# Deployment Architecture

# Deployment Architecture

> AWS deployment topology, Cloudflare edge layer, Docker multi-stage build, CI/CD pipeline, environment strategy, auto-scaling, health checks, and rollback procedures for the Drop fintech platform.

---

## Deployment Topology

> **Note:** AWS App Runner is the PLANNED production deployment target. Current deployment uses Docker Compose only (`docker-compose.yml` and `docker-compose.production.yml`). No CI/CD pipeline, ECR, or GitHub Actions are configured yet.

```
graph TB
    subgraph Internet
        User[End Users]
        Mobile[Mobile App]
    end

    subgraph Cloudflare["Cloudflare Edge"]
        DNS[DNS<br/>getdrop.no]
        CDN[CDN<br/>Static assets cache]
        WAF[WAF Rules<br/>Rate limiting, bot protection,<br/>geo-blocking, OWASP rules]
        DDoS[DDoS Protection<br/>L3/L4/L7 mitigation]
    end

    subgraph AWS["AWS eu-north-1 (Stockholm)"]
        subgraph AppRunner["AWS App Runner (PLANNED)"]
```

```
            WebContainer[drop-web<br/>Next.js 15 standalone<br/>Node.js 22 Alpine<br/>Port
3000]
            APIContainer[drop-api<br/>Hono v4<br/>Node.js 22 Alpine<br/>Port 3001]
        end

        subgraph DataLayer["Data Layer"]
            RDS[(RDS PostgreSQL 16<br/>db.t3.micro → db.r6g.large<br/>Multi-AZ
failover<br/>Automated backups)]
        end

        subgraph Supporting["Supporting Services"]
            ECR[ECR<br/>Container Registry<br/>Image scanning enabled]
            SM[Secrets
Manager<br/>JWT_SECRET<br/>BANKID_CLIENT_SECRET<br/>DATABASE_URL<br/>SENTRY_DSN]
            CW[CloudWatch<br/>Logs + Metrics<br/>Alarm triggers]
        end
    end

    User -->|HTTPS| DNS
    Mobile -->|HTTPS| DNS
    DNS --> CDN
    CDN --> WAF
    WAF --> DDoS
    DDoS -->|Origin pull| WebContainer
    DDoS -->|Origin pull| APIContainer
    WebContainer --> RDS
    APIContainer --> RDS
    AppRunner --> ECR
    AppRunner --> SM
    AppRunner --> CW
```

---

# CI/CD Pipeline

```
flowchart LR
    subgraph Trigger
        Push[git push]
        PR[Pull Request]
```

```
    end

    subgraph Build["Build Stage"]
        Checkout[Checkout code]
        Deps[npm ci]
        TypeCheck[tsc --noEmit]
        Lint[eslint]
        Test[vitest run]
    end

    subgraph Package["Package Stage"]
        DockerBuild[Docker multi-stage build]
        ImageScan[ECR image scan]
        PushECR[Push to ECR]
    end

    subgraph DeployStaging["Deploy: Staging"]
        DeployStagingEnv[Deploy to App Runner<br/>staging service]
        SmokeTest[Smoke test<br/>GET /v1/health]
        E2ETest[E2E test suite]
    end

    subgraph DeployProd["Deploy: Production"]
        Approval[Manual approval gate]
        BlueGreen[Blue/green swap<br/>App Runner traffic shift]
        HealthVerify[Health check verification<br/>3 consecutive passes]
        Rollback{Healthy?}
    end

    Push --> Checkout
    PR --> Checkout
    Checkout --> Deps --> TypeCheck --> Lint --> Test
    Test --> DockerBuild --> ImageScan --> PushECR
    PushECR --> DeployStagingEnv --> SmokeTest --> E2ETest
    E2ETest --> Approval --> BlueGreen --> HealthVerify --> Rollback
    Rollback -->|Yes| Done[Production Live]
    Rollback -->|No| RollbackAction[Revert to previous revision]
```

# Pipeline Stages Detail

| Stage | Tool | Timeout | Failure Action |
|---|---|---|---|
| Checkout | `actions/checkout@v4` | 1m | Fail pipeline |
| Install deps | `npm ci` | 5m | Fail pipeline |
| TypeScript check | `tsc --noEmit` | 3m | Fail pipeline |
| Lint | `eslint .` | 2m | Fail pipeline |
| Unit tests | `vitest run` | 5m | Fail pipeline |
| Docker build | Multi-stage (4 stages: deps, test, builder, runner) | 10m | Fail pipeline |
| Image scan | ECR vulnerability scan | 5m | Warn on HIGH, block on CRITICAL |
| Push to ECR | `docker push` | 3m | Fail pipeline |
| Deploy staging | App Runner update | 10m | Fail pipeline |
| Smoke test | `curl /v1/health` | 1m | Rollback staging |
| Manual approval | GitHub environment protection | 24h | Pipeline expires |
| Production deploy | App Runner traffic shift | 10m | Auto-rollback |
| Health verification | 3x `GET /v1/health` at 10s intervals | 1m | Auto-rollback |

---

# Docker Multi-Stage Build

Source: `src/drop-app/Dockerfile`

```
┌──────────────────────────────────────────────┐
│ Stage 1: deps (node:22-alpine)               │
│                                              │
│ • Install python3, make, g++ (native deps)   │
│ • COPY package*.json                         │
│ • npm ci (production + dev deps)             │
│ • Output: /app/node_modules                  │
├──────────────────────────────────────────────┤
│ Stage 2: test (node:22-alpine)               │
│                                              │
│ • COPY node_modules from deps                │
│ • COPY source code                           │
│ • Run vitest + coverage checks               │
```

```
│ • Mandatory test gate — blocks build on     │
│   failure                                   │
├─────────────────────────────────────────────┤
│ Stage 3: builder (node:22-alpine)           │
│                                             │
│ • COPY node_modules from deps               │
│ • COPY source code                          │
│ • npm run build (Next.js standalone output) │
│ • Output: .next/standalone, .next/static    │
├─────────────────────────────────────────────┤
│ Stage 4: runner (node:22-alpine)            │
│                                             │
│ • Non-root user: nextjs (UID 1001)          │
│ • Install python3, make, g++ (native deps)  │
│ • COPY public/ from builder                 │
│ • COPY .next/standalone from builder        │
│ • COPY .next/static from builder            │
│ • Data dir: /app/data (owned by nextjs)     │
│ • No source code                            │
│ • CMD: node server.js                       │
└─────────────────────────────────────────────┘
```

**Security features in runner stage:**

- Non-root user `nextjs` (UID 1001, GID `nodejs` 1001)
- **Note:** Runner stage currently includes `python3`, `make`, `g++` (installed via `apk add` for native dependency rebuilds). These should be removed in a future optimization.
- No source code — only compiled standalone output
- Data directory `/app/data` owned by `nextjs:nodejs`

---

# Environment Configuration

| Variable | Dev | Staging | Production | Source |
|---|---|---|---|---|
| `NODE_ENV` | `development` | `production` | `production` | Dockerfile ENV |
| `JWT_SECRET` | Dev fallback (static string `'dev-secret-change-in-production'`) | Secrets Manager | Secrets Manager | `auth.ts:8` |

| Variable | Dev | Staging | Production | Source |
|---|---|---|---|---|
| `DATABASE_URL` | Not set (SQLite) | RDS connection string | RDS connection string | Secrets Manager |
| `BANKID_CLIENT_ID` | Not set | BankID test env | BankID prod env | Secrets Manager |
| `BANKID_CLIENT_SECRET` | Not set | BankID test env | BankID prod env | Secrets Manager |
| `BANKID_MOCK` | `true` | `false` | `false` | App Runner env |
| `BANKID_CALLBACK_URL` | `http://localhost:3000/api/auth/bankid/callback` | `https://staging.getdrop.no/...` | `https://getdrop.no/...` | App Runner env |
| `NEXT_PUBLIC_SERVICE_MODE` | `demo` | `mock` or `live` | `live` | Build-time env |
| `SEED_DEMO` | implicit (non-prod) | `true` | Not set | App Runner env |
| `SENTRY_DSN` | Not set | Sentry staging project | Sentry prod project | Secrets Manager |
| `APP_URL` | `http://localhost:3000` | `https://staging.getdrop.no` | `https://getdrop.no` | App Runner env |
| `PORT` | `3000` | `3000` | `3000` | App Runner default |

# Environment Strategy

| Environment | Purpose | Database | BankID | Data |
|---|---|---|---|---|
| **Development** | Local development, `docker compose up` | SQLite at `./data/drop.db` | Mock (`BANKID_MOCK=true`) | Demo seed data |
| **Staging** | Pre-release validation, QA, E2E tests | RDS PostgreSQL (separate instance) | BankID test environment | Demo seed data (`SEED_DEMO=true`) |
| **Production** | Live service | RDS PostgreSQL (Multi-AZ, automated backups) | BankID production | Real user data only |

---

# Scaling Configuration

## App Runner Auto-Scaling

| Parameter | Web Container | API Container |
|---|---|---|
| Min instances | 1 | 1 |
| Max instances | 5 | 10 |

| Parameter | Web Container | API Container |
|---|---|---|
| Concurrency target | 50 req/instance | 100 req/instance |
| Scale-up cooldown | 30s | 30s |
| Scale-down cooldown | 300s | 300s |
| CPU | 1 vCPU | 1 vCPU |
| Memory | 2 GB | 2 GB |

# Scaling Triggers

| Metric | Threshold | Action |
|---|---|---|
| Concurrent requests per instance | > 80% of target | Scale up |
| Concurrent requests per instance | < 25% of target for 5m | Scale down |
| Response time p95 | > 500ms for 3m | Scale up + CloudWatch alarm |
| Error rate (5xx) | > 5% for 2m | CloudWatch alarm, no auto-scale |
| CPU utilization | > 80% for 3m | Scale up |

# RDS PostgreSQL Scaling

| Parameter | Staging | Production |
|---|---|---|
| Instance class | `db.t3.micro` | `db.t3.medium` (initial) → `db.r6g.large` |
| Storage | 20 GB gp3 | 100 GB gp3, auto-scaling to 500 GB |
| Multi-AZ | No | Yes |
| Read replicas | 0 | 0 (add when needed) |
| Backup retention | 7 days | 30 days |
| Maintenance window | Sunday 03:00 UTC | Sunday 03:00 UTC |

---

# Health Check Endpoints

## API Health Check

**Endpoint:** `GET /v1/health` (Hono API) **Source:** `routes/health.ts`

```
// Success (200)
{
  "status": "ok",
  "version": "0.1.0",
  "uptime": 3600,
  "db": "connected",
  "dbLatencyMs": 1,
  "timestamp": "2026-02-21T12:00:00.000Z"
}

// Failure (503)
{
  "status": "error",
  "db": "disconnected",
  "timestamp": "2026-02-21T12:00:00.000Z"
}
```

**Health check performs:** `SELECT 1 as ok` query to verify database connectivity and measure latency.

# Health Check Configuration

| Component | Interval | Timeout | Retries | Grace Period |
|---|---|---|---|---|
| App Runner (web) | 10s | 5s | 3 | 30s |
| App Runner (API) | 10s | 5s | 3 | 30s |
| Docker Compose (dev) | 30s | 10s | 3 | 10s |
| Cloudflare origin health | 60s | 10s | 2 | N/A |

---

# Blue/Green Deployment (Aspirational)

> **Note:** App Runner does NOT have built-in blue/green deployment (see ADR-012). The following describes an aspirational traffic-shifting strategy that would need custom implementation. App Runner performs rolling updates by default.

```
1. New revision deployed alongside current (blue)
2. New revision (green) starts and passes health checks
3. Traffic gradually shifted: 0% → 10% → 50% → 100%
4. If health checks pass for 60s at 100% → old revision drained
5. If health checks fail → immediate rollback to blue
```

# Deployment Checklist

1. All CI checks pass (TypeScript, lint, tests)
2. Docker image built and scanned (no CRITICAL vulnerabilities)
3. Image pushed to ECR
4. Staging deployment succeeds
5. Smoke tests pass (`GET /v1/health` returns 200)
6. Manual approval (production deployments only)
7. Production deployment with health verification
8. Post-deployment monitoring (15 minutes)

---

# Rollback Procedures

## Automatic Rollback

App Runner automatically rolls back if:

- New revision fails health checks within grace period
- Health check failure rate exceeds threshold during traffic shift
- Container crashes on startup (exit code != 0)

## Manual Rollback

```
# List recent revisions
aws apprunner list-operations --service-arn $SERVICE_ARN

# Rollback to previous revision
aws apprunner update-service \
  --service-arn $SERVICE_ARN \
  --source-configuration '{"ImageRepository":{"ImageIdentifier":"<previous-ecr-image>"}}'

# Verify rollback
```

```
curl https://getdrop.no/v1/health
```

# Database Rollback

For database schema changes, migrations are forward-only. In case of issues:

1. **SQLite (dev/staging):** Restore from backup (see DEPLOYMENT.md backup section)
2. **PostgreSQL (prod):** RDS point-in-time recovery to any second within retention window (30 days)

```
# RDS point-in-time restore
aws rds restore-db-instance-to-point-in-time \
  --source-db-instance-identifier drop-prod \
  --target-db-instance-identifier drop-prod-restored \
  --restore-time "2026-02-21T11:00:00Z"
```

---

# Cloudflare Configuration

| Feature | Configuration | Purpose |
|---|---|---|
| DNS | `getdrop.no` → App Runner CNAME (proxied) | Domain routing |
| SSL/TLS | Full (strict) | End-to-end encryption |
| CDN | Cache static assets (`/_next/static/*`, `/public/*`) | Performance |
| WAF | OWASP Core Rule Set, rate limiting rules | Security |
| DDoS | L3/L4/L7 auto-mitigation | Availability |
| Bot management | Challenge mode for suspicious traffic | Security |
| Geo-blocking | Allow: NO, SE, DK, FI (Scandinavia) + test regions | Compliance |
| Page rules | `/*` → SSL always, HSTS | Security |

# Cloudflare WAF Rules

| Rule | Action | Purpose |
|---|---|---|
| OWASP Core Rule Set | Block | SQL injection, XSS, path traversal |

| Rule | Action | Purpose |
|---|---|---|
| Rate limit: `/v1/auth/*` | Challenge at 20 req/10s | Auth endpoint abuse prevention |
| Rate limit: `/v1/transactions/*` | Block at 30 req/10s | Transaction abuse prevention |
| Country block: Sanctioned countries | Block | OFAC/UN sanctions compliance |
| Bot score < 30 | Challenge | Bot traffic mitigation |

---

# Cross-References

- **Container diagram:** container-diagram.md — C4 Level 2 container architecture
- **Deployment guide:** DEPLOYMENT.md — Docker compose, backup/restore procedures
- **Security architecture:** SECURITY-ARCHITECTURE.md — Security headers, CSRF, rate limiting
- **Feature flags:** FEATURE-FLAGS.md — Environment variable driven feature gating
- **Database schema:** DATABASE-SCHEMA.md — All 19 tables

---

Revision #7
Created 2026-02-21 05:58:48 UTC by John
Updated 2026-05-23 10:51:37 UTC by John