Summary:

Enterprises often deploy hundreds or even thousands of container images, with each image potentially representing a different version of an application or a distinct microservice. For developers, it is essential to identify both known and unknown vulnerabilities before the software is deployed to production. This proactive approach allows developers to address vulnerabilities early, saving time during subsequent security assessments and avoiding the need for ad hoc requests to address issues after deployment and avoid security threats from bad actors. Hence there is a requirement for a feature that will help to scan the container images for known and unknown vulnerabilities.

Target users:

Primary users:

These are the main users who actively use the produt as a part of their core responsibilities.

1. **DevOps Engineers**: They integrate scanning tools into CI/CD pipelines to automate vulnerability checks and ensure secure deployments.
2. **Security Engineers**: They use scanning tools to continuously monitor and assess vulnerabilities, ensuring the overall security posture of containerized applications.
3. **Developers**: They address vulnerabilities in the code and dependencies during the development process to prevent security issues from reaching production.

Secondary users:

These users use container scanning for supporting and oversight activities.

1. **System administrators** : They manage the infrastructure and ensure that container images deployed on their systems are secure
2. **Compliance**: They ensure regulatory and policy compliance, reviewing scan results to confirm that containerized applications meet necessary standards.
3. **IT operations**: They monitor and support the IT infrastructure, ensuring that containerized applications remain secure in production.

Opportunity:

These users may benefit from the product but do not use them directly

1. **Incident response teams**: They could leverage scan results during investigations of security incidents, but their primary focus is on response and remediation rather than scanning.
2. **Product managers**: They may review security reports and results to make decisions about product releases and risk management but are not directly involved in the scanning process.

Use cases aimed to solve:

1. Provide users with an overview of all container images in their repository.
2. Identify and prioritize container images based on the severity of vulnerabilities.
3. Enable users to focus on critical and high-severity vulnerabilities to improve the security of their environments.
4. Offer actionable insights to help users mitigate vulnerabilities quickly.

User stories:

As a user, I want to integrate a container image scanning tool with my system so that it can scan container images for known vulnerabilities and report them for remediation.

1. Assuming I have an account with the security product and the necessary entitlements to use the container image scanning product, I should be able to seamlessly integrate the image scanning tool with my system.
2. Furthermore, I must be able to integrate the scanning tool with the CI/CD pipeline, enabling automatic scanning of images for vulnerabilities during the build process and reporting to the scanning dashboard.
3. Additionally, I should have the capability to utilize relevant APIs to integrate with other systems, allowing image scans to be triggered in response to events such as the deployment or update of a new image.
4. Moreover, the tool must be able to integrate with container orchestration platforms (like Kubernetes) and container registries, enabling it to detect newly deployed or updated images and trigger scans accordingly.
5. Once successfully integrated with my system, the container image scanning tool should download scans for known vulnerabilities and report them to me.

6. The scanning tool must utilize the National Vulnerability Database (NVD) to identify known Common Vulnerabilities and Exposures (CVEs), and it should also include GitHub advisories in its database.

7. Following the identification of vulnerabilities, they should be categorized based on their severity (critical, high, medium, low, and unknown) using the CVSS (Common Vulnerability Scoring System).

8. Upon completion of the scan, the system (scanning tool) should generate a report containing the name of the vulnerable package, the installed version, the version in which the vulnerability is fixed, and the severity of the vulnerability.

As a user, I want a clear and concise view of which images contain vulnerabilities, categorized by severity.

1. The security product should provide a dashboard that offers comprehensive information about vulnerabilities, meeting the user's needs.

2. The dashboard must display a summary of the total number of vulnerabilities, the number of container images, critical vulnerabilities, and high vulnerabilities (Image-01).

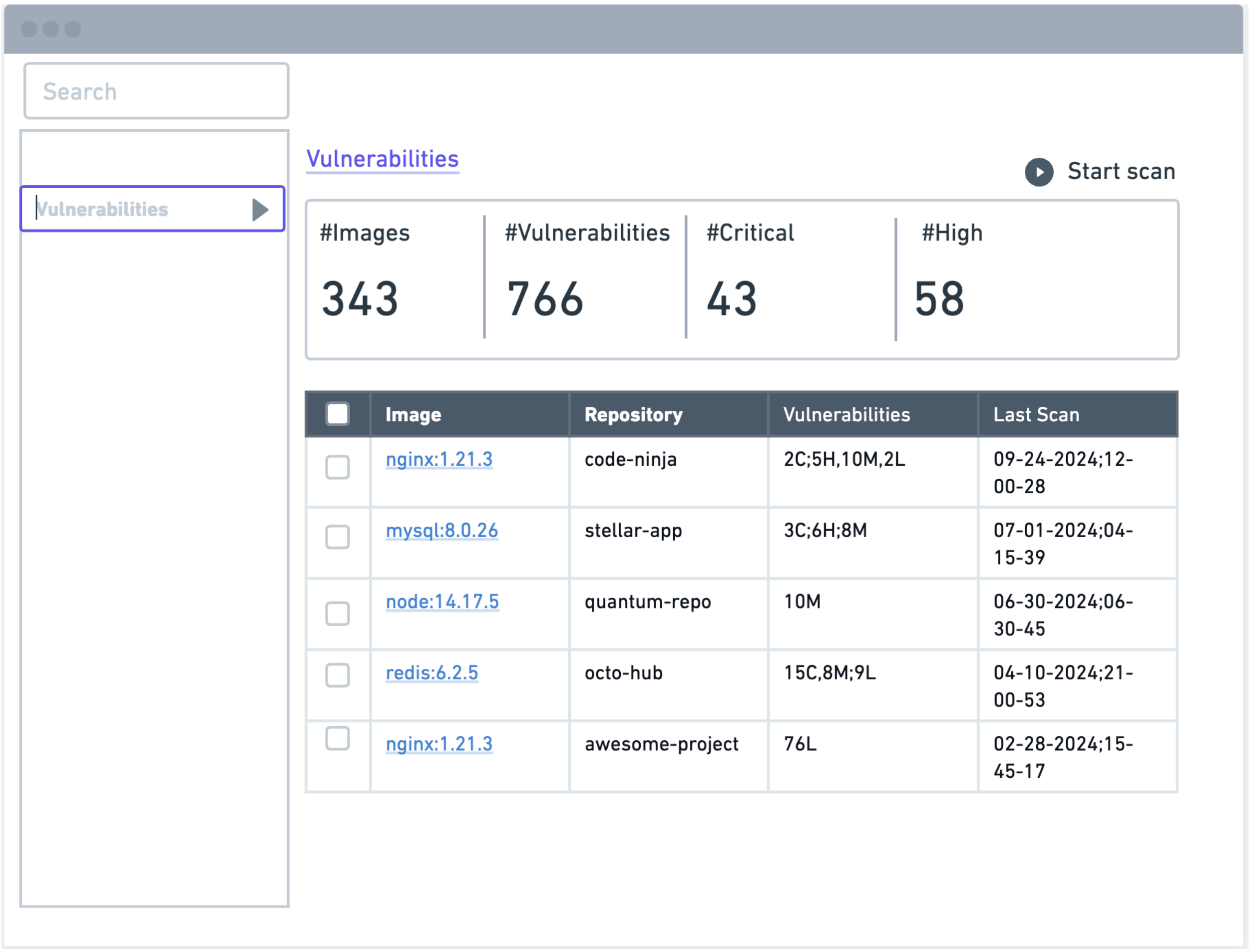
3. The dashboard must show the list of vulnerabilities (Image-01)

    a. Image name

    b. repository

    c. Vulnerabilities (with breakdown by severity)

    d. Last Scan



#Images	#Vulnerabilities	#Critical	#High
343	766	43	58

	Image	Repository	Vulnerabilities	Last Scan
☐	nginx:1.21.3	code-ninja	2C;5H,10M,2L	09-24-2024;12-00-28
☐	mysql:8.0.26	stellar-app	3C;6H;8M	07-01-2024;04-15-39
☐	node:14.17.5	quantum-repo	10M	06-30-2024;06-30-45
☐	redis:6.2.5	octo-hub	15C,8M;9L	04-10-2024;21-00-53
☐	nginx:1.21.3	awesome-project	76L	02-28-2024;15-45-17

(Image-01)

4. I must have the ability to choose the images for rescanning and verifying the resolution of issues from the Vulnerabilities page. After rerunning the scan and resolving the issues, the count in the table on the Vulnerabilities page should be updated. The Vulnerabilities column must display the existing vulnerabilities and not the ones that have been fixed.

5. I must be able to sort the images by scan date or severity.

6. I must be able to deep dive into each image for additional information. I must see the following information when in the additional details section

    a. Severity (Critical, High, Medium, Low)

    b. Score

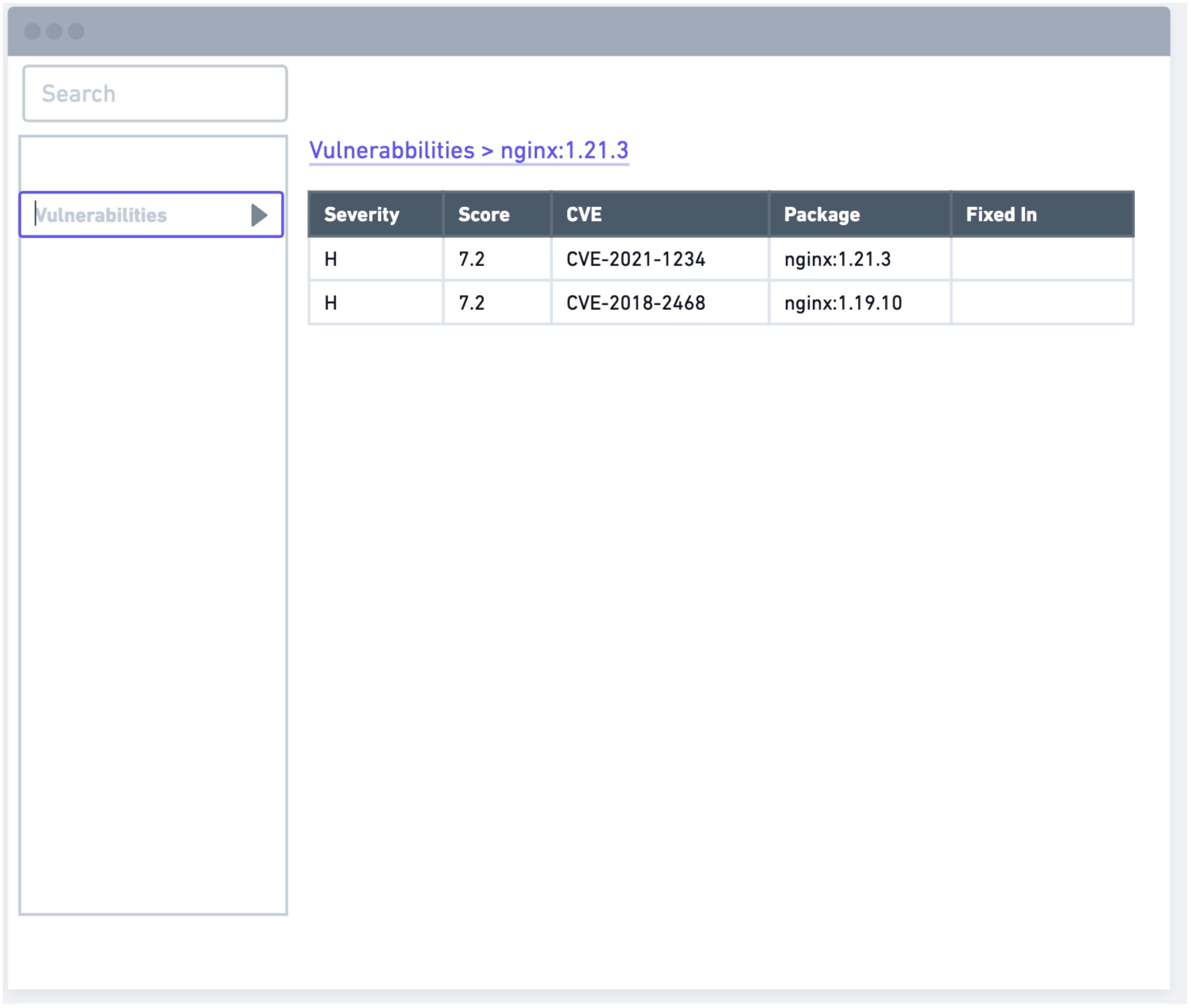
    c. CVE reference

    d. Package/Version

    e. Fixed In (Which version fixed the vulnerabilities)

7. After reviewing the details of each vulnerability and addressing the issues by updating the image with the latest packages, users should be able to rerun the scan from the Vulnerabilities page (Image-01).

8. Once the scan is completed and the issues are resolved, the system should populate the "Fixed In" version column with the package version that addresses the vulnerabilities.



Severity	Score	CVE	Package	Fixed In
H	7.2	CVE-2021-1234	nginx:1.21.3	
H	7.2	CVE-2018-2468	nginx:1.19.10	

9. While the scan is in progress, I must know that the scanning is in progress and I should be able to stop it whenever I need.

As a user, I want to set alerts for vulnerabilities, so I can take immediate action when a new vulnerability is identified by the system while scanning the image.

1. I must be able to set alert conditions in the system when there are any new vulnerabilities found.

2. I must be able to configure the alert condition within the system.

3. I must receive a email notification when the alert condition is triggered
4. The admin must be able to configure batched alerts and real time alerts.
5. The email notification must contain the following information
    a. Image name
    b. Scan date/time
    c. Vulnerability list
        i. Component
        ii. CVE reference
        iii. Severity
        iv. Available fix
6. The admin of the account must be able to set the alert conditions and add users to the list who must receive the email notification.

Development action items (Needs discussion with the development team)

1 Vulnerability Scanning Service

1. **Action**: Implement a service to automatically scan container images when they are added or modified in the repository.
    a. **Dependencies**: Integration with vulnerability databases (e.g., NVD, MITRE) and container registries (e.g., Docker Hub, AWS ECR, Google GCR).
    b. **Considerations**: How frequently should the scan run? Do we need support for custom security databases?

2 API Design

2. **Action**: Develop REST APIs for:
    a. Fetching container images and their vulnerability status.
    b. Triggering scans (manual or automatic).
    c. Managing bulk actions like applying fixes or rescanning multiple images.

    d. Integrating with CI/CD pipelines for build/push triggers.

    e. Exposing scan data for external integrations or custom scripts.

    Questions: Do we need GraphQL or will REST suffice? What pagination strategy should we use to handle large datasets?

3 Prioritization Mechanism

3. **Action**: Implement logic to prioritize and sort images based on the severity of vulnerabilities (e.g., show critical vulnerabilities first).
    - **Considerations**: Should users be able to configure their own priority thresholds (e.g., high only or high + medium)?
    - **Technical Detail**: Can we optimize database queries for sorting/filtering by severity?

4 Dashboard UI

4. **Action**: Build the main dashboard page to list all container images, including filters, sorting, and visual indicators for vulnerability severity.
    a. **UI Components**:
        i. Summary bar for quick vulnerability stats.
        ii. Image table with sortable columns for severity, last scanned date, etc.
    b. **Considerations**:
        i. Infinite scrolling vs. pagination?
        ii. How to efficiently handle thousands of container images without performance degradation?
        iii. Which frontend framework (React, Vue, Angular) will the team use?

5 Bulk Actions

5. **Action**: Implement bulk action features to allow users to rescan or apply fixes to multiple images at once.
    a. **Considerations**: How will the UX handle large selections (e.g., selecting hundreds of images)?
    b. **Technical Questions**: Will the backend support batch operations to handle bulk scans/fixes efficiently?

6 Notification System

6. **Action**: Develop a notification/alert system to inform users about new critical vulnerabilities.
    a. **Frontend**:
        i. Notifications as banners or modals.
        ii. Integration with user-defined thresholds for notifications.
    b. **Backend**: Define criteria to trigger alerts based on real-time scan results.