Solution Document for Product Management & Kubernetes Tasks

Problem Statement 1: Product Requirement and Low-Fidelity Wireframes

🎯 Goal:

Help users understand which container images have vulnerabilities and how severe they are.

📌 Product Requirements Document:

1. Problem:
- Users manage container images with potentially vulnerable dependencies.
- They need a way to detect and fix vulnerabilities efficiently.

2. Users:
- DevOps engineers
- Security analysts
- Developers working with containerized apps

3. User Goals:
- Scan all container images in their repository.
- Identify vulnerabilities with severity levels (e.g., Critical, High, Medium, Low).
- View detailed reports for each image.
- Filter/sort images based on vulnerability severity, image name, etc.

4. Key Features:
- Upload/scan container images
- Dashboard with vulnerability summary
- Filters by severity, date, repository
- Detailed vulnerability report per image
- Fix recommendations or links
- Export reports (PDF, JSON)

5. Metrics:
- Scan completion time
- Number of vulnerabilities detected
- User engagement (reports viewed/exported)

📦 Low-Fidelity Wireframes (Sketch Ideas):

Dashboard View:

```
-------------------------------------
| Vulnerability Summary             |
| - Critical: 10  High: 15          |
| - Medium: 30  Low: 50             |
-------------------------------------
| Filters: [Severity] [Date] [Name] |
-------------------------------------
| Image Name | Vuln Count | Action  |
| ---------- | -----------|-------- |
| app1       | 15         | [View]  |
| app2       | 2          | [View]  |
-------------------------------------
```

Image Details View:

```
-----------------------------------
| Image: app1                     |
| Vulnerabilities:                |
| - CVE-2023-xxxx (Critical)      |
| - CVE-2023-yyyy (Medium)        |
|                                 |
| Fix Instructions:               |
| - Upgrade library xyz to 1.2.3  |
-----------------------------------
```

🧰 Bonus: Development Action Items:

- Select container image scanner (e.g., Trivy, Clair, Anchore)
- Backend API to retrieve vulnerability data
- Frontend components for dashboard and image details
- Auth and access control for multi-user usage

Problem Statement 2: Kubernetes Security Scan

🎯 Goal:

Scan a local K8s cluster for vulnerabilities and output findings in a JSON file.

✅ Steps:

1. Install K8s Cluster:
  - Tools: Minikube, Kind, K3s
  - Example:
   minikube start

2. Run Kubescape:
  - Example:
   kubescape scan --submit=false --format=json > k8s_findings.json

3. Deliverable:
  - A JSON file `k8s_findings.json` containing scan results like CVEs, RBAC issues, workload security gaps, etc.

Problem Statement 3: GoLang App + Docker + K8s

Step #1: GoLang App & Docker

GoLang Program:

```
package main
import (
   "fmt"
   "net/http"
   "time"
)

func handler(w http.ResponseWriter, r *http.Request) {
   fmt.Fprintf(w, "Current date & time: %s", time.Now().Format(time.RFC1123))
}

func main() {
   http.HandleFunc("/", handler)
   http.ListenAndServe(":8080", nil)
}
```

Dockerfile:

```
FROM golang:1.20-alpine
WORKDIR /app
COPY . .
RUN go build -o datetimeapp .
```

```
CMD ["./datetimeapp"]
```

Push to DockerHub:

```
docker build -t yourdockerhub/clock-app .
docker push yourdockerhub/clock-app
```

Step #2: Deploy to K8s with 2 Replicas

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: datetime-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: datetime
  template:
    metadata:
      labels:
        app: datetime
    spec:
      containers:
      - name: datetime
        image: yourdockerhub/clock-app
        ports:
        - containerPort: 8080
```

Step #3: Expose to WAN

service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: datetime-service
spec:
  type: LoadBalancer
  selector:
    app: datetime
  ports:
    - protocol: TCP
```

port: 80
targetPort: 8080

Resources:

- https://www.qwiklabs.com/
- https://labs.play-with-k8s.com