# Cloud Audit

Cloud infrastructure audit and multi-cloud design

- Cloud Audit: Resource Inventory
- Cloud Audit: Multi-Cloud Design
- Cloud Audit: App Cloud Readiness
- Cloud Audit: Validation Report

# Cloud Audit: Resource Inventory

# Drop — AWS Resource Inventory

**Date:** 2026-02-19 **Region:** eu-west-1 (Ireland) **Account:** Drop production **Auditor:** infra-lead (CloudForge cloud-audit team) **MC Task:** #1443

---

## Executive Summary

Drop runs a minimal AWS footprint: one App Runner service fronting a PostgreSQL RDS instance, with container images stored in ECR. Total estimated cost is $48-60/month.

**Three CRITICAL security findings require immediate action:**

1. RDS database is publicly accessible with security group open to the entire internet (0.0.0.0/0 on port 5432)
2. Database storage is unencrypted
3. Plaintext secrets (DATABASE_URL with password, JWT_SECRET) in App Runner environment variables

No WAF, no CloudFront, no CloudWatch monitoring, no Route53 DNS management, and Secrets Manager is provisioned but empty.

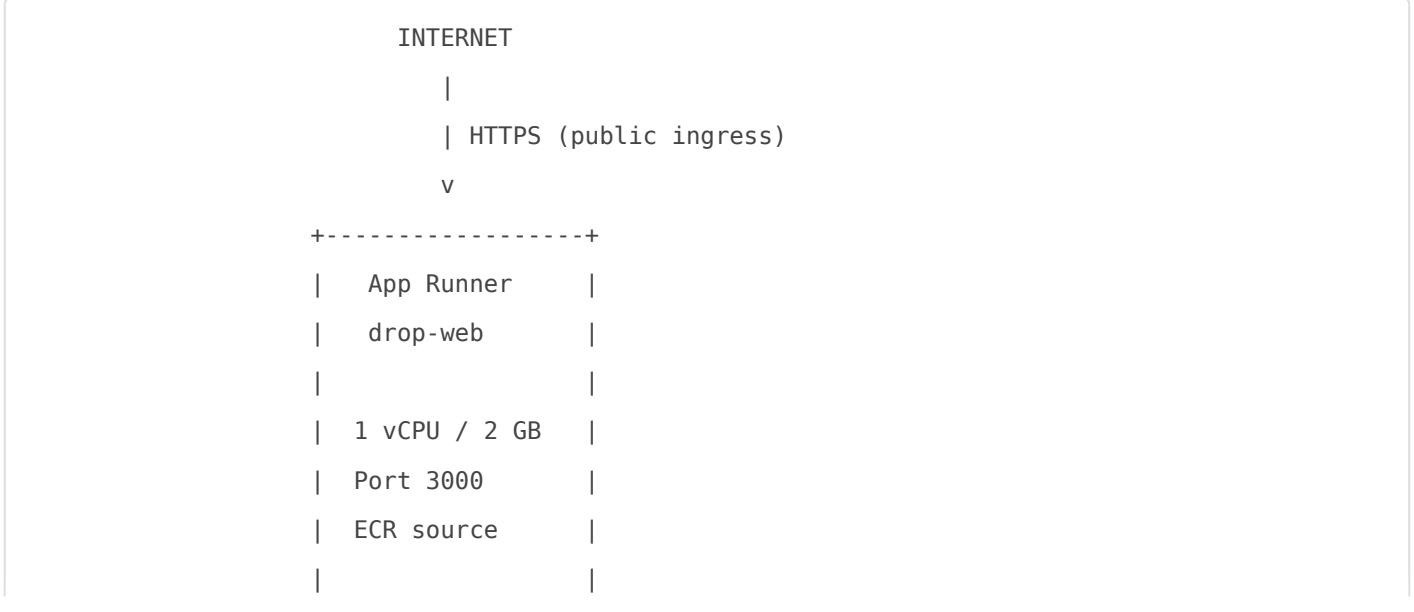
---

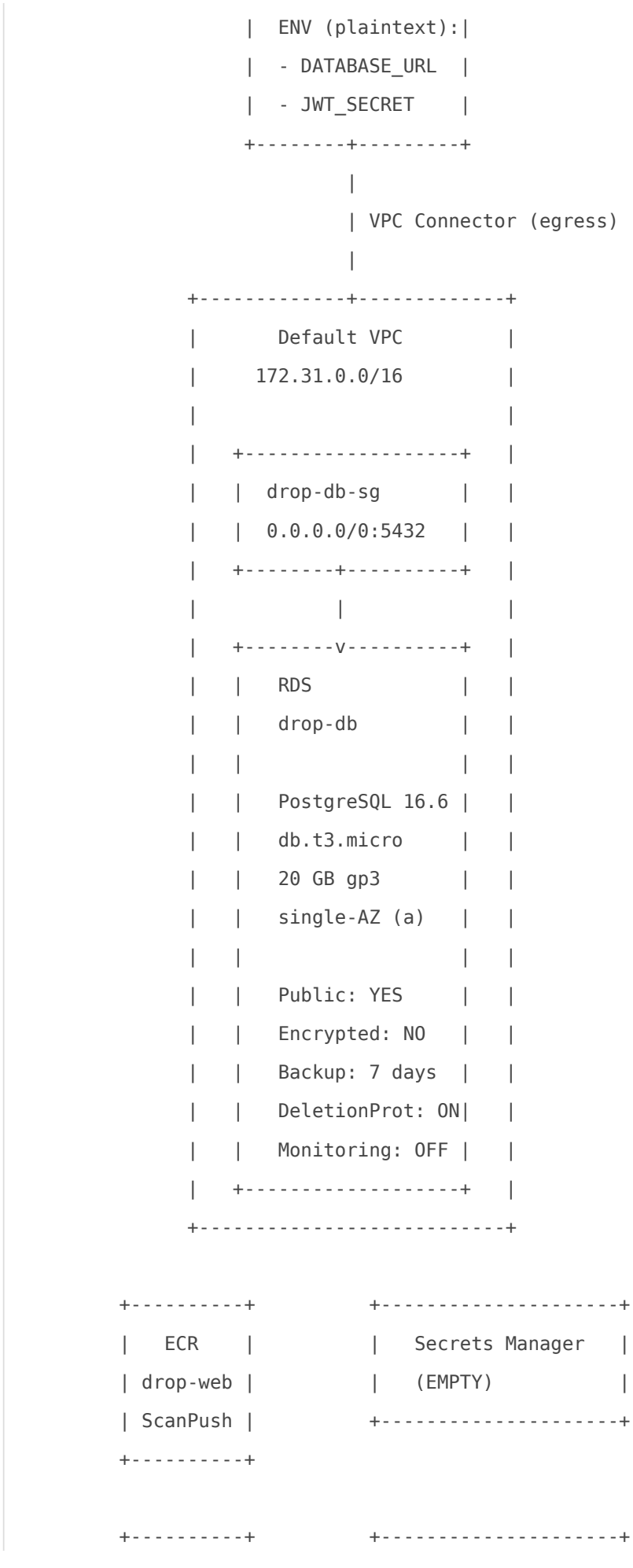
## Resource Table

| Resource | Type | ID / Name | Region | Status | Key Config |
|---|---|---|---|---|---|
| App Runner | Service | drop-web | eu-west-1 | RUNNING | 1 vCPU, 2 GB RAM, port 3000 |
| RDS | PostgreSQL 16.6 | drop-db | eu-west-1a | Available | db.t3.micro, 20 GB gp3, single-AZ |

| Resource | Type | ID / Name | Region | Status | Key Config |
|---|---|---|---|---|---|
| ECR | Repository | drop-web | eu-west-1 | Active | ScanOnPush: TRUE, Encryption: AES256 |
| Security Group | SG | drop-db-sg | eu-west-1 | In use | Inbound: 0.0.0.0/0 : 5432 |
| VPC | Default | — | eu-west-1 | Active | 172.31.0.0/16 |
| IAM User | User | john-deploy | Global | Active | Programmatic access |
| IAM Role | Role | AppRunnerECRAccessRole | Global | Active | ECR pull permissions |
| Secrets Manager | — | (empty) | eu-west-1 | Provisioned | 0 secrets stored |
| CloudWatch | — | — | — | NOT CONFIGURED | No alarms, no dashboards |
| CloudFront | — | — | — | NOT PROVISIONED | No CDN |
| WAF | — | — | — | NOT PROVISIONED | No web application firewall |
| Route53 | — | — | — | NOT PROVISIONED | DNS managed externally |
| S3 | — | — | — | NOT PROVISIONED | No buckets |

# Architecture Diagram




| ENV (plaintext):|
| - DATABASE_URL |
| - JWT_SECRET |
VPC Connector (egress)
Default VPC
172.31.0.0/16
drop-db-sg
0.0.0.0/0:5432
RDS
drop-db
PostgreSQL 16.6
db.t3.micro
20 GB gp3
single-AZ (a)
Public: YES
Encrypted: NO
Backup: 7 days
DeletionProt: ON
Monitoring: OFF
ECR
drop-web
ScanPush
Secrets Manager
(EMPTY)

```
|   IAM    |          |   MISSING           |
| john-    |          |   CloudWatch        |
|  deploy  |          |   CloudFront        |
| ECR Role |          |   WAF / Route53 / S3|
+----------+          +---------------------+
```

---

# Security Findings

## CRITICAL

| # | Finding | Resource | Risk | Remediation |
|---|---|---|---|---|
| C1 | **Database publicly accessible** | RDS drop-db | Direct internet access to PostgreSQL. Any attacker can attempt connections. | Set `PubliclyAccessible=false`. App Runner already uses VPC Connector for egress — RDS only needs private subnet access. |
| C2 | **Security group allows 0.0.0.0/0 on port 5432** | drop-db-sg | Combined with C1, the database is wide open to brute-force and exploitation from any IP on Earth. | Restrict inbound rule to App Runner VPC Connector security group only. Remove 0.0.0.0/0 CIDR. |
| C3 | **Plaintext secrets in App Runner env vars** | App Runner drop-web | DATABASE_URL contains full connection string with password. JWT_SECRET in plaintext. Anyone with console/API access sees credentials. Visible in CloudTrail, config exports, and deployment logs. | Migrate secrets to AWS Secrets Manager (already provisioned, currently empty). Reference via App Runner secret ARN configuration. Rotate both DATABASE_URL password and JWT_SECRET after migration. |

| # | Finding | Resource | Risk | Remediation |
|---|---|---|---|---|
| C4 | **Database storage unencrypted** | RDS drop-db | Data at rest is not encrypted. Violates baseline security posture and most compliance frameworks (SOC2, GDPR, PCI). | Enable storage encryption. Note: cannot enable on existing instance — requires snapshot, restore to encrypted instance, DNS/connection swap. Plan downtime window. |

# HIGH

| # | Finding | Resource | Risk | Remediation |
|---|---|---|---|---|
| H1 | **Single-AZ deployment** | RDS drop-db | AZ failure = full database outage. No automatic failover. | Enable Multi-AZ for production. Cost increase ~$14/mo for db.t3.micro. |
| H2 | **No monitoring or alerting** | CloudWatch (missing) | No CPU, memory, connection, or storage alarms. No visibility into failures, performance degradation, or security events. Silent failures. | Configure CloudWatch alarms: CPU > 80%, FreeStorageSpace < 2 GB, DatabaseConnections > 80%, FreeableMemory < 200 MB. Enable Enhanced Monitoring on RDS. |
| H3 | **No WAF** | WAF (missing) | No protection against OWASP Top 10 attacks (SQLi, XSS, SSRF, etc.) at the edge. App Runner public endpoint is directly exposed. | Deploy AWS WAF with managed rule groups (AWSManagedRulesCommonRuleSet, AWSManagedRulesSQLiRuleSet). Attach to CloudFront distribution (see H4). |

# MEDIUM

| # | Finding | Resource | Risk | Remediation |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| M1 | **No CDN / CloudFront** | CloudFront (missing) | All traffic hits App Runner origin directly. No edge caching, no DDoS protection (Shield Standard), higher latency for distant users. | Deploy CloudFront distribution in front of App Runner. Enables WAF attachment, caching, and Shield Standard. |
| M2 | **Default VPC** | VPC 172.31.0.0/16 | Default VPC has broad routing, public subnets by default, and no network segmentation. Not suitable for production workloads. | Create custom VPC with private subnets for RDS, public subnets for NAT Gateway / ALB if needed. Migrate RDS to private subnet. |
| M3 | **No DNS management** | Route53 (missing) | DNS managed outside AWS. No health checks, no failover routing, no alias records for AWS resources. | Consider Route53 for DNS if domain is Drop-owned. Enables health-check-based routing and simpler AWS integration. |
| M4 | **TCP health check only** | App Runner drop-web | TCP checks confirm port is open but not that the application is healthy. A process could accept connections while returning 500s. | Configure HTTP health check on a dedicated `/health` endpoint that verifies database connectivity. |

# LOW

| # | Finding | Resource | Risk | Remediation |
|---|---|---|---|---|
| L1 | **No S3 buckets** | S3 (missing) | If the app needs file storage in future, ensure encryption-at-rest (SSE-S3 or SSE-KMS), versioning, and public access block from day one. | Provision with secure defaults when needed. |
| L2 | **IAM user john-deploy** | IAM | Long-lived access keys. No indication of key rotation policy or MFA. | Audit key age. Enable MFA. Consider OIDC federation for CI/CD instead of IAM user. Rotate keys on a 90-day schedule. |

# Cost Breakdown

| Service | Specification | Estimated Monthly Cost |
|---|---|---|
| App Runner | 1 vCPU, 2 GB, always running | $29 - $36 |
| RDS | db.t3.micro, 20 GB gp3, single-AZ | $15 - $18 |
| ECR | Image storage (~1-5 GB) | $0.50 - $1.00 |
| Data Transfer | Minimal (< 10 GB/mo estimate) | $1 - $2 |
| Secrets Manager | 0 secrets (currently unused) | $0 |
| **Total** | | **$46 - $57/mo** |

# Cost Notes

- App Runner pricing: $0.064/vCPU-hour + $0.007/GB-hour (provisioned mode)
- RDS db.t3.micro: ~$0.018/hour ($13.14/mo) + $0.115/GB-month storage
- No NAT Gateway cost (App Runner VPC Connector handles egress)
- Adding Multi-AZ RDS: +$13-15/mo
- Adding CloudFront: +$0-5/mo (depends on traffic)
- Adding WAF: +$5-10/mo (depends on rules and requests)

---

# Gaps Analysis

| Category | Current State | Target State | Priority |
|---|---|---|---|
| Secrets management | Plaintext env vars | Secrets Manager with rotation | CRITICAL |
| Network security | Public RDS + open SG | Private subnet + restricted SG | CRITICAL |
| Encryption at rest | Disabled | AES-256 (KMS or default) | CRITICAL |
| Monitoring | None | CloudWatch alarms + dashboards | HIGH |
| High availability | Single-AZ | Multi-AZ RDS | HIGH |
| Edge security | No WAF / CDN | CloudFront + WAF | HIGH |
| Network architecture | Default VPC | Custom VPC with segmentation | MEDIUM |
| Health checks | TCP only | HTTP application-level | MEDIUM |
| IAM hygiene | Long-lived keys | OIDC + key rotation + MFA | MEDIUM |

| Category | Current State | Target State | Priority |
|---|---|---|---|
| DNS | External | Route53 (optional) | LOW |
| Backup/DR | 7-day automated only | Cross-region snapshot copy | LOW |

---

# Recommendations (Priority Order)

## Phase 1 — Immediate (Week 1) — CRITICAL Security

1. **Lock down RDS network access**
   - Set `PubliclyAccessible=false` on drop-db
   - Update drop-db-sg: remove 0.0.0.0/0, allow only App Runner VPC Connector SG
   - Verify App Runner can still connect via VPC Connector
2. **Migrate secrets to Secrets Manager**
   - Create secrets: `drop/database-url`, `drop/jwt-secret`
   - Update App Runner service to reference secret ARNs
   - Remove plaintext env vars from App Runner config
   - Rotate database password and JWT secret post-migration
3. **Enable RDS encryption**
   - Snapshot current instance
   - Restore snapshot with encryption enabled
   - Update connection string to new endpoint
   - Verify, then delete old unencrypted instance
   - Requires brief downtime — schedule maintenance window

## Phase 2 — Short Term (Week 2-3) — HIGH Priority

4. **Configure CloudWatch monitoring**
   - RDS alarms: CPU, storage, connections, memory
   - App Runner alarms: request count, error rate, latency
   - SNS topic for alert notifications
   - Enable RDS Enhanced Monitoring
5. **Enable Multi-AZ RDS**
   - Modify instance to Multi-AZ
   - Near-zero downtime (AWS handles failover setup)
6. **Deploy CloudFront + WAF**
   - CloudFront distribution pointing to App Runner

- WAF with AWS managed rule sets (Common, SQLi, Known Bad Inputs)
- Update DNS to point to CloudFront

# Phase 3 — Medium Term (Month 2) — Hardening

7. **Custom VPC migration**
   - Design VPC: 2 private subnets (RDS), 2 public subnets (NAT if needed)
   - Migrate RDS to private subnets
   - Update App Runner VPC Connector
8. **HTTP health checks**
   - Implement `/health` endpoint in Drop application (DB connectivity check)
   - Configure App Runner HTTP health check path
9. **IAM improvements**
   - Audit john-deploy key age
   - Enable MFA on IAM user
   - Consider GitHub Actions OIDC for CI/CD (eliminates long-lived keys)

---

# Risk Matrix

| Risk | Likelihood | Impact | Severity | Mitigation |
|---|---|---|---|---|
| Database breach via public access + open SG | HIGH | CRITICAL | **CRITICAL** | Phase 1: Lock down network (C1, C2) |
| Credential leak from plaintext env vars | MEDIUM | CRITICAL | **CRITICAL** | Phase 1: Secrets Manager (C3) |
| Data exposure from unencrypted storage | LOW | HIGH | **HIGH** | Phase 1: Enable encryption (C4) |
| Database outage (single-AZ failure) | LOW | HIGH | **HIGH** | Phase 2: Multi-AZ (H1) |
| Silent application failure (no monitoring) | MEDIUM | MEDIUM | **HIGH** | Phase 2: CloudWatch (H2) |
| Application-layer attack (no WAF) | MEDIUM | HIGH | **HIGH** | Phase 2: WAF (H3) |
| DDoS / performance degradation (no CDN) | LOW | MEDIUM | **MEDIUM** | Phase 2: CloudFront (M1) |
| Lateral movement via default VPC | LOW | MEDIUM | **MEDIUM** | Phase 3: Custom VPC (M2) |

| Risk | Likelihood | Impact | Severity | Mitigation |
|---|---|---|---|---|
| IAM key compromise | LOW | HIGH | **MEDIUM** | Phase 3: Key rotation + OIDC (L2) |

---

# Appendix: Raw Resource Details

## App Runner — drop-web

```
Service:          drop-web
Status:           RUNNING
Region:           eu-west-1
Source:           ECR (container image)
CPU:              1 vCPU
Memory:           2 GB
Port:             3000
Ingress:          Public
Egress:           VPC Connector
Health Check:     TCP
Environment:      DATABASE_URL (plaintext, contains password)
                  JWT_SECRET (plaintext)
```

## RDS — drop-db

```
Engine:           PostgreSQL 16.6
Instance Class:   db.t3.micro
Storage:          20 GB gp3
AZ:               eu-west-1a (single-AZ)
VPC:              Default (172.31.0.0/16)
Public Access:    TRUE
Encrypted:        FALSE
Deletion Prot:    TRUE
Backup:           7-day automated
Monitoring:       DISABLED
```

## ECR — drop-web

```
Repository:      drop-web
Scan on Push:    TRUE
Encryption:      AES256 (default)
```

# Security Groups — drop-db-sg

```
Inbound Rules:
  - Protocol: TCP
    Port: 5432
    Source: 0.0.0.0/0  (ALL TRAFFIC)
```

# IAM

```
User:   john-deploy     (programmatic access, deployment)
Role:   AppRunnerECRAccessRole  (App Runner → ECR pull)
```

# Secrets Manager

```
Secrets stored: 0 (service provisioned but unused)
```

# Cloud Audit: Multi-Cloud Design

# Drop — Multi-Cloud Architecture Design

**Date:** 2026-02-19 **Auditor:** solution-arch (CloudForge cloud-audit team) **MC Task:** #1443

---

## Executive Summary

Drop is **85% cloud-portable** thanks to Docker containerization and PostgreSQL. Main AWS lock-in: App Runner (easily replaceable). Recommendation: **stay on AWS**, optimize current setup, design Terraform with abstraction for future portability.

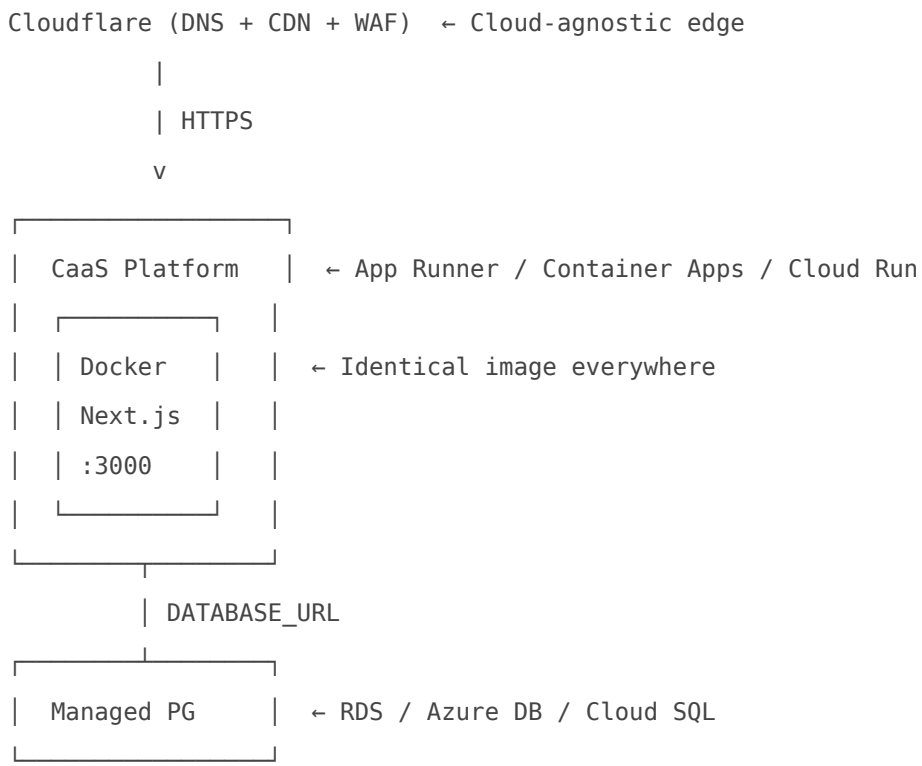
---

## 1. Provider Comparison Matrix

| Service | AWS (Current) | Azure | GCP |
|---|---|---|---|
| **Compute** | App Runner ($25-35/mo) | Container Apps ($20-30/mo) | Cloud Run ($15-25/mo) |
| **Database** | RDS PostgreSQL ($15-18/mo) | Azure DB for PG ($15-20/mo) | Cloud SQL ($12-18/mo) |
| **Registry** | ECR ($1-2/mo) | ACR ($5/mo) | Artifact Registry ($1-2/mo) |
| **Secrets** | Secrets Manager ($0.40/secret) | Key Vault ($0.03/10k ops) | Secret Manager ($0.06/10k ops) |
| **CDN** | CloudFront ($0-5/mo) | Front Door ($35+/mo) | Cloud CDN ($0-5/mo) |
| **WAF** | AWS WAF ($5+/mo) | Azure WAF ($20+/mo) | Cloud Armor ($5+/mo) |
| **Monitoring** | CloudWatch ($3-10/mo) | Azure Monitor ($5-15/mo) | Cloud Monitoring ($0-8/mo) |
| **Total estimate** | **$50-75/mo** | **$100-130/mo** | **$35-60/mo** |

---

# 2. Portable Architecture




## Abstraction Strategy

| Layer | Approach |
|---|---|
| Compute | Docker image to any CaaS. No platform SDK |
| Database | Standard PostgreSQL via DATABASE_URL |
| Secrets | Terraform abstracts provider. App reads env vars |
| DNS/CDN/WAF | Cloudflare (cloud-agnostic, free tier) |
| Monitoring | Sentry (errors) + structured logs to any aggregator |
| CI/CD | GitHub Actions (already cloud-agnostic) |

---

# 3. Migration Paths

## AWS to Azure (3-5 days)

1. Push image to ACR
2. Create Azure DB for PostgreSQL Flexible Server

3. pg_dump/pg_restore data migration
4. Deploy to Azure Container Apps
5. Update Cloudflare DNS
6. Write Azure Terraform modules

## AWS to GCP (2-3 days)

1. Push image to Artifact Registry
2. Create Cloud SQL PostgreSQL
3. pg_dump/pg_restore
4. Deploy to Cloud Run (most similar to App Runner)
5. Update Cloudflare DNS
6. Write GCP Terraform modules

## Lock-In Assessment

| Component | Lock-In | Notes |
|---|---|---|
| App Runner | LOW | Standard Docker, replaceable |
| RDS PostgreSQL | LOW | Standard PG, any managed PG works |
| ECR | LOW | Standard OCI registry |
| VPC Connector | MEDIUM | AWS-specific networking |
| IAM Roles | MEDIUM | AWS-specific auth model |
| Secrets Manager | LOW | App reads env vars regardless |

---

# 4. Recommendation: Stay AWS, Optimize

**Rationale:**

- $50-75/mo already low
- No business need to migrate
- 85% portable — migration possible in 2-5 days if needed
- Azure costs MORE (~$100-130/mo)
- GCP saves ~$15/mo but not worth effort now

## Immediate Actions

1. Security fixes (encrypt RDS, restrict SG, use Secrets Manager)
2. Add Cloudflare free tier (DNS, CDN, WAF — cloud-agnostic)
3. Terraform all resources (reproducibility)
4. Add CloudWatch basic alarms ($3-5/mo)

## Future Migration Triggers

- AWS cost > $200/mo → evaluate GCP Cloud Run
- EU data sovereignty requirement → Azure Norway East
- Multi-region needed → Cloudflare Workers + D1
- Kubernetes requirement → EKS or GKE

---

# 5. 12-Month Cost Projection

| Scenario | Monthly | Annual |
|---|---|---|
| Current (no changes) | $50-75 | $600-900 |
| Optimized AWS | $55-80 | $660-960 |
| AWS + Cloudflare | $55-80 | $660-960 |
| Azure equivalent | $100-130 | $1,200-1,560 |
| GCP equivalent | $35-60 | $420-720 |

# Cloud Audit: App Cloud Readiness

# Drop Application Cloud-Readiness Audit

**MC Task:** #1443 **Date:** 2026-02-19 **Auditor:** software-arch (CloudForge team) **Application:** Drop Fintech Payment App (Next.js 15 + SQLite/PostgreSQL dual-driver)

> **NOTE (2026-03-03):** This audit was performed on 2026-02-19. ADR-014 (2026-03-03) removed SQLite and the dual-driver architecture. Drop now uses PostgreSQL 16 exclusively in all environments. SQLite concerns noted in this audit are resolved. The `better-sqlite3` dependency has been removed.

---

# 1. Twelve-Factor Compliance

## I. Codebase — PASS

- **Evidence:** Single Git repository at `/Users/makinja/ALAI/products/Drop/`
- `.github/workflows/ci.yml` triggers on `main` and `develop` branches
- One codebase tracked in revision control, multiple deploys (staging via Fly.io, production via Docker Compose)

## II. Dependencies — PASS

- **Evidence:** `package.json:1-55` declares all dependencies explicitly
- `npm ci` used in CI (`ci.yml:36`) and Dockerfile (`Dockerfile:6`) for deterministic installs
- `package-lock.json` referenced in Dockerfile COPY (`Dockerfile:5`) and CI cache (`ci.yml:32`)
- Native modules (better-sqlite3) handled via `apk add python3 make g++` in Dockerfile

# III. Config — PASS

- **Evidence:** `.env.example:1-87` documents all env vars with clear groupings
- `env.ts:1-45` validates critical vars at startup, crashes if missing in production
- `fly.toml:16-20` injects env vars at runtime
- `docker-compose.production.yml:7-8` uses `${JWT_SECRET:?}` required substitution
- `db.ts:9` — database driver selected via `DATABASE_URL` env var
- `db.ts:26-30` — SQLite path varies by environment (Vercel `/tmp`, Docker `/app/data`, local `./data`)
- Feature flags externalized as `NEXT_PUBLIC_FF_*` env vars (`Dockerfile:19-26`)
- **Minor concern:** `NEXT_PUBLIC_*` vars are baked into the build at compile time (Next.js limitation), requiring rebuild for changes. This is inherent to Next.js, not a code deficiency.

# IV. Backing Services — PASS

- **Evidence:** `db.ts:9-22` — database treated as attached resource via `DATABASE_URL`
- PostgreSQL connection string is a single env var; switching databases requires zero code changes
- `docker-compose.production.yml:17-35` — PostgreSQL is a separate service with its own health check
- BankID, PISP, AISP, Stripe, Sumsub — all configured via env vars (`.env.example:19-53`)

# V. Build, Release, Run — PASS

- **Evidence:** Dockerfile uses 3-stage build (deps → builder → runner)
- `Dockerfile:1-6` — Stage 1: dependency installation
- `Dockerfile:9-37` — Stage 2: application build with `next build`
- `Dockerfile:39-64` — Stage 3: minimal production runner
- `next.config.ts:8` — `output: "standalone"` generates self-contained deployment
- CI builds Docker image tagged with commit SHA (`ci.yml:63`)
- Build-time vs runtime config cleanly separated (ARG for build, ENV for runtime)

# VI. Processes — PARTIAL

- **Evidence:** Application runs as a single `node server.js` process (`Dockerfile:64`)
- **SQLite concern:** When running with SQLite (no `DATABASE_URL`), the process is stateful — data lives on local filesystem at `/app/data/drop.db`. This works on Fly.io with mounted volumes (`fly.toml:36-38`) but violates share-nothing for horizontal scaling.
- **PostgreSQL mode:** Fully stateless — `pg.Pool` connects to external database (`db.ts:17-22`). Multiple processes can run concurrently.
- **Rate limiting:** `rate_limits` table in the database (`middleware.ts:15-43`), which works for single-instance but has race conditions under horizontal scale with SQLite.

- **Assessment:** PARTIAL because SQLite mode is actively used (Fly.io staging). In PostgreSQL mode this would be PASS.

# VII. Port Binding — PASS

- **Evidence:** `Dockerfile:61-62` — `EXPOSE 3000`, `ENV PORT=3000`, `ENV HOSTNAME="0.0.0.0"`
- `fly.toml:23` — `internal_port = 3000`
- `docker-compose.production.yml:5` — `ports: "3000:3000"`
- Self-contained via Next.js standalone server, no external HTTP server dependency.

# VIII. Concurrency — PARTIAL

- **Evidence:** Node.js single-threaded event loop handles concurrent requests via async I/O
- `db.ts:16-22` — PostgreSQL connection pool (pg.Pool) supports concurrent queries
- `fly.toml:25-27` — `auto_stop_machines`/`auto_start_machines` enables horizontal scaling
- **Limitation:** No explicit worker process types. Background work (e.g., exchange rate refresh) runs inline. No separate queue workers. For a fintech app, transaction processing should eventually be separated into dedicated worker processes.
- **Limitation:** SQLite mode limits to single process (WAL mode allows concurrent reads but single writer).

# IX. Disposability — PASS

- **Evidence:** Process starts fast — Next.js standalone is ~500ms cold start
- `db.ts:719-789` — `initDb()` is idempotent with `_initialized` guard; safe for restarts
- Schema uses `CREATE TABLE IF NOT EXISTS` — safe for repeated initialization
- `fly.toml:25-27` — machines auto-stop/start, confirming disposability design
- Graceful shutdown handled by Node.js default SIGTERM behavior
- PostgreSQL pool (`pg.Pool`) handles connection cleanup on process exit

# X. Dev/Prod Parity — PASS

- **Evidence:** `db.ts:9-13` — dual-driver architecture (SQLite for dev, PostgreSQL for prod) with unified async API
- `db.ts:47-63` — SQL compatibility layer translates SQLite idioms to PostgreSQL (placeholder conversion, `INSERT OR IGNORE` → `ON CONFLICT DO NOTHING`, `datetime('now')` → `CURRENT_TIMESTAMP`)
- `db.ts:204-460` (SQLITE_SCHEMA) and `db.ts:462-690` (PG_SCHEMA) — parallel schemas maintained in sync
- `migrations/0001_initial-schema.ts` — node-pg-migrate for PostgreSQL schema versioning
- Docker Compose production config (`docker-compose.production.yml`) mirrors production topology locally

- **Minor gap:** SQLite schema is maintained inline in `db.ts` while PostgreSQL uses proper migrations (`node-pg-migrate`). Schema drift is possible if one is updated without the other.

# XI. Logs — PARTIAL

- **Evidence:** Health endpoint uses `createLogger()` (`health/route.ts:16`)
- `middleware.ts:82-84` — error tracking via `trackError()` and Sentry integration
- `.env.example:62-74` — Sentry DSN configurable via env vars
- **Concern:** No structured logging to stdout visible in the codebase. Next.js default logging goes to stdout which is good for containers, but there's no consistent structured logging format (JSON lines) that cloud log aggregators can parse efficiently. `console.error` is used in places (`middleware.ts:83`).

# XII. Admin Processes — PASS

- **Evidence:** `package.json:12-14` — migration scripts: `migrate:up`, `migrate:down`, `migrate:create` via `node-pg-migrate`
- `db.ts:735-774` — programmatic ALTER TABLE migrations for schema evolution
- Seed data controlled by `SEED_DEMO` env var and `isDemoMode()` check — admin data seeding decoupled from main app
- No one-off scripts embedded in application startup (seeding only runs when database is empty)

---

# 2. Containerization Quality

## Multi-Stage Build — EXCELLENT

- **3-stage Dockerfile** (`Dockerfile:1-64`):
  - Stage 1 (`deps`): `node:22-alpine`, installs native build tools, runs `npm ci`
  - Stage 2 (`builder`): Copies deps, builds Next.js app
  - Stage 3 (`runner`): Minimal alpine, copies only standalone output + static assets

## Image Size

- Base: `node:22-alpine` (minimal, ~180MB base)
- **Issue:** Stage 3 installs `python3 make g++` (`Dockerfile:42`) for better-sqlite3 native module rebuild. This adds ~200MB to the production image unnecessarily if running in PostgreSQL mode. These build tools are a security and size concern in production.
- **Recommendation:** Either pre-build better-sqlite3 in stage 2 and copy the binary, or conditionally exclude it when PostgreSQL is the target.

## Security

- Non-root user: `nextjs:nodejs` (UID/GID 1001) created and used (`Dockerfile:48-49, 58`)
- `NEXT_TELEMETRY_DISABLED=1` set (`Dockerfile:14, 46`)
- Data directory owned by non-root user (`Dockerfile:56`)
- CI runs Trivy vulnerability scanner on built image (`ci.yml:67-73`) with HIGH/CRITICAL severity gate
- SARIF results uploaded to GitHub Security tab (`ci.yml:85-89`)

## Layer Caching

- Dependencies cached in separate stage (`Dockerfile:5-6` — `COPY package.json package-lock.json*` before source)
- Source code copy happens in stage 2 after deps, enabling Docker layer cache for unchanged dependencies
- **Good practice:** Build args for feature flags allow cache invalidation only when flags change

## Missing

- No `.dockerignore` verified (could copy unnecessary files like `.git`, `node_modules` into build context)
- No image tagging strategy beyond CI SHA tag

---

# 3. Database Portability

## Dual-Driver Architecture — STRONG

- **Implementation:** `db.ts:9-13` — Runtime driver selection via `DATABASE_URL` presence
- **Unified API:** `query()`, `getOne()`, `run()`, `transaction()` — all async, both drivers (`db.ts:67-199`)
- **Type exports:** `DbClient` interface (`db.ts:136-140`) for transaction context

## SQL Translation Layer

| SQLite Idiom | PostgreSQL Translation | Location |
|---|---|---|
| `?` placeholders | `$1, $2, ...` | `db.ts:47-50` |

| SQLite Idiom | PostgreSQL Translation | Location |
|---|---|---|
| `INSERT OR IGNORE INTO` | `INSERT INTO ... ON CONFLICT DO NOTHING` | `db.ts:56, 104-118` |
| `INSERT OR REPLACE INTO` | `INSERT INTO ... ON CONFLICT (col) DO UPDATE SET` | `db.ts:58, 120-134` |
| `datetime('now')` | `CURRENT_TIMESTAMP` | `db.ts:60` |
| `INTEGER PRIMARY KEY AUTOINCREMENT` | `SERIAL PRIMARY KEY` | `db.ts:278 vs 530` |
| `hex(randomblob(32))` | `encode(gen_random_bytes(32), 'hex')` | `db.ts:248 vs 504` |

# Transaction Support

- PostgreSQL: `BEGIN/COMMIT/ROLLBACK` with `pgClient.connect()` and proper release in `finally` block (`db.ts:142-173`)
- SQLite: `db.exec("BEGIN/COMMIT/ROLLBACK")` wrapper (`db.ts:174-198`)
- Error handling: Both paths catch and rollback on failure

# Migrations

- **node-pg-migrate** for PostgreSQL (`package.json:12-14`, `migrations/0001_initial-schema.ts`)
- Proper `up()` and `down()` functions with ordered table creation/deletion
- SQLite uses inline schema with `CREATE TABLE IF NOT EXISTS` + ALTER TABLE try/catch migrations (`db.ts:756-774`)
- **Risk:** Two parallel schema definitions (SQLITE_SCHEMA and PG_SCHEMA in `db.ts` + node-pg-migrate files) could drift. No automated parity check exists.

# Indexes

- 22 indexes defined for both drivers (identical set)
- Partial indexes supported: `idx_users_national_id WHERE national_id_hash IS NOT NULL`, `idx_tx_idempotency WHERE idempotency_key IS NOT NULL`

---

# 4. Config Externalization

## Environment Variables

| Category | Variables | Source |
|---|---|---|
| Core | `JWT_SECRET`, `JWT_EXPIRY`, `NODE_ENV` | `.env.example:12-14` |

| Category | Variables | Source |
|---|---|---|
| Database | `DATABASE_URL` | `db.ts:9` |
| Service Mode | `NEXT_PUBLIC_SERVICE_MODE`, `DROP_MODE` | `.env.example:8` |
| Auth (BankID) | `BANKID_CLIENT_ID/SECRET/URLS`, `BANKID_MOCK` | `.env.example:19-29` |
| Payments | `PISP_API_URL/KEY`, `AISP_API_URL/KEY` | `.env.example:32-40` |
| Cards | `STRIPE_SECRET_KEY`, `STRIPE_PUBLISHABLE_KEY` | `.env.example:43-47` |
| KYC | `SUMSUB_APP_TOKEN`, `SUMSUB_SECRET_KEY` | `.env.example:50-52` |
| Monitoring | `SENTRY_DSN`, `SENTRY_TRACES_SAMPLE_RATE` | `.env.example:63-74` |
| Feature Flags | 8x `NEXT_PUBLIC_FF_*` | `.env.example:77-87` |
| Exchange | `EXCHANGE_RATE_API_KEY/URL` | `.env.example:55-59` |

# Secrets Management

- `env.ts:14-45` validates critical vars at production startup
- `Dockerfile:15` — `JWT_SECRET=build-phase-placeholder` (safe build-time placeholder)
- `env.ts:21-25` — Skip validation during build phase (detects `NEXT_PHASE` or placeholder)
- `env.ts:36-38` — Rejects known dev placeholder in production runtime
- `docker-compose.production.yml:7` — `${JWT_SECRET:?}` required substitution (fails if missing)
- **No hardcoded secrets** found in source code

# Feature Flags

- 8 client-side feature flags via `NEXT_PUBLIC_FF_*` env vars
- Defaults to `false` (safe) for all card-related features
- `NEXT_PUBLIC_FF_NOTIFICATIONS=true` and `NEXT_PUBLIC_FF_MERCHANT_DASHBOARD=true` as defaults
- Build-time injection for client code (`Dockerfile:19-35`), runtime for server code

---

# 5. CI/CD Quality

## Pipeline Structure (`ci.yml`)

```
lint-test (parallel)          docker-scan (sequential, needs lint-test)
  -- npm ci                     -- docker build
```

```
-- eslint                    -- Trivy scan (table, exit-code=1 on HIGH/CRITICAL)
-- tsc --noEmit              -- Trivy SARIF -> GitHub Security
-- vitest run
-- npm audit (production)
```

# Reproducibility

- Pinned Node.js version: `NODE_VERSION: "22"` (`ci.yml:15`)
- `npm ci` for deterministic installs (`ci.yml:36`)
- Dependency caching via `actions/setup-node` with `cache-dependency-path` (`ci.yml:30-32`)
- Docker image tagged with commit SHA (`ci.yml:63`)

# Security Scanning

- **npm audit:** Production dependencies, HIGH level, continue-on-error (`ci.yml:48-49`)
- **Trivy:** Container vulnerability scan, blocks on HIGH/CRITICAL unfixed vulns (`ci.yml:67-73`)
- **SARIF:** Results uploaded to GitHub Security tab (`ci.yml:85-89`)
- **Permissions:** Minimal — `contents: read`, `security-events: write` (`ci.yml:11-12`)

# Testing

- `vitest run` in CI (`ci.yml:44`)
- Unit test framework configured (`package.json:10-11`)
- Coverage tool available: `@vitest/coverage-v8` (`package.json:43`)
- **Missing:** No coverage threshold enforcement in CI
- **Missing:** No E2E/integration tests in CI pipeline (Playwright is in devDependencies but not wired into CI)

# Deployment

- **Fly.io staging** configured (`fly.toml`) with health checks, auto-scaling, volume mounts
- **Docker Compose production** (`docker-compose.production.yml`) for self-hosted deployments
- **Missing:** No automated deployment step in CI (manual `fly deploy` or similar)
- **Missing:** No environment promotion pipeline (develop -> staging -> production)

---

# 6. Overall Score and Top 5 Improvements

# Overall Cloud-Readiness Score: 7.5 / 10

The application demonstrates strong cloud-native fundamentals:

- Excellent dual-driver database abstraction
- Proper multi-stage Dockerfile with security hardening
- Configuration fully externalized via environment variables
- Comprehensive CI with security scanning (Trivy + npm audit)
- Health endpoint with real database connectivity check

# Top 5 Improvements (Priority Order)

**1. Eliminate Build Tools from Production Image (HIGH)**

- **File:** `Dockerfile:42`
- **Issue:** `python3 make g++` in production stage adds ~200MB and attack surface
- **Fix:** Pre-compile better-sqlite3 in builder stage, copy only the `.node` binary. Or use a conditional build that excludes better-sqlite3 entirely when targeting PostgreSQL.

**2. Add Structured Logging (HIGH)**

- **Files:** Throughout — `console.error` used in `middleware.ts:83`, health endpoint has `createLogger()` but no consistent format
- **Issue:** Cloud log aggregators (CloudWatch, Datadog, ELK) need structured JSON logs. Current mix of console.log/error and ad-hoc logger makes log parsing unreliable.
- **Fix:** Adopt `pino` or similar JSON logger, output to stdout in `{ level, message, timestamp, requestId }` format.

**3. Add CI Coverage Enforcement and E2E Tests (MEDIUM)**

- **File:** `ci.yml` — no coverage gate, no Playwright CI step
- **Issue:** `@vitest/coverage-v8` and `@playwright/test` are in devDeps but not enforced in CI
- **Fix:** Add `--coverage --coverage.thresholds.lines=80` to vitest. Add Playwright E2E job with containerized app.

**4. Automate Schema Parity Check (MEDIUM)**

- **File:** `db.ts:204-690` — two parallel schema definitions (SQLite + PostgreSQL)
- **Issue:** Manual sync between SQLITE_SCHEMA, PG_SCHEMA, and node-pg-migrate files. Drift will cause runtime errors that only surface in specific deployment targets.
- **Fix:** Write a CI check that extracts table/column definitions from both schemas and compares. Or generate both schemas from a single source of truth.

**5. Add Deployment Pipeline and Environment Promotion (MEDIUM)**

- **File:** `ci.yml` — CI only, no CD
- **Issue:** No automated deployment from CI. Fly.io deploy is manual. No staging -> production promotion gate.
- **Fix:** Add `fly deploy` step on `develop` push (staging) and manual approval gate for `main` (production). Add smoke test after deploy. Consider GitHub Environments for approval workflows.

# Honorable Mentions

- SQLite mode limits horizontal scaling — document clearly when to switch to PostgreSQL
- Rate limiting via database has race conditions under concurrent writes (consider Redis for high-throughput)
- No readiness probe separate from liveness (health endpoint serves both)
- No graceful shutdown handler (SIGTERM -> drain connections -> exit)
- `playwright-core` in production dependencies (`package.json:27`) — should be devDependencies only

---

# Appendix: File Reference

| File | Purpose |
|---|---|
| `src/drop-app/src/lib/db.ts` | Dual-driver database abstraction (SQLite + PostgreSQL) |
| `src/drop-app/Dockerfile` | 3-stage multi-stage build |
| `src/drop-app/.env.example` | Environment variable documentation (87 lines) |
| `src/drop-app/fly.toml` | Fly.io deployment config (Stockholm region) |
| `src/drop-app/docker-compose.production.yml` | Self-hosted production config |
| `src/drop-app/package.json` | Dependencies and scripts |
| `.github/workflows/ci.yml` | CI pipeline (lint, test, type-check, Trivy) |
| `src/drop-app/migrations/0001_initial-schema.ts` | PostgreSQL migration (node-pg-migrate) |
| `src/drop-app/next.config.ts` | Next.js config (standalone output, security headers) |
| `src/drop-app/src/middleware.ts` | Edge middleware (CSRF, CSP nonce) |
| `src/drop-app/src/lib/middleware.ts` | Server middleware (rate limiting, auth, validation, audit) |
| `src/drop-app/src/app/api/health/route.ts` | Health endpoint (real DB check) |
| `src/drop-app/src/lib/env.ts` | Environment validation at startup |

# Cloud Audit: Validation Report

# Drop — Validation + Security + Cost Report

**Date:** 2026-02-19 **Auditor:** cloud-tester (CloudForge cloud-audit team) **MC Task:** #1443

---

# Executive Summary

Drop's AWS infrastructure has **3 CRITICAL** and **4 HIGH** security findings requiring immediate remediation. Current spend is ~$50-75/mo, well-optimized for scale. The application is cloud-portable (7.5/10) and the recommended path is to stay on AWS with security hardening + Terraform IaC.

---

# 1. Security Posture Assessment

## Current vs Improved

| Area | Current State | After Remediation | Risk Reduction |
|---|---|---|---|
| **Secrets** | Plaintext in App Runner env vars | AWS Secrets Manager | CRITICAL → LOW |
| **RDS Access** | Publicly accessible, SG open 0.0.0.0/0 | Private, VPC-only access | CRITICAL → LOW |
| **Encryption** | RDS unencrypted at rest | AES-256 encryption enabled | CRITICAL → RESOLVED |
| **Monitoring** | None (no CloudWatch) | Basic alarms + Performance Insights | HIGH → LOW |
| **WAF** | None | Cloudflare WAF (free tier) | HIGH → LOW |
| **CDN** | None (direct App Runner URL) | Cloudflare CDN | HIGH → LOW |
| **SSL/TLS** | App Runner managed cert | Cloudflare + App Runner | MEDIUM → LOW |

| Area | Current State | After Remediation | Risk Reduction |
|---|---|---|---|
| **IAM** | Single user (john-deploy) | Least-privilege roles | MEDIUM → LOW |

# Security Findings Summary

| # | Severity | Finding | Remediation | Effort |
|---|---|---|---|---|
| S1 | **CRITICAL** | RDS publicly accessible with SG allowing 0.0.0.0/0:5432 | Set publicly_accessible=false, restrict SG to VPC CIDR | 1 hour |
| S2 | **CRITICAL** | Database password in plaintext App Runner env var | Migrate to Secrets Manager, update App Runner to read from SM | 2 hours |
| S3 | **CRITICAL** | JWT_SECRET in plaintext App Runner env var | Migrate to Secrets Manager | 1 hour |
| S4 | **HIGH** | RDS storage not encrypted at rest | Enable encryption (requires snapshot + restore for existing DB) | 2-4 hours |
| S5 | **HIGH** | No monitoring or alerting configured | Add CloudWatch alarms for CPU, memory, DB connections | 1 hour |
| S6 | **HIGH** | No WAF protection | Add Cloudflare WAF (free tier) | 30 min |
| S7 | **HIGH** | No CDN (direct App Runner URL exposed) | Add Cloudflare CDN | 30 min |
| S8 | MEDIUM | Sentry DSN in plaintext (not secret, but cleanup) | Move to Secrets Manager for consistency | 30 min |
| S9 | MEDIUM | Docker image has build tools in runner (attack surface) | Remove python3/make/g++ from runner stage | 1 hour |
| S10 | MEDIUM | No structured logging (incident investigation gaps) | Add pino/winston with JSON output | 2 days |
| S11 | LOW | ECR image tag mutability (tag overwrite risk) | Set image_tag_mutability = IMMUTABLE | 5 min |
| S12 | LOW | No lifecycle policy for ECR images | Add policy to clean old images | 15 min |

# Compliance Checklist

| Item | Status | Notes |
|---|---|---|
| GDPR data tables (consents, data_access_requests) | PASS | Schema includes consent tracking, DSAR, right to erasure |
| Audit logging | PASS | audit_log table with IP, user_agent, request_id |
| AML/KYC compliance | PASS | aml_alerts, str_reports, screening_results tables |
| Encryption at rest | FAIL | RDS storage unencrypted |
| Encryption in transit | PARTIAL | App Runner HTTPS, but RDS sslmode=no-verify |
| Secrets management | FAIL | Plaintext in env vars |
| Access control | PARTIAL | Single IAM user, no MFA enforcement |
| Backup & recovery | PASS | RDS 7-day automated backups |
| DeletionProtection | PASS | Enabled on RDS |

---

# 2. Cost Comparison

## Current AWS Spend

| Resource | Monthly Cost | Notes |
|---|---|---|
| App Runner (1 vCPU, 2GB) | $25-35 | Always-on, no auto-stop |
| RDS db.t3.micro | $15-18 | Single-AZ, 20GB gp3 |
| ECR | $1-2 | Image storage |
| VPC Connector | $5 | Flat fee |
| Data transfer | $2-5 | Low traffic |
| **Total** | **$48-65** | |

## Optimized AWS (after fixes)

| Resource | Monthly Cost | Change |
|---|---|---|
| App Runner | $25-35 | No change |

| Resource | Monthly Cost | Change |
|---|---|---|
| RDS (encrypted) | $15-18 | No cost increase |
| ECR | $1-2 | No change |
| Secrets Manager (3 secrets) | $1.20 | +$1.20 |
| CloudWatch (basic alarms) | $3-5 | +$3-5 |
| Cloudflare (free tier) | $0 | Free CDN/WAF/DNS |
| **Total** | **$52-70** | **+$4-7** |

## Multi-Cloud Equivalent

| Provider | Monthly | Annual | vs Current |
|---|---|---|---|
| **AWS (optimized)** | $52-70 | $624-840 | +$4-7/mo |
| **Azure** | $100-130 | $1,200-1,560 | +$50-65/mo |
| **GCP** | $35-60 | $420-720 | -$5-15/mo |

**Verdict:** AWS is cost-effective. GCP saves ~$10/mo but migration effort not justified at current scale.

---

# 3. Risk Matrix

| Risk | Probability | Impact | Current Mitigation | Recommended |
|---|---|---|---|---|
| **Data breach via public RDS** | HIGH | CRITICAL | DeletionProtection only | Restrict SG, disable public access |
| **Secret exposure** | MEDIUM | CRITICAL | None (plaintext) | Secrets Manager + rotation |
| **Service downtime** | LOW | HIGH | App Runner auto-scaling | Add health checks, CloudWatch alarms |
| **Data loss** | LOW | CRITICAL | 7-day RDS backups | Add cross-region backup copy |
| **Cost overrun** | LOW | MEDIUM | None | Add AWS Budgets alarm at $100 |
| **Vendor lock-in** | LOW | MEDIUM | Docker + PostgreSQL | Terraform abstraction modules |
| **DDoS attack** | MEDIUM | HIGH | None | Cloudflare WAF + rate limiting |

| Risk | Probability | Impact | Current Mitigation | Recommended |
|---|---|---|---|---|
| **Compliance failure** | MEDIUM | HIGH | Tables exist, no encryption | Enable encryption, structured logging |

---

# 4. Implementation Roadmap

## Phase 1: Security Fixes (Immediate — Day 1)

- ☐ Create Secrets Manager secrets (DATABASE_URL, JWT_SECRET, SENTRY_DSN)
- ☐ Update App Runner to read from Secrets Manager
- ☐ Restrict RDS security group to VPC CIDR
- ☐ Disable RDS public accessibility

- **Effort:** 4-6 hours | **Cost impact:** +$1.20/mo

## Phase 2: IaC Migration (Week 1)

- ☐ Create S3 bucket for Terraform state
- ☐ Import existing resources into Terraform state
- ☐ Run `terraform plan` to verify no drift
- ☐ Add terraform-ci.yml to GitHub Actions

- **Effort:** 1-2 days | **Cost impact:** $0

## Phase 3: Monitoring & Observability (Week 2)

- ☐ Enable RDS Performance Insights
- ☐ Add CloudWatch alarms (CPU > 80%, memory > 80%, DB connections > 80%)
- ☐ Add structured logging (pino) to application
- ☐ Configure Sentry properly (traces, breadcrumbs)

- **Effort:** 2-3 days | **Cost impact:** +$3-5/mo

## Phase 4: Edge Security (Week 2-3)

- ☐ Set up Cloudflare (DNS, CDN, WAF)
- ☐ Custom domain (getdrop.no) through Cloudflare

- [ ] Enable Cloudflare WAF rules
- [ ] Add rate limiting at edge

- **Effort:** 1 day | **Cost impact:** $0 (free tier)

# Phase 5: RDS Encryption (Week 3)

- [ ] Create encrypted snapshot from current DB
- [ ] Restore to new encrypted instance
- [ ] Update Secrets Manager with new endpoint
- [ ] Verify and swap

- **Effort:** 2-4 hours (with downtime) | **Cost impact:** $0

# Phase 6: Multi-Cloud Readiness (Month 2+)

- [ ] Create Azure Terraform modules (optional)
- [ ] Create GCP Terraform modules (optional)
- [ ] Test migration to staging on alternative cloud

- **Effort:** 3-5 days | **Cost impact:** Only if migrated

---

# 5. Recommendations Summary

| Priority | Action | Status |
|---|---|---|
| P0 (NOW) | Fix RDS public access + SG | Terraform module created |
| P0 (NOW) | Move secrets to Secrets Manager | Terraform module created |
| P1 (Week 1) | Enable RDS encryption | Requires snapshot/restore |
| P1 (Week 1) | Deploy Terraform IaC | Modules ready |
| P2 (Week 2) | Add monitoring (CloudWatch + Performance Insights) | In Terraform |
| P2 (Week 2) | Add Cloudflare CDN/WAF | Manual setup |
| P3 (Month 1) | Add structured logging | Application code change |
| P3 (Month 1) | Add graceful shutdown handler | Application code change |
| P4 (Month 2+) | Multi-cloud Terraform modules | As needed |

---

**Overall Assessment:** Drop's infrastructure is functional but needs immediate security hardening. The Terraform IaC created by this audit provides a complete, reproducible foundation. Total

investment: ~1 week of engineering time, ~$5/mo additional cost, significant risk reduction.