**Problem Statement 1:**

**Product Requirements (Simple Version)**

**1. Purpose of the Product:**

* Help users scan all their container images.
* Show which ones have security issues (vulnerabilities).
* Allow users to easily identify and fix the most dangerous ones (critical or high).

**2. Key Features:**

1. **Image List Dashboard**
   * Show all container images in a table/list.
   * Show number of vulnerabilities for each image.
   * Show severity levels: Critical, High, Medium, Low.
2. **Filter & Search**
   * Let users search by image name.
   * Let users filter by severity (e.g., show only Critical or High vulnerabilities).
3. **Details Page for Each Image**
   * Show all vulnerabilities for a selected image.
   * Show affected components, severity, and how to fix them (e.g., update version).
4. **Fix Suggestions**
   * Show possible fixes (like upgrading to a safer version).
   * Mark which images need urgent attention.
5. **Status Indicator**
   * Color-coded or status icons to show urgency (e.g., red for Critical).

**Low-Fidelity Wireframes (Sketch Style)**

Let’s imagine some basic wireframes.

**1. Image List Page**

pgsql

------------------------------------------------------------

| Search [\_\_\_\_\_\_\_\_\_\_] Filter: [Critical ▼] |

------------------------------------------------------------

| Image Name | Total Vulns | Critical | High | Fix |

------------------------------------------------------------

| backend:v1.2 | 12 | 3 | 4 | 🔧 |

| frontend:v3.4 | 4 | 0 | 1 | |

| db:v5.1 | 20 | 10 | 5 | 🔧 |

**2. Image Detail Page**

markdown

------------------------------------------------------------

| Image: backend:v1.2 |

| Total Vulns: 12 Critical: 3 High: 4 Medium: 5 |

------------------------------------------------------------

| Vulnerability ID | Severity | Component | Fix Available |

------------------------------------------------------------

| CVE-1234 | Critical | openssl | Yes (update) |

| CVE-5678 | High | libxml | Yes (patch) |

------------------------------------------------------------

[Suggest Fixes Button]

**Bonus: Development Action Items**

1. **Backend**
   * Build an API to scan container images and store scan results.
   * Fetch vulnerability data from sources like NVD or vendor advisories.
2. **Frontend**
   * Build dashboard UI to list all images and vulnerabilities.
   * Implement filters and search.
   * Show vulnerability details per image.
3. **Security Scanning**
   * Integrate scanning tools (e.g., Trivy, Clair, or Snyk) to analyze images.
4. **Fix Suggestions**
   * Recommend safer versions based on CVE data.

**Problem Statement 2:**

**Step-by-Step Guide (Human-friendly):**

**Step 1: Install a Local Kubernetes Cluster**

Use **Minikube** because it's beginner-friendly.

1. **Install Minikube:**
   * On Linux:

bash

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube

* + On macOS:

bash

brew install minikube

1. **Start Minikube:**

bash

minikube start

1. Check if it’s working:

bash

kubectl get nodes

**Step 2: Install a Security Scanning Tool (Kubescape)**

1. **Install Kubescape:**

bash

curl -s https://raw.githubusercontent.com/kubescape/kubescape/master/install.sh | /bin/bash

1. **Verify Installation:**

bash

kubescape version

**Step 3: Run the Scan**

1. Run the scan on the cluster:

bash

kubescape scan framework nsa --format json --output results.json

nsa is one of the available scanning frameworks (you could also use mitre, cis, etc.)

1. This command will scan the cluster and save results in a JSON file called results.json

**📦 Deliverable**

You now have the results.json file — this is what you submit.

**Problem Statement 3 (Technical):**

**Step 1: GoLang App + Docker + GitHub + DockerHub**

**🔹 Write a Simple Go App**

Create a file called main.go:

go

CopyEdit

package main

import (

"fmt"

"net/http"

"time"

)

func handler(w http.ResponseWriter, r \*http.Request) {

currentTime := time.Now().Format("2006-01-02 15:04:05")

fmt.Fprintf(w, "Current Date & Time: %s", currentTime)

}

func main() {

http.HandleFunc("/", handler)

http.ListenAndServe(":8080", nil)

}

**🔹 Dockerfile**

Create a Dockerfile:

Dockerfile

# Use official Go image

FROM golang:1.20

# Create app directory

WORKDIR /app

# Copy the Go app

COPY . .

# Build it

RUN go build -o main .

# Run it

CMD ["./main"]

**🔹 Build & Test Locally**

bash

CopyEdit

docker build -t datetime-app .

docker run -p 8080:8080 datetime-app

Check it by visiting: http://localhost:8080

**🔹 Push to GitHub**

1. Create a repo on GitHub (e.g. datetime-go-app)
2. Push your code:

bash

CopyEdit

git init

git remote add origin https://github.com/your-username/datetime-go-app.git

git add .

git commit -m "Initial commit"

git push -u origin master

**🔹 Push Docker Image to DockerHub**

1. Login:

bash

docker login

1. Tag & push:

bash

docker tag datetime-app yourdockerhubusername/datetime-app

docker push yourdockerhubusername/datetime-app

**Step 2: Deploy with K8s (Declarative)**

**🔹 Create Deployment YAML (deployment.yaml):**

yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: datetime-deployment

spec:

replicas: 2

selector:

matchLabels:

app: datetime

template:

metadata:

labels:

app: datetime

spec:

containers:

- name: datetime

image: yourdockerhubusername/datetime-app

ports:

- containerPort: 8080

**🔹 Apply it:**

bash

kubectl apply -f deployment.yaml

**🔹 Check if running:**

bash

kubectl get pods

**Step 3: Expose to Internet**

**🔹 Create a Service (service.yaml):**

yaml

apiVersion: v1

kind: Service

metadata:

name: datetime-service

spec:

type: LoadBalancer

selector:

app: datetime

ports:

- protocol: TCP

port: 80

targetPort: 8080

**🔹 Apply the Service:**

bash

CopyEdit

kubectl apply -f service.yaml

**🔹 Get External IP:**

bash

kubectl get service datetime-service

Look under the EXTERNAL-IP column — that’s your public IP. Open it in the browser.

**✅ TL;DR (Summary)**

* **Write Go code**
* **Containerize it**
* **Push to GitHub & DockerHub**
* **Deploy to k8s with 2 replicas using YAML**
* **Expose with a LoadBalancer**

**✅ Extra Tip: Use Others' Real Examples**

You can look for these on GitHub by searching:

sql

go webapp current time dockerfile

Or search on Google/YouTube:

vbnet

“Deploy Go app to Kubernetes using Docker” step-by-step tutorial