EduNetGuard — Codi font dels scripts principals

Document complementari de la memòria del projecte | CFGS ASIX · Institut Puig Castellar · 2025–2026

Codi principal del worker i del seed

Aquest document conté el codi font dels dos scripts Python principals del projecte. S'adjunta com a document independent per mantenir la memòria principal neta de codi, tal com indica la guia de bones pràctiques. El codi complet es lliura també com a fitxers .py independents.

1 sync_pihole_from_db.py — Worker principal

Script que s'executa cada minut via cron. Connecta amb MariaDB, calcula les franges actives, actualitza el camp enabled a gravity.db de Pi-hole i recarrega les llistes. És el component que uneix la configuració guardada a la base de dades amb el filtratge DNS real.

```
#!/usr/bin/env python3

# Script principal del worker d'EduNetGuard.
# La seva funció és llegir les regles guardades a MariaDB i aplicar l'estat real a
Pi-hole.
# La web només guarda la configuració, però qui toca Pi-hole és aquest script.

import os
import re
import sqlite3
import subprocess
from datetime import datetime
import pymysql
import pymysql.cursors



# Configuració de connexió a MariaDB.
MYSQL = {
    "host": "127.0.0.1",
    "user": "edunetguard",
    "password": "********",  # contrasenya real al servidor
    "database": "edunetguard",
    "cursorclass": pymysql.cursors.DictCursor,
    "autocommit": False,
}

PIHOLE_DB = "/etc/pihole/gravity.db"
PIHOLE_CMD = "/usr/local/bin/pihole"

COMMENT_ALLOW = "EDUNET:ALLOW"
COMMENT_EXTRA_BLOCK = "EDUNET:EXTRA_BLOCK"



def run(cmd, timeout=20):
    try:
        result = subprocess.run(cmd, capture_output=True, text=True,
                                timeout=timeout, check=False)
        if result.stdout.strip():
            print(result.stdout.strip())
        if result.stderr.strip():
            print(result.stderr.strip())
```

```
        return result.returncode
    except subprocess.TimeoutExpired:
        print("[WARN] Timeout executant:", " ".join(cmd))
        return 124


def normalize_domain(domain: str) -> str:
    domain = domain.strip().lower()
    domain = re.sub(r"^https?://", "", domain)
    domain = re.sub(r"/.*$", "", domain)
    domain = re.sub(r":\d+$", "", domain)
    domain = domain.strip(".")
    if not domain or not re.match(r"^[a-z0-9.-]+$", domain):
        raise ValueError(f"Domini no vàlid: {domain}")
    return domain


def domain_to_regex(domain: str) -> str:
    domain = normalize_domain(domain)
    return r"(\.\|^)" + domain.replace(".", r"\.") + r"$"


def get_now():
    now = datetime.now()
    return now.isoweekday(), now.strftime("%H:%M:%S")


def get_conn():
    return pymysql.connect(**MYSQL)


def get_active_category_ids(cur, dia, hora):
    cur.execute("""
        SELECT DISTINCT b.id, b.nombre
        FROM reglas r
        JOIN horarios h ON r.horario_id = h.id
        JOIN bloqueos b ON r.bloqueo_id = b.id
        WHERE r.activa = 1
          AND h.dia = %s
          AND %s >= h.hora_inicio
          AND %s < h.hora_fin
    """, (dia, hora, hora))
    return cur.fetchall()


def get_active_extra_domains(cur, dia, hora):
    cur.execute("""
        SELECT DISTINCT e.dominio, e.tipo
        FROM regla_dominios_extra e
        JOIN horarios h ON e.horario_id = h.id
        WHERE h.dia = %s
          AND %s >= h.hora_inicio
          AND %s < h.hora_fin
    """, (dia, hora, hora))
    return cur.fetchall()


def get_active_allowed_domains(cur, dia, hora):
    cur.execute("""
        SELECT DISTINCT p.dominio, p.tipo
        FROM permitidos_regla p
        JOIN horarios h ON p.horario_id = h.id
        WHERE h.dia = %s
          AND %s >= h.hora_inicio
```

```
          AND %s < h.hora_fin
    """, (dia, hora, hora))
    return cur.fetchall()


def get_all_bloqueos(cur):
    cur.execute("SELECT id, nombre FROM bloqueos ORDER BY id")
    return cur.fetchall()


def comment_for_category(name: str) -> str:
    return "EDUNET:" + name.strip().upper().replace(" ", "_")


def ensure_entry(sql, entry_type: int, value: str, comment: str):
    row = sql.execute("""
        SELECT id, comment, enabled FROM domainlist
        WHERE type = ? AND domain = ? LIMIT 1
    """, (entry_type, value)).fetchone()
    if row:
        current_comment = row[1] or ""
        if current_comment in ("", COMMENT_ALLOW, COMMENT_EXTRA_BLOCK):
            sql.execute("UPDATE domainlist SET comment = ? WHERE id = ?",
                        (comment, row[0]))
        return
    sql.execute("""
        INSERT INTO domainlist (type, domain, enabled, comment)
        VALUES (?, ?, 0, ?)
    """, (entry_type, value, comment))


def load_managed_entries(sql):
    rows = sql.execute("""
        SELECT id, type, domain, enabled, comment
        FROM domainlist WHERE comment LIKE 'EDUNET:%'
    """).fetchall()
    managed = {}
    for row in rows:
        managed[(row[1], row[2])] = {
            "id": row[0], "type": row[1], "domain": row[2],
            "enabled": row[3], "comment": row[4] or "",
        }
    return managed


def main():
    if os.geteuid() != 0:
        print("Executa amb sudo")
        return
    dia, hora = get_now()
    print(f"[INFO] Dia {dia} Hora {hora}")
    db = get_conn()
    cur = db.cursor()
    sql = sqlite3.connect(PIHOLE_DB)
    try:
        active_categories = get_active_category_ids(cur, dia, hora)
        active_category_ids = {row["id"] for row in active_categories}
        active_category_comments = {comment_for_category(row["nombre"]) for row in
active_categories}
        active_extra = get_active_extra_domains(cur, dia, hora)
        active_allow = get_active_allowed_domains(cur, dia, hora)
        for row in active_extra:
            value = domain_to_regex(row["dominio"]) if row["tipo"] == "regex" else
normalize_domain(row["dominio"])
            ensure_entry(sql, 3, value, COMMENT_EXTRA_BLOCK)
```

```
        for row in active_allow:
            value = domain_to_regex(row["dominio"]) if row["tipo"] == "regex" else
normalize_domain(row["dominio"])
            ensure_entry(sql, 2, value, COMMENT_ALLOW)
        managed = load_managed_entries(sql)
        desired = {key: 0 for key in managed.keys()}
        for key, row in managed.items():
            if row["comment"] in active_category_comments:
                desired[key] = 1
        for row in active_extra:
            value = domain_to_regex(row["dominio"]) if row["tipo"] == "regex" else
normalize_domain(row["dominio"])
            desired[(3, value)] = 1
        for row in active_allow:
            value = domain_to_regex(row["dominio"]) if row["tipo"] == "regex" else
normalize_domain(row["dominio"])
            desired[(2, value)] = 1
        changed = False
        for key, row in managed.items():
            new_enabled = desired.get(key, 0)
            if int(row["enabled"]) != int(new_enabled):
                print(f"[CANVI] {row['comment']} | {row['domain']} ->
{new_enabled}")
                sql.execute("UPDATE domainlist SET enabled = ? WHERE id = ?",
                            (new_enabled, row["id"]))
                changed = True
        if changed:
            sql.commit()
            run([PIHOLE_CMD, "reloadlists"])
            print("[OK] Pi-hole actualitzat")
        else:
            print("[=] Sense canvis")
        all_bloqueos = get_all_bloqueos(cur)
        for row in all_bloqueos:
            aplicado = 1 if row["id"] in active_category_ids else 0
            cur.execute("""
                INSERT INTO estado_bloqueos (bloqueo_id, aplicado) VALUES (%s, %s)
                ON DUPLICATE KEY UPDATE aplicado = VALUES(aplicado)
            """, (row["id"], aplicado))
        db.commit()
    finally:
        sql.close()
        db.close()


if __name__ == "__main__":
    main()
```

Nota: *La contrasenya de MariaDB apareix emmascarada (********). La contrasenya real es troba al fitxer /var/www/html/backend/config.php al servidor.*

2 seed_pihole_categories.py — Script d'inicialització

Script que s'executa una sola vegada abans de posar en marxa el sistema per primera vegada. Crea totes les entrades EDUNET a gravity.db amb enabled=0 i fa una còpia de seguretat de la base de dades de Pi-hole. Sense aquest pas previ, el worker no tindria entrades per activar o desactivar.

```
#!/usr/bin/env python3

import os, sys, time, shutil, sqlite3, subprocess

DB_PATH = "/etc/pihole/gravity.db"
PIHOLE  = "/usr/local/bin/pihole"
COMMENT_PREFIX = "EDUNET"

CATEGORIES = {
    "IA": [
        r"(\.\|^)chatgpt\.com$", r"(\.\|^)openai\.com$",
        r"(\.\|^)claude\.ai$", r"(\.\|^)anthropic\.com$",
        r"(\.\|^)gemini\.google\.com$", r"(\.\|^)copilot\.microsoft\.com$",
        r"(\.\|^)perplexity\.ai$", r"(\.\|^)meta\.ai$",
        r"(\.\|^)deepseek\.com$", r"(\.\|^)grok\.com$",
        r"(\.\|^)x\.ai$", r"(\.\|^)poe\.com$",
        r"(\.\|^)mistral\.ai$",
    ],
    "JUEGOS": [
        r"(\.\|^)roblox\.com$", r"(\.\|^)rbxcdn\.com$",
        r"(\.\|^)steamcommunity\.com$", r"(\.\|^)steampowered\.com$",
        r"(\.\|^)friv\.com$", r"(\.\|^)crazygames\.com$",
        r"(\.\|^)poki\.com$", r"(\.\|^)miniclip\.com$",
    ],
    "REDES_SOCIAL": [
        r"(\.\|^)tiktok\.com$", r"(\.\|^)instagram\.com$",
        r"(\.\|^)x\.com$", r"(\.\|^)twitter\.com$",
        r"(\.\|^)facebook\.com$", r"(\.\|^)fbcdn\.net$",
        r"(\.\|^)pinterest\.com$",
    ],
    "STREAMING": [
        r"(\.\|^)youtube\.com$", r"(\.\|^)youtu\.be$",
        r"(\.\|^)twitch\.tv$", r"(\.\|^)netflix\.com$",
        r"(\.\|^)spotify\.com$",
    ],
    "DESCARGAS": [
        r"(\.\|^)mega\.nz$", r"(\.\|^)mediafire\.com$",
        r"(\.\|^)uptodown\.com$", r"(\.\|^)softonic\.com$",
    ],
    "COMPRAS_ONLINE": [
        r"(\.\|^)amazon\.es$", r"(\.\|^)aliexpress\.com$",
        r"(\.\|^)shein\.com$", r"(\.\|^)temu\.com$",
    ],
    "FOROS_OCIO": [
        r"(\.\|^)reddit\.com$", r"(\.\|^)9gag\.com$",
        r"(\.\|^)forocoches\.com$",
    ],
}


def die(msg, code=1):
    print(f'[ERROR] {msg}'); sys.exit(code)

def require_root():
    if os.geteuid() != 0: die('Executa amb sudo')

def run(cmd):
    r = subprocess.run(cmd, capture_output=True, text=True, check=False)
```

```
    if r.stdout.strip(): print(r.stdout.strip())
    if r.stderr.strip(): print(r.stderr.strip())
    return r.returncode

def backup_db():
    ts  = time.strftime('%Y%m%d-%H%M%S')
    dst = f'{DB_PATH}.bak.seed.{ts}'
    shutil.copy2(DB_PATH, dst)
    print(f'[OK] Backup: {dst}')

def ensure_schema(conn):
    cur  = conn.cursor()
    cur.execute('PRAGMA table_info(domainlist)')
    cols = {row[1] for row in cur.fetchall()}
    missing = {'id','type','domain','enabled','comment'} - cols
    if missing: die(f'Falten columnes: {missing}')

def upsert_regex(conn, category, pattern):
    comment = f'{COMMENT_PREFIX}:{category}'
    cur     = conn.cursor()
    cur.execute('SELECT id FROM domainlist WHERE type=3 AND domain=?', (pattern,))
    row = cur.fetchone()
    if row is None:
        cur.execute('INSERT INTO domainlist (type,domain,enabled,comment) VALUES
(3,?,0,?)',
                    (pattern, comment))
        return 'insertat'
    cur.execute('UPDATE domainlist SET enabled=0, comment=? WHERE id=?',
                (comment, row[0]))
    return 'actualitzat'

def main():
    require_root()
    if not os.path.exists(DB_PATH): die(f'No existeix {DB_PATH}')
    if not os.path.exists(PIHOLE):  die(f'No existeix {PIHOLE}')
    backup_db()
    conn = sqlite3.connect(DB_PATH)
    try:
        ensure_schema(conn)
        resum = {}
        for category, patterns in CATEGORIES.items():
            resum[category] = {'insertat': 0, 'actualitzat': 0}
            for pattern in patterns:
                estat = upsert_regex(conn, category, pattern)
                resum[category][estat] += 1
        conn.commit()
        print('\n[OK] Resum:')
        for cat, d in resum.items():
            print(f'  - {cat}: {d["insertat"]} insertat(s), {d["actualitzat"]}
actualitzat(s)')
    finally:
        conn.close()
    rc = run([PIHOLE, 'reloadlists'])
    if rc != 0: die('Ha fallat pihole reloadlists')
    print('\n[OK] Pi-hole recarregat')
    print('[OK] Totes les entrades EDUNET queden desactivades (enabled=0)')

if __name__ == '__main__':
    main()
```

Nota: *Les llistes completes de dominis per categoria es poden consultar a l'Annex G de la memòria principal.*