() {
  $('login-card').classList.add('hidden');
  $('register-card').classList.add('hidden');
  $('app').classList.remove('hidden');
  $('btn-logout').classList.remove('hidden');
  loadRooms(); pollRooms = setInterval(loadRooms, 3000);
}
$('btn-logout').onclick = () => { localStorage.removeItem('chat_jwt'); location.reload(); };

async function loadRooms() {
  let r = await fetch(API + '/api/rooms', {headers:{Authorization:'Bearer '+token}});
  let rooms = await r.json(); $('rooms').innerHTML = '';
  rooms.forEach(s => {
    let li = document.createElement('li');
    let label = s.id.split('-').filter(n=>n!==user).join(',');
    li.textContent = label||s.id;
    li.onclick = () => openRoom(s.id, label);
    $('rooms').appendChild(li);
  });
}

$('btn-new-room').onclick = () => {
  const names = prompt('Participantes separados por coma'); if (!names) return;
  let parts = [...new Set(names.split(',').map(s=>s.trim()).filter(Boolean))];
  if (!parts.includes(user)) parts.push(user);
  const id = parts.sort().join('-');
  fetch(API + '/api/room', {method:'POST',headers:{'Content-Type':'application/json',
      Authorization:'Bearer '+token},body:JSON.stringify({id,participants:parts})}).then(
      loadRooms).catch(()=>alert('No se pudo crear'));
};

function openRoom(id,label) {
  room = id; participantCount = id.split('-').length;
  $('room-title').textContent = label;
  $('chat').classList.remove('hidden');
  fetchMsgs(); clearInterval(pollMsgs); pollMsgs = setInterval(fetchMsgs, 800);
}

async function fetchMsgs() {
  if (!room) return;
  let r = await fetch(`${API}/api/history/${room}?limit=200`, {headers:{Authorization:'
      Bearer '+token}});
  if (!r.ok) return;
```

```
  let arr = await r.json();
  $('messages').innerHTML = '';
  arr.reverse().forEach(renderMsg);
}

function renderMsg(m) {
  const wrap = document.createElement('div'); wrap.classList.add('msg-wrap', m.from===user?'
      mine':'other');
  const bubble = document.createElement('div'); bubble.classList.add('bubble');
  if (m.content.startsWith(STORAGE) && /\.(png|jpe?g|gif)$/i.test(m.content)) {
    const img = new Image(); img.src = m.content; bubble.appendChild(img);
  } else if (m.content.startsWith(STORAGE)) {
    const a = document.createElement('a'); a.href = m.content;
    a.textContent = getFileName(m.content);
    a.target = '_blank'; bubble.appendChild(a);
  } else {
    bubble.textContent = m.content;
  }
  wrap.appendChild(bubble);
  if (participantCount>2) {
    const name = document.createElement('span'); name.className='name';
    name.textContent = m.from===user?'T  ':m.from;
    wrap.appendChild(name);
  }
  $('messages').appendChild(wrap);
  $('messages').scrollTop = $('messages').scrollHeight;
}

$('btn-send').onclick = async () => { const txt = $('msg-input').value.trim(); if (!txt)
    return; $('msg-input').value = ''; await postMsg(txt); };
$('msg-input').addEventListener('keyup', e=>{ if (e.key==='Enter') $('btn-send').click(); })
    ;

$('btn-upload').onclick = () => $('file-input').click();
$('file-input').onchange = async e => {
  const f = e.target.files[0]; if (!f) return;
  const fd = new FormData(); fd.append('f', f);
  let up = await fetch(API + '/api/upload', {method:'POST',headers:{Authorization:'Bearer '+
      token},body:fd});
  if (!up.ok) return alert('Error al subir');
  const {url} = await up.json(); await postMsg(url);
};

async function postMsg(content) {
  await fetch(API+'/api/send',{method:'POST',headers:{'Content-Type':'application/json',
      Authorization:'Bearer '+token},body:JSON.stringify({room,content})});
  fetchMsgs();
}
</script>
</body>
</html>
```

5.3. Servicio systemd

```
# /etc/systemd/system/chat-api.service
[Unit]
Description=Chat API (FastAPI + Uvicorn)
After=network.target

[Service]
User=administrador
WorkingDirectory=/home/administrador/srv/chat-api
Environment="PATH=/home/administrador/venv/bin:/usr/bin"
Environment="PYTHONUNBUFFERED=1"
ExecStart=/home/administrador/venv/bin/uvicorn \
  main:app --host 0.0.0.0 --port 8000 --lifespan off --workers 4
```

```
Restart=always

[Install]
WantedBy=multi-user.target
```

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable --now chat-api.service
```

6. Balanceador L7: Caddy

6.1. Instalación

```
$ sudo apt install -y debian-keyring debian-archive-keyring \
  apt-transport-https curl

$ curl -1sLf 'https://dl.cloudsmith.io/public/caddy/stable/gpg.key' | \
  sudo gpg --dearmor -o /usr/share/keyrings/caddy-stable-archive-keyring.gpg

$ curl -1sLf 'https://dl.cloudsmith.io/public/caddy/stable/debian.deb.txt' | \
  sudo tee /etc/apt/sources.list.d/caddy-stable.list

$ sudo apt update
$ sudo apt install -y caddy
```

6.2. Caddyfile (sin cambios)

```
:80 {
    reverse_proxy {
        to 192.168.122.102:8000 192.168.122.103:8000

        lb_policy random
        lb_retries 2
        lb_try_duration 3s

        transport http {
            versions h1
        }
    }
}
```

```
$ sudo systemctl enable --now caddy.service
$ sudo systemctl reload caddy.service
```

Ahora `http://192.168.122.101` balancea tráfico entre ambas APIs.

7. Pruebas de extremo a extremo

1. Navega a `http://192.168.122.101` y carga la interfaz.
2. Regístrate, inicia sesión; verifica en DevTools que las peticiones van a `/api/...`.
3. Abre otro navegador y otro usuario; crea una sala y envía mensajes.
4. Detén `api1`: `sudo systemctl stop chat-api.service` en la 102. Caddy debe redirigir todo a `api2` sin errores 5xx.
5. Sube un archivo y confirma su presencia en `http://192.168.122.106:9000/chat/`.

Referencias

- MinIO – https://min.io/docs/minio/linux/index.html
- RethinkDB – https://rethinkdb.com/docs/install/ubuntu/
- FastAPI – https://fastapi.tiangolo.com/
- Caddy – https://caddyserver.com/docs/

Repositorio GitHub con codigos de main.py y la Web UI de chat:
https://github.com/juancitucs/ChatWebSocker.git