AWS re:Invent

NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

CON316

Container image signing on AWS

Niaz Khan (he/him)

General Manager, AWS Signer
AWS

Michael Brown (he/him)

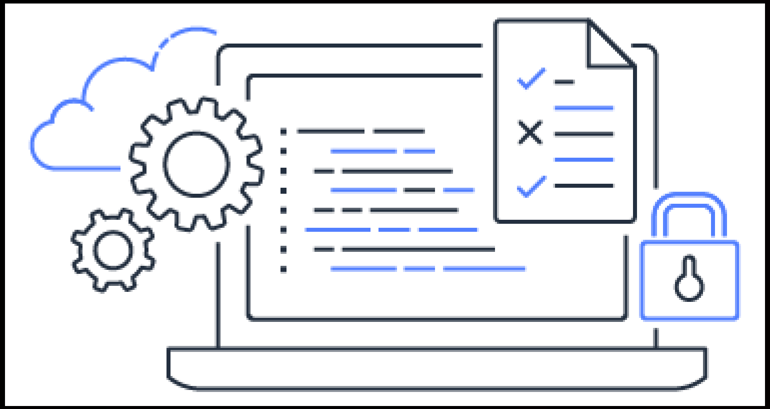
Sr. Software Development Engineer, Amazon ECR
AWS

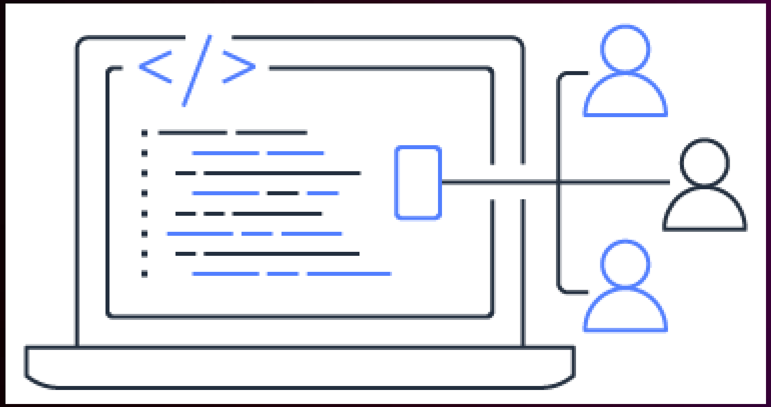
Agenda

- Introduction to image signing
- Planning for image signing
- Image signing tooling
- Adopting and using image signing

Introduction to image signing

What problems does image signing solve?





- Controlling who gets to run what in which environment, using enforceable policies that are evaluated against software artifacts in addition to roles

Existing mechanisms – Network and AWS Identity and Access Management (IAM) policies

Restrict traffic to within the VPC and use VPC endpoints

Use a proxy to restrict requests to a specific list of domains

IAM policies can deny pulls from untrusted Amazon ECR registries

Existing solutions have gaps, and none protect from a trusted registry being compromised

Existing challenges – Notary v1

Additional infrastructure needs to store signatures

Trust on first use weakens trust model

Tagging requires signing as tags included in signature

Goals for Notary v2

No new infrastructure; native registry support

Open source tooling

Flexible trust policy

Planning for image signing

Signing identities

Who or what is making attestations?

- Reviewers – “I reviewed this image for code correctness”
- Automated approval workflows – “I checked this image for licenses”
- Organizations – “I published this image”

Approval workflows

Multiple signatures can attest to specific requirements

- Build system – “I built this image”
- Vulnerability scanner – ”I scanned this image”
- Bill of materials – “I know what’s in this image”
- Integration tests – “I tested this image”

Verification workflows

What attestations are needed for deployment?

- "Who reviewed this image?"
- "What systems ran checks?"
- "Who published this image?"

What do I do if a check is expired?

- Fail open or fail closed

Image signing tooling

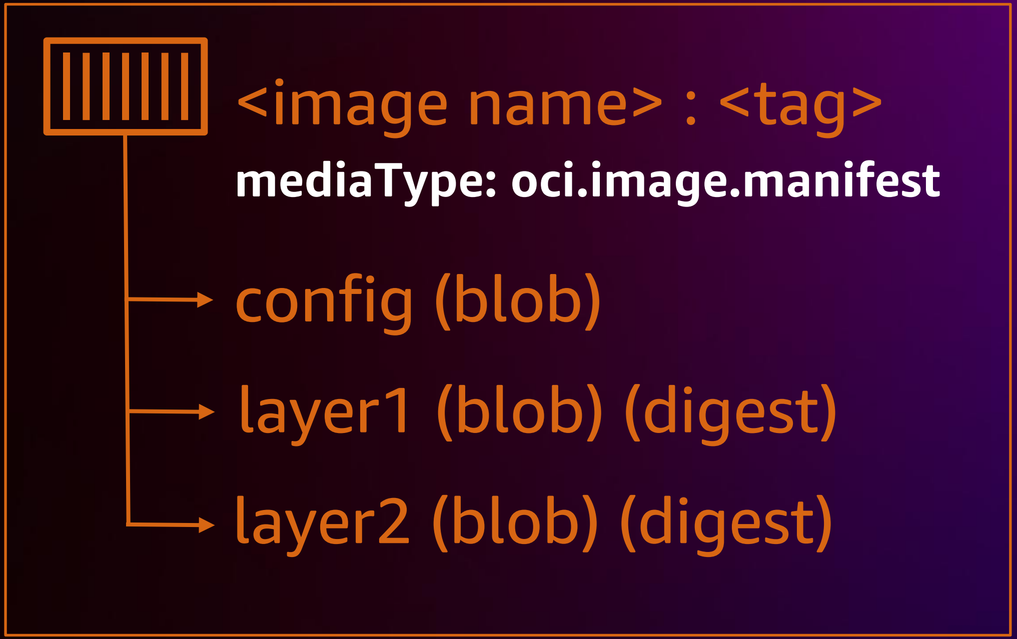
Reference types – What is an OCI image?

Images are identified by the digest of their contents (content-addressable storage)

A manifest is a JSON file that describes the image and its layers, identified by the SHA256 digest

Layers, similarly, are identified by the digest of each layer's bytes

Changing anything in the layer or manifest changes the digest, creating an entirely new object



Reference types – Example image manifest

Example manifest:

```
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.oci.image.manifest.v1+json",
  "config": {
    "mediaType": "application/vnd.oci.image.config.v1+json",
    "size": 7023,
    "digest": "sha256:b5b2b2c507a0944348e0303114d8d93aaaa081732b86451d9bce1f432a537bc7"
  },
  "layers": [
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "size": 32654,
      ”digest": "sha256:9834876dcfb05cb167a5c24953eba58c4ac89b1adf57f28f2f9d09af107ee8f0"
    },
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "size": 16724,
      "digest": "sha256:3c3a4604a545cdc127456d94e421cd355bca5b528f4a9c1905b15da2eb4a4c6b"
    }
  ],
  "annotations": {...}
}
```

Reference types – Working group

OCI reference types working group was formed at the beginning of the year

Representatives from cloud providers, tooling maintainers, and container users collaborated to extend the existing OCI image and distribution specs to support the new reference types use case

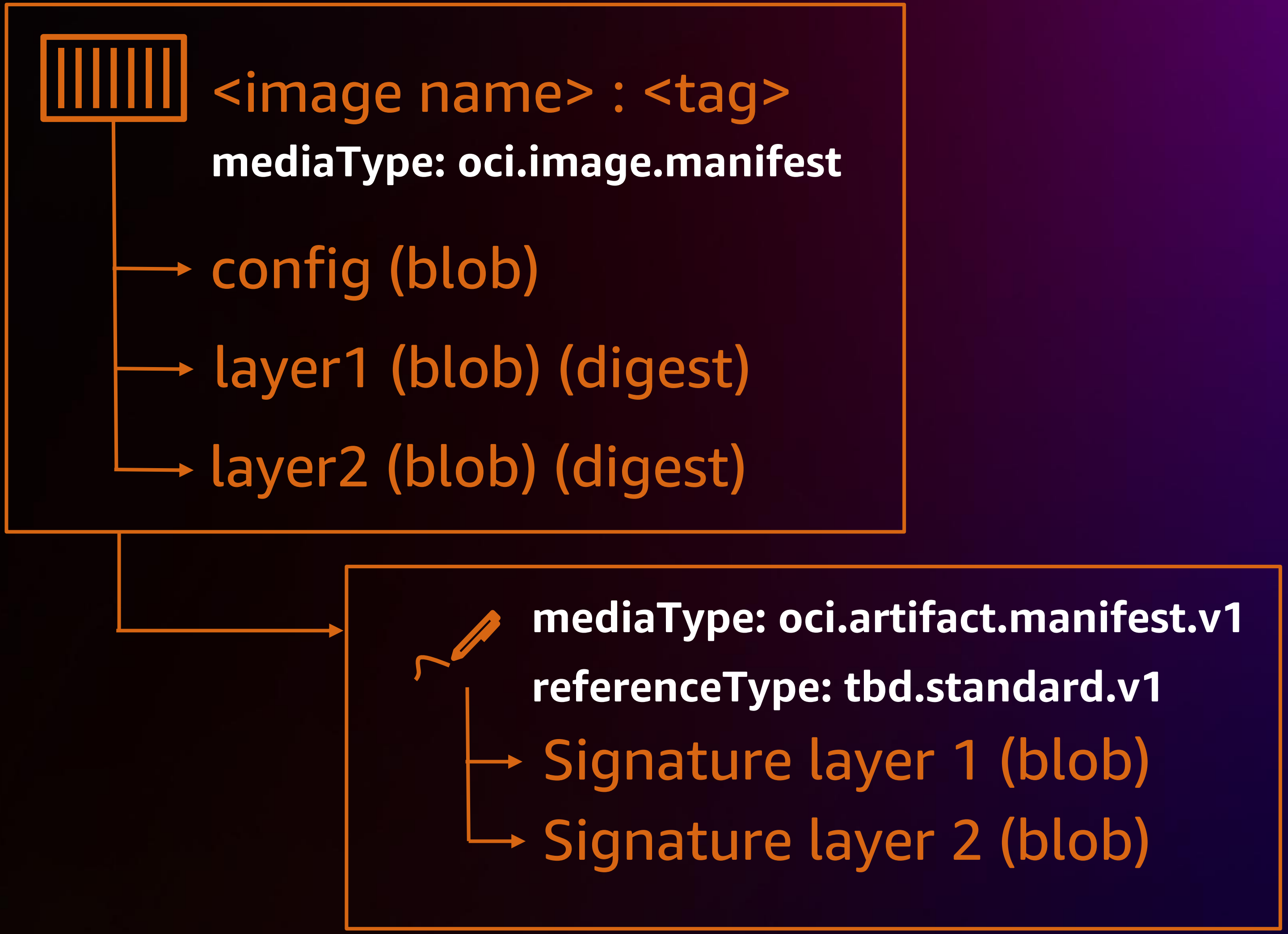
Needed a way to attach information to an image after build/push time without changing the digest

Reference types – How to store additional information alongside an image in a registry

New manifest type – artifact manifest

- Config no longer required
- Top-level artifactType field

New subject field, allowing an artifact to reference an image



Reference types – Example artifact manifest

Example artifact manifest:

```
{
  "mediaType": "application/vnd.oci.artifact.manifest.v1+json",
  ”artifactType": "application/vnd.example.sbom.v1",
  ”blobs": [
    {
      "mediaType": "application/gzip",
      "size": 32654,
      ”digest": "sha256:9834876dcfb05cb167a5c24953eba58c4ac89b1adf57f28f2f9d09af107ee8f0"
    }
  ],
  ”subject": {
    "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
    "size": 16724,
    "digest": "sha256:3c3a4604a545cdc127456d94e421cd355bca5b528f4a9c1905b15da2eb4a4c6b"
  },
  "annotations": {...}
}
```

Reference types – How to discover information related to an image

New API – the referrers API

Able to query for artifacts that reference any subject image

Filter by artifactType

```
GET
/<name>/referrers/<digest>
```

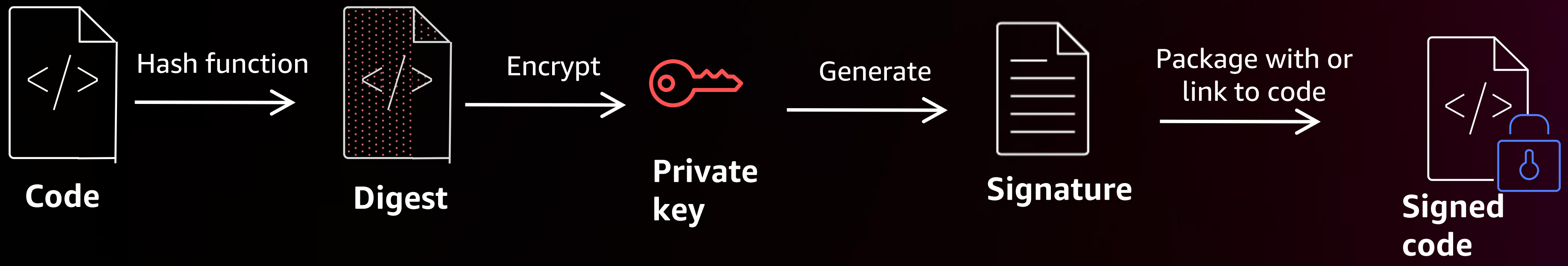
```
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.oci.image.index.v1+json",
  "manifests": [
    {
      "mediaType": "application/vnd.oci.artifact.manifest.v1+json",
      "size": 1234,
      "digest": "sha256:a1a1a1...",
      "artifactType": "application/vnd.example.sbom.v1",
      "annotations": {
        "org.opencontainers.artifact.created": "2022-01-01T14:42:55Z",
        "org.example.sbom.format": "json"
      }
    },
    {
      "mediaType": "application/vnd.oci.artifact.manifest.v1+json",
      "size": 1234,
      "digest": "sha256:a2a2a2...",
      "artifactType": "application/vnd.example.signature.v1",
      "annotations": {
        "org.opencontainers.artifact.created": "2022-01-01T07:21:33Z",
        "org.example.signature.fingerprint": "abcd"
      }
    }
  ]
}
```

Implementations of reference type

- ORAS
- Notary v2
- go-containerregistry
- Zot
- Distribution

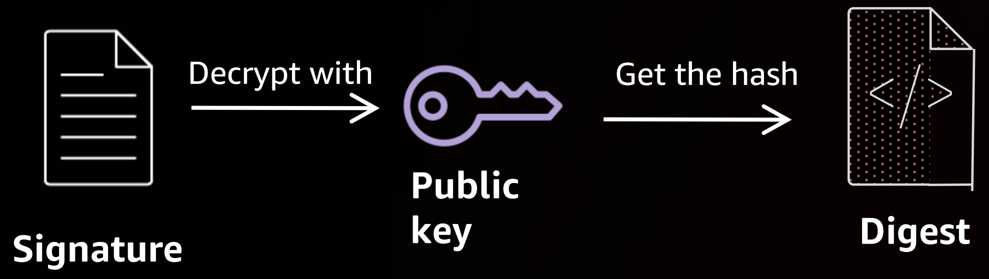
Notary v2 – What is code signing?

- Code signing is an industry standard technique used to confirm that the code is unaltered and comes from an approved source



Notary v2 – Verifying signatures

Recipient of signed code verifies hash using public key



```
If (decrypted hash  == hash of original code)
{
        // The code is correctly signed
}
```

Notary v2 trust policy document

```
{
    "version": "1.0",
    "trustPolicies": [
        {
            "name": “<policy_name>",   // Name of the policy.
            "registryScopes": [ <image_name> ],  // The registry artifacts to which the policy applies.
            "signatureVerification": {// The level of verification - strict, permissive, audit, skip.
              "level" : “<strict>"
            },
            "trustStores": ["ca:<trust_store_name>"], // The trust stores that contain the X.509 trusted roots.
            "trustedIdentities": [// Identities that are trusted to sign the artifact.
              "x509.subject: C=US, ST=WA, L=Seattle, O=<organization>, OU=<organization_unit>, CN=<common_name>"
            ]
        }
    ]
}
```

Adopting and using image signing

Set up signing keys

```
> notation certificate generate-test <certificate_identity>
> notation key add <key_name> <key_file_path> <certificate_file_path>
```

Sign and push image

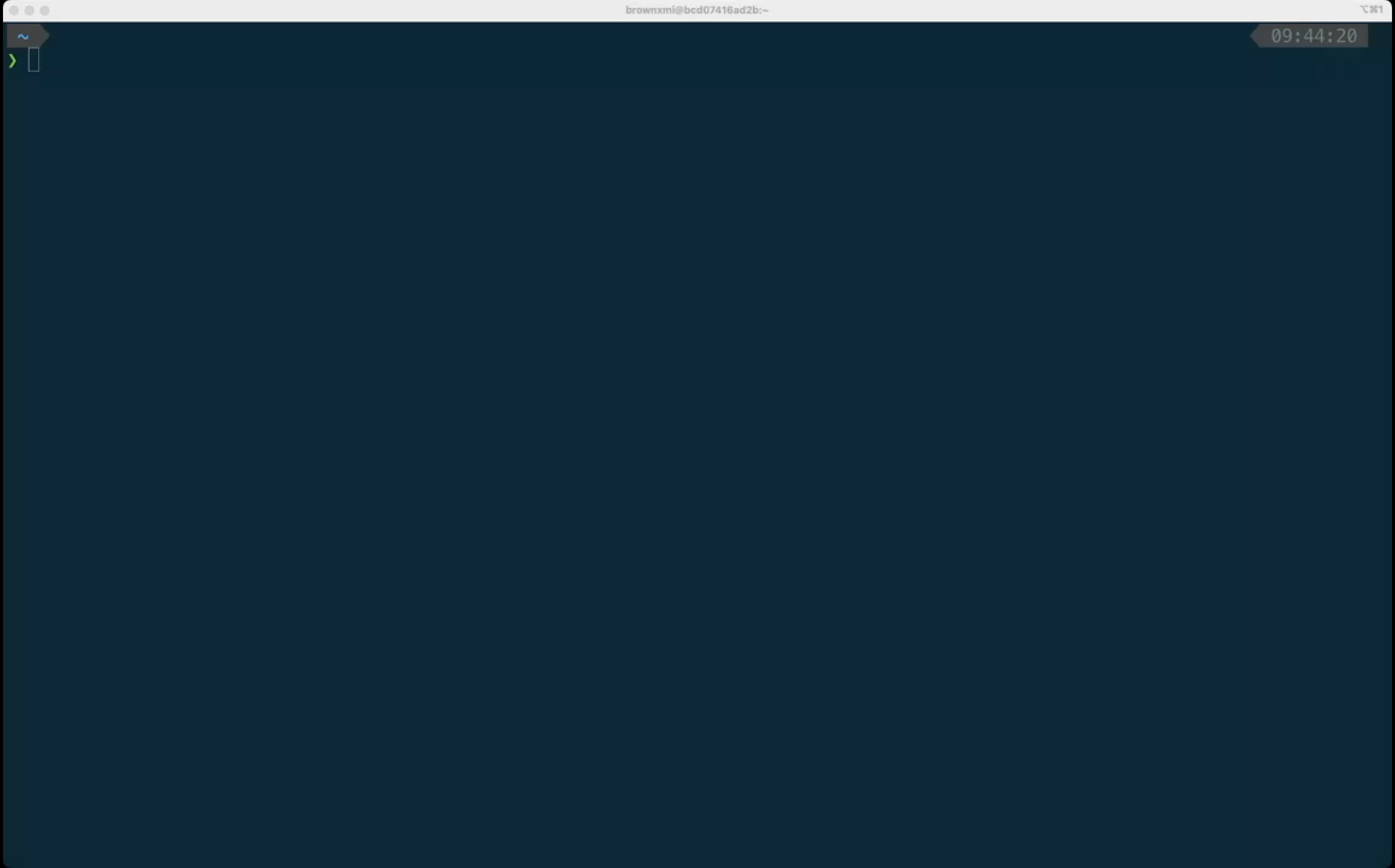
```
> docker push <image_name>
> notation sign <image_name> --key <key_name>
```

Set up trust policy

```
> notation certificate add --type <certificate_type>
      --store <certificate_file_path>
> cp <trust_policy_json> $HOME/.config/notation/trustpolicy.json
```

Verify and pull image

```
> notation verify <image-name>
Verified signature for <image_digest>
> docker pull <image_name>:<image_digest>
```



```
brownxmi@bcd07416ad2b:~                                                   ⌥⌘1
 ~                                                                   09:44:20
❯
```

Thank you!

Please complete the session survey in the **mobile app**