ADR-012: AWS App Runner Deploy

ADR-012: AWS App Runner for Deployment

Status: Accepted **Date:** 2026-02-21 **Deciders:** John (AI Director), Alem (CEO) **Category:** Infrastructure

---

Context

Drop needs a deployment target for its backend services (Next.js BFF and Hono mobile API). The deployment platform must support Docker containers, auto-scaling, HTTPS termination, and be cost-effective at low initial traffic with the ability to scale.

Deployment options considered:

Platform	Container Support	Auto-scaling	Min Cost	Operational Overhead	Cold Start
AWS App Runner	Yes (ECR/source)	Automatic	~$5/mo (min instances)	Very low	Warm (min instance)
AWS ECS/Fargate	Yes (ECR)	Manual config (target tracking)	~$10/mo (Fargate)	Medium (task defs, services, ALB)	Warm
AWS Lambda	Yes (container image)	Automatic (per-request)	~$0 (free tier)	Low	Cold start problem
Vercel	No (serverless functions)	Automatic	Free tier	Very low	Cold start for API
Railway	Yes (Dockerfile)	Automatic	~$5/mo	Very low	Warm
Fly.io	Yes (Dockerfile)	Automatic	~$5/mo	Low	Warm

Key factors:

1. **WebSocket/long connections:** App Runner supports them; Lambda does not (29s timeout)
2. **PostgreSQL connectivity:** App Runner runs in VPC, can connect to RDS; Lambda requires NAT gateway ($32/mo)
3. **Operational simplicity:** App Runner is "push container, get HTTPS endpoint" -- no load balancer, target group, or service mesh to configure
4. **Cost at scale:** App Runner pricing is straightforward (vCPU-hour + memory-hour); ECS/Fargate pricing is similar but with more configuration
5. **AWS ecosystem:** PostgreSQL on RDS, secrets in Secrets Manager, logs in CloudWatch -- all in same account

Vercel was used for the landing page (static) and is excellent for Next.js, but its serverless function model is not ideal for the Hono API or long-running database connections.

Decision

Use AWS App Runner for backend deployment. Keep Vercel for the landing page (static site).

```
graph TB
    subgraph edge["Edge Layer"]
        cf["Cloudflare<br/>DNS + CDN + WAF + DDoS"]
    end

    subgraph aws["AWS (eu-north-1)"]
        subgraph apprunner["App Runner"]
            nextjs["Next.js BFF<br/>Web app + API routes<br/>(1-10 instances)"]
            hono["Hono API<br/>Mobile REST API<br/>(1-10 instances)"]
        end

        subgraph data["Data Layer"]
            rds["RDS PostgreSQL<br/>(production DB)"]
            secrets["Secrets Manager<br/>(JWT_SECRET, BANKID creds)"]
        end

        subgraph monitoring["Monitoring"]
            cw["CloudWatch<br/>(logs, metrics)"]
        end
    end
```

```
    subgraph vercel["Vercel"]
        landing["Landing Page<br/>getdrop.no<br/>(static)"]
    end

    cf --> nextjs
    cf --> hono
    cf --> landing
    nextjs --> rds
    hono --> rds
    nextjs --> secrets
    hono --> secrets
    nextjs --> cw
    hono --> cw

    classDef edge_style fill:#FFF3E0,stroke:#E65100
    classDef aws_style fill:#E3F2FD,stroke:#1565C0
    classDef vercel_style fill:#F3E5F5,stroke:#6A1B9A

    class cf edge_style
    class nextjs,hono,rds,secrets,cw aws_style
    class landing vercel_style
```

App Runner Configuration

Setting	Value	Rationale
Region	`eu-north-1` (Stockholm)	Closest AWS region to Norway; GDPR data residency
Source	ECR (Docker image)	Pushed by GitHub Actions CI/CD
CPU	1 vCPU	Sufficient for current load
Memory	2 GB	Room for Node.js heap + DB connections
Min instances	1	Eliminates cold start; ~$5/mo baseline
Max instances	10	Auto-scales based on concurrent requests
Port	3000 (Next.js), 3001 (Hono)	Default Node.js ports
Health check	`GET /api/health` (also available at `/v1/health`)	Returns DB connectivity status

Setting	Value	Rationale
Auto deploy	Yes (on ECR push)	CI/CD pushes new image, App Runner deploys

Deployment Pipeline

```
graph LR
    push["git push"] --> gha["GitHub Actions"]
    gha --> build["Docker build<br/>+ TypeScript check<br/>+ Lint + Test"]
    build --> ecr["Push to ECR"]
    ecr --> apprunner["App Runner<br/>auto-deploy"]
    apprunner --> health["Health check<br/>GET /api/health"]
    health -->|"Healthy"| live["Live traffic"]
    health -->|"Unhealthy"| rollback["Auto-rollback<br/>to previous version"]
```

Consequences

Positive

- Minimal operational overhead: no load balancers, target groups, or service meshes to manage
- Automatic HTTPS with AWS-managed TLS certificates
- Auto-scaling based on concurrent requests (0 config beyond min/max instances)
- VPC connectivity to RDS PostgreSQL without NAT gateway
- Automatic rollback on failed health checks
- CloudWatch integration for logs and metrics out of the box
- Cost-effective: ~$5/mo baseline with 1 min instance, scales linearly

Negative

- Less configurability than ECS/Fargate (no custom networking, task placement, or sidecar containers)
- Limited to HTTP/HTTPS workloads (no TCP/UDP services)
- Newer AWS service with fewer community resources and examples
- No built-in blue/green deployment (App Runner does rolling updates). **Note:** `deployment-architecture.md` describes blue/green as aspirational — it would need custom implementation.
- Vendor lock-in to AWS (container is portable, but App Runner config is not)

Risks

- **App Runner regional availability:** Service may not be available in all regions. Mitigation: `eu-north-1` (Stockholm) is supported.
- **Scaling latency:** New instances take 30-60 seconds to provision. Mitigation: maintain 1 min instance for baseline traffic; pre-scale before expected traffic events.
- **Cost at scale:** App Runner pricing can exceed ECS/Fargate for high-throughput workloads. Mitigation: evaluate migration to ECS/Fargate if monthly compute exceeds $200.

References

- Deployment Architecture -- Full deployment topology
- System Context (C4 Level 1) -- Infrastructure components
- ADR-005: Monolith First -- Single deployment model
- ADR-008: Hono API Framework -- Mobile API deployment
- AWS App Runner documentation

Revision #5
Created 2026-02-21 05:59:02 UTC by John
Updated 2026-05-23 10:56:56 UTC by John