Documentación de Infraestructura y Servicios

Este documento describe la arquitectura, orquestación y funcionamiento de cada servicio definido en `docker-compose.yml`, explicando los Dockerfiles bajo `services/`, la configuración especial del proxy de base de datos (`haproxy.cfg`), el directorio `scripts/` (backups, restore y cleanup) y los archivos clave de la base de datos (en especial `pg-hba-fixer.sh`).

1. Visión general (docker-compose)

- Redes definidas:
  - `frontend-network`: tráfico de frontends y gateway.
  - `backend-network`: tráfico entre gateway y microservicios, y monitoreo.
  - `database-network`: capa de datos (PostgreSQL maestro/réplica, proxy) y servicios que acceden a DB.
- Volúmenes:
  - `postgres-master-data`, `postgres-replica-data`: persistencia de PostgreSQL.
  - `redis-data`: persistencia de Redis (AOF).
  - `prometheus-data`, `grafana-data`, `loki-data`: monitoreo y logs.
  - `backups`: artefactos de respaldo de DB y configuraciones.
- Salud y dependencias:
  - Healthchecks para Postgres, Redis, AI y otros.
  - `depends_on` y un job `pg-hba-fixer` para preparar el maestro antes de la réplica.
- API Gateway (Nginx) enruta a microservicios y frontends estáticos.
- Algunos servicios tienen una segunda réplica (`*-service-2`) para balanceo a nivel del gateway.

Archivo: `docker-compose.yml`

2. Capa de datos

2.1 PostgreSQL maestro

- Servicio: `postgres-master` (imagen `postgres:15-alpine`).
- Archivos montados:
  - `infrastructure/database/init.sql`: esquema, índices, triggers y datos de ejemplo.
  - `infrastructure/database/master.conf`: parámetros de réplica (`wal_level=replica`, `hot_standby=on`, etc).
- Variables: `POSTGRES_DB`, `POSTGRES_USER`, `POSTGRES_PASSWORD`, `REPLICATION_USER`, `REPLICATION_PASSWORD`.
- Healthcheck: `pg_isready -U admin -d ${DB_NAME}`.

Archivos relevantes:

- `infrastructure/database/master.conf`
- `infrastructure/database/init.sql`

2.2 Preparación de `pg_hba.conf` y rol de réplica (pg-hba-fixer)

- Servicio: `pg-hba-fixer` (job de una sola corrida, imagen `postgres:15-alpine`).
- Función: prepara el maestro para aceptar conexiones de replicación y asegura el rol de replicación.
- Script ejecutado: `infrastructure/database/pg-hba-fixer.sh`.
  - Espera al maestro con `pg_isready`.
  - Preprende a `pg_hba.conf` la línea (si no existe):
    - `host replication <REPLICATION_USER> 0.0.0.0/0 scram-sha-256`.
  - Crea el rol de replicación si no existe:
    - `CREATE ROLE <REPLICATION_USER> WITH LOGIN REPLICATION PASSWORD '<REPLICATION_PASSWORD>';`
  - `SELECT pg_reload_conf();` para recargar configuración.
- Variables usadas: `PGDATA`, `REPLICATION_USER`, `REPLICATION_PASSWORD`, `DB_USER`, `DB_NAME`, `PGPASSWORD`.

Archivo: `infrastructure/database/pg-hba-fixer.sh`

Nota: En el repositorio el archivo es `.sh` (shell), no `.hs`.

2.3 PostgreSQL réplica

- Servicio: `postgres-replica` (build propio).
- Dockerfile: `services/postgres-replica/Dockerfile`.
- Entrypoint: `services/postgres-replica/entrypoint.sh`.
  - Si el directorio de datos está vacío, espera al maestro, limpia datos y realiza `pg_basebackup` con `-R` (genera `primary_conninfo`).
  - Arranca PostgreSQL como réplica.
- Variables: `REPLICATION_USER`, `REPLICATION_PASSWORD`, `POSTGRES_PASSWORD`.

Archivos: `services/postgres-replica/Dockerfile`, `services/postgres-replica/entrypoint.sh`

2.4 Proxy de base de datos (HAProxy)

- Servicio: `db-proxy` (build de `infrastructure/db-proxy`).
- Objetivo: exponer un endpoint único para las apps y manejar failover hacia la réplica si el maestro cae.
- Puerto host: `5433` mapeado al `5432` del contenedor (HAProxy escucha en 5432 dentro).
- Dockerfile: `infrastructure/db-proxy/Dockerfile` (instala cliente `psql` y copia `haproxy.cfg`).
- Configuración HAProxy: `infrastructure/db-proxy/haproxy.cfg`

Explicación de `haproxy.cfg`:

- `global` y `defaults`:
  - Logs a stdout; sockets admin.
  - `mode tcp`, `option tcplog`, timeouts largos (sesiones de DB), `retries 3`.
- `frontend postgres_frontend`:
  - `bind *:5432` y `default_backend postgres_backend`.
- `backend postgres_backend`:
  - `mode tcp`, `balance roundrobin`.
  - Keepalives TCP: `option clitcpka`, `option srvtcpka`.
  - `default-server on-marked-down shutdown-sessions on-marked-up shutdown-backup-sessions`:

        - Si un servidor se marca down, cierra sesiones hacia él.
        - Si uno vuelve up, cierra sesiones de backup.
    - Servidores:
        - `server postgres-master postgres-master:5432 check inter 2000 rise 2 fall 3 maxconn 200`
        - `server postgres-replica postgres-replica:5432 check inter 2000 rise 2 fall 3 backup maxconn 200`
    - Efecto: el tráfico normal va al maestro; la réplica entra como backup ante caída del maestro.

Archivos: `infrastructure/db-proxy/Dockerfile`, `infrastructure/db-proxy/haproxy.cfg`

2.5 Redis

- Servicio: `redis` (imagen `redis:7-alpine`).
- Comando: `redis-server --appendonly yes --requirepass ${REDIS_PASSWORD}`.
- Volumen: `redis-data:/data`.
- Puerto: `6379:6379`.
- Healthcheck con `redis-cli ping`.

3. API Gateway (Nginx)

- Servicio: `api-gateway` (imagen `nginx:alpine`).
- Configuración: `services/api-gateway/nginx.conf`.
- Upstreams para cada backend (algunos con 2 réplicas).
- Rutas API:
    - `/api/auth/` → `auth-service`
    - `/api/config/` → `config-service`
    - `/api/appointments/` → `appointment-service`
    - `/api/queue/` → `queue-service`
    - `/api/notifications/` → `notification-service`
    - `/api/statistics/` → `statistics-service`
    - `/api/ai/` → `ai-service`
- Rutas Frontend:
    - `/auth/` → `auth-frontend`
    - `/config/` → `config-frontend`
    - `/admin/` → `admin-frontend`
    - `/totem/` → `totem-frontend`
    - `/` → `lobby-frontend`

Archivo: `services/api-gateway/nginx.conf`

4. Frontends estáticos (Nginx)

- `config-frontend`: sirve `services/config-service/frontend`.
- `auth-frontend`: sirve `services/auth-service/frontend`.
- `lobby-frontend`: sirve `services/lobby-frontend`.
- `admin-frontend`: sirve `services/admin-service/frontend`.
- `totem-frontend`: sirve `services/totem-service/frontend`.

Cada uno monta el directorio correspondiente como `/usr/share/nginx/html` (solo lectura) y se conecta a `frontend-network`.

5. Microservicios (Node.js)

Patrón común en Dockerfiles:

- Multi-stage (builder con `npm install` y runtime con `npm ci --only=production`).
- Usuario no privilegiado `nodejs` (uid 1001) y `chown` del directorio de trabajo.
- `WORKDIR /app`, copia de `package*.json` y luego el código.
- Exponen su puerto y arrancan con `node src/index.js` o `npm start`.

Servicios:

- Auth
  - Servicio: `auth-service` (y `auth-service-2`). Puerto contenedor: `3001`.
  - Dockerfile: `services/auth-service/Dockerfile`.
  - Env: DB (vía `db-proxy`), Redis, JWT (`JWT_SECRET`, `JWT_EXPIRATION`).
  - Healthcheck HTTP a `/health` dentro del contenedor.
- Config
  - Servicio: `config-service` (y `config-service-2`). Puerto: `3002`.
  - Dockerfile: `services/config-service/Dockerfile`.
  - Env: DB, Redis.
- Appointments
  - Servicio: `appointment-service` (y `appointment-service-2`). Puertos: `3000`/`3003`.
  - Dockerfile: `services/appointment-service/Dockerfile`.
  - Env: DB, Redis, JWT.
- Queue
  - Servicio: `queue-service` (y `queue-service-2`). Puerto: `3004`.
  - Dockerfile: `services/queue-service/Dockerfile`.
  - Env: DB, Redis.
- Notifications
  - Servicio: `notification-service` (y `notification-service-2`). Puerto: `3005`.
  - Dockerfile: `services/notification-service/Dockerfile`.
  - Env: DB y SMTP (`SMTP_HOST`, `SMTP_PORT`, `SMTP_USER`, `SMTP_PASSWORD`).
- Statistics
  - Servicio: `statistics-service` (y `statistics-service-2`). Puerto: `3006`.
  - Dockerfile: `services/statistics-service/Dockerfile`.
  - Env: DB.
- AI

    - Servicio: `ai-service`. Puerto: `3007`.
    - Dockerfile: `services/ai-service/Dockerfile` (single-stage con `npm install --production`).
    - Env: `GEMINI_API_KEY` y `NODE_ENV`.

6. Monitoreo

- Loki: `grafana/loki`, volumen `loki-data`, puerto `3100:3100`.
- Promtail: lee `docker.sock` y envía a Loki. Config: `infrastructure/monitoring/promtail/promtail-config.yml`.
- Prometheus: config en `infrastructure/monitoring/prometheus/prometheus.yml`, datos en `prometheus-data`, puerto `9090:9090`.
- Grafana: `grafana/grafana`, datasources en `infrastructure/monitoring/grafana/provisioning/datasources`, puerto `3010:3000`, admin password `GRAFANA_PASSWORD`.

7. Scripts (backups, restore, limpieza)

Los scripts viven en `scripts/` (host) y también existen copias equivalentes en `services/backup-service/scripts/` (para la imagen con cron). En el `docker-compose.yml` se monta `./scripts` dentro del contenedor `backup-service` en `/scripts` y el volumen `backups` en `/backups`.

- Backup de base de datos
    - Archivo: `scripts/backup_db.sh`
    - Hace `pg_dump` contra `db-proxy` y comprime a `/backups/db_backup_YYYYmmddHHMMSS.sql.gz`.
    - Requiere `DB_NAME`, `DB_USER`, `DB_PASSWORD` y que el stack esté arriba.
    - Ejecución:
        - `docker compose exec backup-service sh -lc "/scripts/backup_db.sh"`
- Backup de configuraciones
    - Archivo: `scripts/backup_configs.sh`
    - Copia configs críticas a un directorio temporal y empaqueta a `/backups/configs_backup_YYYYmmddHHMMSS.tar.gz`.
    - Incluye: `/etc/nginx/nginx.conf`, `/etc/prometheus/prometheus.yml` (extensible para Loki/Promtail/Grafana).
    - Ejecución:
        - `docker compose exec backup-service sh -lc "/scripts/backup_configs.sh"`
- Limpieza de backups antiguos
    - Archivo: `scripts/cleanup_backups.sh`
    - Borra archivos en `/backups` con más de 7 días (`RETENTION_DAYS=7`).
    - Ejecución:
        - `docker compose exec backup-service sh -lc "/scripts/cleanup_backups.sh"`
- Restaurar base de datos
    - Archivo: `scripts/restore_db.sh`

  - Uso: `restore_db.sh <path_to_backup_file.sql.gz>`.
  - Elimina y recrea la DB indicada y restaura desde el backup comprimido.
  - Ejecución (suponiendo el archivo está en el volumen `backups`):
    - `docker compose exec backup-service sh -lc "/scripts/restore_db.sh /backups/db_backup_YYYYmmddHHMMSS.sql.gz"`
  - Advertencia: en producción, evalúa restauraciones parciales o a otra DB para evitar pérdida de datos.

- Restaurar configuraciones
  - Archivo: `scripts/restore_configs.sh`
  - Uso: `restore_configs.sh <path_to_backup_file.tar.gz>`.
  - Descomprime a un directorio temporal y repone archivos en `/etc/nginx/nginx.conf` y `/etc/prometheus/prometheus.yml` (extensible).
  - Ejecución:
    - `docker compose exec backup-service sh -lc "/scripts/restore_configs.sh /backups/configs_backup_YYYYmmddHHMMSS.tar.gz"`

8. Servicio de Backups en compose vs imagen con cron

- En `docker-compose.yml`, `backup-service` usa `postgres:15-alpine` y queda en espera (`tail -f /dev/null`) para ejecutar scripts bajo demanda. Montajes importantes:
  - `./scripts:/scripts`, `backups:/backups`, `nginx.conf` y `prometheus.yml` (solo lectura), `./.env:/app/.env:ro`.
- Existe un Dockerfile alternativo: `services/backup-service/Dockerfile` que:
  - Instala `dcron`, `postgresql-client`, `gzip`, `tar`, `findutils`, `bash`.
  - Copia y da permisos a los scripts en `/usr/local/bin`.
  - Registra un cron diario 02:00: `backup_db.sh && backup_configs.sh && cleanup_backups.sh`.
  - Comando: `crond -f -L /var/log/cron.log`.
- Para usar el modo automático (cron), habría que modificar el servicio `backup-service` en `docker-compose.yml` para que haga `build` con `services/backup-service/` y no mantener `tail` en primer plano.

9. Archivos de base de datos (detalle)

- `infrastructure/database/init.sql`
  - Crea extensiones (`uuid-ossp`, `pgcrypto`).
  - Define tablas: `usuarios`, `direcciones_municipales`, `unidades`, `servicios`, `agendamientos`, `turnos`, `estaciones`, `estadisticas`, `notificaciones`.
  - Índices para rendimiento: emails, RUT, estado/fechas, etc.
  - Triggers para actualizar `ultima_actualizacion` y funciones auxiliares.
  - Incluye inserciones de ejemplo/comentarios para desarrollo.

- `infrastructure/database/master.conf`
  - `wal_level = replica`, `max_wal_senders = 5`, `hot_standby = on`, `password_encryption = scram-sha-256`.

- `infrastructure/database/pg-hba-fixer.sh`
  - Job idempotente que garantiza que `pg_hba.conf` permita la replicación y que exista el rol de replicación.
  - Trabaja sobre el volumen del maestro (`postgres-master-data`).
- `infrastructure/database/init-master.sh`
  - Script alternativo para preparar el maestro (no montado por defecto en compose): crea rol de replicación, modifica `pg_hba.conf` y recarga configuración.

10. Puertos y acceso

- Base de datos para clientes:
  - Dentro de la red de Docker: usar `db-proxy:5432`.
  - Desde host: `localhost:5433` (mapeado a HAProxy).
- Redis: `localhost:6379` (si se requiere desde host), dentro de Docker usar `redis:6379`.
- API Gateway: `localhost:8080`.
- Grafana: `localhost:3010`, Prometheus: `localhost:9090`, Loki: `localhost:3100`.
- Frontends: `config-frontend` expone `80:80`; los demás frontends están en red interna y accesibles vía gateway (`/auth`, `/admin`, etc.).

11. Operaciones comunes

- Levantar todo: `docker compose up -d`.
- Ver logs de un servicio: `docker compose logs -f <servicio>`.
- Ejecutar un script de backup: `docker compose exec backup-service sh -lc "/scripts/backup_db.sh"`.
- Restaurar DB: `docker compose exec backup-service sh -lc "/scripts/restore_db.sh /backups/archivo.sql.gz"`.
- Limpiar backups: `docker compose exec backup-service sh -lc "/scripts/cleanup_backups.sh"`.

12. Recomendaciones de seguridad y operación

- `pg_hba.conf`: la regla `0.0.0.0/0` es amplia y adecuada para entornos de laboratorio con Docker; en producción restringir a la subred de Docker o a IPs específicas.
- Credenciales: mantener `.env` fuera de control de versiones y rotar secretos en producción.
- Restauraciones: en producción usar restauración a una DB alterna o bajo ventanas de mantenimiento.
- HAProxy: para failover real activo en primario/replicación, contemplar promoción controlada del standby o un mecanismo externo (patroni/keepalived) si se requiere HA automática.

Cualquier cambio en `docker-compose.yml` o en los Dockerfiles debe realizarse con cuidado para mantener compatibilidad entre servicios, redes y variables de entorno.