Proof of Concept (POC)

Docker Content Trust

By Som Kartikey (Cloud and DevOps) August 2024

OVERVIEW

The objective of this PoC is to showcase how Docker Content Trust (DCT) enhances container security by allowing only signed and verified Docker images to be deployed. By leveraging digital signatures through Notary, DCT ensures that images are authenticated and have not been tampered with, protecting against threats like image spoofing and unauthorized modifications.

This PoC demonstrates the practical steps to enable DCT, manage signing keys, and integrate it into CI/CD pipelines, emphasizing its effectiveness in creating a secure Docker image supply chain, reducing the risk of supply chain attacks, and reinforcing overall container security practices within an organization.

Problem Statement

The primary problem Docker Content Trust (DCT) aims to solve is ensuring the integrity and authenticity of Docker images throughout the software development lifecycle, from development to production deployment.

Problem Breakdown:

- **Image Integrity**: Docker Images can be tampered with by unauthorized parties, either during transit or while storing the docker image in the repository. Ensuring that the image built by the developer is the same as received by the user is important in maintaining reliability and security.

- **Image Authenticity:** It's essential to verify that Docker images come from trusted sources. Without this verification, there is a risk of pulling and running images that might be malicious, vulnerabilities, or other security threats.

- **Trust in CI/CD Pipelines:** In automated CI/CD pipelines, images are often built, pushed, and pulled without manual intervention. This automation can be exploited if there’s no mechanism to enforce that only trusted and verified images are being used, leading to potential security breaches.

- **Compliance Requirements:** Many industries have strict compliance and regulatory requirements that mandate secure software supply chains. Ensuring that only signed and verified images are used is often a key component of meeting these requirements.

Objective and Scope

The primary objective of Docker Content Trust (DCT) is to enhance the security of Docker images by ensuring that only trusted and verified images are used throughout the software development and deployment process. DCT aims to protect against the risks of using compromised or tampered images by providing a mechanism for image signing and verification, ensuring that the images deployed in production are authentic and have not been altered. It provides following features:

- **Guarantee Image Integrity**
- **Ensure Image Authenticity**
- **Insure Secure CI/CD Pipelines**
- **Meet Compliance Needs**
- **Integration with Docker Ecosystem**
- **Key Management (Cryptographic hash key generation)**
- **User and Access Control (Organization can add or remove key signers)**
- **Further Integration with CI/CD Pipelines**

Working

Docker Content Trust (DCT) ensures the security and integrity of Docker images by implementing cryptographic signing and verification mechanisms. When a Docker image is built, it can be signed using a private key, creating a digital signature that uniquely identifies it. When the image is pulled or deployed, DCT checks this signature against a trusted key to confirm the image's authenticity and integrity, ensuring it hasn't been tampered with or altered. By enforcing these trust policies, DCT prevents unauthorized or malicious images from running, secures the software supply chain, supports compliance requirements, and integrates smoothly with existing CI/CD pipelines and Docker registries.

Following are the steps to do so:

- Build an Image using docker build command from a docker file.
- CMD: $docker build .

```
 => => sha256:d58137ded569a53fbc27bc227ab40807d2306033d1b7eed800f2b2f0c8c083bf 6.38kB / 6.38kB                    0.0s
 => => sha256:c6a83fedfae6ed8a4f5f7cbb6a7b6f1c1ec3d86fea8cb9e5ba2e5e6673fde9f6 3.62MB / 3.62MB                    1.6s
 => => sha256:731f5f878c76e7151b9c66968cbb701e7e815fd48718033cf20ba30a2317cff1 42.31MB / 42.31MB                  6.3s
 => => sha256:17ea4cb95273cd126d8bd46096946efa3d2bad82680b0c838ead8400a98c055a 1.39MB / 1.39MB                    1.8s
 => => extracting sha256:c6a83fedfae6ed8a4f5f7cbb6a7b6f1c1ec3d86fea8cb9e5ba2e5e6673fde9f6                         0.4s
 => => extracting sha256:731f5f878c76e7151b9c66968cbb701e7e815fd48718033cf20ba30a2317cff1                         1.8s
 => => extracting sha256:17ea4cb95273cd126d8bd46096946efa3d2bad82680b0c838ead8400a98c055a                         0.1s
 => => extracting sha256:2ba186860f733cc6abe783afaec3d55af4852d15cafc3f030544609c22d6e217                         0.0s
 => [internal] load build context                                                                                 0.5s
 => => transferring context: 4.59MB                                                                               0.3s
 => [3/4] COPY . .                                                                                                0.4s
 => [4/4] RUN yarn install --production                                                                          32.8s
 => exporting to image                                                                                            1.5s
 => => exporting layers                                                                                           1.4s
 => => writing image sha256:8dce551efeccb3cd8e32ef981f05d6fe09e27e4cbc151d267955390cc4aa9fbf                      0.1s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/w56hh912y11izbay5bzfq55if

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
```

```
PS C:\Users\skartikey\Downloads\Docker\app> cd app
PS C:\Users\skartikey\Downloads\Docker\app\app> docker build .
[+] Building 64.6s (10/10) FINISHED                                                              docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                          0.5s
 => => transferring dockerfile: 138B                                                                          0.0s
 => [internal] load metadata for docker.io/library/node:20-alpine                                             3.4s
 => [auth] library/node:pull token for registry-1.docker.io                                                   0.0s
 => [internal] load .dockerignore                                                                             0.2s
 => => transferring context: 2B                                                                               0.0s
 => [1/4] FROM docker.io/library/node:20-alpine@sha256:1a526b97cace6b4006256570efa1a29cd1fe4b96a5301f8d48e87c5139438a45   9.9s
 => => resolve docker.io/library/node:20-alpine@sha256:1a526b97cace6b4006256570efa1a29cd1fe4b96a5301f8d48e87c5139438a45   0.1s
 => => sha256:1a526b97cace6b4006256570efa1a29cd1fe4b96a5301f8d48e87c5139438a45 7.67kB / 7.67kB                0.0s
 => => sha256:2cc3d19887bfea8bf52574716d5f16d4668e35158de866099711ddfb2b16b6e0 1.72kB / 1.72kB                0.0s
 => => sha256:d58137ded569a53fbc27bc227ab40807d2306033d1b7eed800f2b2f0c8c083bf 6.38kB / 6.38kB                0.0s
 => => sha256:c6a83fedfae6ed8a4f5f7cbb6a7b6f1c1ec3d86fea8cb9e5ba2e5e6673fde9f6 3.62MB / 3.62MB                1.6s
 => => sha256:731f5f878c76e7151b9c66968cbb701e7e815fd48718033cf20ba30a2317cff1 42.31MB / 42.31MB              6.3s
 => => sha256:17ea4cb95273cd126d8bd46096946efa3d2bad82680b0c838ead8400a98c055a 1.39MB / 1.39MB                1.8s
 => => extracting sha256:c6a83fedfae6ed8a4f5f7cbb6a7b6f1c1ec3d86fea8cb9e5ba2e5e6673fde9f6                     0.4s
 => => extracting sha256:731f5f878c76e7151b9c66968cbb701e7e815fd48718033cf20ba30a2317cff1                     1.8s
 => => extracting sha256:17ea4cb95273cd126d8bd46096946efa3d2bad82680b0c838ead8400a98c055a                     0.1s
```

- Generate Trust keys Using the following command:
- CMD: $ docker trust key generate larry.
  $ “ls” list to check whether the key is generated or not.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\skartikey\Downloads\Docker\app\app> docker trust key generate larry
```

```
PS C:\Users\skartikey\Downloads\Docker> ls


    Directory: C:\Users\skartikey\Downloads\Docker


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         8/29/2024  12:33 PM                app
-a----         8/29/2024  12:47 PM            191 larry.pub
```

- Add the user/yourself as the signer of the key.
- CMD: $docker trust signer add - -key larry.pub somkartikey somkartikey/dockercontenttext
- Enter the Passphrase.

```
PS C:\Users\skartikey\Downloads\Docker> docker trust signer add --key larry.pub somkartikey somkartikey/dockercontenttext
Adding signer "somkartikey" to somkartikey/dockercontenttext...
Initializing signed repository for somkartikey/dockercontenttext...
You are about to create a new root signing key passphrase. This passphrase
will be used to protect the most sensitive key in your signing system. Please
choose a long, complex passphrase and be careful to keep the password and the
key file itself secure and backed up. It is highly recommended that you use a
password manager to generate the passphrase and keep it safe. There will be no
way to recover this key. You can find the key in your config directory.
Enter passphrase for new root key with ID 7d28660:
Repeat passphrase for new root key with ID 7d28660:
Enter passphrase for new repository key with ID d886ec9:
Repeat passphrase for new repository key with ID d886ec9:
Successfully initialized "somkartikey/dockercontenttext"
Successfully added signer: somkartikey to somkartikey/dockercontenttext
```

- Enable the Docker Content Trust.
- CMD: set DOCKER_CONTENT_TRUST=1(For Windows)
- export DOCKER_CONTENT_TRUST=1(For Linux)

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\skartikey\Downloads\Docker> set DOCKER_CONTENT_TRUST=1
PS C:\Users\skartikey\Downloads\Docker>
```

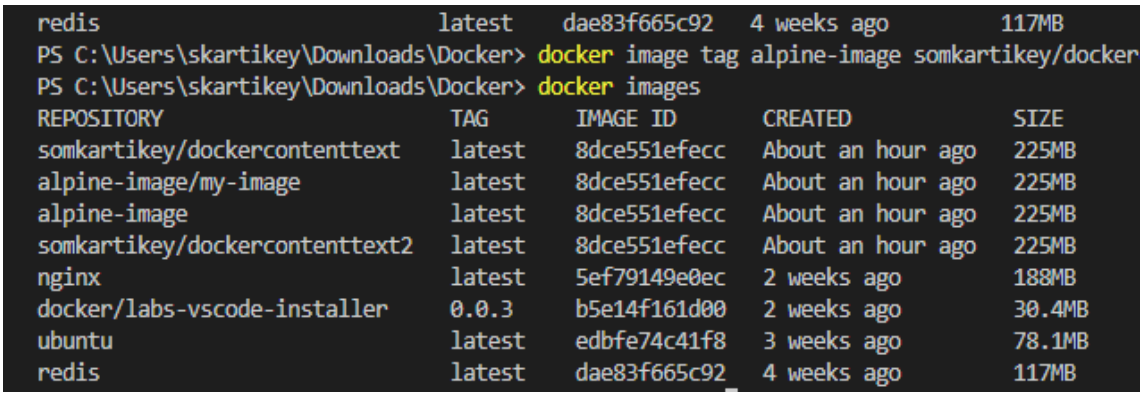
- Sign the Image using the keys.
- CMD: docker trust sign somkartikey/myalpine:latest(somkartikey/myalpine is the tagged alpine image)
- Enter the Passphrase to verify.

```
Enter passphrase for root key with ID 7d28660:
Enter passphrase for new repository key with ID 846bee4:
Repeat passphrase for new repository key with ID 846bee4:
Enter passphrase for new somkartikey key with ID a7f2257:
Repeat passphrase for new somkartikey key with ID a7f2257:
Passphrases do not match. Please retry.
Enter passphrase for new somkartikey key with ID a7f2257:
Repeat passphrase for new somkartikey key with ID a7f2257:
Created signer: somkartikey
Finished initializing signed repository for somkartikey/dockercontenttext2:latest
Signing and pushing trust data for local image somkartikey/dockercontenttext2:latest, may overwrite remote trust data
```

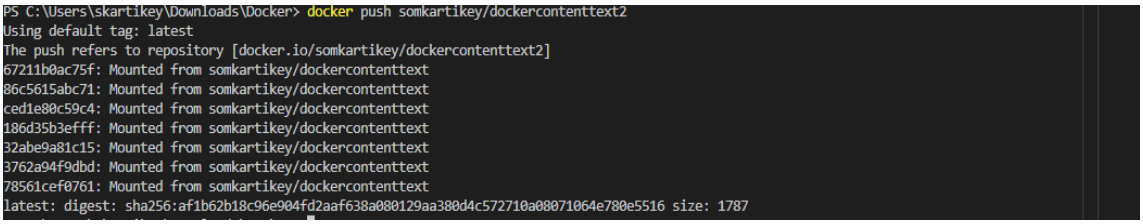
```
The push refers to repository [docker.io/somkartikey/dockercontenttext2]
67211b0ac75f: Layer already exists
86c5615abc71: Layer already exists
ced1e80c59c4: Layer already exists
186d35b3efff: Layer already exists
32abe9a81c15: Layer already exists
3762a94f9dbd: Layer already exists
78561cef0761: Layer already exists
latest: digest: sha256:af1b62b18c96e904fd2aaf638a080129aa380d4c572710a08071064e780e5516 size: 1787
Signing and pushing trust metadata
Enter passphrase for somkartikey key with ID a7f2257:
Successfully signed docker.io/somkartikey/dockercontenttext2:latest
```

- Tag the Image using the command ‘tag’ option.

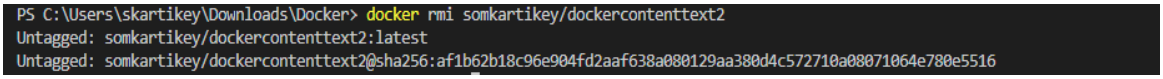
- CMD: docker image tag alpine-image somkartikey/dockercontenttext(here alpine-image is the image name and somkartikey/docker is the repository name)
- List the images the command docker images.



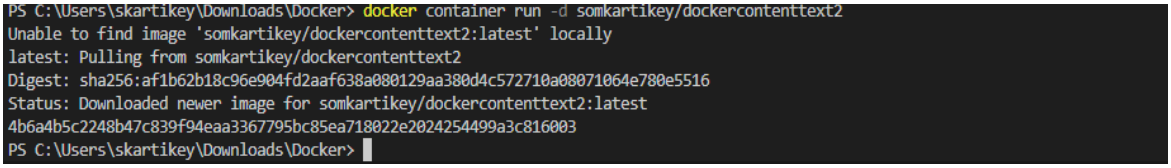
- Push the image to the repository.
- CMD: docker push somkartikey/dockercontenttext2(repository name that has been tagged to an image).



- Remove the image from the local machine so that we can pull it from the repository.
- CMD: docker rmi somkartikey/dockercontnettext



- Pull the image from the repository.
- CMD: docker container run –d somkartikey/dockercontenttext



Automation Using DCT

It is very common for Docker Content Trust to be built into existing automation systems. To allow tools to wrap Docker and push trusted content, there are environment variables that can be passed through to the client.

Working

- **SET DCT REPOSITORY PASSPHARSE**
- Export DOCKER_CONTENT_TRUST_REPOITORY_PASSPHRASE=”abcdxyz”

```
root@ip-172-31-14-9:/home/ubuntu# export DOCKER_CONTENT_TRUST_REPOSITORY_PASSPHRASE="Portal@23"
```

- **CREATING A PRIVATE KEY AND PUBLIC KEY USING OPEN SSL**
- CMD: openssl genrsa –out key.pem 1024 (private key creation)
  openssl rsa -in key.pem -pubout -out pubkey.pem(public key creation)

- **CREATING A DELEGATION ‘CRT’ ‘KEY’ ‘CSR’ FILE USING OPEN SSL**
- CMD: openssl genrsa -out delegation.key 2048
  openssl req -new -sha256 -key delegation.key -out delegation.csr
  openssl x509 -req -sha256 -days 365 -in delegation.csr -signkey delegation.key
  delegation.crt

```
root@ip-172-31-14-9: /home/ubuntu
root@ip-172-31-14-9:/home/ubuntu# ls
Dockerfile  delegation.crt  delegation.csr  delegation.key  example-voting-app  localhost_dct  mykey.pem  mypublickey.pem  n
root@ip-172-31-14-9:/home/ubuntu#
```

- **LOADING AND ADDING SIGNER**

- CMD: docker trust load delegation.key --name somkartikey(name of the trust key / name of the .pub file)
  docker trust signer add –key delegation.crt somkartikey somkartikey/dct(somkartikey is the signer and somkartikey/dct is the repository name).

```
Dockerfile  delegation.crt  delegation.csr  delegation.key  example-voting-app  localhost_dct  mykey.pem  mypu
root@ip-172-31-14-9:/home/ubuntu# docker trust key load delegation.key --name somkartikey
Loading key from "delegation.key"...
Successfully imported key from delegation.key
root@ip-172-31-14-9:/home/ubuntu# export DOCKER_CONTENT_TRUST_ROOT_PASSPHRASE="Portal@23"
root@ip-172-31-14-9:/home/ubuntu# docker trust signer add --key delegation.crt somkartikey somkartikey/dct
Adding signer "somkartikey" to somkartikey/dct...
Successfully added signer: somkartikey to somkartikey/dct

root@ip-172-31-14-9:/home/ubuntu#
```

- **SIGN AN IMAGE**
- CMD: docker trust sign somkartikey/mynginx:latest

```
root@ip-172-31-14-9:/home/ubuntu# docker images
REPOSITORY              TAG             IMAGE ID       CREATED          SIZE
notary-server           latest          7b1c12050e72   15 minutes ago   1.03GB
notary-signer           latest          4f187512d589   16 minutes ago   1.02GB
<none>                  <none>          5439669008d3   7 hours ago      1.03GB
<none>                  <none>          d6412a00f9e9   7 hours ago      1.02GB
somkartikey/mynginx2    latest          5ef79149e0ec   2 weeks ago      188MB
nginx                   latest          5ef79149e0ec   2 weeks ago      188MB
somkartikey/mynginx     latest          0d3cc6e1bac7   2 weeks ago      188MB
alpine                  latest          324bc02ae123   5 weeks ago      7.8MB
somkartikey/myalpine    latest          324bc02ae123   5 weeks ago      7.8MB
mariadb                 10.4            db1c64954e0a   2 months ago     387MB
registry                latest          cfb4d9904335   11 months ago    25.4MB
golang                  1.17.13-alpine  270c4f58750f   2 years ago      314MB
root@ip-172-31-14-9:/home/ubuntu# docker trust sign somkartikey/mynginx:latest
Created signer: somkartikey
Finished initializing signed repository for somkartikey/mynginx:latest
Signing and pushing trust data for local image somkartikey/mynginx:latest, may overwrite remote trust data
The push refers to repository [docker.io/somkartikey/mynginx]
5f0272c6e96d: Layer already exists
f4f00eaedec7: Layer already exists
55e54df86207: Layer already exists
ec1a2ca4ac87: Layer already exists
8b87c0c66524: Layer already exists
72db5db515fd: Layer already exists
9853575bc4f9: Layer already exists
latest: digest: sha256:66e78fcb91a13bcff13c862e6a65fe49ff4c4e55eedf35132d4173793e1b225b size: 1778
Signing and pushing trust metadata
Successfully signed docker.io/somkartikey/mynginx:latest
```

- **BUILD WITH CONTENT TRUST**

- CMD: touch Dockerfile (create's Dockerfile)
  Nano Dockerfile (to open the docker file in editor)
  FROM somkartikey/mynginx(FROM specifies the image)
  CMD "echo"
  docker build .(building an image, you can specify –t as a tag to name your images)


```
root@ip-172-31-14-9:/home/ubuntu# docker build -t somkartikey/mynginx:latest .
[+] Building 0.1s (5/5) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 73B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/somkartikey/mynginx:latest
 => CACHED [1/1] FROM docker.io/somkartikey/mynginx
 => exporting to image
 => => exporting layers
 => => writing image sha256:774a76515d93a21c7a922c725c37c24210162aebf17dc01241343eb5a744ea50
 => => naming to docker.io/somkartikey/mynginx:latest
root@ip-172-31-14-9:/home/ubuntu#
```

CONCLUSION

Docker Content Trust (DCT) is vital for ensuring the integrity and authenticity of Docker images by using digital signatures to verify that images are from trusted sources and have not been tampered with.

For organizations, understanding and implementing DCT is crucial to secure their container supply chain, especially as containerization becomes more widespread. DCT helps prevent security threats like image spoofing and unauthorized deployments, significantly reducing the risk of supply chain attacks.

Implementing DCT requires a strategic approach, including proper key management, defining trusted signers, integrating DCT into CI/CD pipelines, and maintaining secure communication with the Notary server.

Organizations must assess their current security posture and align their policies to enforce DCT effectively across the development lifecycle. By adopting Docker Content Trust, companies not only safeguard their software delivery pipeline but also enhance compliance and build trust with their customers.

DCT thus plays a foundational role in an organization's broader security strategy, ensuring a reliable and secure deployment of containerized applications.

REFERENCE

1. Docker-Content-Trust Manuel (***https://docs.docker.com/engine/security/trust/***)
2. DCT Automation (***https://docs.docker.com/engine/security/trust/trust_automation/***).
3. Notary (***https://github.com/notaryproject/notary***)
4. The Update Framework (***https://github.com/theupdateframework***).
5. OpenSSL (https://docs.openssl.org/master/man7/ossl-guide-libcrypto-introduction/#fetching-examples).