1. Pod:

(YAML)

apiVersion: v1

kind: Pod

metadata:

name: vulnerability-scanner

labels:

app: vulnerability-scanner

spec:

containers:

- name: vulnerability-scanner-container

image: your-vulnerability-scanner-image

command: ["/bin/sh"]

args:

- -c

- |

# Install and run vulnerability scanning tools

apt-get update && apt-get install -y tool1 tool2 tool3

tool1 scan > tool1\_report.txt

tool2 scan > tool2\_report.txt

tool3 scan > tool3\_report.txt

# Collect scan reports into one log

cat tool1\_report.txt tool2\_report.txt tool3\_report.txt > scan\_report.log

# Encrypt and securely store the log (adjust as per your setup)

gpg --recipient your\_key\_id --output encrypted\_report.gpg --encrypt scan\_report.log

volumeMounts:

- name: report-storage

mountPath: /reports

volumes:

- name: report-storage

emptyDir: {}

1. InitContainers:

InitContainers are executed before the main containers in a pod start. You can use them to run vulnerability scanning tools before your application containers start. This approach ensures that the scanning is performed before any services become active.

1. CronJobs:

If you want to perform regular vulnerability scans, you can create a CronJob resource. A CronJob allows you to schedule jobs to run periodically. You can define the scanning process and save the results securely after each scan.

(YAML)

apiVersion: batch/v1beta1

kind: CronJob

metadata:

name: vulnerability-scanner-cron

spec:

schedule: "0 0 \* \* \*" # Run daily at midnight

jobTemplate:

spec:

template:

spec:

containers:

- name: vulnerability-scanner-container

image: your-vulnerability-scanner-image

command: ["/bin/sh"]

args:

- -c

- |

# Install and run vulnerability scanning tools

apt-get update && apt-get install -y tool1 tool2 tool3

tool1 scan > tool1\_report.txt

tool2 scan > tool2\_report.txt

tool3 scan > tool3\_report.txt

# Collect scan reports into one log

cat tool1\_report.txt tool2\_report.txt tool3\_report.txt > scan\_report.log

# Encrypt and securely store the log (adjust as per your setup)

gpg --recipient your\_key\_id --output encrypted\_report.gpg --encrypt scan\_report.log

volumes:

- name: report-storage

emptyDir: {}

volumeMounts:

- name: report-storage

mountPath: /reports

1. Docker Image:

This image can be deployed as a separate container within your Kubernetes cluster, and you can run it manually or schedule it to perform vulnerability scans as needed.

1. Create a Docker Image:

# Use a base image with the required tools pre-installed

FROM ubuntu:latest

# Install necessary packages

RUN apt-get update && apt-get install -y tool1 tool2 tool3 gnupg

# Copy your vulnerability scanning scripts to the image

COPY scan.sh /usr/local/bin/

RUN chmod +x /usr/local/bin/scan.sh

# Set the entry point to your scanning script

ENTRYPOINT ["/usr/local/bin/scan.sh"]

1. Build the Docker Image:

Navigate to the directory containing your Dockerfile and scanning script, and run the following command to build the image:

-docker build -t vulnerability-scanner-image .

1. Push the Image:

Push the built Docker image to a container registry, such as Docker Hub or a private registry that your Kubernetes cluster has access to.

-docker push vulnerability-scanner-image

1. Deploy the Image as a Job:

Once the image is available in a registry, you can create and run a Kubernetes Job to perform the vulnerability scan. The Job definition might look like this:

apiVersion: batch/v1

kind: Job

metadata:

name: vulnerability-scan-job

spec:

template:

spec:

containers:

- name: vulnerability-scanner-container

image: vulnerability-scanner-image

restartPolicy: Never

The job will run the vulnerability scanner image, execute the scanning script within the container, and generate the scan report as per your script's logic.

Using a custom Docker image for vulnerability scanning allows you to encapsulate the scanning tools and scripts, making it more portable and manageable.