Container-Image-Signing-and-Verification-System

This project focuses on **securely building, signing, and verifying a Docker image** that contains a **simple Node.js application**.

In this project:

- We are creating a **Node.js** based application.
- We are **dockerizing** the app by writing a Dockerfile.
- After building the image, we **digitally sign** it using **Cosign**.
- Finally, we **verify** the image to ensure its integrity and authenticity.

Commands and Setup

1. Prerequisites

- **Docker Desktop**: Installed and running.
- **Docker Hub Account**: For pushing images.
- **kubectl**: Installed (comes with Docker Desktop).
- **Basic CLI Knowledge**: Familiarity with terminal commands.

2. Build and Sign a Docker Image

2.1 Create DockerFile

- We created dockerfile in vs code as below written command which has necessary command for building base image.

  Command in dockerfile:

  ```
  FROM node:16
  WORKDIR /app
  COPY package*.json ./
  RUN npm install
  COPY . .
  CMD ["node","index.js"
  ```

- This Dockerfile sets up a simple Node.js application inside a Docker container. It installs all dependencies and runs the app using node index.js inside a Node 16 environment.

2.2Create Sample Node.js Files and package.json file

- We write simple JS code for NODE.js application in index.js file aand package.json file for starting this application as below:

Code for index.js :

```
const http=require("http");

const server=http.createServer((req,res)=>{
   res.writeHead(200,{'Content-Type':'text/plain'});
   res.end("Hello from Node.js \n");
});

const PORT=3000;
server.listen(PORT,()=>{
   console.log(`Server running on port ${PORT}`);
})
```

Code for package.json :

```
{
   "name": "node-app",
   "version":"1.0.0",
   "description": "A simple NOde.js App",
   "main": "index.js",
   "scripts": {
      "start": "node index.js"
   }
}
```

2.3 Build the Docker Image

- Create a Docker image named node-image:v1 using below command.

  docker build -t node-image:v1**.**

2.4 Install Cosign and Generate Keys

- Install Cosign

  curl -sSfL https://raw.githubusercontent.com/sigstore/cosign/main/install.sh | sh

  sudo mv cosign /usr/local/bin

- Generate Key Pair

  cosign generate-key-pair

- **Output :**

- cosign.key (private key, keep secure!).
- cosign.pub (public key, used for verification).

2.5 Sign the Image

- **cosign sign --key cosign.key node-image:v1**

- **Output :**
- Signing the image will attach a cryptographic signature to the image.

3.Push Image to Docker Hub

3.1 Tag and Push the Image

- Log in to Docker Hub
  docker login

- Tag the image

docker tag my-safe-image:v1 <DOCKERHUB_USERNAME>// node-image:v1

- Push to Docker Hub
  docker push <DOCKERHUB_USERNAME>//node-image:v1

- **Output :**
  One can see the pushed image from docker hub.

4.Verify the signed Image

- **cosign verify --key cosign.pub <DOCKERHUB_USERNAME>//node-image:v1**

- **Output :**
- Using the public key cosign.pub, verify that the Docker image sanket8933/node-image:v1 was signed by the correct private key and has not been tampered with."

5. Set Up Kubernetes and Kyverno

5.1Enable Kubernetes in Docker Desktop

- Open Docker Desktop → **Settings** → **Kubernetes** → **Enable Kubernetes** → **Apply**.

5.2Install Kyverno

- Installs Kyverno, a policy engine for Kubernetes.
  kubectl create -f https://github.com/kyverno/kyverno/releases/latest/download/install.yaml

5.3List all Pods running inside kyverno

- This command lists all the Kyverno pods running inside the kyverno namespace to verify if Kyverno is properly installed and active.
  kubectl get pods -n kyverno

5.4Get Public Key

- This command displays the contents of the cosign.pub public key file used for verifying Docker image signatures.
  type cosign.pub

- **Output :**
  It will give public key , save this and paste it to image-policy file.

6. Create Signature Verification Policy

6.1 create image-policy.yaml write following code.

Code for image-policy.yaml :

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: check-image-signature
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    - name: verify-image
      match:
        any:
          - resources:
              kinds:
                - Pod
      verifyImages:
        - image: "*/*"
          key: |-
            -----BEGIN PUBLIC KEY-----
```

```
//paste your public key here.
-----END PUBLIC KEY-----
```

6.2 Apply the policy

- This command applies the image-policy.yaml file to the Kubernetes cluster to enforce security policies on container images.
  kubectl apply -f image-policy.yaml

7. Test the workflow

7.1 Deploy a Signed Image

- This command creates and runs a new Kubernetes pod named safe-pod using the signed Docker image you uploaded to Docker Hub.
  kubectl run safe-pod --image=<DOCKERHUB_USERNAME>/node-image:v1

7.2 verify

- This command retrieves the list of pods in the current Kubernetes namespace, displaying their status, including whether they are "Running" and we run signed image on safe pod so safe-pod's status should be running.
  kubectl get pods

7.3 Deploy an Unsigned Image

- This command creates a new pod named unsafe-pod using the nginx:latest Docker image, running the NGINX web server.
  kubectl run unsafe-pod --image=nginx:latest

as this is not signed images so we should get error like this :

```
Error: admission webhook "validate.kyverno.svc" denied the request:
image signature verification failed for nginx:latest
```