Scani5 – End-to-End PRD

Hybrid Architecture: Agent in **Go**, Core API in **Go**, Enrichment/Microservices in **Python**.
Modules Covered: Agent, Server/API, Discovery/Enrichment, Database, Web App, Security, Deployment, Observability, QA.
Goal: Ship a scalable, low-overhead **asset discovery + topology visualization** feature tightly integrated with Scani5 VMDR (vulnerabilities from app list/NVD + external sources), with near real-time updates and enterprise-grade controls.

---

0) Executive Summary

Scani5 will extend its VMDR capabilities with a topology subsystem that: (1) augments agents to collect neighbor/service/connection data and full software inventory, (2) ingests and correlates internal telemetry with **OWASP Amass** external discovery, (3) computes and stores a **graph** of assets and interconnections, (4) renders an **interactive visualization** with risk overlays and agent coverage, and (5) continuously enriches nodes with **NVD/Exploit/Advisory** data via Python microservices.

---

1) Scope & Objectives

In scope (v1): - Scani5 Agent (Go): cross-platform lightweight collectors for **OS, software inventory, services, neighbors, connections, interfaces**. - Internal + external discovery: integrate **Amass** to pull domains/subdomains/IPs/ASNs. - Data platform: PostgreSQL/Timescale for OLTP + time-series; optional Neo4j plugin or graph tables. - Core API (Go): secure ingest, query, export, and WebSocket updates; RBAC/OIDC. - Enrichment services (Python): NVD (CPE/CVE), ExploitDB, vendor advisories; risk scoring. - Web App (React + Cytoscape.js): topology canvas + drilldowns + filters.

Out of scope (v1): SNMP/NetFlow, active wide-range port scanning, mobile agents, auto-remediation.

Success metrics: - Agent overhead: **≤3% CPU**, **≤150 MB RSS** during collection; network payload ≤300 KB typical. - Freshness: **≥95%** of agented nodes visible within **5 minutes** of check-in. - Scale: render **25k nodes / 200k edges** with <2s pan/zoom interactions (server-side tiling + LOD).

---

2) Personas & Top Use Cases

- **SecOps Analyst**: visualize high-risk clusters, see exploitability, spot unagented assets.
- **IT Operations**: validate deployment coverage by site/floor/subnet; plan maintenance.
- **IR/Threat Hunter**: trace lateral movement and blast radius from compromised host.

Key scenarios 1) Color nodes by **max CVSS**; filter to **site=factory-01 floor=2**; expand neighbors.
2) Highlight **unagented** nodes (discovered via neighbors/Amass) to find coverage gaps.
3) Click node → details: OS, software list, vuln rollup, services, connections timeline.
4) Live rollout: observe agents appearing via **WebSocket delta stream**.
5) Export current view to JSON/CSV for audit; API access for SIEM/SOAR.

---

3) High-Level Architecture

Language split - **Go**: Agent; Core API (high throughput, mTLS, WS, persistence).
- **Python**: Enrichment microservices (NVD/CPE matching, NLP heuristics, dedupe, scoring).

Core components 1. **Scani5 Agent (Go)** → mTLS → **Ingest API (Go)** → **DB (Postgres/Timescale)**.
2. **Amass Connector (Python)** schedules discovery → normalized ingest → DB.
3. **Correlation/Graph Service (Go)** computes edges/clusters/tiles.
4. **Enrichment Workers (Python)** consume app lists → match CPEs → pull CVEs → compute risk → write back.
5. **Web App (React/Cytoscape)** pulls tiles via REST, subscribes to WS for deltas.

Messaging (optional, at scale): Kafka/NATS for ingest buffering and enrichment fan-out.

---

4) Agent (Go) – Requirements & Design

Platforms: Windows 10/11, Server 2016+; Ubuntu 20.04+, RHEL 8+, Amazon Linux 2; macOS 12+.

Collectors (policy-controlled): - **System**: hostname, OS name/version, kernel/build, uptime, UUID, site/floor tags. - **Interfaces**: name, MAC, IPs (v4/v6), VLAN, MTU; default route; DNS servers. - **Neighbors**: ARP/NDP, L2 adjacencies; DHCP/Router mac if available. - **Connections (lite)**: 5-tuple summaries (src/dst ip/port, proto), counters, first/last seen. - **Services (listening)**: TCP/UDP ports → process name/PID; optional binary hash. - **Software inventory**: package/program list + versions (per-OS method below).
- **Resource telemetry**: optional CPU%, mem, payload size (for SRE).

OS-specific collection
- **Windows**: WMI/CIM + PowerShell (`Get-CimInstance Win32_Product` is avoided; use registry & `winget` / `wmic` alternative), `Get-NetTCPConnection`, `Get-NetNeighbor`, Win32_NetworkAdapter.
- **Linux**: dpkg/rpm/apk queries; `/proc` for sockets; `ip -j addr`, `ip neigh`, `ss -lntu`.
- **macOS**: `system_profiler SPApplicationsDataType` (parsed), `pkgutil --pkgs`, `arp -an`, `lsof -i -P -n`.

Resource controls - Scheduler with jitter (default 5 min; min 1, max 15).
- Per-collector timeout (2–5s typical).
- Adaptive backoff on high CPU/IO.
- Payload gzip + protobuf/JSON, hard cap (e.g., 1.5 MB).

Security & config - **mTLS** client auth; cert bootstrap flow; periodic rotation.
- Payload **Ed25519** signature + nonce to prevent replay.
- Remote policy from `/agent/config` (JSON).
- PII guard: hash usernames/process names; redact paths; toggles per policy.

Agent Payload (JSON Schema v1)

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
```

```
  "$id": "https://scani5.io/schemas/agent-topology-v1.json",
  "title": "Scani5 Agent Topology Payload",
  "type": "object",
  "required": ["agent_id", "ts", "host", "interfaces"],
  "properties": {
    "schema_version": {"type": "string", "const": "1.0"},
    "agent_id": {"type": "string", "format": "uuid"},
    "ts": {"type": "string", "format": "date-time"},
    "host": {
      "type": "object",
      "required": ["hostname","os","uuid"],
      "properties": {
        "hostname": {"type": "string"},
        "uuid": {"type": "string"},
        "os": {"type": "object", "properties": {"name": {"type":"string"},
"version": {"type":"string"}, "kernel": {"type":"string"}}},
        "site": {"type": "string"},
        "floor": {"type": "string"}
      }
    },
    "interfaces": {
      "type": "array",
      "items": {"type":"object","properties": {"name":
{"type":"string"},"mac":{"type":"string"},"ips":{"type":"array","items":
{"type":"string"}},"vlan":{"type":"integer"}}}
    },
    "neighbors": {"type": "array", "items": {"type":"object","properties":
{"ip":{"type":"string"},"mac":{"type":"string"},"via":
{"type":"string","enum":["arp","ndp"]},"last_seen":
{"type":"string","format":"date-time"}}}},
    "services": {"type": "array", "items": {"type":"object","properties":
{"proto":{"type":"string","enum":["tcp","udp"]},"port":
{"type":"integer"},"process":{"type":"string"},"hash":
{"type":"string"},"first_seen":{"type":"string","format":"date-
time"},"last_seen":{"type":"string","format":"date-time"}}}},
    "connections": {"type":"array","items":{"type":"object","properties":
{"src_ip":{"type":"string"},"src_port":{"type":"integer"},"dst_ip":
{"type":"string"},"dst_port":{"type":"integer"},"proto":
{"type":"string"},"count":{"type":"integer"},"first":
{"type":"string","format":"date-time"},"last":
{"type":"string","format":"date-time"}}}},
    "apps": {"type":"array","items":{"type":"object","required":
["name","version"],"properties":{"name":{"type":"string"},"version":
{"type":"string"},"vendor":{"type":"string"},"install_source":
{"type":"string"},"candidates":{"type":"array","items":{"type":"string"}}}}},
    "telemetry": {"type":"object","properties":{"cpu_pct":
{"type":"number"},"mem_mb":{"type":"number"},"payload_bytes":
{"type":"integer"}}},
    "sig": {"type": "string"}
  }
}
```

Notes: `apps[*].candidates` may store speculative CPE strings from local heuristics to aid server-side matching.

---

5) Core Server/API (Go) – Requirements & Contracts

Responsibilities: - Authenticate agents via mTLS; validate payload signature/nonce.
- Normalize & upsert assets, interfaces, services, edges, applications.
- Serve graph/tiles, summaries, node details; broadcast deltas over WS.
- Authorize UI/API users via OIDC (Cognito), JWT scopes, RBAC + row-level scoping.

External Interfaces: REST + WebSocket; optional gRPC for internal services.

5.1 REST/WS Endpoints (OpenAPI snippets)

```
openapi: 3.0.3
info: { title: Scani5 Topology API, version: 1.0.0 }
servers: [{ url: https://api.scani5.internal }]
components:
  securitySchemes:
    mtls: { type: mutualTLS }
    bearerAuth: { type: http, scheme: bearer, bearerFormat: JWT }
  schemas:
    AgentTopologyBatch: { type: object, properties: { items: { type: array,
items: { $ref: '#/components/schemas/AgentTopology' }}}}
    AgentTopology: { $ref: 'https://scani5.io/schemas/agent-topology-
v1.json' }
    GraphTile:
      type: object
      properties:
        tile: { type: string }
        nodes: { type: array, items: { $ref: '#/components/schemas/Node' } }
        edges: { type: array, items: { $ref: '#/components/schemas/Edge' } }
    Node:
      type: object
      properties:
        id: { type: string, format: uuid }
        label: { type: string }
        os: { type: string }
        agented: { type: boolean }
        risk: { type: number }
        ips: { type: array, items: { type: string } }
        last_seen: { type: string, format: date-time }
    Edge:
      type: object
      properties:
        src: { type: string, format: uuid }
        dst: { type: string, format: uuid }
        kind: { type: string, enum: [neighbor, connection] }
        weight: { type: integer }
```

```
        last_seen: { type: string, format: date-time }
security: []
paths:
  /ingest/topology:
    post:
      security: [{ mtls: [] }]
      requestBody:
        required: true
        content:
          application/json:
            schema: { $ref: '#/components/schemas/AgentTopologyBatch' }
      responses:
        '202': { description: Accepted }
  /agent/config:
    get:
      security: [{ mtls: [] }]
      parameters: [{ name: agent_id, in: query, schema: { type: string,
format: uuid } }]
      responses: { '200': { description: OK } }
  /topology/summary:
    get:
      security: [{ bearerAuth: [] }]
      parameters:
        - { name: site, in: query, schema: { type: string } }
        - { name: floor, in: query, schema: { type: string } }
      responses: { '200': { description: OK } }
  /topology/graph:
    get:
      security: [{ bearerAuth: [] }]
      parameters:
        - { name: site, in: query, schema: { type: string } }
        - { name: floor, in: query, schema: { type: string } }
        - { name: only_agented, in: query, schema: { type: boolean } }
        - { name: min_severity, in: query, schema: { type: number } }
        - { name: tile, in: query, schema: { type: string } }
      responses:
        '200':
          content:
            application/json:
              schema: { $ref: '#/components/schemas/GraphTile' }
  /topology/node/{id}:
    get:
      security: [{ bearerAuth: [] }]
      parameters: [{ name: id, in: path, required: true, schema: { type:
string, format: uuid } }]
      responses: { '200': { description: OK } }
  /export/topology:
    get:
      security: [{ bearerAuth: [] }]
      responses: { '200': { description: file } }
  /ws/topology:
```

```
    get:
      security: [{ bearerAuth: [] }]
      responses: { '101': { description: Switching Protocols } }
```

Notes: Keep API response payloads **tile-based** to avoid sending entire graphs.

5.2 Ingest & Processing Flow

1) Validate mTLS + payload signature; dedupe via nonce cache (Redis).
2) Normalize MAC/IP formats; upsert **assets**, **interfaces**, **services**, **apps**.
3) Build/merge **edges** (`neighbor` / `connection`) with weights + timestamps.
4) Publish **delta events** (Redis pub/sub) → WS broadcaster.
5) Queue **apps** for enrichment (Kafka topic: `apps.to_enrich`).

5.3 Policy Model (example)

```
{
  "collect": {
    "interval_seconds": 300,
    "neighbors": true,
    "services": true,
    "connections": true,
    "apps": true
  },
  "privacy": { "hash_process_names": true, "redact_paths": true },
  "limits": { "max_neighbors": 5000, "max_connections": 2000 }
}
```

---

6) Enrichment & Discovery (Python microservices)

Services (FastAPI workers / Celery/Arq): - **CPE Matcher**: transform `(name,version,vendor)` to candidate **CPEs** using rules + fuzzy match.
- **NVD Fetcher**: pull CVEs by CPE from NVD JSON feeds; cache; store **CVSS v3.1**, CWE, references.
- **ExploitDB/Vendor Advisories**: scrape/API where allowed; mark **public exploit availability**.
- **Risk Scorer**: compute `risk_score` per asset: `max(cvss) + exploit_bonus + age_factor + exposure (services/open ports)`.

Data contracts

```
{
  "app_key": {"name":"openssl","version":"1.1.1k","vendor":"OpenSSL"},
  "cpe_candidates": ["cpe:2.3:a:openssl:openssl:1.1.1k:*:*:*:*:*:*:*"],
  "cves": [{"cve":"CVE-2023-12345","cvss":8.8,"exploitable":true}],
  "risk": {"score": 9.3, "factors": {"cvss":8.8, "exploit":0.3, "exposure":
0.2}}
}
```

Storage: tables `apps`, `app_cpe_links`, `cves`, `app_cves`, `asset_risk` (see DB section).

Amass Connector - Scheduled (cron) to run **passive** by default; configurable **active**.
- Parse JSON output; upsert `amass_assets (fqdn, ip, asn, first_seen/last_seen)`; correlate with internal `assets` (by IP/FQDN) → `asset_links (confidence)`.

---

7) Database Design (PostgreSQL + Timescale)

```
-- Assets
CREATE TABLE assets (
  id UUID PRIMARY KEY,
  agent_id UUID,
  hostname TEXT,
  os_name TEXT,
  os_version TEXT,
  kernel TEXT,
  site TEXT,
  floor TEXT,
  mac TEXT,
  ips INET[],
  agented BOOLEAN DEFAULT TRUE,
  risk_score NUMERIC(4,1) DEFAULT 0.0,
  last_seen TIMESTAMPTZ NOT NULL
);
CREATE INDEX assets_site_floor_idx ON assets(site, floor);
CREATE INDEX assets_ips_gin ON assets USING GIN (ips);

-- Interfaces
CREATE TABLE interfaces (
  id UUID PRIMARY KEY,
  asset_id UUID REFERENCES assets(id) ON DELETE CASCADE,
  name TEXT, mac TEXT, ips INET[], vlan INT
);

-- Services
CREATE TABLE services (
  id UUID PRIMARY KEY,
  asset_id UUID REFERENCES assets(id) ON DELETE CASCADE,
  proto TEXT, port INT, process TEXT, hash TEXT,
  first_seen TIMESTAMPTZ, last_seen TIMESTAMPTZ
);
CREATE INDEX services_asset_port_idx ON services(asset_id, port);

-- Applications
CREATE TABLE apps (
  id UUID PRIMARY KEY,
  asset_id UUID REFERENCES assets(id) ON DELETE CASCADE,
  name TEXT, version TEXT, vendor TEXT, install_source TEXT,
  first_seen TIMESTAMPTZ, last_seen TIMESTAMPTZ
```

```
);
CREATE INDEX apps_asset_name_idx ON apps(asset_id, name);

-- Observed Edges
CREATE TABLE edges (
  id BIGSERIAL PRIMARY KEY,
  src_asset UUID REFERENCES assets(id) ON DELETE CASCADE,
  dst_asset UUID REFERENCES assets(id) ON DELETE CASCADE,
  kind TEXT CHECK (kind IN ('neighbor','connection')),
  weight BIGINT DEFAULT 1,
  first_seen TIMESTAMPTZ,
  last_seen TIMESTAMPTZ
);
CREATE INDEX edges_src_dst_idx ON edges(src_asset, dst_asset, kind);

-- NVD/CPE/CVE
CREATE TABLE cpes (
  id UUID PRIMARY KEY,
  cpe TEXT UNIQUE,
  vendor TEXT, product TEXT, version TEXT
);
CREATE TABLE cves (
  id UUID PRIMARY KEY,
  cve TEXT UNIQUE,
  cvss NUMERIC(3,1), severity TEXT, published TIMESTAMPTZ, last_modified
TIMESTAMPTZ,
  has_exploit BOOLEAN DEFAULT FALSE, summary TEXT
);
CREATE TABLE app_cpe_links (
  app_id UUID REFERENCES apps(id) ON DELETE CASCADE,
  cpe_id UUID REFERENCES cpes(id) ON DELETE CASCADE,
  confidence NUMERIC(3,2),
  PRIMARY KEY (app_id, cpe_id)
);
CREATE TABLE app_cves (
  app_id UUID REFERENCES apps(id) ON DELETE CASCADE,
  cve_id UUID REFERENCES cves(id) ON DELETE CASCADE,
  PRIMARY KEY (app_id, cve_id)
);

-- External (Amass)
CREATE TABLE amass_assets (
  id UUID PRIMARY KEY,
  fqdn TEXT,
  ip INET,
  asn INT,
  source TEXT,
  first_seen TIMESTAMPTZ,
  last_seen TIMESTAMPTZ
);
CREATE INDEX amass_fqdn_idx ON amass_assets(fqdn);
```

```
CREATE TABLE asset_links (
  amass_id UUID REFERENCES amass_assets(id) ON DELETE CASCADE,
  asset_id UUID REFERENCES assets(id) ON DELETE CASCADE,
  confidence NUMERIC(3,2),
  PRIMARY KEY (amass_id, asset_id)
);

-- Time-series for agent health
CREATE TABLE agent_checkins (
  agent_id UUID,
  ts TIMESTAMPTZ,
  site TEXT,
  floor TEXT,
  cpu_pct REAL,
  mem_mb REAL,
  payload_bytes INT
);
SELECT create_hypertable('agent_checkins','ts');
```

Views/rollups

```
CREATE MATERIALIZED VIEW asset_vuln_rollup AS
SELECT a.id AS asset_id, COALESCE(MAX(c.cvss),0) AS max_cvss, COUNT(DISTINCT
c.cve) AS cve_count
FROM assets a
LEFT JOIN apps p ON p.asset_id=a.id
LEFT JOIN app_cves ac ON ac.app_id=p.id
LEFT JOIN cves c ON c.id=ac.cve_id
GROUP BY a.id;
```

---

8) Web App (React + Vite + Cytoscape.js)

Pages - **Topology** (default): infinite canvas w/ mini-map; tiles fetched from `/topology/graph?tile=...`; WS deltas for live updates.
- **Assets**: searchable table; drill into node details.
- **Risk Explorer**: filters by CVSS/exploitability; heat overlays.
- **Admin**: Policy editor, Amass settings, RBAC, audit logs.

UI Interactions - Node click → Drawer with OS, apps, services, max CVSS, last seen, tags.
- Double-click node → expand N neighbors (configurable).
- Lasso select → group/pin; export selection.
- Filters: site, floor, subnet, OS, risk, agented.
- Layouts: COSE (force-directed) default; hierarchical by subnet; geo/floor grid.

Performance strategies - Server-side **tile & cluster** aggregation; send only visible subset.
- Client **LOD**: hide labels on zoom-out; edge bundling; cap max render counts.

Security - OIDC login (Cognito), JWT in SPA; per-API site scoping; ETag caching.

---

9) Security, Privacy, Compliance

- Transport: TLS1.3, **mTLS** for agents, JWT for UI; cert pinning in agent.
- Payload security: Ed25519 signature + nonce; replay cache in Redis.
- AuthZ: RBAC (`admin`, `secops`, `itops`, `viewer`); row-level scoping by site.
- Data protection: encryption at rest; PII minimization; hashing of sensitive fields.
- Supply chain: signed agent binaries; SBOM; SLSA level targets; CI attestation.
- Auditing: export events, policy changes, admin actions recorded with actor + timestamp.

---

10) Deployment & DevOps

Environments: Dev, Staging, Prod.
Containerization: Docker images for Go API and Python workers; SPA built to Nginx.

Kubernetes (Helm) key objects - Deployments: `api`, `ws-broadcaster`, `graph-service`, `amass-connector`, `cpe-matcher`, `nvd-fetcher`, `risk-scorer`, `web`.
- Stateful: `postgres` (Timescale), optional `neo4j`.
- Add-ons: `redis`, optional `kafka`.

Env/Secrets: AWS SSM/Secrets Manager; per-env config maps.
CI/CD: GitHub Actions/GitLab CI: build, test, SAST, container scan, sign, deploy.

Reference docker-compose (dev)

```
version: '3.9'
services:
  api:
    image: scani5/api:latest
    ports: ["8080:8080"]
    env_file: .env
    depends_on: [db, redis]
  web:
    image: scani5/web:latest
    ports: ["3000:80"]
    depends_on: [api]
  db:
    image: timescale/timescaledb:pg16
    environment: { POSTGRES_PASSWORD: postgres }
    ports: ["5432:5432"]
  redis:
    image: redis:7
    ports: ["6379:6379"]
  nvd:
    image: scani5/nvd-fetcher:latest
    depends_on: [db]
```

```
  risk:
    image: scani5/risk-scorer:latest
    depends_on: [db]
```

---

11) Observability & SLOs

- **Metrics**: ingest TPS, payload sizes, WS subscribers, render tile latency, DB qps.
- **Logs**: structured JSON; correlation IDs; redaction filters.
- **Tracing**: OpenTelemetry across API → DB → enrichment → WS.
- **SLOs**: 99p ingest < 2s; 95p WS fan-out < 500ms; 95p tile fetch < 300ms.

---

12) QA Plan

- **Unit tests**: parsers, dedupe, schema validation, RBAC.
- **Contract tests**: JSON schema for agent payload; OpenAPI for API.
- **Load tests**: 10k agents synthetic; 200k edges; verify latency budgets.
- **Security tests**: mTLS failure, replay attempts, RLS leaks.
- **UI e2e**: Playwright scenarios for topology interactions and filters.

---

13) Implementation Blueprint (Milestones)

M1 – Agent & Ingest (3–4 weeks) - Go agent collectors (system, interfaces, apps, neighbors, services).
- mTLS bootstrap + signed payloads; `/ingest/topology` upsert path.
- Basic graph build; tiles endpoint; minimal UI prototype.

M2 – Enrichment & Risk (3–4 weeks) - Python CPE matcher + NVD fetcher; CVE store; risk rollups.
- Node coloring by risk; node detail drawer; exports.

M3 – Amass & Coverage (2–3 weeks) - Amass connector; external assets; coverage view (unagented highlight).
- Community clustering; performance tuning (LOD/tiles, indexes).

M4 – Security/RBAC & Admin (2 weeks) - OIDC integration; RBAC; policy editor; audit logs; SLO dashboards.

---

14) Reference Code Stubs

Go – Agent neighbor collector (sketch)

```go
func CollectNeighbors(ctx context.Context) ([]Neighbor, error) {
  switch runtime.GOOS {
  case "windows": return collectWinNeighbors()
  case "darwin": return collectDarwinNeighbors()
```

```
  default: return collectLinuxNeighbors()
  }
}
```

Go – Ingest handler (gin/fiber/chi)

```
func IngestTopology(w http.ResponseWriter, r *http.Request) {
  // 1) mTLS already enforced at proxy; 2) verify signature+nonce
  var batch AgentTopologyBatch
  if err := json.NewDecoder(r.Body).Decode(&batch); err != nil {
http.Error(w, "bad", 400); return }
  for _, item := range batch.Items {
    upsertAsset(tx, item)
    upsertEdges(tx, item)
    enqueueApps(item.Apps)
  }
  w.WriteHeader(http.StatusAccepted)
}
```

Python – CPE match & NVD fetch (FastAPI worker)

```
from fastapi import FastAPI
app = FastAPI()

@app.post("/enrich/apps")
async def enrich_apps(apps: list[dict]):
    matches = [match_cpe(a) for a in apps]
    cves = fetch_cves_for(matches)
    risk = score(cves, exposure_hint=apps)
    save(matches, cves, risk)
    return {"ok": True}
```

React – fetch tile

```
const res = await fetch(`/api/topology/graph?site=${site}&tile=${tileId}`, {
headers: { Authorization: `Bearer ${token}` } });
const tile = await res.json();
cy.add(tile.nodes.map(n => ({ data: n }))); cy.add(tile.edges.map(e => ({
data: e })));
```

---

15) Admin & Governance

- **Config registry** per site/floor (JSON policies); versioned; approvals for changes.
- **Data retention**: connections TTL 30 days; apps/services 180 days; assets forever (soft delete).

- **Exports**: watermark + audit trail; optional PII stripping for reports.

---

16) Risks & Mitigations

- **Volume spikes** → backpressure via queue; batch upserts; payload caps.
- **CPE ambiguity** → heuristic + human override mapping table.
- **UI performance** → strict LOD, clustering, pagination/tiles.
- **Privacy** → hashing, redaction, RBAC scopes; config as code.
- **Supply chain** → signed builds, SBOM, SLSA provenance, dependency pinning.

---

17) Appendices

A) Agent Policy JSON Schema

B) OpenAPI full spec (expand later)

C) Database migrations (Liquibase/Flyway scripts)

D) Helm Charts values.yaml templates

E) Sample synthetic data generator (Go/Python)

Deliverable: This PRD/spec is intentionally **implementation-ready** for AI coding agents: schemas, endpoints, DB DDL, milestones, and code stubs are all provided for scaffolding end-to-end.