ARCHITECTURE.md — Pokémon Pricer (GPT-5 Pro Revis

System Architecture • Signals & Backtesting Integration • Build-Ready Blueprint

Generated: 2025-08-25 07:09 UTC

This PDF was generated from the canonical ARCHITECTURE.md for internal reference.

ARCHITECTURE.md — Pokémon Pricer (GPT-5 Pro Revision)

> **Purpose**
> This document is the canonical, implementation-grade system architecture for Pokémon Pricer. It aligns the **SRS**, **PRD**, and the new **Signals Methodology** + **Backtesting Engine** specs into a single blueprint engineers can build from and SRE/ML teams can operate. It covers stack decisions, data models, pipelines, ML ops, security, observability, and cost controls.

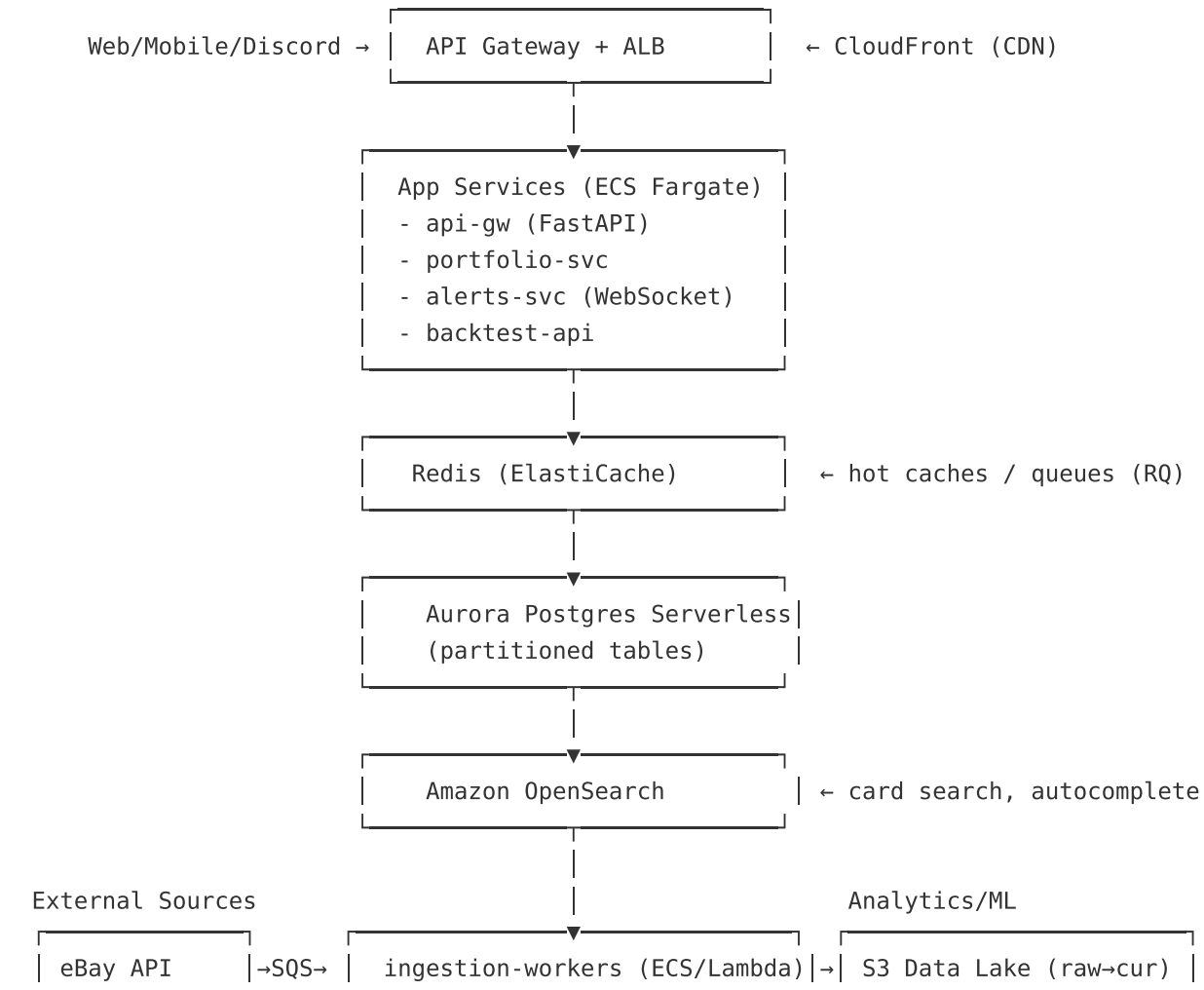
---

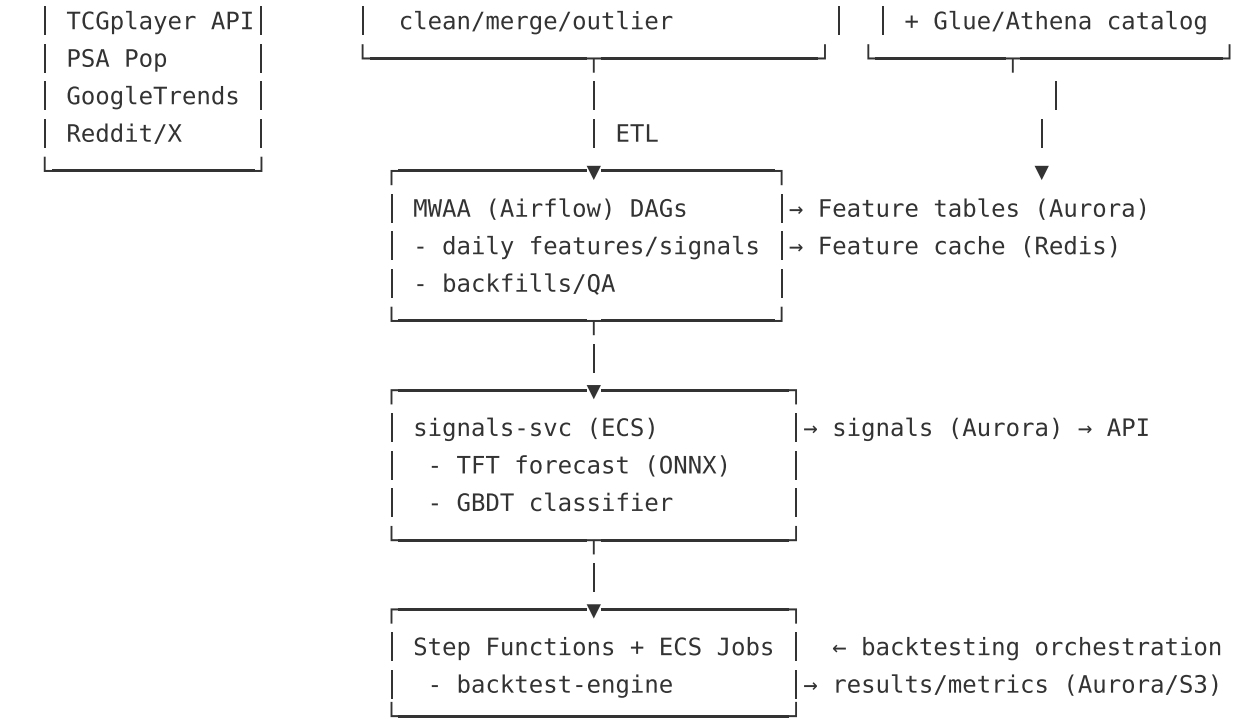
0) North-star principles

- **Data > opinions:** Source, unify, and quality-gate marketplace, grading, and sentiment data to drive every feature.
- **Stateless edge, stateful core:** Scale web & APIs horizontally; keep authoritative state in managed services (Aurora Postgres Serverless v2, S3, OpenSearch, Redis).
- **Batch + streaming pragmatism:** Nightly signals + on-demand refresh for high-impact cards; stream minimal deltas to keep UX current.
- **Interpretability at the glass:** Simple BUY/HOLD/SELL for hobbyists; full factor/forecast decomposition for pros.
- **Cost before complexity:** Prefer managed/serverless, scale-to-zero where feasible, and partition hot vs. cold workloads.

---

1) High-level system overview






```

---

## 2) Finalized tech stack (build now)

**Frontend**
- **Next.js 14** (App Router, ISR/SSG/SSR), **TypeScript**, **TailwindCSS**, **Radix UI**.
- **Data fetching:** `@tanstack/react-query` (client) + server actions.
- **Charts:** `echarts` (web) with accessibility helpers.
- **Auth:** OAuth/email/password to API; JWT baked into HttpOnly cookies.

**Backend / Services**
- **Python 3.11 / FastAPI** (+ Uvicorn) for `api-gw`, `portfolio-svc`, `alerts-svc`, `backtest-api`.
- **Background tasks:** ECS Fargate workers + Redis Queue (RQ) for short jobs; **AWS Step Functions** for long backtests.
- **Orchestration:** **MWAA (Apache Airflow)** for ETL, nightly signals, backfills.
- **Model serving:** ONNX Runtime (CPU) inside `signals-svc`; PyTorch for offline training.

**Data & Infra**
- **Primary DB:** **Amazon Aurora PostgreSQL Serverless v2** (multi-AZ). Native partitioning for trades/price tables.
- **Search:** **Amazon OpenSearch Serverless** for card metadata + autocomplete.
- **Cache:** **Amazon ElastiCache Redis** for hot keys + feature cache + WebSocket fanout.
- **Object store / lake:** **Amazon S3** (raw/trusted/curated zones) + **Glue** catalog + **Athena** for ad-hoc SQL.
- **Queues:** **Amazon SQS** (ingestion), **SNS** (event fanout).
- **CDN:** **CloudFront** fronting Next.js & images (S3 origin).
- **Runtime:** **AWS ECS Fargate** (spot for non-critical workers), **ECR** for images.
- **IaC/CI/CD:** **Terraform** modules + **GitHub Actions** (build→test→scan→image push→deploy via ECS/ArgoCD if EKS).
- **Observability:** **OpenTelemetry** → **AWS Distro for OpenTelemetry** → CloudWatch metrics/logs + **Grafana**/**Prometheus** (AMP) + **Sentry**.

**Security**
```

- **Secrets:** AWS Secrets Manager (+ auto rotation for DB).
- **AuthN/Z:** JWT (PASETO v4 recommended) + Argon2id password hashing, RBAC & Postgres RLS.
- **Network:** Private subnets for DB/caches; ALB + AWS WAF on public edges.
- **Compliance:** Data retention & Right-to-Delete flows; affiliate disclosures.

---

3) Domain model & storage (core tables)

> **Notation:** PK ⋆, FK →, NN (NOT NULL). Partitioned where noted. All monetary values in **USD cents** (integers).

3.1 Entities (summary)
- **cards**: canonical card master.
- **sets**: TCG sets/expansions.
- **grades**: grading scales (raw, PSA10, PSA9…).
- **trades** (partitioned by month): atomic sales from marketplaces.
- **prices_daily** (partitioned by month): one OHLC per day per (card,grade).
- **features_daily** / **features_latest**: engineered factors per day/latest snapshot.
- **signals**: model outputs (B/H/S, conviction, expected return, risk).
- **users**, **portfolios**, **holdings**, **alerts**, **watchlists**.
- **backtest_runs**, **backtest_trades**, **metrics** (summary).
- **affiliate_clicks**: outbound monetization telemetry.

3.2 DDL (extracts)

```sql
-- cards & sets
create table sets (
  set_id text primary key,
  name text not null,
  release_date date,
  series text
);

create table cards (
  card_id text primary key,
  name text not null,
  set_id text not null references sets(set_id),
  number text not null,
  rarity text,
  supertype text,
  subtype text,
  image_small text,
  image_large text,
  unique (set_id, number)
);

-- grades (enumerated universe)
create table grades (
  grade_code text primary key,        -- 'raw','psa10','psa9',...
  description text not null
);

-- trades: atomic marketplace sales (monthly partitions)
create table trades (
```

```
  trade_id bigserial primary key,
  card_id text not null references cards(card_id),
  grade_code text not null references grades(grade_code),
  ts timestamptz not null,
  source text not null,               -- 'ebay','tcgplayer',...
  price_cents int not null check (price_cents > 0),
  currency text default 'USD',
  url text,
  meta jsonb,
  unique (card_id, grade_code, ts, source, price_cents)
) partition by range (ts);

-- partition helper (create per month)
create table trades_2025_08 partition of trades
  for values from ('2025-08-01') to ('2025-09-01');

-- daily price OHLC (by day)
create table prices_daily (
  card_id text not null references cards(card_id),
  grade_code text not null references grades(grade_code),
  day date not null,
  open_cents int,
  high_cents int,
  low_cents int,
  close_cents int,
  volume int,
  primary key (card_id, grade_code, day)
);

-- features (latest snapshot per card/grade)
create table features_latest (
  card_id text not null references cards(card_id),
  grade_code text not null references grades(grade_code),
  asof_ts timestamptz not null,
  features jsonb not null,            -- normalized vector & raw features
  recency_days int not null default 0,
  primary key (card_id, grade_code)
);

-- signals (BUY/HOLD/SELL)
create table signals (
  signal_id bigserial primary key,
  card_id text not null references cards(card_id),
  grade_code text not null references grades(grade_code),
  horizon_days int not null default 90,
  asof_ts timestamptz not null,
  model_version text not null,
  action text not null check (action in ('BUY','HOLD','SELL')),
  conviction numeric not null check (conviction >= 0 and conviction <= 1),
  expected_return numeric,            -- e.g. 0.08 for +8%
  risk numeric,                       -- annualized volatility proxy
  utility numeric,                    -- risk-adjusted score
  features jsonb,                     -- snapshot used for this signal
  unique (card_id, grade_code, horizon_days, asof_ts, model_version)
);
```

```
-- users, portfolios, holdings
create table users (
  user_id uuid primary key,
  email citext unique not null,
  password_hash text,                 -- Argon2id if password auth
  tier text not null default 'free',  -- free|premium|elite|admin
  created_at timestamptz not null default now()
);

create table portfolios (
  portfolio_id uuid primary key,
  user_id uuid not null references users(user_id),
  name text not null,
  created_at timestamptz not null default now()
);

create table holdings (
  holding_id uuid primary key,
  portfolio_id uuid not null references portfolios(portfolio_id),
  card_id text not null references cards(card_id),
  grade_code text not null references grades(grade_code),
  qty int not null check (qty > 0),
  cost_basis_cents int,
  purchase_date date,
  created_at timestamptz not null default now()
);

-- alerts
create table alerts (
  alert_id uuid primary key,
  user_id uuid not null references users(user_id),
  card_id text references cards(card_id),
  grade_code text references grades(grade_code),
  type text not null,                 -- threshold_above, pct_change, signal_change, etc.
  threshold numeric,
  window text,
  channel text not null,              -- email, in_app, discord
  active boolean not null default true,
  cooldown_minutes int not null default 60,
  created_at timestamptz not null default now()
);

-- backtests
create table backtest_runs (
  run_id uuid primary key,
  user_id uuid references users(user_id),
  name text,
  strategy_dsl text not null,
  universe jsonb not null,              -- list of (card_id, grade_code)
  start_date date not null,
  end_date date not null,
  initial_capital_cents int not null,
  assumptions jsonb,                    -- fees, slippage, delay, etc.
  status text not null,                 -- queued|running|done|error
  created_at timestamptz not null default now()
);
```

```
create table metrics (
  metric_id bigserial primary key,
  run_id uuid references backtest_runs(run_id),
  card_id text references cards(card_id),
  grade_code text references grades(grade_code),
  start_date date,
  end_date date,
  cumulative_return numeric,
  cagr numeric,
  volatility numeric,
  sharpe numeric,
  sortino numeric,
  max_drawdown numeric,
  win_rate numeric,
  avg_trade_return numeric,
  trade_count int,
  details jsonb,
  created_at timestamptz not null default now()
);
```

Indexes (representative)
- `trades`: `(card_id, grade_code, ts)`, `(source, ts)`.
- `prices_daily`: `(card_id, grade_code, day desc)` include `(close_cents, volume)`.
- `signals`: `(card_id, grade_code, asof_ts desc)`.
- `holdings`: `(portfolio_id)`; `alerts`: `(user_id, active)`.

Row-level Security
- Enable RLS on `portfolios`, `holdings`, `alerts`, `backtest_runs`, `metrics`.
- Policies: `user_id = current_setting('app.user_id')::uuid`.

---

4) Service decomposition & responsibilities

4.1 `api-gw` (FastAPI)
- Public REST: search, card detail, portfolio CRUD, alerts CRUD, signals read, backtest submit/get.
- Auth (login, refresh, OAuth callbacks). Issues short-lived JWT in **HttpOnly** cookie.
- Rate limits via API Gateway + app middleware (per IP + per user tier).

4.2 `portfolio-svc`
- Aggregates holdings with latest prices to compute portfolio P&L, allocation, and risk (caches per user).
- Exposes endpoints consumed by dashboard.

4.3 `alerts-svc`
- Evaluates alert rules against latest signals/prices (Redis streams or periodic tick).
- Delivers via **WebSocket** (Socket.IO or raw WS), email (SES), and Discord (bot webhook).
- De-duplicate via cooldown, persist deliveries (for audit & UX).

4.4 `ingestion-workers`
- Pull marketplace data (eBay, TCGplayer), PSA pop, Google Trends, Reddit/X.
- Push raw JSON to S3 (raw zone), then normalize → `trades` with outlier filters, reconcile duplicates.

- Fanout SQS messages when a card’s daily close changes → update `prices_daily` and OpenSearch doc.

4.5 `signals-svc`
- Nightly (Airflow) and on-demand inference per (card, grade):
  - Build features from `prices_daily` + external signals (sentiment, trends, pop deltas).
  - TFT (ONNX) forecast expected return + uncertainty.
  - GBDT classifier maps features to BUY/HOLD/SELL probabilities.
  - Compute risk-adjusted utility & conviction; write to `signals`.
  - Cache hot signals in Redis (`card:signal:{card_id}:{grade}`).
- Exposes explain API for premium users (feature attributions, forecast bands).

4.6 `backtest-api` + `backtest-engine`
- Accepts Strategy DSL + universe + assumptions; validates and enqueues.
- **Step Functions** orchestrates chunked ECS jobs over date ranges; accumulates equity curve.
- Writes trade log (S3 parquet), metrics (Aurora). Returns summary, S3 link for CSV.

---

5) Search architecture (OpenSearch)

Index: `cards_v1`
- **Mapping:** `name` (text, edge-ngram analyzer for autocomplete, lowercase/ASCII), `set_name`, `number`, `rarity`, `types`, `series`, `aliases`, `id`, `image_small`.
- **Autocomplete:** `/search?q=mew` uses multi-match over `name.autocomplete`, boosts exact phrase; filter facets by set/rarity.
- **Sync:** Airflow nightly full sync; incremental updates on `cards` changes.

---

6) Data pipelines

6.1 ETL zones (S3)
- **raw/**: source snapshots (`ebay/{date}/...jsonl`, `tcg/{date}/...`).
- **trusted/**: deduped, schema-validated records.
- **curated/**: analytic parquet (trades, prices_daily, sentiment_daily).

6.2 Airflow DAGs
- `dag_ingest_market`: pull APIs (respect rate limits), store raw, validate (Great Expectations), write `trades`.
- `dag_prices_ohlc`: resample to daily OHLC per (card,grade), update `prices_daily`, OpenSearch refresh.
- `dag_features_signals`: compute feature vectors, write `features_latest`, call `signals-svc` batch.
- `dag_pop_trends_sentiment`: scrape/ingest external signals; update derived features.
- `dag_backfill`: parametric replays for new cards/grades.
- **Schedule:** Business-day nightly (~02:00 UTC), on-demand for hot cards.

Outlier policy: Tukey fences + z-score; flag to `meta.outlier=true` but preserve in S3/trusted for audit; exclude from `prices_daily` unless confirmed.

---

7) API surface (selected)

- `GET /search?q=&set=&page=&limit=` → `CardSummary[]`

- `GET /cards/{cardId}?grade=psa10` → details, images, latest price, **latest signal**
- `GET /cards/{cardId}/history?grade=psa10&window=365d` → OHLC & feature overlays
- `GET /cards/{cardId}/signal?grade=psa10&horizon=90` → `{action, conviction, expected_return, risk, utility, asof_ts}`
- `GET /portfolio` / `POST /portfolio/holdings` / `DELETE ...`
- `GET /alerts` / `POST /alerts` / `PATCH /alerts/{id}`
- `POST /backtests` (DSL + universe + assumptions) → `{run_id}`
- `GET /backtests/{run_id}` → status + summary; `GET /backtests/{run_id}/trades` (CSV link)

Auth
- `POST /auth/login`, `POST /auth/register`, `POST /auth/refresh`, `POST /auth/logout`.
- JWT (short TTL) + refresh token rotation; scopes by tier (`free`, `premium`, `elite`, `admin`).

Rate limits (default)
- Unauthed: 60 req/min/IP; Authed Free: 600 req/min; Premium/Elite: higher; backtests: 3 concurrent/user (queue).

---

8) Caching & performance

- **Next.js ISR:** Card pages use ISR (revalidate 5–15 min) + client SWR for live numbers.
- **Redis keys:**
  - `card:signal:{card}:{grade}` TTL 24h (auto-invalidated nightly or on update)
  - `card:price:daily:{card}:{grade}` (compressed series)
  - `portfolio:summary:{user}:{hash}` TTL 60s
- **DB optimizations:** Partition `trades`, `prices_daily`; `VACUUM` & `ANALYZE` nightly; HOT data in memory via `pg_prewarm`.
- **CDN:** CloudFront caches images/static; origin shield; WebP/AVIF variants.

SLOs
- Search p95 < **2.0 s**; Card detail p95 < **2.5 s**; Signal fetch p95 < **300 ms** (from cache).
- Alerts delivery (threshold) **≤ 2 min** end-to-end.

---

9) Signals & backtesting integration (from quant specs)

Signals runtime
1. Airflow selects changed (card,grade) from `prices_daily` Δ.
2. Build feature vector (momentum, vol, drawdown, liquidity, sentiment, trends, pop; normalized & clipped).
3. **TFT (ONNX)** multi-horizon forecast → expected return + uncertainty.
4. **GBDT** classifier → class probabilities.
5. Risk-adjusted utility `U = R̂ − λσ̂` → logistic to **conviction**; **B/H/S** via thresholds & hysteresis.
6. Persist `signals`, cache hot keys, notify `alerts-svc` of signal flips (via SNS).

Backtesting runtime
1. `POST /backtests` → validate DSL + universe + assumptions; write `backtest_runs(queued)`.
2. Step Functions fan-out per card chunk → ECS jobs run **daily-bar simulator** (next-day execution, fees/slippage).
3. Persist trade log (S3 parquet), compute metrics (Sharpe, Sortino, MDD, CAGR, hit rate, turnover).

4. Aggregate & write `metrics`, mark run `done`; expose CSV link.

No-lookahead guarantees: Indicators computed with past bars; positions shifted one day; next available sale price execution.

---

10) MLOps lifecycle

- **Data readiness:** Great Expectations checks on features (missingness, drift, freshness).
- **Training:** Offline jobs (ECS GPU optional) retrain TFT & GBDT monthly or on drift; store artifacts in **S3 model registry** with `model_version`.
- **Validation:** Backtest on rolling OOS windows; acceptance gates on Sharpe/Drawdown vs. baseline; calibration (Platt/temperature).
- **Deployment:** Promote to `signals-svc` via canary (percentage of cards); shadow inference for a week; roll forward on green.
- **Explainability:** SHAP/attention summaries stored per cohort for premium “Why” panels.
- **Monitoring:** Inference latency, feature drift, signal flip rate, realized vs. expected return gaps.

---

11) Observability & SRE

- **Tracing:** OpenTelemetry trace IDs injected from edge → services → DB calls.
- **Metrics:**
  - API: p50/p95/p99 latency, error rates per route/tier.
  - Pipelines: DAG success %, task duration, lag from source.
  - Signals: cards processed/min, inference latency, failure rate.
  - Backtests: queue depth, runtime, success %, cost/job.
- **Dashboards:** Grafana boards for API perf, DB health (connections, locks), Redis (hit ratio), Airflow DAGs.
- **Alerting:** PagerDuty/on-call for SLO breaches, DAG failures >N runs, replication lag, OpenSearch cluster health.
- **Log policy:** JSON logs with `corr_id`, PII redaction, 30-day hot, 180-day cold in S3 (lifecycle).

---

12) Security, privacy, compliance

- **AuthN:** Email/password (Argon2id), OAuth (Google/Discord). Device-bound refresh tokens.
- **AuthZ:** RBAC + Postgres RLS; tier gating in middleware (feature flags).
- **Secrets:** AWS Secrets Manager (DB creds, API keys, JWT signing), rotated; no plaintext in env.
- **Crypto:** TLS 1.2+ everywhere; AES-256 at rest (S3/Aurora/Redis encryption).
- **WAF:** AWS WAF rules (SQLi/XSS), bot control, rate limiting bursts.
- **Headers:** CSP, HSTS, X-Frame-Options, X-Content-Type, Referrer-Policy.
- **PII & GDPR/CCPA:** Data map; access/export/delete endpoints; 30-day delete SLA; consent for cookies/affiliates.
- **Audit:** Admin actions & data exports logged; backtest/alert changes versioned.
- **Marketplace policies:** Affiliate disclosure banners, tracking parameters audited; “not investment advice” disclaimers.

---

13) Cost management

- **Serverless & scale-to-zero:** Aurora Serverless v2 auto-scales; Fargate task min counts per environment; Lambda for light ETL.
- **Spot where safe:** Fargate Spot for ingestion & backtests; on failure, requeue.
- **Tiered storage:** S3 lifecycle (30-day to infrequent access; 180-day to Glacier); prune old raw except compliance snapshots.
- **OpenSearch Serverless:** Limit replicas, ILM to delete stale indices; card index < few GB.
- **Right-size caches:** Redis memory alerts; evict LRU; compress Redis payloads.
- **CI budgets:** Cost visibility per workflow; destroy preview stacks post-merge.

---

14) Environments & networking

- **Envs:** `dev` (shared), `stage` (pre-prod), `prod` (isolated).
- **VPC:** Private subnets (Aurora/Redis/OpenSearch), public for ALB; NAT for egress.
- **Security groups:** Least privilege east-west; only ALB opens 443 to the world.
- **Endpoints:** VPC endpoints for S3/Secrets/CloudWatch to avoid NAT costs.
- **Data residency:** Primary region `us-east-1`; read-only replica (later) for EU if needed.

---

15) CI/CD & IaC

Git strategy
- `main`: prod; `develop`: stage; `feature/*`, `hotfix/*`.

Pipelines (GitHub Actions)
1. **PR:** lint (ruff/black/mypy, eslint), unit tests, coverage gate, SAST (CodeQL), docker build (multi-arch), trivy scan.
2. **Merge to develop:** Terraform plan/apply (stage), deploy ECS tasks (blue/green), run smoke & e2e (Playwright).
3. **Promote to main:** Same to prod with manual approval and canary rollout; DB migrations via Alembic/`psql` job.

Terraform modules
- `vpc/`, `aurora/`, `redis/`, `opensearch/`, `ecs/`, `mwaa/`, `s3-buckets/`, `cloudfront/`, `waf/`, `secrets/`, `monitoring/`.

Migrations
- Alembic revisions; zero-downtime pattern (add columns → backfill → switch reads → drop old).

---

16) Sequence diagrams

16.1 Card page load (with signal)

```
User → CloudFront: GET /cards/{id}?grade=psa10
CloudFront → api-gw: cache miss → origin fetch
api-gw → Redis: GET card:signal:{id}:psa10  (hit? return)
  miss:
    api-gw → Aurora: SELECT latest signal row
    api-gw → Redis: SETEX card:signal:...
```

```
api-gw → Aurora: SELECT prices_daily window + metadata
api-gw → OpenSearch: GET card doc (fallback if needed)
api-gw → User: JSON (card, latest signal, chart)
```

16.2 Nightly signals

```
Airflow → Aurora: list updated (card,grade)
Airflow → signals-svc (ECS): batch infer
signals-svc → Aurora: INSERT signals rows
signals-svc → Redis: refresh cache hot keys
signals-svc → SNS: publish signal_flip events
alerts-svc → WS/Email/Discord: deliver notifications
```

16.3 Backtest

```
User → backtest-api: POST DSL+universe
backtest-api → Aurora: INSERT backtest_runs (queued)
Step Functions: start → map per chunk
  ECS job: run engine → write trades(S3), metrics(Aurora)
Step Functions: aggregate → set status=done
User → GET /backtests/{run_id}: summary + CSV link
```

---

17) UX integration notes

- **Casual view:** expose `{action, conviction badge}` plus “simple reason” (top 2 drivers).
- **Pro view:** toggle shows factor panel (momentum/vol/drawdown/sentiment), forecast fan, expected return, risk, utility, Sharpe of signal history.
- **Portfolio:** P&L waterfall (by card, set), exposure by grade, risk (volatility) bubble chart.
- **Backtest UI:** DSL builder (IF/THEN), run summary (equity curve + drawdown + trade table), export CSV.

---

18) Runbooks (abbrev)

- **Data lag:** If `prices_daily` behind > 6h → check ingestion workers, source rate limits, SQS dead letters; backfill DAG.
- **Signal failure spike:** Roll back `model_version` pointer; re-pin to previous; requeue failures.
- **OpenSearch red:** Scale shards, reduce replica, snapshot & restore, or throttle writes.
- **Aurora hot partition:** Create new partitions, run `ANALYZE`, offload heavy reads to read endpoint, add covering indexes.

---

19) Roadmap hooks

- **Mobile (Phase 2):** Reuse Next.js API; add camera scanning (on-device model) → upload

candidate → resolve to `card_id`.
- **Discord bot:** Thin client to `api-gw` with bot token scopes; rate limit per guild; slash commands (`/price`, `/signal`).
- **Public API (later):** API keys + per-key quotas; docs via OpenAPI + portal.
- **Multi-TCG expansion:** Reuse schemas; add `game` column to `sets`/`cards`; reindex OpenSearch.

---

20) Appendix — key config

```yaml
app:
  base_currency: USD
  tiers:
    free:
      alerts_max: 5
      portfolios_max: 1
    premium:
      alerts_max: 50
      portfolios_max: 5
    elite:
      alerts_max: 1000
      portfolios_max: 50
signals:
  horizon_days: 90
  lambda_risk_aversion: 4.0
  hysteresis: 0.02        # utility threshold buffer
  refresh:
    nightly_utc: "02:00"
    hot_cards_cron: "*/15 * * * *"
backtests:
  max_concurrent_per_user: 3
  default_fees:
    buy_pct: 0.05
    sell_pct: 0.10
  slippage_pct: 0.00
etl:
  sources:
    ebay:
      rate_limit_per_min: 120
    tcgplayer:
      rate_limit_per_min: 60
  schedules:
    nightly_utc: "01:00"
    pop_trends_weekly: "Sun 03:00"
```

Implementation status & next steps

1) **Provision infra (Terraform):** VPC, Aurora Serverless v2, ElastiCache Redis, OpenSearch Serverless, ECS clusters, S3 buckets (raw/trusted/curated), MWAA, CloudFront+WAF, Secrets.
2) **Scaffold services:** `api-gw`, `portfolio-svc`, `alerts-svc`, `ingestion-workers`, `signals-svc`, `backtest-api`, `backtest-engine`.

3) **Ship MVP slices:** search → card detail → portfolio → alerts → nightly signals; then backtesting UI & Step Functions.
4) **Wire MLops:** model registry, baseline TFT+GBDT, nightly inference; add explainability endpoints.
5) **Observability & SLOs:** dashboards + alert rules; runbooks in `/docs/ops/`.

> This architecture is **build-ready** and already aligned with the quant specs: the signals and backtesting contracts above are the interfaces the web/app and data teams will rely on.