Container Image Vulnerability Scanner – Product Requirements Document (PRD)

1. Problem Statement

Organizations running containerized workloads need rapid visibility into which container images contain known vulnerabilities and how severe those issues are. With potentially thousands of images across multiple registries, security engineers require an intuitive way to:

1. Identify images with critical/high CVEs.
2. Prioritize remediation efforts.
3. Track remediation progress over time.

2. Goals & Success Metrics

Goal	Metric	Target
Fast vulnerability visibility	Time from image push to scan result available	< 5 min
Remediation prioritization	% critical CVEs patched within 7 days	> 80 %
Scalability	Images scanned per hour	≥ 5 000

3. Personas

- **DevOps Engineer (Priya)** – maintains CI/CD pipelines, wants automated scanning.
- **Security Engineer (Arun)** – triages CVEs, needs dashboards & drill-downs.
- **Platform Admin (Sara)** – tracks policy compliance across teams.

4. User Stories (excerpt)

1. *As a Security Engineer, I can view a sortable list of all container images with counts of critical/high CVEs so that I know which ones to patch first.*
2. *As a DevOps Engineer, I receive a CI pipeline break if a new image introduces critical vulnerabilities, so that insecure images never reach production.*
3. *As a Platform Admin, I export the latest scan results as CSV for audit reporting.*

5. Functional Requirements

- **Image Inventory** — ingest images from multiple registries (Docker Hub, ECR, GCR).
- **Automated Scanning** — trigger on image push & nightly re-scans.
- **Results Storage** — indexed by image digest, tag, registry.
- **Dashboard** — list view with severity columns, search & filters.
- **Drill-Down View** — vulnerability table, CVE details & remediation guidance.

- **Bulk Actions** — rerun scan, export report, open Jira ticket.
- **Role-Based Access** — viewer vs. admin.
- **API** — REST/GraphQL for integrations.

6. Non-Functional Requirements

- Process ≥ 10 000 images without perceptible UI lag (< 500 ms query time).
- Auth via SSO (OIDC).
- Encryption-in-transit & at-rest.
- 99.9 % UI availability.
- Audit logging.

7. MVP Scope & Release Plan

Sprint 1–2 (Weeks 1-4)

- Registry connector (Docker Hub)
- Async scan worker (Trivy)
- PostgreSQL schema
- Dashboard list & drill-down UI (React)

Sprint 3 (Weeks 5-6)

- Filters/search, export CSV
- Role-based auth
- Slack/Jira webhook POC

8. Wireframes (Low-Fidelity)

```
+-------------------------------------------------------------+
| Container Images – Vulnerability Dashboard                  |
+-----------+---------+----------+-------+------+------------+
| Image     | Tag     | Critical | High  | Med  | Last Scan  |
+-----------+---------+----------+-------+------+------------+
| nginx     | 1.21    |    2     |   5   |  1   | 18-Jul-25  |
| redis     | 7.0     |    0     |   3   |  1   | 18-Jul-25  |
| ...                                                     ... |
+-------------------------------------------------------------+
[Filters] [Bulk Rescan] [Export]
```

```
Breadcrumb: Images / nginx:1.21
+---------------- Summary ----------------+
| Critical: 2 | High: 5 | Med: 1 | Low: 0 |
+-----------------------------------------+
Tabs: [Vulnerabilities] [Layers] [Fixes] [History]
```

```
+--------------------------------------------------------------------------+
| Sev | CVE-ID        | Package  | Version | Fixed In | Description       |
+--------------------------------------------------------------------------+
| CRI | CVE-2024-1234 | openssl  | 1.1.1   | 1.1.1k  | buffer overflow…  |
| ...                                                                       |
+--------------------------------------------------------------------------+
[Generate Fix PR]
```

9. Engineering Action Items (Kick-Off)

- **Backend**
- REST endpoints: `/images`, `/images/{id}/vulns`.
- gRPC scan micro-service using Trivy CLI.
- **Database**
- Tables: `images`, `scan_reports`, `vulnerabilities`.
- **Frontend**
- React table component with virtual scroll.
- Severity pill component (critical=red, high=orange…).
- **DevOps**
- Helm chart for deployment.
- GitHub Actions CI → build, test, push container, run Trivy policy gate.

10. Open Questions & Risks

- How frequently should historical scan data be retained?
- Will customers need on-prem deployment?
- Risk: large registries (\~100 k images) may require sharding.