# CollectIQ — AWS Services, Rationale & Cost Model

Scope: Pokémon TCG MVP (1-week hackathon) + post-hackathon scale-up. Audience: DevOps / Cloud Engineering.

## 1) Purpose

This document details the AWS services CollectIQ will use, why each is required, how it is configured, and a pragmatic cost model with levers and example monthly scenarios. Pricing varies by region and time; treat dollar values as directional. Use AWS Cost Explorer and Budgets to track actuals.

### Amazon S3 (Object Storage)

Why we need it:

Stores raw uploads (card images), thumbnails, and AI inference artifacts. Presigned URLs enable secure client uploads without exposing credentials.

How we’ll configure it:

One bucket per stage (e.g., dev/prod). Block public access, enforce SSL, S3-managed encryption, short-lived presigned URLs (~60s). Separate prefixes: uploads/, thumbs/, artifacts/. Lifecycle: raw→Standard-IA (30d), tmp→expire(7d).

Cost model (directional):

Billed per GB-month stored + PUT/GET requests + data egress. For MVP: tens of GBs + thousands of requests → low monthly cost. Egress is near-zero if served via same region and CloudFront; spikes if external downloads are large.

Cost levers and guardrails:

• Aggressive lifecycle policies for tmp/ and artifacts/.  
• Compress thumbnails; limit max upload size.  
• Use CloudFront if public delivery grows; keep S3 requester-pays OFF.  
• Monitor with Storage Lens.

### AWS Lambda (Serverless Compute)

Why we need it:

Runs identification orchestration, valuation normalization, and persistence logic with near-zero ops overhead.

How we’ll configure it:

Node.js 20 functions for identify, price normalization, persist, and presign. Memory 512–1024MB; timeouts 10–30s. X-Ray tracing on. One function per bounded context to keep IAM tight.

Cost model (directional):

Billed per request + GB-seconds. MVP traffic (hundreds to thousands of invocations/day) typically low cost. Heavier Bedrock/Rekognition calls dominate overall cost, not Lambda itself.

Cost levers and guardrails:

• Right-size memory; avoid over-long timeouts.  
• Reuse connections (keep-alive) and cache config in init scope.  
• Avoid chatty downstream calls; batch writes to DynamoDB where possible.

### Amazon API Gateway or Next.js Route Handlers

Why we need it:

Expose secure APIs for upload, identify, valuation, and vault access. For MVP we prefer Next.js route handlers; API Gateway is a clean fallback.

How we’ll configure it:

If Next.js routes suffice, deploy via Amplify/CloudFront. If using API Gateway: HTTP API (cheaper) → Lambda proxy, auth via Cognito JWT. Enable WAF on public entrypoint.

Cost model (directional):

Next.js handlers cost is embedded in hosting; API Gateway charges per million calls. With hackathon traffic, cost remains low. WAF adds small fixed + request fees if enabled.

Cost levers and guardrails:

• Prefer route handlers for simplicity; move to API Gateway as traffic/latency patterns require.  
• Cache GETs at edge via CloudFront.  
• Enable gzip/brotli.

### Amazon DynamoDB (NoSQL)

Why we need it:

Primary store for user vault, card records, and price snapshots. Single-table design scales elastically and fits event-shaped writes.

How we’ll configure it:

On-demand capacity (PAY\_PER\_REQUEST). Keys: PK=USER#{userId}, SK=CARD#{cardId}; secondary entity for price snapshots (PK=CARD#{cardId}, SK=PRICE#ISO8601). PITR on; TTL for ephemeral items.

Cost model (directional):

Billed per read/write request unit + storage. On-demand is ideal for spiky hackathon loads. With thousands of items and light reads, cost is low; heavy analytics later suggests DAX or export-to-S3.

Cost levers and guardrails:

• Prefer batch writes/queries; avoid hot partitions.  
• Compress large attributes; avoid unbounded item growth.  
• Use GSIs only when needed.

### Amazon Cognito (Auth)

Why we need it:

Managed user pools for sign‑up/sign‑in, JWT issuance. Avoids building auth from scratch.

How we’ll configure it:

User Pool + App Client per stage. Email sign-in; optional MFA in prod. Frontend via Amplify Auth UI or cognito-auth libs.

Cost model (directional):

Charges for MAUs and optional features. Hackathon scale (hundreds–low thousands users) is typically minimal.

Cost levers and guardrails:

• Use Hosted UI where possible.  
• Enforce password policy; enable email verification.  
• Rotate app client secrets; store in SSM.

### Amazon Bedrock (Foundation Models)

Why we need it:

LLM tasks: metadata extraction, reasoning for valuation summaries; optional multimodal parsing depending on available models.

How we’ll configure it:

Adapter pattern from backend to Bedrock InvokeModel APIs. Cache prompts + outputs for repeat calls. Keep payloads small and redact PII.

Cost model (directional):

Billed per token or per input/output unit; varies by model family. Bedrock usage likely the largest AI cost driver if prompts are long or frequent.

Cost levers and guardrails:

• Keep prompts terse; prefer JSON responses.  
• Batch summarizations.  
• Cache recent inferences; fall back to mock in demo.  
• Monitor throttling/quotas.

### Amazon Rekognition (Vision)

Why we need it:

Computer vision for text/feature detection in card images; assists identification and authenticity heuristics.

How we’ll configure it:

Use DetectText/DetectLabels as a baseline; custom model is out-of-scope for the hackathon. Combine signals with visual-hash heuristics.

Cost model (directional):

Billed per image request tier. MVP volumes remain small → low cost.

Cost levers and guardrails:

• Only analyze required resolutions; pre-resize client side.  
• Skip duplicate images via client-side SHA or ETag checks.

### Amazon CloudWatch & AWS X-Ray (Observability)

Why we need it:

Centralized logs, metrics, dashboards, and traces for Lambdas/APIs. Essential for demo reliability and post-mortems.

How we’ll configure it:

Structured JSON logs; dashboards for p50/p95 latency and error rates; alarms to SNS. X-Ray enabled on Lambdas and API entrypoints.

Cost model (directional):

Charges for logs ingested/retained, custom metrics, and trace storage. Keep retention short in dev; dashboards are inexpensive.

Cost levers and guardrails:

• Trim log verbosity; set retention to 7–14 days in dev.  
• Aggregate metrics; avoid high-cardinality labels.  
• Sample traces (e.g., 10–20%).

### AWS CDK (CloudFormation)

Why we need it:

Infrastructure-as-code for reproducible environments and rapid iteration.

How we’ll configure it:

TypeScript CDK app with stacks per stage (`collectiq-dev`, `collectiq-prod`). Checked into repo; CI deploys on main.

Cost model (directional):

CDK synth/deploy itself has no direct cost; CloudFormation stacks incur no charge beyond the resources they create.

Cost levers and guardrails:

• Reuse constructs; tag resources with Project/Stage/Owner.  
• Store context in cdk.json; parameterize stage.

### AWS Amplify Hosting or CloudFront

Why we need it:

Global delivery of the web app; CI-backed deploys with caching at edge.

How we’ll configure it:

For MVP simplicity use Amplify Hosting (builds Next.js, previews PRs). For higher control, build static + SSR via CloudFront + Lambda@Edge.

Cost model (directional):

Amplify billed by build minutes, storage, and data transfer; CloudFront billed by egress/requests. MVP scale → low.

Cost levers and guardrails:

• Cache static assets aggressively; set immutable headers.  
• Optimize images; use Next/Image.  
• Monitor 4xx/5xx in reports.

### AWS WAF (Optional)

Why we need it:

Layer 7 protection (bot control, IP rate limits) in front of CloudFront/API Gateway as traffic grows.

How we’ll configure it:

Attach managed rule groups; tune rate-based rules for upload/identify endpoints.

Cost model (directional):

Charges include web ACL + request fees; skip for hackathon unless required by policy.

Cost levers and guardrails:

• Start with managed core ruleset.  
• Observe false positives before tightening.

### AWS Systems Manager Parameter Store / Secrets Manager

Why we need it:

Secure configuration and secret storage (API keys, model IDs, DB/table names).

How we’ll configure it:

Parameter Store for non-secret config; Secrets Manager for sensitive values; access via IAM in Lambdas.

Cost model (directional):

Parameter Store standard throughput is low-cost; Secrets Manager charges per secret/month + API calls.

Cost levers and guardrails:

• Prefer Parameter Store for most config; Secrets Manager only for rotating secrets.  
• Cache reads in function init scope.

### Amazon SNS / EventBridge (Optional Alerts)

Why we need it:

Simple notifications (price alert mock) and event fan-out.

How we’ll configure it:

EventBridge bus for rule-based triggers; SNS email for demo alerts. Replace with real push later.

Cost model (directional):

Both are low-cost for small message volumes (pennies).

Cost levers and guardrails:

• Use minimal rules; avoid chatty schedules.  
• Batch where possible.

### Amazon Route 53 (Optional Custom Domain)

Why we need it:

Branded domain for judges and future users.

How we’ll configure it:

Hosted zone + A/AAAA records to Amplify or CloudFront.

Cost model (directional):

Domain registration yearly; hosted zone small monthly fee; DNS queries are low-cost.

Cost levers and guardrails:

• Use subdomain per stage (dev.collectiq.app).

## 2) Example Monthly Cost Scenarios (Directional)

These are estimation frameworks for budgeting; replace figures with actuals from Cost Explorer once live.

| Service | Hackathon (Low) | Early Post-MVP (Med) | Growth (High) | Notes |
| --- | --- | --- | --- | --- |
| S3 | $1–$5 | $5–$25 | $25–$150 | Storage + PUT/GET; lifecycle reduces cost |
| Lambda | $1–$3 | $5–$20 | $20–$150 | Invocations + GB-seconds |
| Next.js Hosting / CloudFront | $2–$10 | $10–$50 | $50–$300 | Data transfer dominates at scale |
| API Gateway (if used) | $0–$5 | $5–$25 | $25–$150 | HTTP API pricing |
| DynamoDB | $2–$10 | $10–$60 | $60–$300 | On-demand R/W + storage |
| Cognito | $0–$5 | $5–$20 | $20–$100 | Based on MAUs |
| Bedrock (LLM) | $5–$50 | $50–$300 | $300–$1500+ | Model & token usage dependent |
| Rekognition | $1–$10 | $10–$60 | $60–$300 | Per image analysis |
| CloudWatch/X-Ray | $3–$15 | $15–$80 | $80–$300 | Logs, metrics, traces |
| WAF (optional) | $0 | $5–$20 | $20–$100 | ACL + request fees |
| SSM/Secrets | $0–$5 | $5–$10 | $10–$20 | Secrets monthly fee + API calls |
| SNS/EventBridge | $0–$2 | $2–$10 | $10–$40 | Low volume messaging |
| Route 53 | $1–$3 | $1–$5 | $1–$10 | Hosted zone + queries; excl. domain reg |

Important: ranges are directional and will vary by region, traffic, and optimizations. Use Budgets with email alerts, and tag all resources (Project=CollectIQ, Stage=dev|prod).

## 3) Cost Guardrails & Monitoring

• Create AWS Budgets for: Monthly Total, Bedrock, CloudWatch Logs, and Data Transfer.  
• Enable Cost Anomaly Detection with email alerts.  
• Tag resources and activate cost allocation tags in Billing.  
• Set CloudWatch alarms for p95 latency, error %, and Lambda throttles; wire to SNS.  
• Keep dev log retention to 7–14 days; auto-delete tmp assets.

## 4) Deployment Checklist (Per Stage)

1. CDK bootstrap and deploy core stack (S3, DDB, Cognito, Lambdas, roles).  
2. Verify presigned upload → S3 PUT → GET success; enforce CORS.  
3. Enable X-Ray on Lambdas; confirm traces in console.  
4. Configure Amplify or CloudFront; set cache policies for static assets.  
5. Store secrets in Secrets Manager; config in SSM Parameter Store.  
6. Create dashboards and budgets; set alarms and anomaly detection.  
7. Run load test on identify endpoint with mock adapters; validate p95 targets.  
8. Record runbook links and escalation contacts.