EzyGas — Current-State Inventory (Redacted)

Purpose: Factual snapshot of the live Replit deployment, to serve as the input for an AWS migration runbook. **Redaction policy:** Secret *names* are listed; **no secret values** appear in this file. Gathered from inside the Repl on 2026-07-09. **Source of truth:** `.replit`, `package.json`, `drizzle.config.ts`, `shared/schema.ts`, `server/*`, `migrations/`.

---

1. Runtime & Platform

Item	Value
Host	Replit (`REPL_ID` `92cc62a7-…`, cluster `picard`)
Nix channel	`stable-24_05`
Modules	`nodejs-20`, `web`, `javascript`, `postgresql-16`
Node / npm	v20.20.0 / 10.8.2
Dev command	`npm run dev` → `NODE_ENV=development tsx server/index.ts`

Build	`npm run build` → `tsx script/build.ts` (esbuild → `dist/index.cjs`)
Prod start	`node ./dist/index.cjs`
Deployment target	`autoscale`
App framework	Express 5 (ESM) + Vite/React 18 client (SSR meta layer)

Ports

localPort	externalPort	Purpose
5000	80	Main web/API server (`PORT=5000`)
8081	8081	Expo mobile (customer) dev/metro
8082	—	Expo mobile (driver / "EzyMan") dev/metro

Workflows (Replit "run" orchestration)

- **Project** (parallel): Start application + Start mobile + Start driver app + regression.
- **Start application**: kills stray `tsx server/index.ts`, runs `npm run dev`, waits on port 5000.
- **Start mobile / driver**: Expo 54 via tunnel, `--legacy-peer-deps`.
- **regression** (validation gate): `npm run test:regression` (Playwright).
- **postMerge hook**: `scripts/post-merge.sh` (60s timeout).

---

2. External Services (what to reprovision on AWS)

Service	Role	Client lib	Wiring file	Secret names (values masked)
Neon Postgres	Primary DB (prod)	`pg` Pool + `drizzle-orm/node-postgres`	`server/db.ts`	`NEON_DATABASE_URL`, `NEON_API_KEY`
Replit/local Postgres	Dev DB fallback	same	`server/db.ts`	`DATABASE_URL`, `PG{HOST,PORT,USER,PASSWORD,DATABASE}`
Upstash Redis	Cache, rate-limit, sessions	`@upstash/redis`, `@upstash/ratelimit`	`cache.ts`, `redis-limiter.ts`, `session-store.ts`	`UPSTASH_REDIS_REST_URL`, `UPSTASH_REDIS_REST_TOKEN`
Cloudflare R2	Object storage (uploads, backups)	`@aws-sdk/client-s3` (S3-compatible)	`server/r2.ts`	`R2_ACCOUNT_ID`, `R2_ACCESS_KEY_ID`, `R2_SECRET_ACCESS_KEY`, `R2_BUCKET_NAME`, `R2_PUBLIC_URL`
Paymob	Payments (card + wallet)	`fetch` + HMAC	`server/paymob.ts`	`PAYMOB_API_KEY`, `PAYMOB_HMAC_SECRET`, `PAYMOB_CARD_INTEGRATION_ID`, `PAYMOB_WALLET_INTEGRATION_ID`, `PAYMOB_IFRAME_ID`
SMS Misr	OTP / transactional SMS	`fetch` URLSearchParams	`server/sms.ts`	`SMSMISR_USERNAME`, `SMSMISR_PASSWORD`, `SMSMISR_SENDER`
SMTP (mail.ezy-gas.com)	Email	`nodemailer`	(mail layer)	`SMTP_HOST`, `SMTP_PORT`, `SMTP_USER`, `SMTP_PASS`

Web Push (VAPID)	Browser push	`web-push`	`push-notifications.ts`	`VAPID_PUBLIC_KEY`, `VAPID_PRIVATE_KEY`, `VAPID_EMAIL`
Zoho Books	Accounting sync	`googleapis`/REST	`zoho-*.ts`	`ZOHO_CLIENT_ID`, `ZOHO_CLIENT_SECRET`, `ZOHO_REFRESH_TOKEN`, `ZOHO_ORG_ID`
Discord	Ops alerts / bot	`discord.js`	`discord-bot.ts`	`DISCORD_BOT_TOKEN`, `DISCORD_ALERT_WEBHOOK`
Sentry	Error monitoring	`@sentry/node`, `@sentry/react`	multiple	`SENTRY_DSN`
Gmail / Google	Mail + Search Console	`googleapis`	`server/gmail.ts`	`GOOGLE_SEARCH_CONSOLE_VERIFICATION`
Replit Auth (OIDC)	Login	`openid-client`, `passport`	auth layer	`ISSUER_URL`, `REPL_ID`, `SESSION_SECRET`

Note: `SLACK_*` env vars still present but Slack integration was removed (commits `a109eff`, `92961a9`, `d6f828b`). Candidate for cleanup — do not carry to AWS.

App-level secrets (not third-party)

Name	Used by	Notes
`NATIONAL_ID_ENCRYPTION_KEY`	`server/encryption.ts`	AES encryption of driver national IDs (PII) — **must migrate the exact key or data is unreadable**
`SESSION_SECRET`	session middleware	rotate-safe (logs everyone out)

`OPS_DASHBOARD_TOKEN`	operational metrics endpoint	bearer auth (commit `950c380`)
`IOT_INGEST_KEY`	cylinder-reading ingest	device auth
`ADMIN_EMAIL` / `ADMIN_PASSWORD` / `ADMIN_DISPLAY_NAME`	bootstrap admin	seed only

---

3. Database

- **ORM:** Drizzle (`drizzle-orm` 0.45, `drizzle-kit` 0.31). Schema: `shared/schema.ts` (~79 KB).
- **Migrations:** `migrations/` — 8 SQL migrations `0000`→`0007` + `meta/` snapshots + `_journal.json`.
- **Prod migration DB:** `NEON_DATABASE_URL`; dev: `DATABASE_URL` (see `drizzle.config.ts`).
- **Pooling:** `server/db.ts` derives a PgBouncer pooler URL in prod (`-pooler` slug + `?pgbouncer=true`).
- **Backups:** `server/backup.ts` runs `pg_dump` → gzip → upload to R2 (scheduled).
- **Tables (47):**

```
addresses, admin_users, areas, b2b_receivables, cash_ledger, cities,

customer_ledger, customer_notes, cylinder_custody_events,
cylinder_readings,

cylinders, deposit_ledger, distributors, driver_commissions,

driver_join_requests, driver_locations, drivers, events_ledger,
governorates,

hubs, incoming_assets, inventory_ledger, inventory_nodes,
job_applications,

job_listings, manager_scopes, mobile_sessions, order_events, orders,

organizations, organization_sites, project_tasks, push_subscriptions,
```

`reconciliations, reseller_applications, reseller_drivers, reseller_orders, resellers, route_plans, route_stops, service_tiers, slack_conversations, slack_memories, slack_messages, tickets, user_profiles, zoho_sync_log`

`slack_conversations` / `slack_memories` / `slack_messages` are orphaned by the Slack removal — verify before carrying over.

EzyGas — 10,000-User Pilot Migration Plan

Goal: Move EzyGas from Replit (single-instance, ephemeral) to a production-grade host capable of serving 10,000 concurrent users reliably.

Estimated total effort: 5–7 days
Target platform: Google Cloud Run (recommended — the codebase already has a Cloud Run-compatible SIGTERM handler)
Fallback options: Railway or Render (zero-config, faster to set up, less ops overhead)

Current State Audit

Area	Current State	Gap
Hosting	Replit single instance	Ephemeral, no auto-scaling

OTP store	Redis + in-memory fallback	Works if Upstash credentials are set
File uploads	R2 + local disk fallback	Works if R2 credentials are set
Rate limiting	In-memory `express-rate-limit`	Not multi-instance safe
CORS origins	`*.replit.app`, `ezy-gas.com`	Must remove Replit origins in prod
CSP headers	Disabled	XSS exposure
Discord bot	401 (invalid token)	Ops visibility broken
Build	`npm run build` + `npm start`	Already works
Graceful shutdown	SIGTERM/SIGINT handler, 8s drain	Already Cloud Run compatible
DB connection	Neon + PgBouncer pooler	Already production-grade
Backups	Daily pg_dump → R2	Already in place
Error tracking	Sentry	Already in place

Phase 0 — Code Fixes (Day 0–1)

These are changes to make in the current repo before deploying anywhere new. None of them are breaking changes.

0-A Switch to Redis-backed rate limiting

`server/redis-limiter.ts` already exists and uses `@upstash/ratelimit` (sliding window, multi-instance safe). It is **not wired up** — `server/index.ts` uses the in-memory `express-rate-limit` instead. Swap them.

In `server/index.ts`, replace the four in-memory limiters with the Redis ones:

```
-import rateLimit from "express-rate-limit";

+import {

+  generalLimiter,

+  authLimiter,

+  orderLimiter,

+  otpLimiter,

+} from "./redis-limiter";

-  const generalLimiter = rateLimit({ windowMs: 15 * 60 * 1000, max: 300,
... });

-  const authLimiter    = rateLimit({ windowMs: 15 * 60 * 1000, max: 20,
... });

-  const orderLimiter   = rateLimit({ windowMs: 15 * 60 * 1000, max: 10,
... });

-  const otpLimiter     = rateLimit({ windowMs: 10 * 60 * 1000, max: 5,
... });
```

The Redis limiter falls back to `next()` (no blocking) if Upstash is unavailable, so it degrades gracefully the same way the old one did.

Also reconcile the limits in `redis-limiter.ts` — they are currently set much higher than the in-memory ones (1000 vs 300 general, 50 vs 20 auth). Set them to the production values that match your SLA and abuse-prevention requirements.

0-B Remove Replit keep-alive ping

`server/index.ts` lines ~400–407 ping `/api/health` every 4 minutes to prevent Replit from throttling the process. This is harmless on other platforms but is dead code. Delete this block after migration.

0-C Tighten CORS for production

Add a `NODE_ENV` guard so Replit origins are only allowed in development:

```
const allowedOrigins = [
  /ezy-gas\.com$/,
  /localhost/,
  /127\.0\.0\.1/,
  // Dev-only
  ...(process.env.NODE_ENV !== "production"
    ? [/\.replit\.app$/, /\.repl\.co$/, /\.expo\.dev$/, /\.worf\.replit\.dev$/]
    : []),
];
```

0-D Re-enable CSP headers

In `server/index.ts`, change `contentSecurityPolicy: false` to a real policy. Start permissive and tighten after verifying no breakage:

```
helmet({
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      scriptSrc:  ["'self'", "'unsafe-inline'"],   // tighten to nonce-based after testing
```

```
      styleSrc:   ["'self'", "'unsafe-inline'"],
      imgSrc:     ["'self'", "data:", "https:"],
      connectSrc: ["'self'", "https:"],
      fontSrc:    ["'self'", "https:"],
      frameSrc:   ["'none'"],
    },
  },
  crossOriginEmbedderPolicy: false,
})
```

0-E Fix Discord bot token

Re-generate the bot token in the Discord Developer Portal and update `DISCORD_BOT_TOKEN` in Replit secrets (and later in your new platform secrets). Re-invite the bot with the `applications.commands` scope.

Phase 1 — Containerise (Day 1–2)

Create a `Dockerfile` at the repo root. The app already has `npm run build` (Vite + tsx bundle → `dist/`) and `npm start` (`node dist/index.cjs`).

```
# ── Build stage ──────────────────────────────────────────────
FROM node:20-alpine AS builder

WORKDIR /app

COPY package*.json ./
```

```
RUN npm ci --ignore-scripts

COPY . .

RUN npm run build

# ── Runtime stage ──────────────────────────────────────────────

FROM node:20-alpine AS runner

WORKDIR /app

ENV NODE_ENV=production

COPY package*.json ./

RUN npm ci --omit=dev --ignore-scripts

COPY --from=builder /app/dist ./dist

EXPOSE 5000

CMD ["node", "dist/index.cjs"]
```

Add a `.dockerignore`:

```
node_modules

.git

uploads/

.local/

*.md

load-tests/

mobile-driver/

mobile-customer/
```

Test locally before deploying:

```
docker build -t ezygas .

docker run --env-file .env.production -p 5000:5000 ezygas
```

Phase 2 — Platform Setup (Day 2–3)

Option A — Google Cloud Run (recommended)

Why: The codebase already has a SIGTERM graceful shutdown (8s drain + DB pool close) that is textbook Cloud Run. It also auto-scales to zero and back up instantly.

```
# One-time setup

gcloud run deploy ezygas \

 --image gcr.io/YOUR_PROJECT/ezygas \

 --platform managed \

 --region me-central1 \          # or europe-west1 (closer to Egypt)

 --port 5000 \

 --min-instances 1 \             # keep 1 warm to avoid cold-start latency

 --max-instances 10 \            # auto-scale up to 10 for burst traffic

 --memory 1Gi \

 --cpu 1 \

 --timeout 60 \

 --set-env-vars NODE_ENV=production \

 --set-secrets "SESSION_SECRET=SESSION_SECRET:latest,DATABASE_URL=..." \
```

```
--allow-unauthenticated
```

Option B — Railway (fastest to ship)

1. Connect GitHub repo to Railway
2. Set `npm run build` as the build command and `npm start` as the start command
3. Add all environment variables in the Railway dashboard
4. Railway auto-deploys on every push to `main`

Option C — Render

Same as Railway. Use `npm run build` + `npm start`, add env vars in dashboard.

Phase 3 — Environment Variables (Day 3)

Migrate all Replit secrets to your new platform. Required:

Variable	Notes
`NODE_ENV`	Set to `production`
`SESSION_SECRET`	Generate a new 64-char random string for prod
`NEON_DATABASE_URL`	Use the Neon **direct** connection URL — the app derives the pooler URL automatically
`NATIONAL_ID_ENCRYPTION_KEY`	Same key as current Replit secret — do NOT rotate during migration
`UPSTASH_REDIS_REST_URL`	**Must be set** — OTP store and rate limiting depend on this

`UPSTASH_REDIS_REST_TOKEN`	Same
`R2_ACCOUNT_ID`	**Must be set** — prevents local-disk fallback
`R2_ACCESS_KEY_ID`	Same
`R2_SECRET_ACCESS_KEY`	Same
`R2_BUCKET_NAME`	Same
`R2_PUBLIC_URL`	Same
`R2_LIFECYCLE_POLICY_CONFIRMED`	Set to `true` after confirming in Cloudflare dashboard
`PAYMOB_API_KEY`	Same
`PAYMOB_HMAC_SECRET`	Same
`PAYMOB_CARD_INTEGRATION_ID`	Same
`PAYMOB_WALLET_INTEGRATION_ID`	Same
`SMSMISR_USERNAME`	Same
`SMSMISR_PASSWORD`	Same
`SMSMISR_SENDER`	Same
`SMTP_PASS`	Same

`VAPID_PRIVATE_KEY`	Same
`VAPID_PUBLIC_KEY`	Same
`VAPID_EMAIL`	Same
`DISCORD_BOT_TOKEN`	New token after Phase 0-E
`SENTRY_DSN`	Same
`IOT_INGEST_KEY`	Same

Phase 4 — Staging Deploy & Load Test (Day 4–5)

Deploy to a staging URL (not `ezy-gas.com` yet) and validate before going live.

4-A Smoke test

```
curl https://staging.ezy-gas.com/api/health

# Expect: 200 OK

curl https://staging.ezy-gas.com/api/geography/governorates

# Expect: JSON array of Egyptian governorates
```

4-B Verify service integrations

Check startup logs for these lines — any `[WARN]` means a credential is missing:

```
[DB] Neon — PgBouncer pooler ✅
```

```
[Cache] Upstash Redis connected ✅

[R2] Cloudflare R2 storage connected ✅

[Sentry] Error tracking enabled ✅

[Migrations] Safety columns verified ✅
```

4-C Run k6 load tests

The repo already has `load-tests/smoke.js` and `load-tests/stress.js`.

```
# Install k6: https://k6.io/docs/get-started/installation/

# Smoke test first (10 VUs, 1 min)

BASE_URL=https://staging.ezy-gas.com k6 run load-tests/smoke.js

# Stress test (ramp to 200 VUs over 10 min)

BASE_URL=https://staging.ezy-gas.com k6 run load-tests/stress.js
```

Pass criteria:

- p95 response time < 2s
- p99 response time < 5s
- Error rate < 1%
- No `[DB] Unexpected pool error` in logs

4-D Test backup restore

```
# Trigger a manual backup

curl -X POST https://staging.ezy-gas.com/api/admin/backup/trigger \

 -H "Authorization: Bearer <admin-token>"

# Restore to a test DB and verify row counts match

# See docs/backup-restore-runbook.md for full procedure
```

4-E Regression test suite

```
BASE_URL=https://staging.ezy-gas.com npm run test:regression
```

Phase 5 — Production Cutover (Day 5–7)

5-A Pre-cutover checklist

- All Phase 0 code changes merged to `main`
- Staging load test passed
- All env vars confirmed in production platform
- R2 lifecycle policy confirmed in Cloudflare dashboard
- Discord bot re-authenticated and online
- Backup restore tested successfully
- Mobile APKs rebuilt via EAS pointing to new production API URL

5-B Deploy production

```
# Tag the release

git tag v1.0.0-pilot

git push origin v1.0.0-pilot
```

Trigger production deploy on your chosen platform.

5-C DNS migration

Update `ezy-gas.com` DNS to point to the new platform's IP/hostname. TTL should be set to 300s (5 min) before the cutover so rollback is fast.

```
# Cloudflare / DNS provider

A   ezy-gas.com   →   <new platform IP>

   TTL: 300
```

5-D Post-cutover smoke test

Repeat the smoke test from 4-A against `ezy-gas.com` (not staging). Verify Sentry is receiving events by intentionally triggering a 404.

5-E Monitor for the first 48 hours

Watch these during the pilot launch:

- Sentry error volume (should be < 0.1% of requests)
- Neon connection pool saturation (stay under 35/40 slots)
- Upstash Redis latency (should be < 50ms)
- Cloud Run / Railway instance count (should stay under 5 for 10k pilot)

Rollback Plan

If production is broken after cutover, revert DNS to Replit:

```
A   ezy-gas.com   →   <old Replit IP>

   TTL: 300
```

DNS propagates within 5 minutes. The Replit instance is kept alive until the pilot is stable (minimum 2 weeks).

Mobile App Update

The Android APKs hardcode (or configure at build time) the API base URL. Before the pilot, rebuild both apps via EAS pointing at the new production URL:

```
# In mobile-customer/ and mobile-driver/

eas build --platform android --profile production
```

Distribute updated APKs to pilot users. The old APKs will continue to work if you keep the old Replit URL alive during the transition.

Summary Timeline

Day	Milestone
0–1	Phase 0 code fixes merged, Dockerfile created
1–2	Staging environment set up on target platform
3	All env vars migrated, staging deployed
4–5	Load tests pass, regression suite green, backup tested
5–6	Mobile APKs rebuilt and distributed
6–7	Production cutover, DNS updated, 48h monitoring window

EzyGas Production Stack Inventory — COMPLETED

Document Owner: CTO Office
Last Updated: 2026-04-02
Environment: Production (ezygas.replit.app)
Purpose: Infrastructure baseline for stress testing
Status: COMPLETE — all TBDs filled, corrections applied

CORRECTIONS TO ORIGINAL DOCUMENT

The following items in the original inventory were marked TBD or were incorrect:

Field	Original	Corrected
Express version	"likely 4.x"	**5.0.1**
Payment gateway	"TBD — Fawry or Paymob"	**None — COD only** (removes entire risk category)
SMS provider	"TBD — Twilio, Nexmo, or local"	**SMS Misr** (Egyptian carrier, `sms4.eg`)

Maps API	"TBD — Google Maps"	**Not integrated** (no geocoding, no routing)
Rate limiting	"❌ No rate limiting"	✅ **Already implemented** (see Section 9)
Health check	"assumed"	✅ **Live** at `/api/health`
Metrics endpoint	"not configured"	✅ **Added** at `/api/metrics`
CDN	"serving from Frankfurt"	**Served from Replit/Cloud Run** (GCP global LB)

---

1. Hosting Infrastructure

Provider

- **Hosting Provider:** Replit Autoscale (Google Cloud Run)
- **Production URL:** `https://ezygas.replit.app`
- **Custom Domain:** `ezy-gas.com` (configured in Replit)
- **Region:** Google-managed (Cloud Run auto-selects; GCP global load balancer)

Compute Resources

- **Instance Type:** Serverless Cloud Run containers
- **vCPUs:** Auto-scaled (0.5–4 vCPU per instance)
- **RAM:** Auto-scaled (512MB–4GB per instance)
- **Storage:** Ephemeral container filesystem (`uploads/` directory — see known issue)
- **Network:** Google Cloud global load balancer, unlimited bandwidth

Auto-Scaling

- **Min instances:** 0 (kept alive via self-ping every 4 minutes)
- **Max instances:** Replit Autoscale default (~100 concurrent instances)
- **Scale trigger:** CPU + concurrent request count (Cloud Run managed)
- **Load balancer:** Google Cloud Load Balancer (automatic)

Keep-Alive

- Self-ping to `/api/health` every 4 minutes via `setInterval` in `server/index.ts`
- Prevents cold starts; keeps at least 1 warm instance at all times

---

2. Web Server Layer

- **Framework:** Express.js **v5.0.1** (latest major — async error handling native)
- **Language:** TypeScript (compiled via esbuild → `dist/index.cjs`)
- **SSL/TLS:** Google-managed (auto-renewed wildcard for `*.replit.app`), TLS 1.3
- **HTTP/2:** Yes (Google Cloud Run default)
- **Gzip:** Yes — `compression` middleware applied to all responses
- **Static assets:** Served from `dist/public/` (Vite build output, hashed filenames)
- **Cache-Control:** `max-age=31536000, immutable` for hashed assets (Vite default)

---

3. Application Server Layer

- **Runtime:** Node.js **v20.20.0** (LTS)
- **Process model:** 1 process per container (horizontal scale via Cloud Run)
- **Build pipeline:**
- `npm run build`
- `  ├── Vite   → dist/public/         (React SPA, gzipped bundles)`
- `  └── esbuild → dist/index.cjs     (~1.9MB CJS bundle)`
- **Run command (production):** `node ./dist/index.cjs`
- **Run command (development):** `tsx server/index.ts`
- **Port:** 5000 (remapped to 80/443 by Cloud Run)

Background services (all in-process)

Service	Interval	Purpose

SLA Engine	60 seconds	Breach detection, auto-ticket, credit
Keep-alive	4 minutes	Prevent idle cold starts
Slack bot	Always-on (Socket Mode)	Operational commands via WebSocket

---

4. Database Layer

- **Engine:** PostgreSQL 15+ (Neon serverless)
- **Provider:** Neon (neon.tech) — serverless, auto-scales, no idle cost
- **Project ID:** `young-bonus-36728518`
- **Region:** `eu-central-1` (Frankfurt, AWS) — ~50ms latency to Cairo
- **Host:** `ep-frosty-band-agb0q22b-pooler.c-2.eu-central-1.aws.neon.tech`
- **DB name:** `neondb`
- **User:** `neondb_owner`
- **Connection mode:** pgBouncer (pooled)
- **SSL:** Required (`sslmode=require`)
- **ORM:** Drizzle ORM v0.39.3

Connection pool (production)

Parameter	Value
Max connections	40
Min connections (warm)	5
Idle timeout	30 seconds
Connection timeout	5 seconds
Query timeout	30 seconds

Stress test implication

At 40 max connections and no cache layer, expect connection pool saturation around **800–1,000 concurrent users** sending DB-touching requests. Adding Redis would multiply effective capacity to 2,000+.

Schema

- **35 tables**, 1,568 lines of schema code (`shared/schema.ts`)
- Full indexes on all foreign keys and status/created_at columns
- Managed via `npm run db:push` (Drizzle Kit)

High availability

- **Failover:** Automatic (Neon manages primary/replica internally)
- **RTO:** < 30 seconds (Neon SLA)
- **Backups:** Continuous PITR (point-in-time recovery), 7-day retention (Free) / 30-day (Pro)
- **RPO:** < 1 minute (WAL archiving to S3)

---

5. Cache Layer

Status: ❌ Not configured

No Redis or Memcached is in use. All requests hit the Neon database directly.

Predicted breaking point without cache: ~800 concurrent users (pool exhaustion)
Predicted capacity with Redis: ~2,000–5,000 concurrent users

Recommended implementation (Upstash Redis — Day 1)

```
npm install ioredis

Secret: REDIS_URL = <Upstash Redis URL>

TTL strategy:

 - User sessions:     3,600 seconds (1 hour)
```

```
- Product catalog:     300 seconds (5 minutes)
- Governorates/areas:  86,400 seconds (24 hours)
- Order status:          60 seconds (1 minute)
```

---

6. CDN

Status: ❌ Not configured

Static assets served directly from Cloud Run origin.

Latency impact: Cairo/Giza users see +50–80ms vs. Cloudflare-cached edge nodes.

Recommended implementation (Cloudflare Free — Day 1)

1. Add `ezy-gas.com` to Cloudflare (free tier)
2. Set DNS to Cloudflare nameservers
3. Proxy traffic through Cloudflare edge
4. Cache rules: hashed JS/CSS assets → 1-year TTL; HTML → no-cache
5. Enable Brotli + minification in Cloudflare dashboard

Expected improvement: 200–500ms faster page loads for Egyptian users

---

7. File Storage

Status: ⚠️ Ephemeral only

User avatar uploads stored in `uploads/avatars/` on the container filesystem. Files are lost on container restart.

Volume of uploads: Low (avatars only — admin/driver profiles)
Risk level: Medium (pilot phase UX impact, not data corruption)

Recommended implementation (Cloudflare R2 — Day 3)

```
npm install @aws-sdk/client-s3

Secrets: R2_ACCOUNT_ID, R2_ACCESS_KEY_ID, R2_SECRET_ACCESS_KEY,
R2_BUCKET_NAME
```

---

8. External APIs & Services

8.1 Payment — CASH ON DELIVERY ONLY

EzyGas uses no payment gateway. All orders are COD.

This eliminates the entire payment gateway risk category from the stress test:

- No Fawry, Paymob, or Stripe integration
- No payment API rate limits to worry about
- No payment gateway timeouts or failures
- Cash collected by driver at delivery, recorded in `cashLedger` table

Payment flow: Consumer places order → Driver collects cash → Driver records in app → reconciled by hub manager

8.2 SMS — SMS Misr

Property	Value
Provider	SMS Misr (sms4.eg — Egyptian carrier)
API URL	`http://sms4.eg/webapi`
Env vars	`SMSMISR_USERNAME`, `SMSMISR_PASSWORD`, `SMSMISR_SENDER`
Encoding	UTF-8 (Arabic supported)

Use cases	Driver OTP login, order status notifications
Rate limit	~100 SMS/min (contact SMS Misr for commercial limits)
Cost	~0.10–0.15 EGP per SMS (standard Egypt market rate)
Delivery SLA	< 5 seconds for domestic Egyptian numbers

Stress test implication: OTP-based mobile auth sends 1 SMS per login. At 3,750 orders/day (pilot peak), expect ~150–500 SMS/day — well within standard limits.

8.3 Email — Gmail / Custom SMTP

Property	Value
Provider	Custom SMTP via `mail.ezy-gas.com`
Host	`mail.ezy-gas.com`
Port	465 (SSL)
From	`admin@ezy-gas.com`
Env var	`SMTP_PASS`
Use cases	Order confirmations, formatted Arabic HTML emails
Rate limit	Depends on hosting — typically 500–1,000/hour

8.4 AI — Anthropic Claude

Property	Value
Model	`claude-opus-4-5`
Max tokens	2,048 per response
Use cases	Slack bot AI answers, code review
Env var	`ANTHROPIC_API_KEY`
Rate limit	Varies by Anthropic tier

Stress test impact: Claude only used for Slack commands — no customer-facing calls. Zero impact on order processing capacity.

8.5 Maps / Geocoding

Status: ❌ Not integrated

No Google Maps, OpenStreetMap, or geocoding API is configured. Address data is stored as plain text + linked to area/city/governorate via the geography tables (`areas`, `cities`, `governorates`).

Delivery distance: Estimated via hub-to-area flat rate table — no real-time geocoding.

8.6 Slack Bot

Property	Value
Framework	@slack/bolt (Socket Mode WebSocket)
Workspace	Mohamed EL Zayat HQ
Bot name	`ezygas_server_admin`

Scopes	`chat:write`, `app_mentions:read`, `commands`, `im:write`
Env vars	`SLACK_BOT_TOKEN`, `SLACK_APP_TOKEN`, `SLACK_SIGNING_SECRET`

8.7 Linktra® IoT Sensors

Property	Value
Broadcast interval	Every 5 seconds
Measurement accuracy	±3%
Low-gas threshold	20% (per cylinder, configurable)
Ingest endpoint	`POST /api/iot/reading`
History endpoint	`GET /api/cylinders/:id/readings`
Frontend polling	Every 10 seconds (consumer portal)

---

9. Security Configuration

Rate Limiting — ✅ ALREADY IMPLEMENTED

```
// General API — 1,000 requests per 15 minutes per IP

app.use("/api", rateLimit({ windowMs: 15 * 60 * 1000, max: 1000 }));

// Auth endpoints — 100 requests per 15 minutes per IP
```

```
app.use("/api/admin/login", authLimiter);

app.use("/api/auth", authLimiter);
```

Note: Limit was raised from 100 to 1,000 for beta (multiple testers on shared IP). Should be tightened for production:

- Recommended: `max: 300` general, `max: 20` auth, `max: 10` order creation

Authentication

Web (Replit Auth):

- OpenID Connect via Replit OAuth
- Sessions in PostgreSQL (`sessions` table, `connect-pg-simple`)
- Session secret: `SESSION_SECRET`

Admin (session-based):

- Login: `POST /api/admin/login` → sets `req.session.adminId`
- Credentials in DB (`admin_accounts` table, bcrypt hashed)

Mobile (Bearer token):

- 90-day tokens stored in `mobile_sessions` table
- User IDs: `mobile_+20XXXXXXXXXX`

SSL/TLS

- Auto-managed by Google Cloud Run (Let's Encrypt / Google CA)
- TLS 1.3, HTTP/2 enabled
- Auto-renewed (no manual action required)

Helmet

- Enabled (`helmet()`) with CSP disabled for Replit compatibility
- HSTS: Cloud Run default

---

10. Monitoring & Observability

Health Check — ✅ LIVE

Endpoint: `GET /api/health` (public, no auth)
Response:

```
{
 "status": "ok",
 "timestamp": "2026-04-02T20:00:00.000Z",
 "uptime": 3600,
 "database": "connected",
 "memory": { "used": "245MB", "total": "512MB" }
}
```

Metrics Endpoint — ✅ ADDED

Endpoint: `GET /api/metrics` (admin auth required)
Data returned:

- System: uptime, Node version, heap/RSS memory %
- Orders: placed / assigned / out_for_delivery / completed / cancelled / last_hour / last_24h
- Cylinders: total / filled / in_transit / delivered / empty / maintenance
- Drivers: total / active / inactive
- Users: total registered
- SLA: breached / at_risk
- Support tickets: open / in_progress / resolved

Slack Bot — ✅ LIVE

Real-time operational visibility via `/ezygas-*` commands (see production-stack.md for full list).

External Monitoring — ❌ Not configured

Recommended (Day 2):

- **UptimeRobot** (free): Monitor `/api/health` every 5 minutes, alert to email if down
- **Sentry** (free tier): Error tracking + performance traces

Logs

- All `console.log/error` output captured in Replit logs
- Structured format: `[Module] message` with timestamps
- Retention: Replit dashboard (7-day rolling)
- Slack: `/ezygas-logs [n]` tails last N lines live

---

11. Performance Baseline (Pre-Stress-Test)

Current traffic: 0 (pre-launch)

Predicted limits (without fixes):

Concurrent Users	Predicted Behavior
0–200	Smooth — DB pool well within limits
200–500	Good — approaching 50% pool usage
500–800	Warning — DB pool near saturation
800–1,000	**Breaking point** — pool exhausted, 503 errors
1,000+	**Failure** — cascading DB timeouts

Predicted limits (with Redis cache + Cloudflare CDN):

Concurrent Users	Predicted Behavior
0–1,500	Smooth — cache absorbs 70–80% of reads

1,500–3,000	Good — within stress test GO criteria
3,000–5,000	Acceptable — Cloud Run auto-scales
5,000+	Cloud Run instance count determines limit

---

12. Known Issues & Gaps

CRITICAL (fix before stress test)

#	Issue	Impact	Fix	ETA
1	No Redis cache	Pool exhaustion at ~800 users	Add Upstash Redis	Day 1
2	No CDN	+50–80ms latency to Egypt	Add Cloudflare Free	Day 1
3	Ephemeral file storage	Avatars lost on restart	Migrate to R2/S3	Day 3
4	Rate limits too generous	DDoS / abuse risk	Tighten to 300/IP/15min	Day 2
5	No external monitoring	Blind to production failures	UptimeRobot + Sentry	Day 2

ALREADY FIXED (corrections to original document)

#	Issue	Status

1	"No rate limiting"	✅ Rate limiting already live
2	"No health check"	✅ `/api/health` already live
3	"No metrics endpoint"	✅ `/api/metrics` added
4	"Payment gateway unknown"	✅ COD only — no gateway
5	"SMS provider unknown"	✅ SMS Misr (documented above)
6	"Maps API unknown"	✅ Not used — geography tables

MEDIUM (fix during Week 1)

#	Issue	Fix
1	No async job processing	SMS/email block request thread — add job queue
2	No structured logging	Add `pino` logger for JSON logs
3	No database read replicas	Enable Neon read replicas for reporting queries
4	No circuit breakers	Add for SMS Misr calls (prevent cascade)
5	No staging environment	Clone Replit project for safe load testing

---

13. Deployment Process

Property	Value
Method	Replit Autoscale (Git push → auto-deploy)
Build time	~2–5 minutes
Deploy time	~10–30 seconds (Cloud Run rolling update)
Downtime	Near-zero (Cloud Run rolls instances)
Rollback	Replit checkpoint → restore (2–5 minutes)
CI/CD	None (direct to production)
Automated tests	None (0% test coverage — technical debt)

---

14. Stress Test Readiness Assessment

Overall: 🟡 YELLOW — 65% ready

Area	Status	Notes
Auto-scaling infra	✅ Green	Cloud Run handles horizontal scale
Database	✅ Green	Neon auto-scales, PITR, failover
SSL/TLS	✅ Green	Auto-managed
Rate limiting	✅ Green	Already implemented

Health check	✅ Green	`/api/health` live
Metrics endpoint	✅ Green	`/api/metrics` added
COD payment model	✅ Green	No payment gateway risk
Cache layer	❌ Red	WILL fail at ~800 concurrent users
CDN	❌ Red	+50ms latency for all Egypt users
File storage	⚠️ Yellow	Uploads lost on restart
External monitoring	⚠️ Yellow	No UptimeRobot / Sentry
Staging environment	⚠️ Yellow	No test environment isolated

Go/No-Go Prediction

Scenario	Prediction
Current state (no fixes)	🔴 NO-GO — breaks at ~800 concurrent
After Redis + CDN (Day 1)	🟢 GO — handles 1,500+ concurrent
Full Week 1 fixes	🟢 CONFIDENT GO — 3,000+ concurrent

---

15. Week 1 Action Plan (Prioritized)

Day 1 (2026-04-03) — Cache + CDN

1. 🔲 Deploy Upstash Redis → add `REDIS_URL` secret → implement caching middleware
2. 🔲 Add Cloudflare CDN in front of `ezy-gas.com`
3. 🔲 Establish performance baseline (k6, 10 users, 5 minutes)

Day 2 (2026-04-04) — Monitoring + Rate Limits

4. 🔲 Add UptimeRobot — monitor `/api/health` every 5 minutes
5. 🔲 Add Sentry — `npm install @sentry/node` + `SENTRY_DSN` secret
6. 🔲 Tighten rate limits (300 general / 20 auth / 10 order-creation)
7. 🔲 Document SMS Misr commercial rate limits (call account manager)

Day 3 (2026-04-05) — File Storage + Test Environment

8. 🔲 Migrate avatar uploads to Cloudflare R2
9. 🔲 Clone Replit project as staging environment for destructive testing

Day 4–5 (2026-04-06–07) — Test Data + Scripts

10. 🔲 Seed staging DB with 100K synthetic users, 10K addresses
11. 🔲 Write k6 scripts for user journeys (order placement, tracking)
12. 🔲 Dry run baseline test on staging

---

16. Infrastructure Team

Role	Person
Chairman / Acting CTO	Dr. Mohamed Fathy EL Zayat
DevOps Lead	TBD (hiring)
Backend Lead	TBD (hiring)
Database Admin	Neon self-service + support

Vendor Support

- **Replit:** https://replit.com/support
- **Neon:** https://neon.tech/docs/introduction/support
- **SMS Misr:** https://sms4.eg (account manager contact required for commercial SLA)
- **Cloudflare:** https://developers.cloudflare.com (self-service)
- **Upstash:** https://upstash.com (self-service Redis)

Document Status: COMPLETE
Next Review: 2026-04-03 EOD (after Day 1 fixes)
Approved by: CTO Office