

Trust But Verify: Bringing Supply Chain Integrity To CD GitOps

Yuji Watanabe & Hirokuni Kitahara, IBM Research

Trust But Verify: Bringing Supply Chain Integrity To CD GitOps



BUILDING FOR THE ROAD AHEAD

DETROIT 2022



KubeCon



CloudNativeCon

North America 2022

BUILDING FOR THE ROAD AHEAD

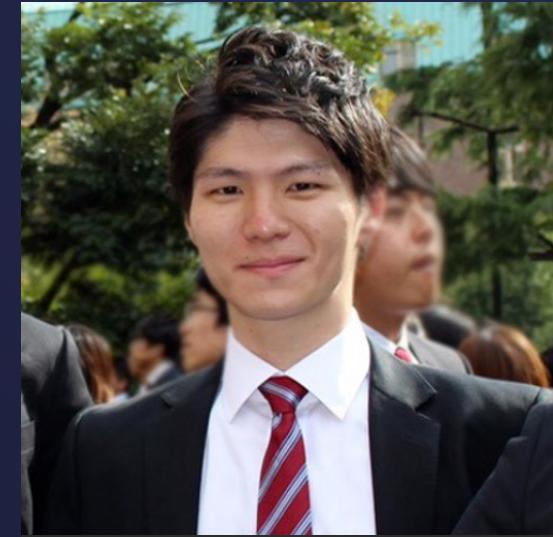
DETROIT 2022

October 24-28, 2021



Yuji Watanabe

Senior Technical
Staff Member
IBM Research



Hirokuni Kitahara

Research Scientist

IBM Research

Agenda

- End-to-end software supply chain integrity
- YAML manifest signature and enforcement
- Gaps in CD GitOps - manifest build
- ArgoCD Interlace – extension to fill the gap
- Demo

End-to-end supply chain

source



build



package



deliver



maintain

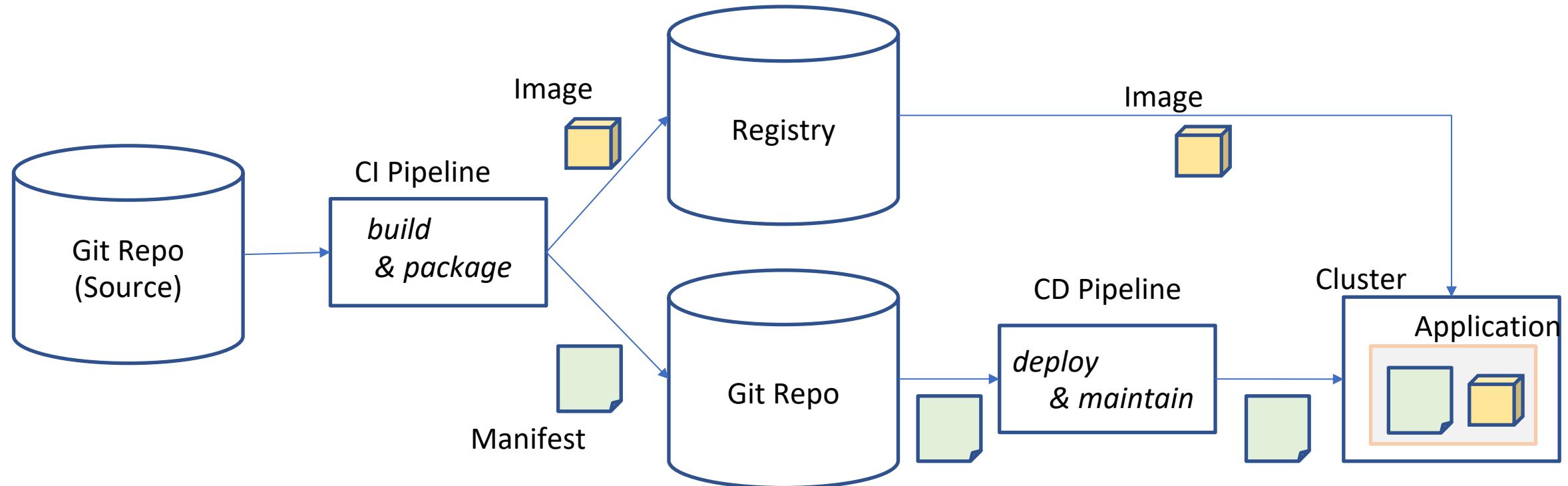


use

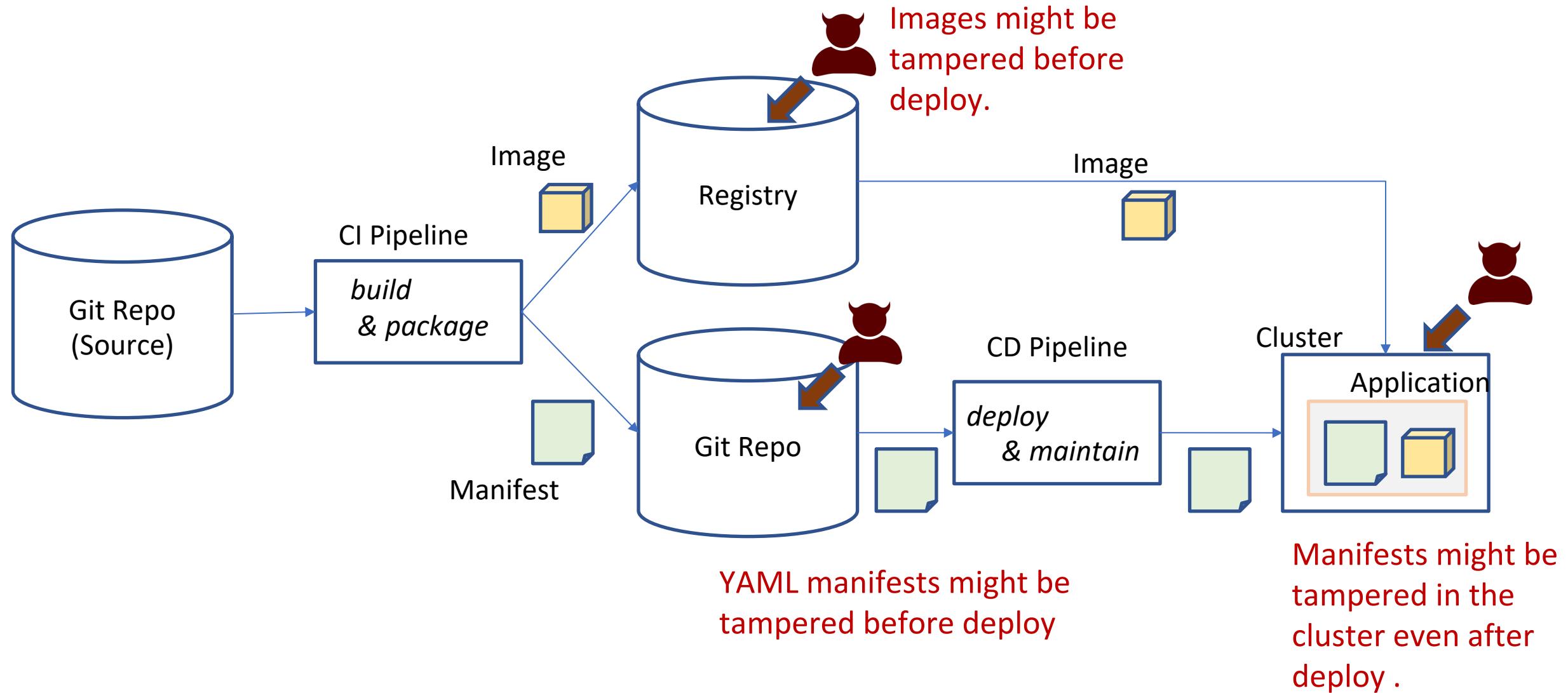


Delivery application to clusters

Images and YAML manifests are delivered to deploy an application.

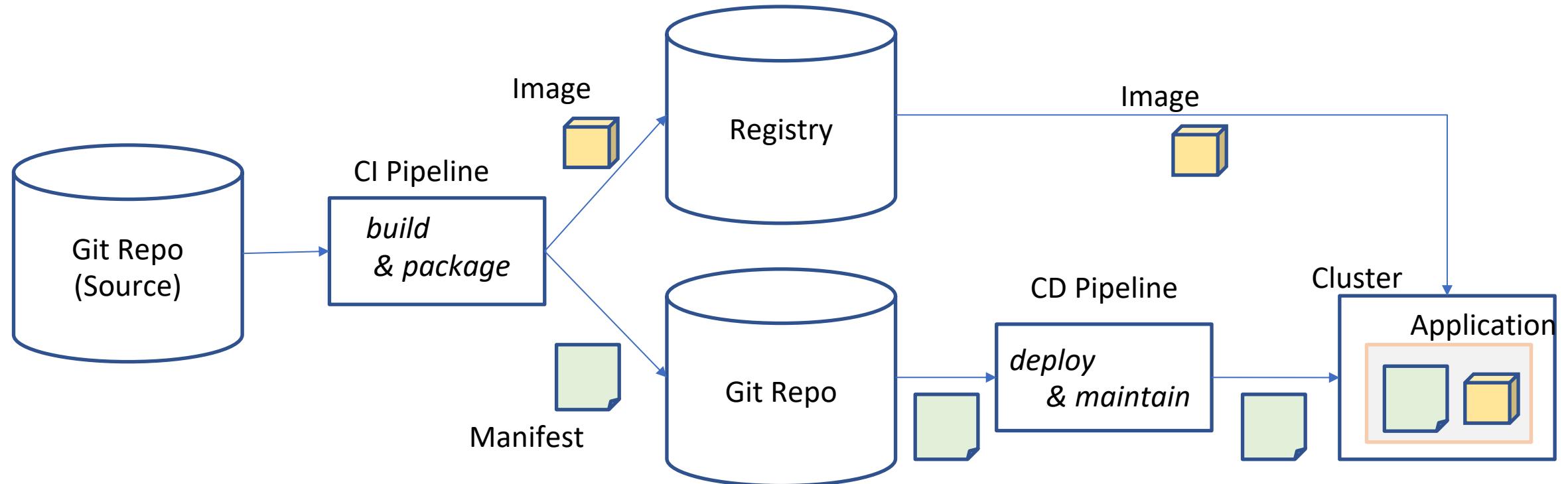


Integrity issues in delivery



End-to-end software supply chain integrity

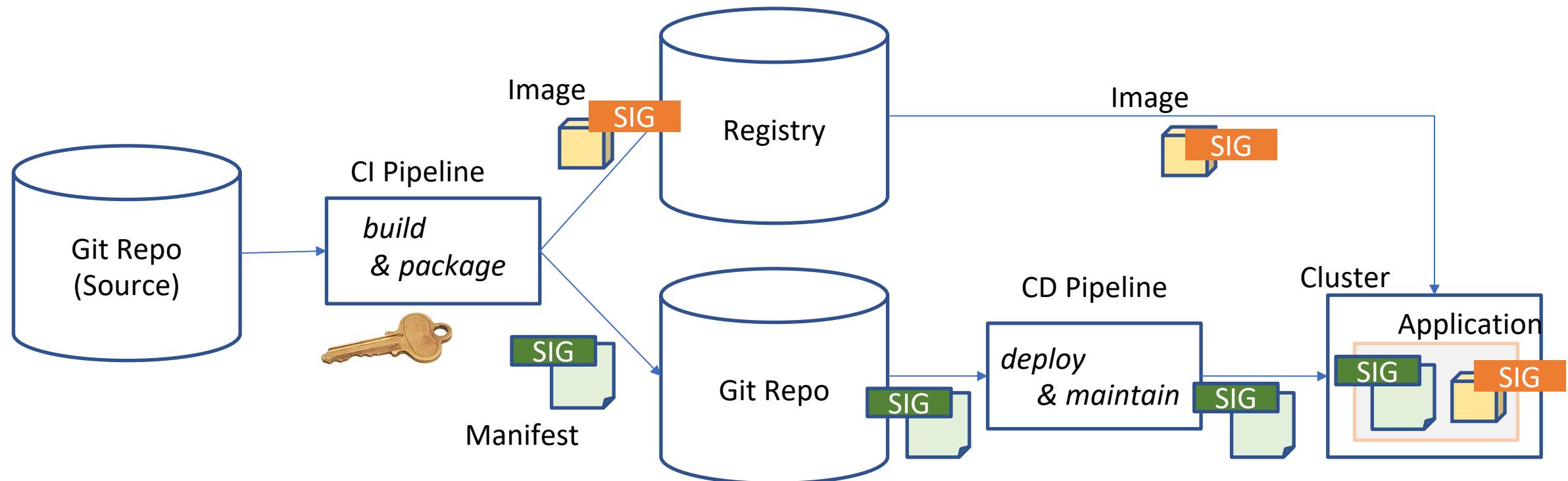
To protect delivery ...



End-to-end software supply chain integrity

To protect delivery ...

Use signature for integrity assurance and source verification of manifests and images



YAML manifest signing

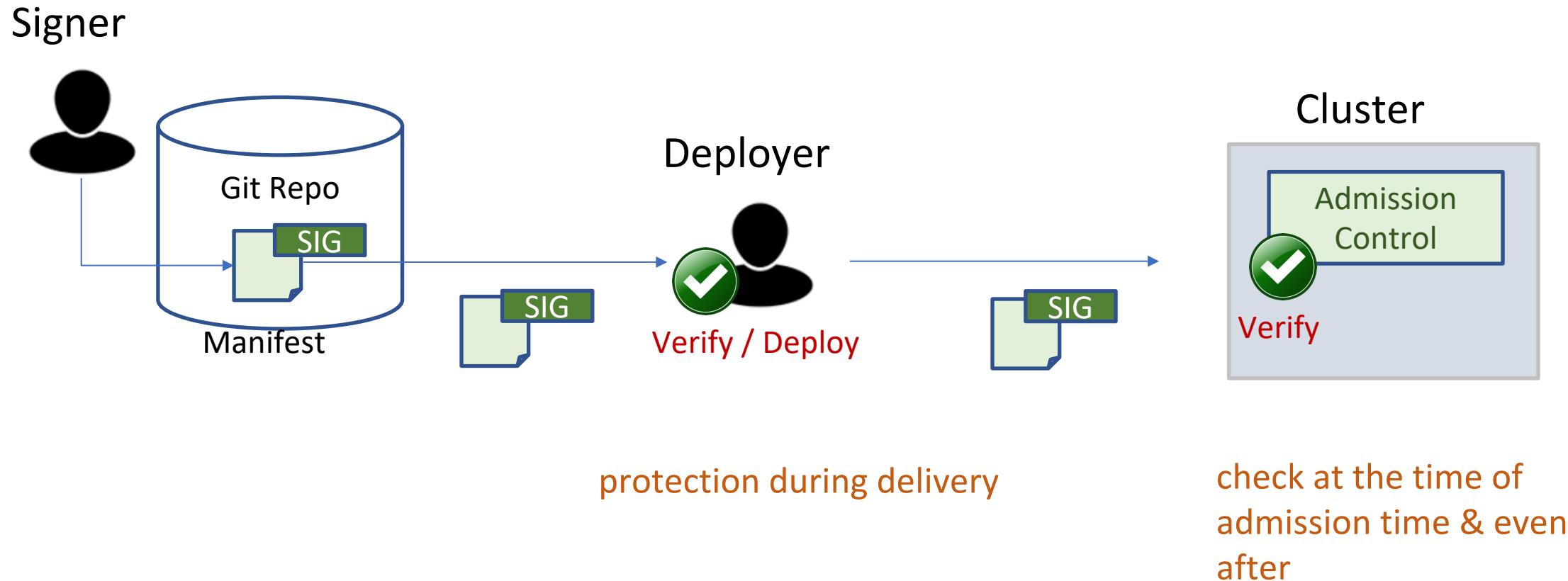
<https://github.com/sigstore/k8s-manifest-sigstore>

kubectl sigstore sign -f test-deployment.yaml -k cosign.key

```
metadata:
  annotations:
    cosign.sigstore.dev/message: H4sIAAAAAAA/wBZAab+H4sIAAAAAAA/+ySu87bMAyFPfsp+A
    cosign.sigstore.dev/signature: MEUCIQCetBNqM7j4JBBgewjUIMsIPlR3gxKF+fLZ8rGYBZE2
  labels:
    app: nginx
  name: test-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
```

Verify signed YAML manifests

- Both deployer and cluster side admission control can verify the signature.

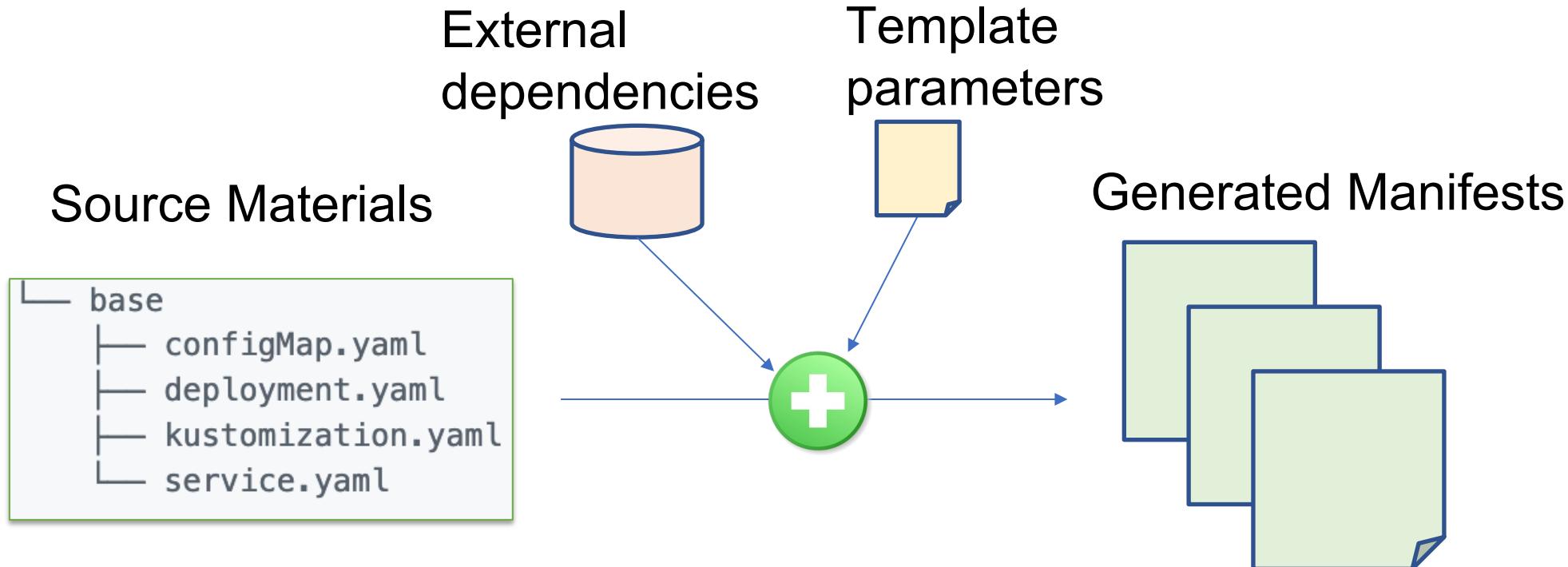


Admission control for signed YAML manifests

- Integrity Shield
 - OPA/Gatekeeper (PEP) + Integrity Shield API backend(PDP)
 - Signature verification at admission time + continuous check
- Kyverno
 - validate.manifests() to enable YAML manifest signature validation (v1.8.0-)
- You can enable protection of your YAML manifests with signature using k8s-manifest-sigstore tool via CLI or pipeline.



Manifest Build (kustomize)



```
kubectl apply -k <base_dir>
```

```
kustomize build <base_dir> | kubectl apply -f -
```

Nested dependency in manifest build

App repo (parent)

- guestbook-ui-deployment.yaml
- guestbook-ui-svc.yaml
- kustomization.yaml

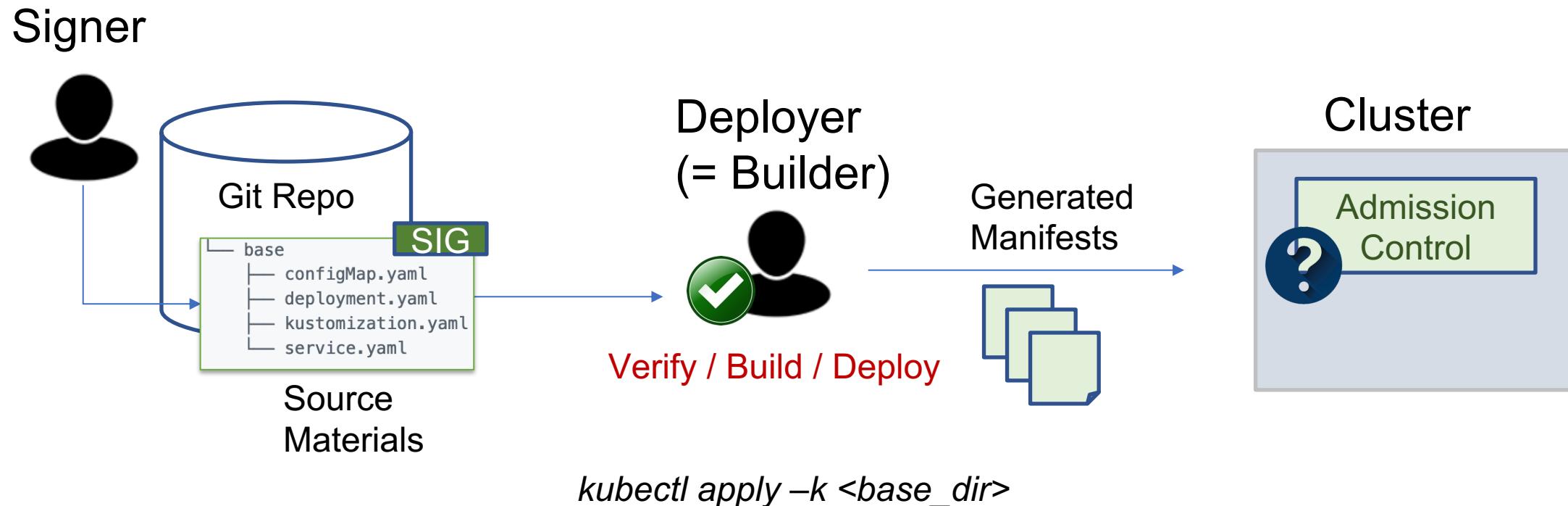
```
namePrefix: sample-kustomize-
commonLabels:
  project: sample-kustomize-app
resources:
  - https://github.com/hirokuni-kitahara/sample-kustomize-base.git
  - guestbook-ui-deployment.yaml
  - guestbook-ui-svc.yaml
```

Config repo (child)

- base-configmap.yaml
- kustomization.yaml

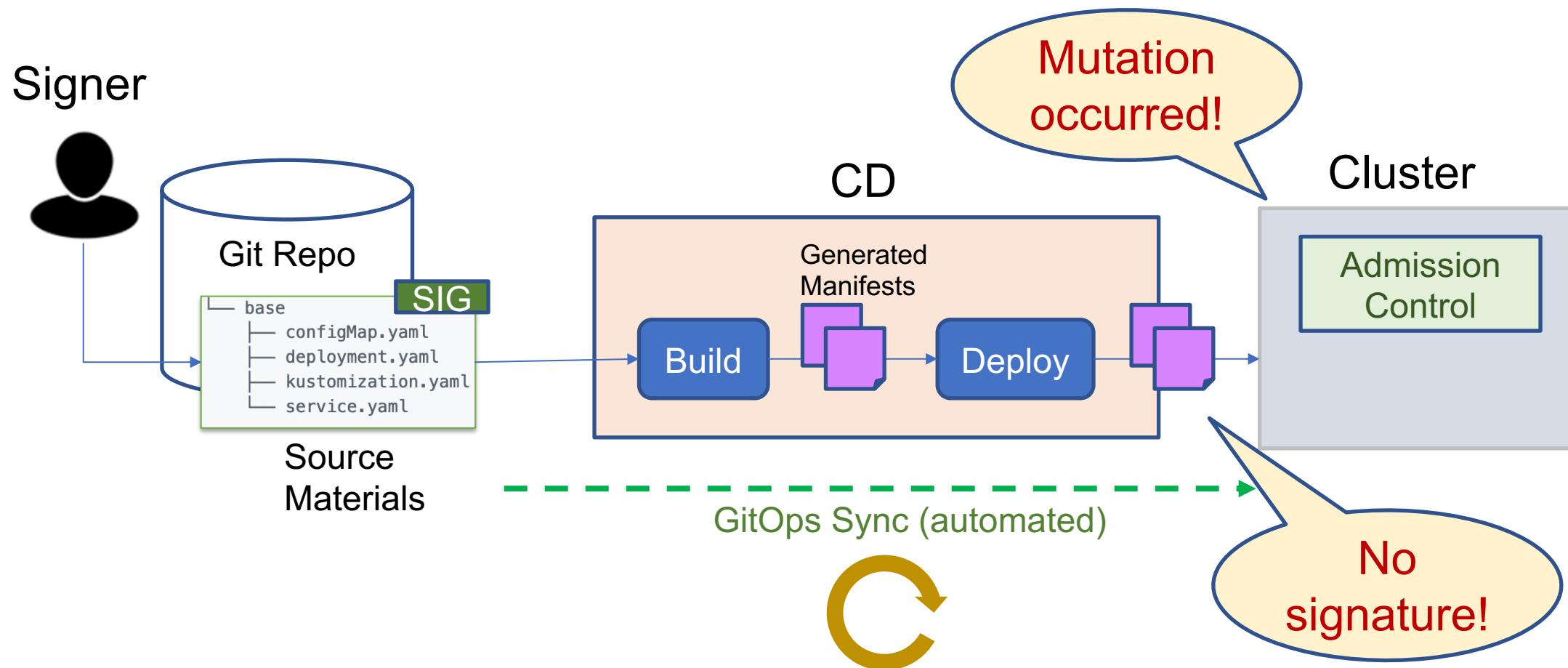
Cluster needs to rely on deployer

- Deployer can verify before build, but the cluster side cannot.
→ Cluster side need to rely on deployer's pre-check

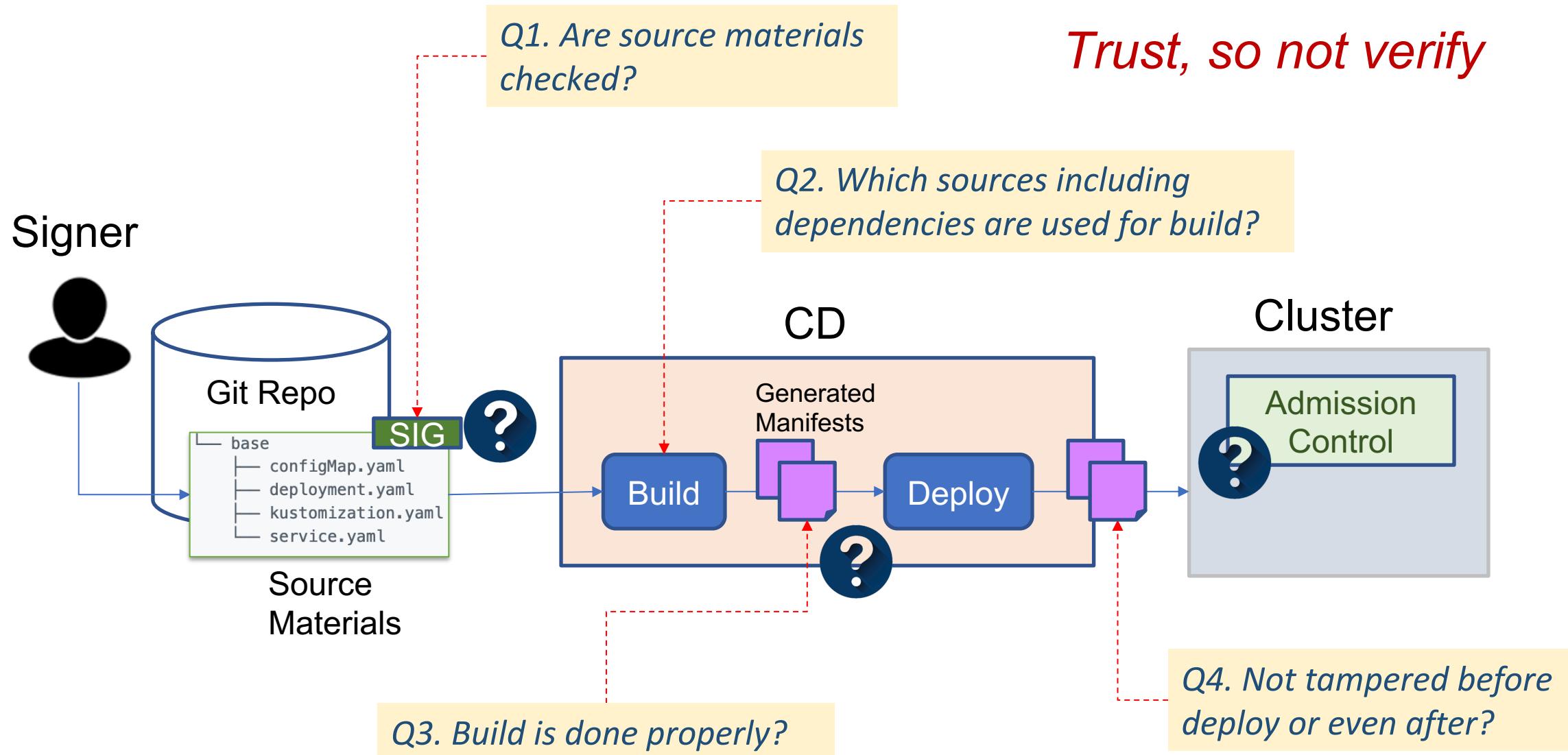


CD GitOps

Modern software deployment mutated the source materials before deploying final manifests.



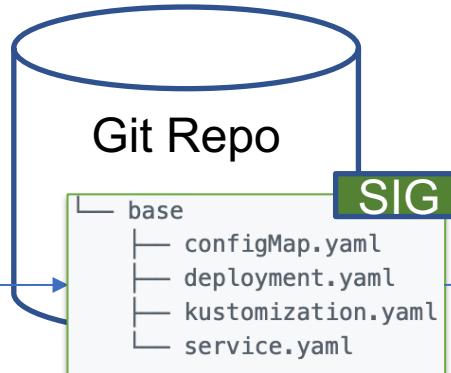
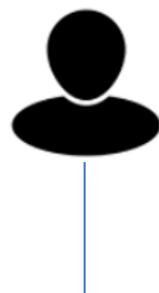
Gaps in CD GitOps



Interlace – "trust but verify" approach

Interlace controller extends GitOps sync in ArgoCD by three functions - verify, sign, provenance.

Signer



1. Verify signature

SIG

ArgoCD

Build

Generated Manifests

Deploy

SIG

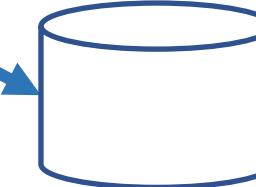
2. Sign manifests

Interlace

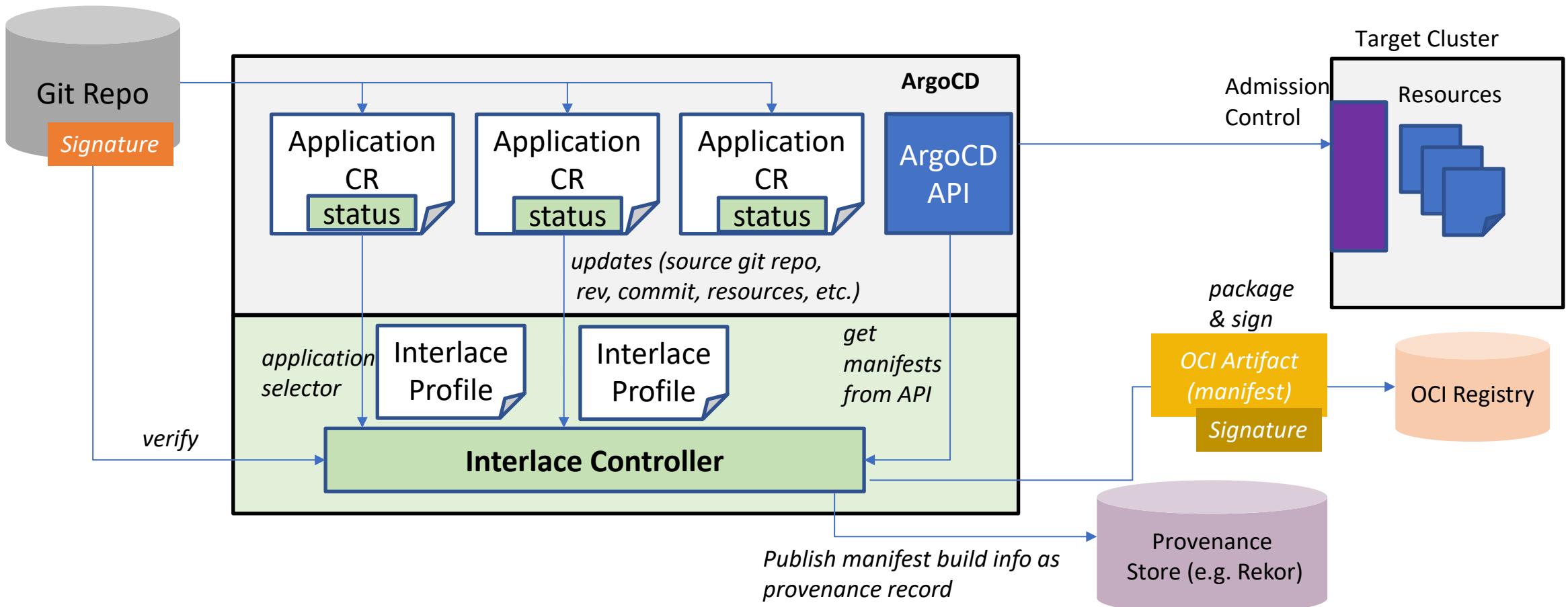


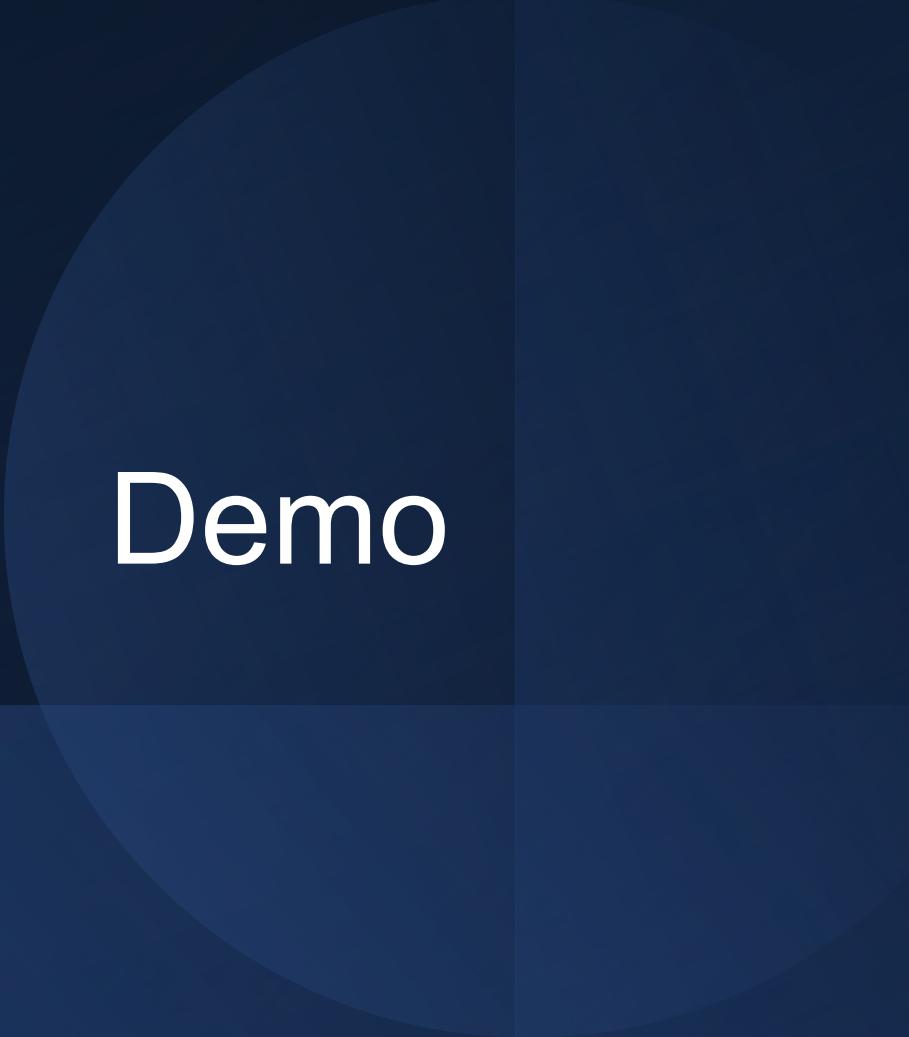
3. Record provenance

Cluster



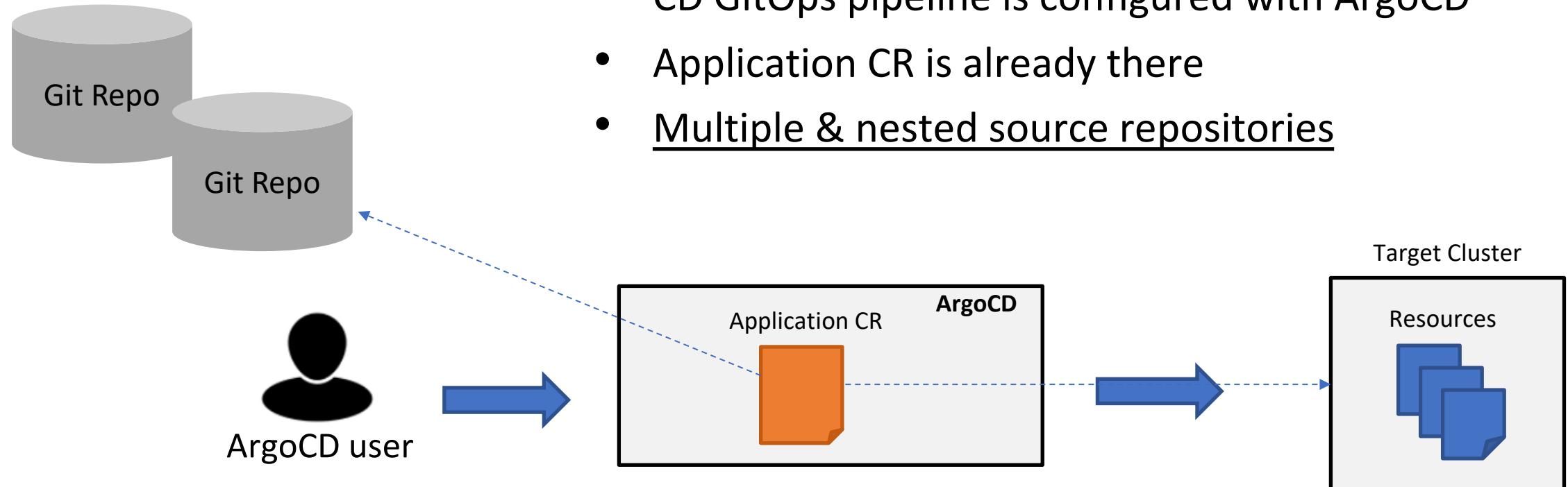
- An Application custom resource defines source (git repo) and target (cluster).
- ArgoCD observes source git repo, then auto-sync with target.
- Interlace detects the sync event and then do the verify/sign/provenance recording.





Demo

Demo Scenario



Source repositories in demo

hirokuni-kitahara / sample-kustomize-app Public

Code Issues Pull requests Actions Projects View

main sample-kustomize-app / kustomization.yaml

hirokuni-kitahara use external base repo ...

1 contributor

7 lines (7 sloc) | 207 Bytes

```
1 namePrefix: sample-kustomize-
2 commonLabels:
3   project: sample-kustomize-app
4 resources:
5 - https://github.com/hirokuni-kitahara/sample-kustomize-base.git
6 - guestbook-ui-deployment.yaml
7 - guestbook-ui-svc.yaml
```

External dependency

hirokuni-kitahara / sample-kustomize-base

Code Issues Pull requests Actions

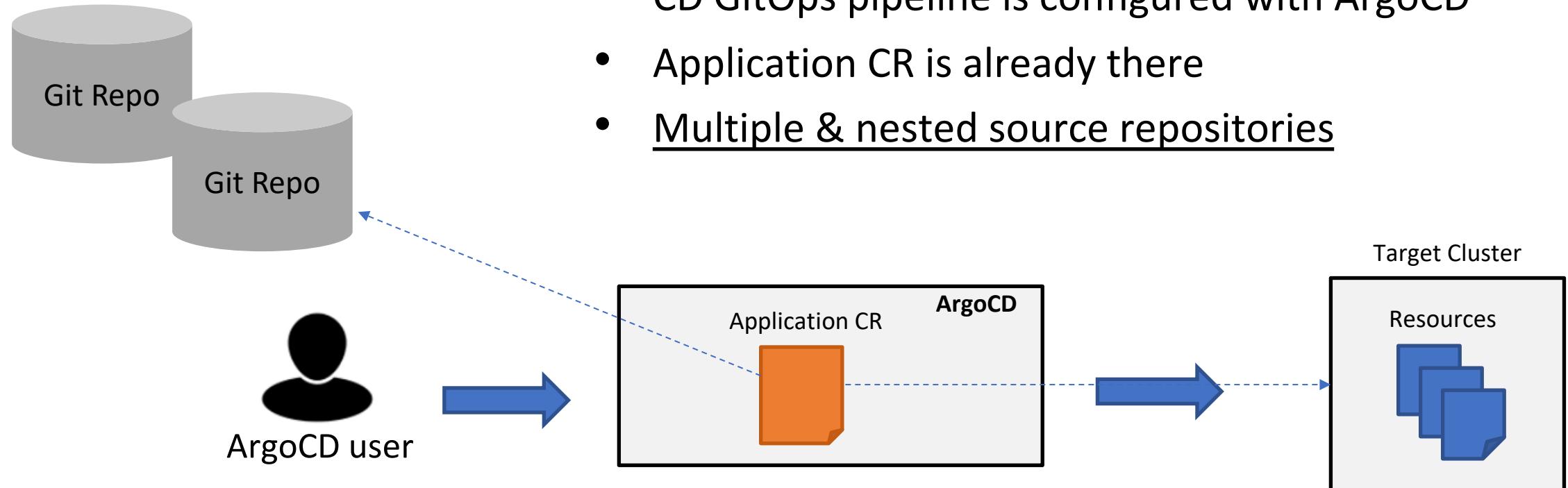
main 1 branch 0 tags

hirokuni-kitahara initial commit ...

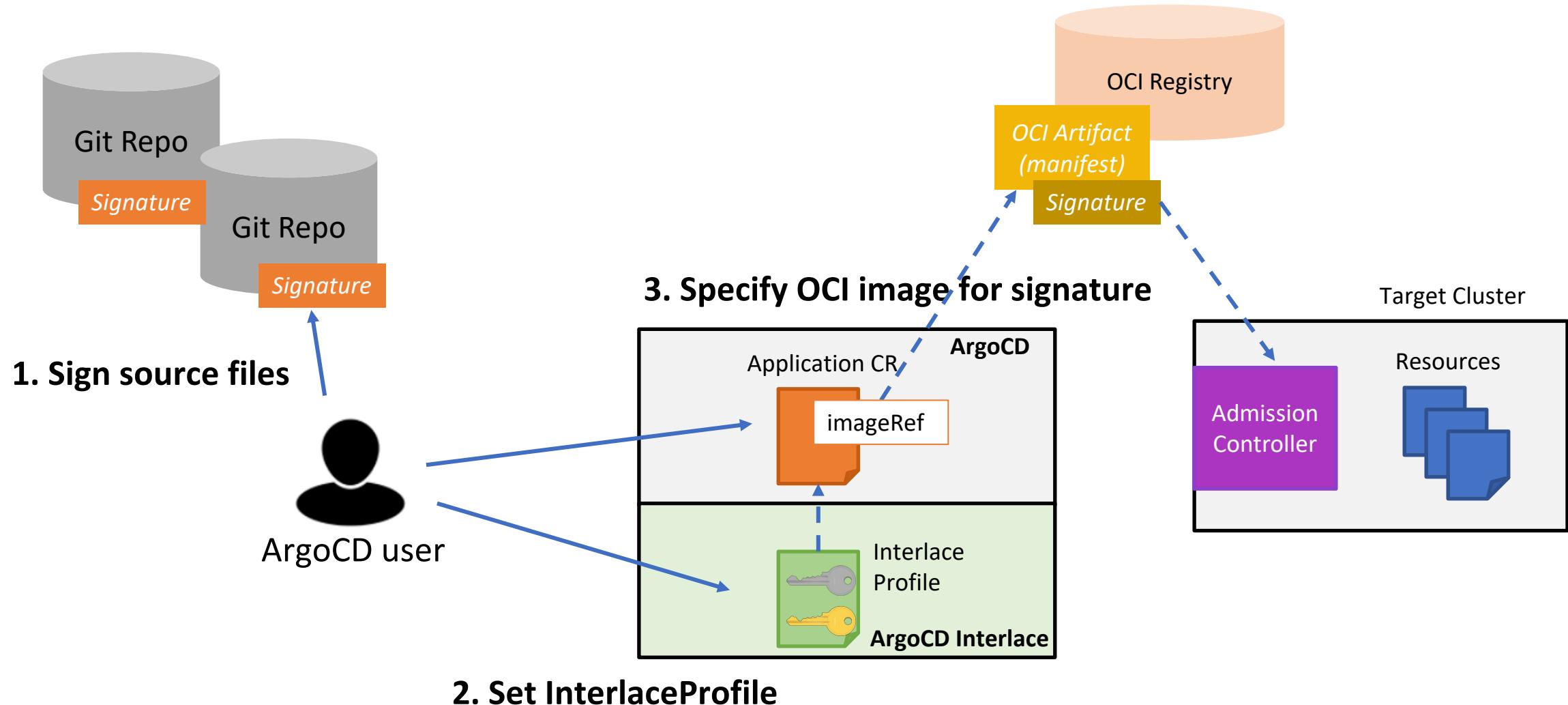
base-configmap.yaml initial commit

kustomization.yaml initial commit

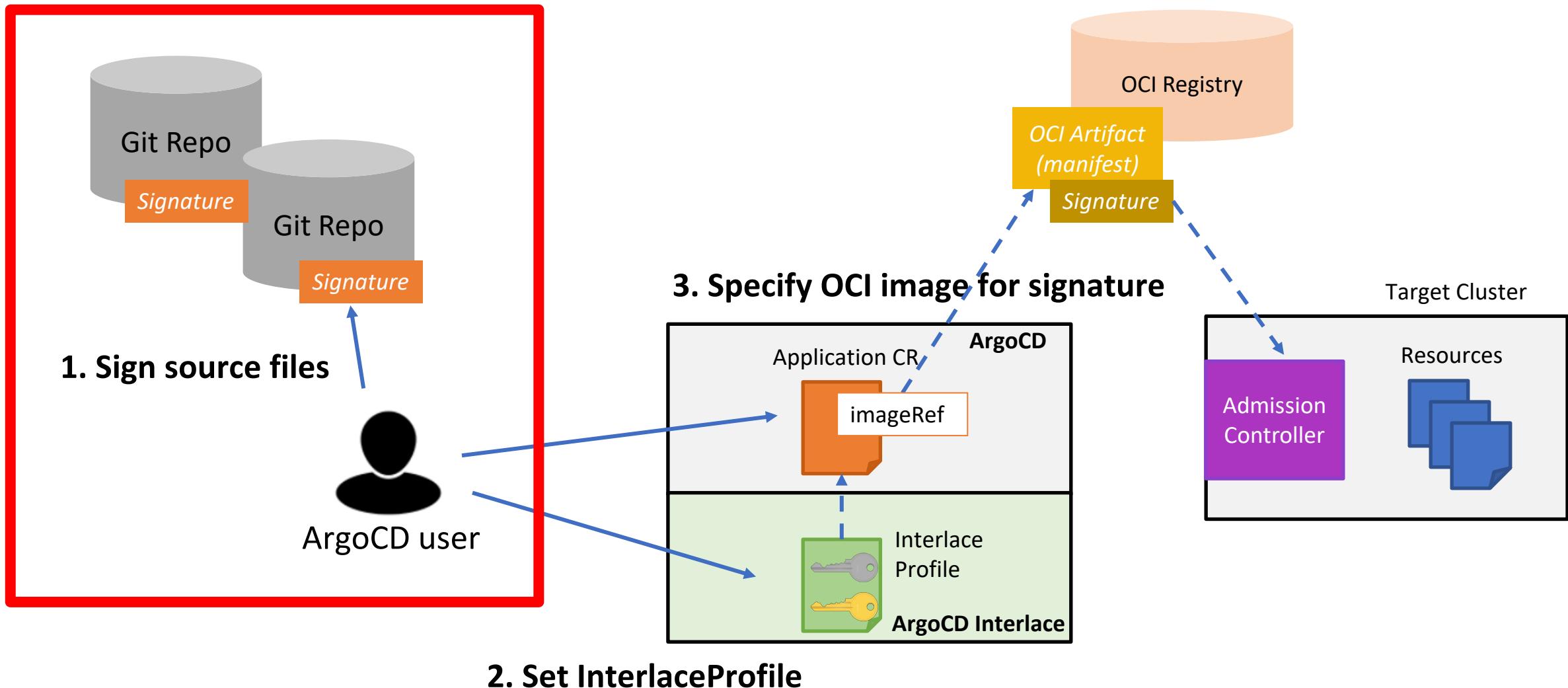
Demo Scenario



Demo Scenario



Demo Scenario



Sign source

```
$ git ls-tree -r HEAD --name-only | grep -v source-materials | xargs sha256sum > source-materials
```

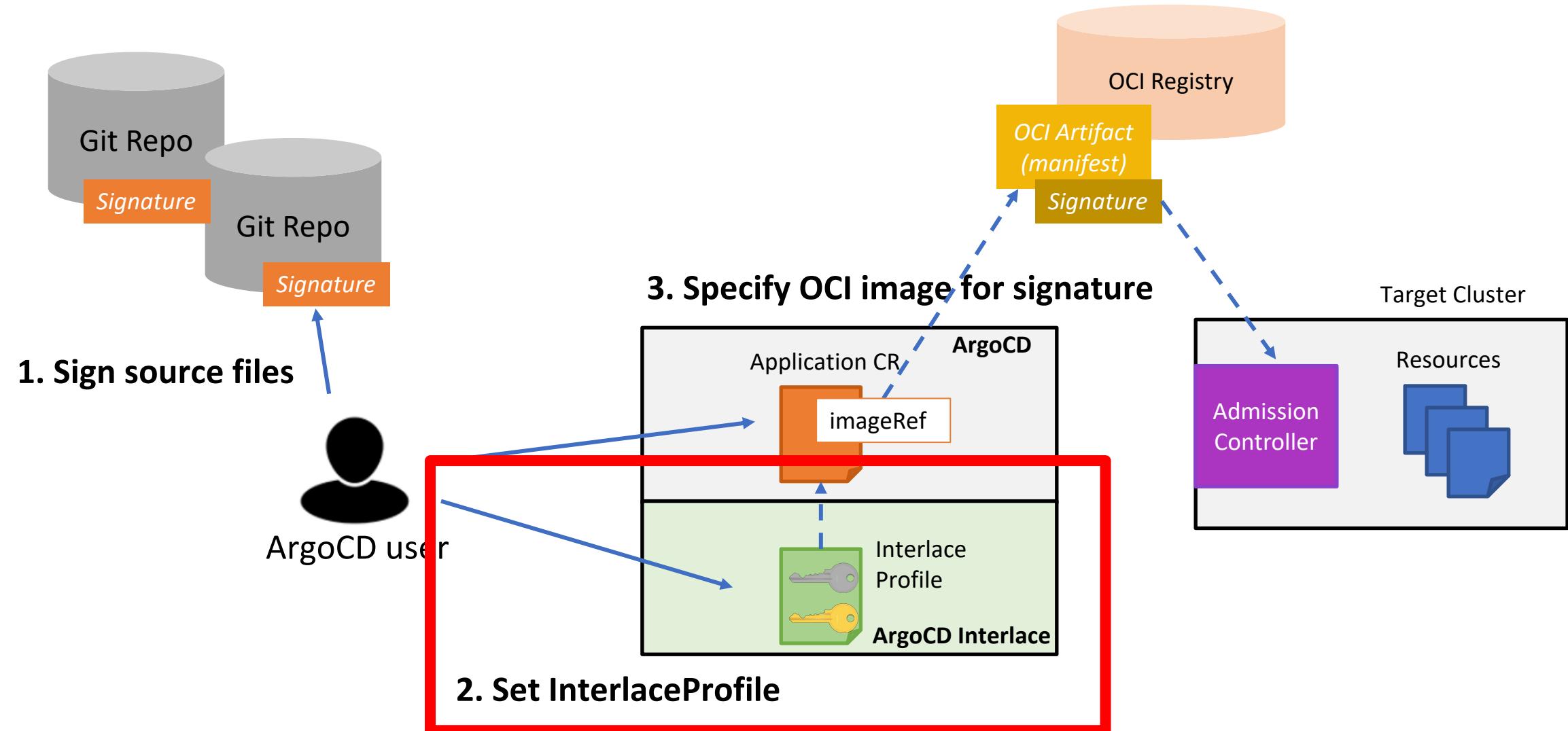
```
$ cosign sign-blob --key <SIGNING_KEY> --output-signature source-materials.sig source-materials
```



hirokuni-kitahara sign files		
	2972ead 2 minutes ago	⌚ 85 commits
📄 guestbook-ui-deploym... reset		3 hours ago
📄 guestbook-ui-svc.yaml initial commit		14 months ago
📄 kustomization.yaml use external base repo		27 days ago
📄 source-materials sign files		2 minutes ago
📄 source-materials.sig sign files		2 minutes ago

Signature files

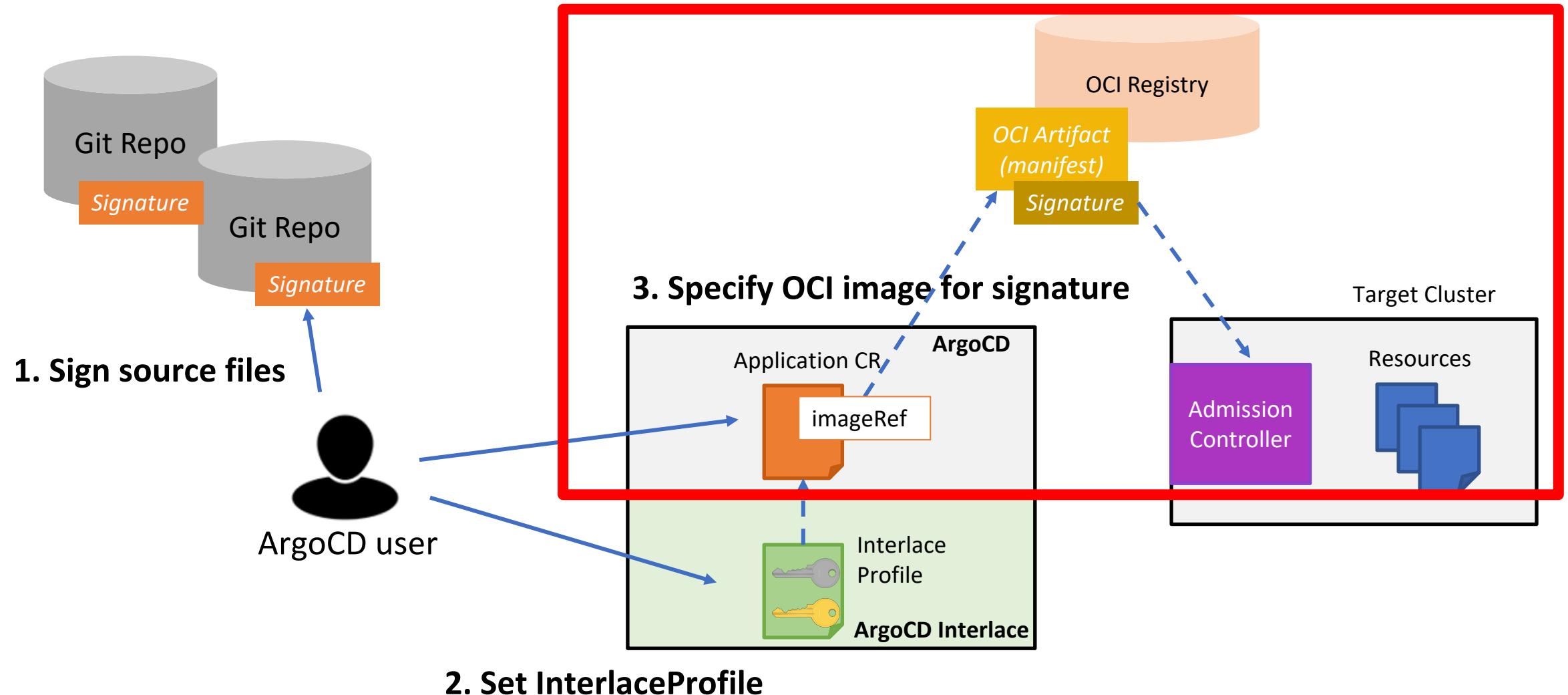
Demo Scenario



Set InterlaceProfile

```
1  apiVersion: interlace.argocd.dev/v1beta1
2  kind: InterlaceProfile
3  metadata:
4    name: kubecon-demo-profile
5    namespace: argocd-interlace
6  spec:
7    applicationSelector:
8      - matchLabels:
9        kubecon: 'demo'
10   verifyConfig:
11     key:
12       PEM: |
13         -----BEGIN PUBLIC KEY-----
14         MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE5znfqEW3LfyMW
15         jl7n/wWiZxWxtDskyTwFUgWYtILbpJ+j9lJqrPfBlmyp4Lxt0
16         -----END PUBLIC KEY-----
17   signConfig:
18     key:
19       secret: kubecon-demo-sign-key
20     match:
21       - kind: Deployment
22       - kind: Service
```

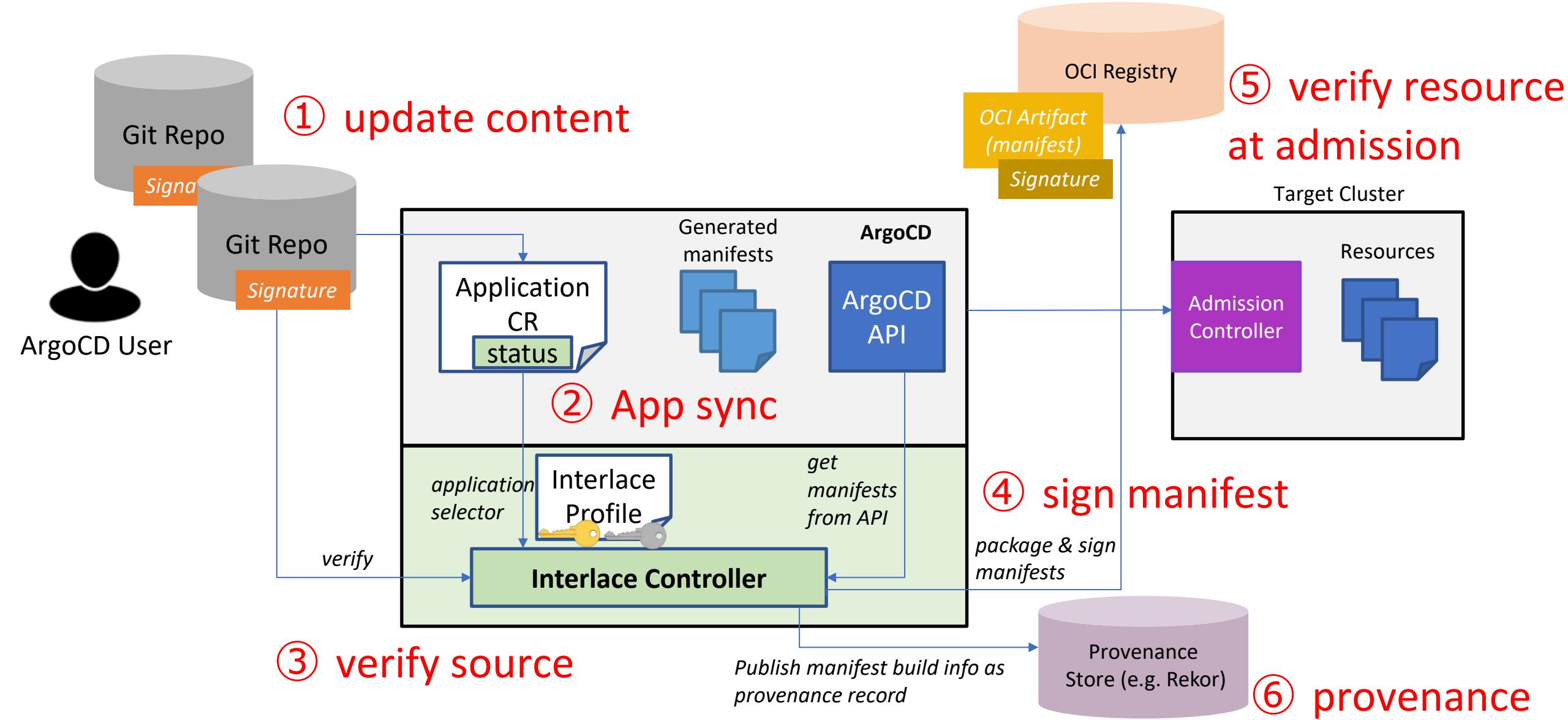
Demo Scenario



Specify OCI Image for signature

```
1 apiVersion: argoproj.io/v1alpha1
2 kind: Application
3 metadata:
4   name: kubecon-demo-app-cluster1
5   namespace: argocd
6   annotations:
7     interlace.argocd.dev/manifestImage: ghcr.io/hirokuni-kitahara/kubecon-demo-app-manifest:dev
8   labels:
9     kubecon: demo
10  spec:
```

End-to-end signature protection



Successful deployment with valid signature

Applications / **kubecon-demo-app-cluster1**

APP DETAILS APP DIFF SYNC SYNC STATUS HISTORY AND ROLLBACK DELETE REFRESH ▾

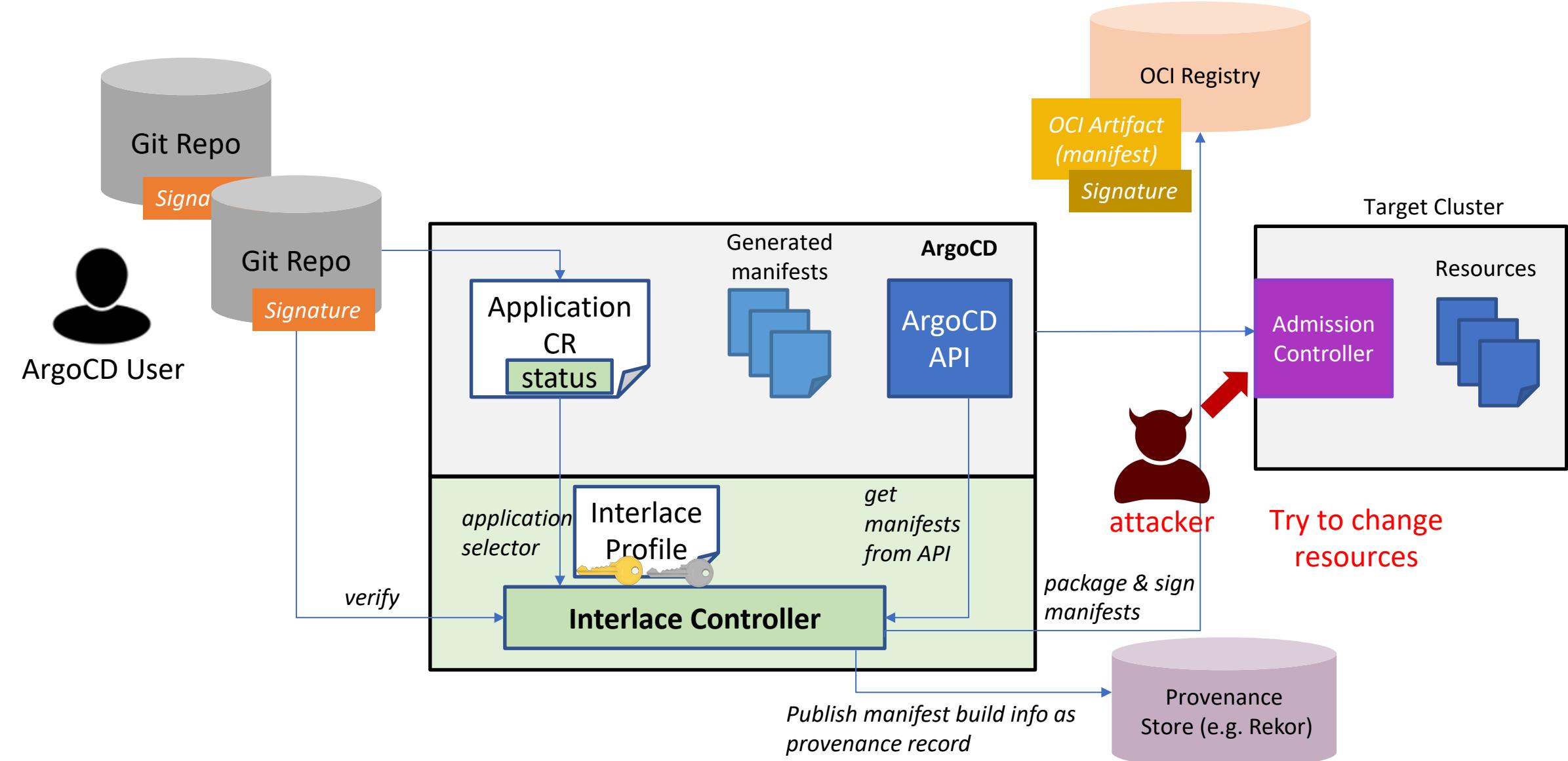
APP HEALTH **Healthy**

CURRENT SYNC STATUS **Synced** **To main (2972ead)** **MORE**
Author: Hirokuni-Kitahara1 <hirokuni.kitahara1@ibm.co...
Comment: sign files

100%

The diagram illustrates the deployment flow from a source cluster to a target cluster. It starts with a source cluster icon labeled "kubecon-demo-app-cluster1" containing two green heart icons. A dashed arrow points from this cluster to a "sample-kustomize-base-config" component (cm icon), which is marked as "Synced". Another dashed arrow points from the source cluster to a "sample-kustomize-guestbook-SVC" component (SVC icon), also marked as "Synced". From the "sample-kustomize-guestbook-SVC" component, a solid arrow leads to a "sample-kustomize-guestbook-ep" component (ep icon), which is also marked as "Synced". This is followed by a "sample-kustomize-guestbook-ES" component (ES icon) and finally a "sample-kustomize-guestbook-rs" component (rs icon). Each of these components has a green heart icon. The deployment process took "a few seconds" for the first step and "11 hours" for the final step, with revision "rev:1".

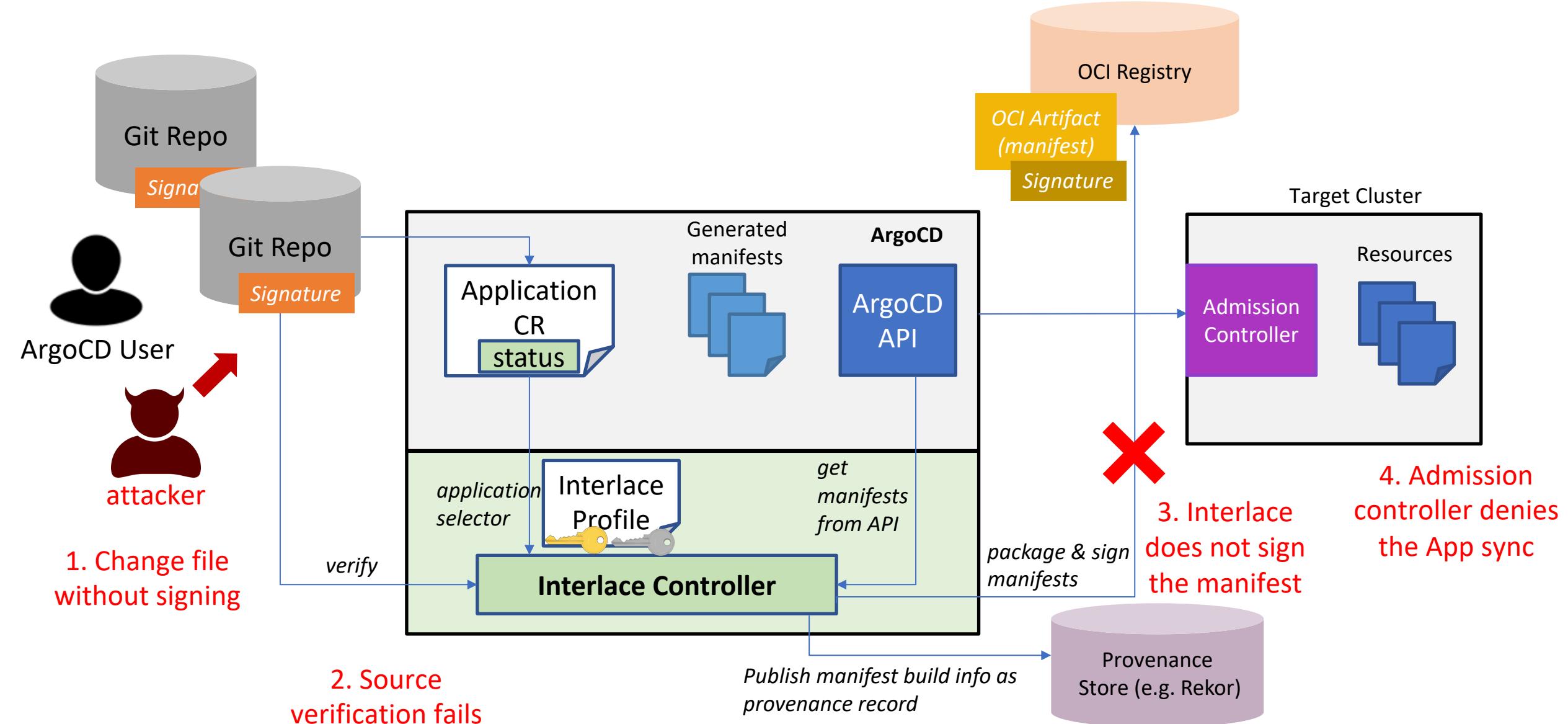
Unauthorized change to resource



Blocked by admission controller

```
target-cluster:$ kubectl edit deploy -n kubecon-demo sample-kustomize-guestbook-ui
error: deployments.apps "sample-kustomize-guestbook-ui" could not be patched:
admission webhook "validation.gatekeeper.sh" denied the request: [enforce-rule-sample-profile-0] denied; {"allow": false, "message": "Signature verification is required for this request, but failed to verify signature. diff found: {\"items\":[{\"key\": \"spec.template.spec.containers.0.ports.0.containerPort\", \"values\": {\"after\": 12345, \"before\": 80}}]}"]
You can run `kubectl replace -f /tmp/kubectl-edit-4020396247.yaml` to try this update again.
target-cluster:$
```

Sync failed because Interlace does not sign



Sync failed because Interlace does not sign

Applications / **kubecon-demo-invalid-sig-app-cluster1** APPLICATIONS

APP DETAILS APP DIFF SYNC SYNC STATUS HISTORY AND ROLLBACK DELETE REFRESH ▾

APP HEALTH **Missing** CURRENT SYNC STATUS **OutOfSync** From **invalid-sig (05da132)** MORE

Author: Hirokuni-Kitahara1 <hirokuni.kitahara1@ibm.co...
Comment: change without signature

LAST SYNC RESULT **Sync failed** MORE To **05da132**
Failed a day ago (Tue Oct 25 2022 11:38:43 GMT-0400)
Author: Hirokuni-Kitahara1 <hirokuni.kitahara1@ibm.co...
Comment: change without signature

APP CONDITIONS **1 Error**

100%

```
graph LR; Root["kubecon-demo-invalid-sig-ap..."] --> CM["sample-kustomize-base-confi... cm"]; Root --> SVC["sample-kustomize-guestbook... svc"]; Root --> Deploy["sample-kustomize-guestbook... deploy"]; CM --> EP["sample-kustomize-guestbook... ep"]; EP --> ES["sample-kustomize-guestbook... endpointslice"]
```

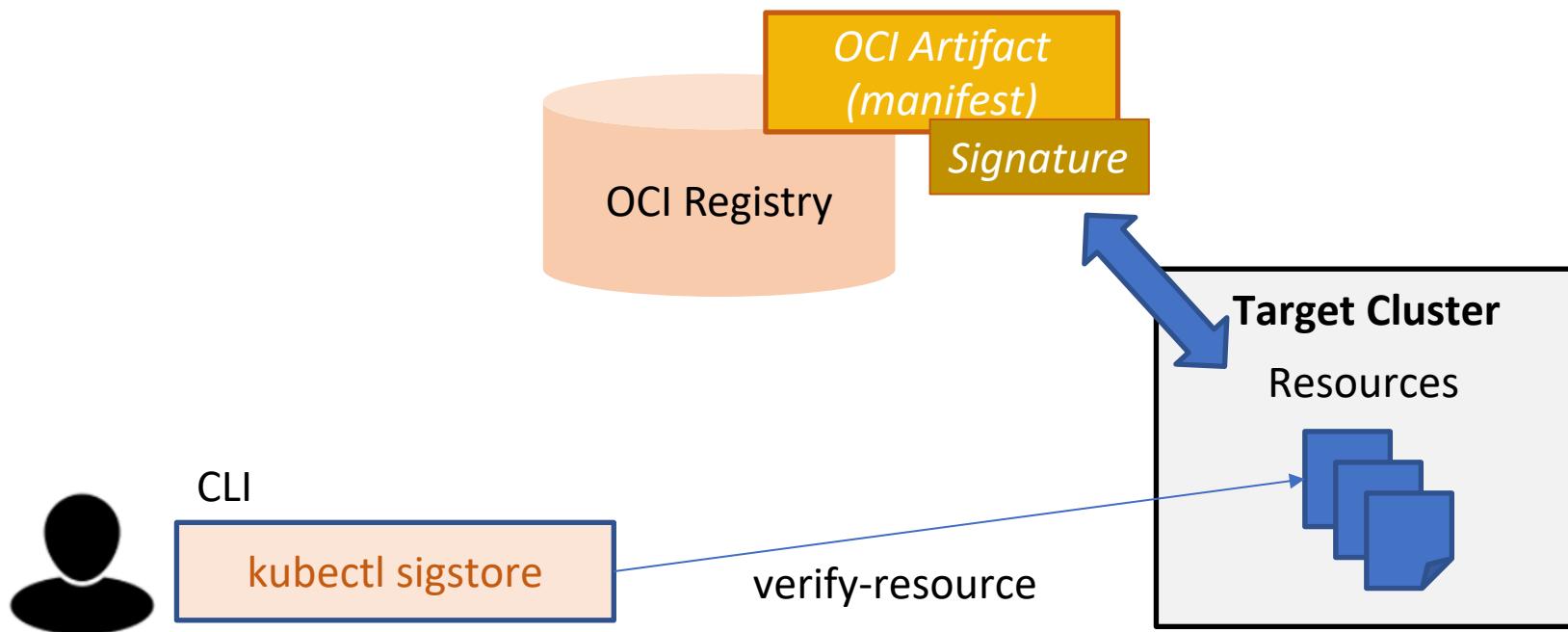
The screenshot shows the Kubeapps interface for managing applications in a Kubernetes cluster. The application 'kubecon-demo-invalid-sig-app-cluster1' is displayed. The 'SYNC' tab is active, showing the current sync status as 'OutOfSync' from 'invalid-sig (05da132)'. The 'LAST SYNC RESULT' section is highlighted with a red box and displays a failed sync message: 'Sync failed' (Failed a day ago), with the author being 'Hirokuni-Kitahara1' and a comment about a 'change without signature'. Below the sync status, there's a network diagram showing dependencies between various Kubernetes resources: 'sample-kustomize-base-config' (cm), 'sample-kustomize-guestbook' (svc), and 'sample-kustomize-guestbook' (deploy). The 'sample-kustomize-base-config' resource has a green checkmark icon, while the others have yellow warning icons. Arrows indicate dependencies from the base config to the svc and deploy resources, and from the svc to the endpointslice resource.

Verify resource with OCI artifact at cluster

- Verify the deployed resources with public key & OCI artifact

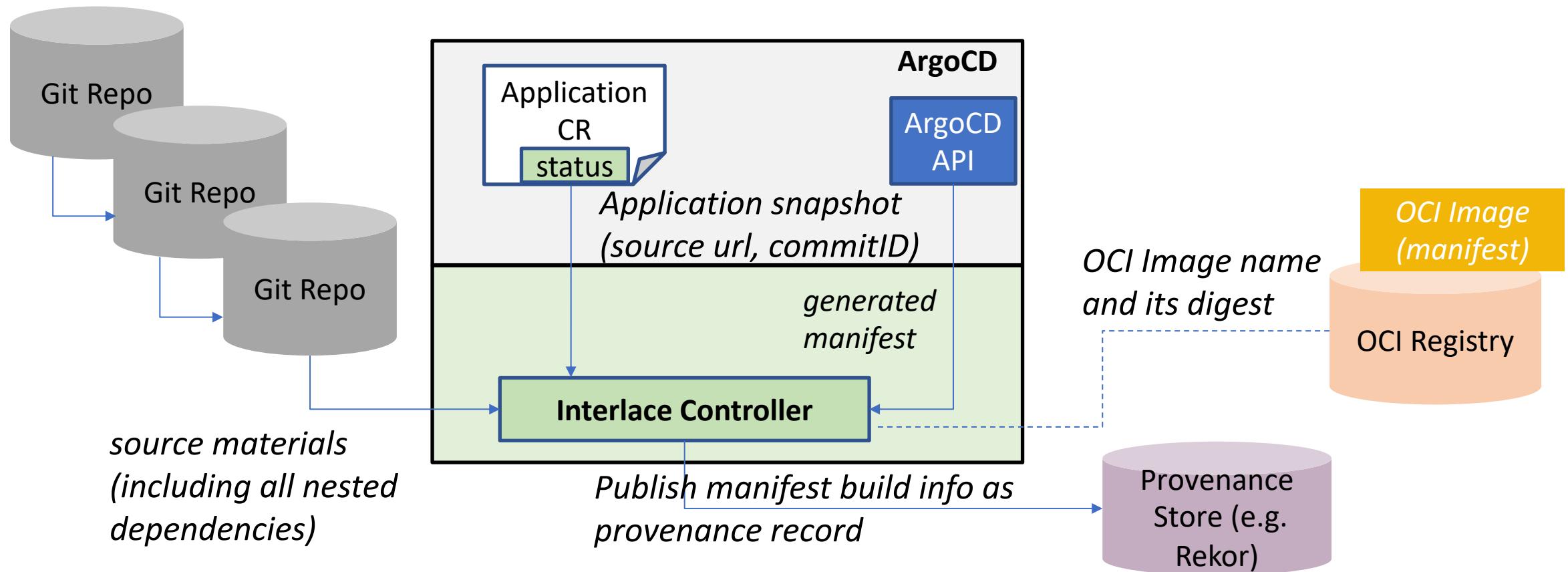
```
$ kubectl sigstore verify-resource <KIND> -k <PUBLIC_KEY> -i <OCI_ARTIFACT>
```

```
$ kubectl sigstore verify-resource deploy -n sample-ns -k ~/.sigstore/cosign.pub -i ghcr.io/hirokuni-kitahara/sample-app-manifest:dev --allow-insecure-registry
NAME          SIGNED   MANIFEST IMAGE           AGE
sample-kustomize-guestbook-ui  true      ghcr.io/hirokuni-kitahara/sample-app-manifest:dev  14h
```



Recording provenance

- Source materials (URL, commitID) including nested dependencies
- Snapshot of Application CR
- Manifest build artifacts (OCI image name and digest)



Get provenance from provenance store

```
$ provenance.sh -i ghcr.io/hirokuni-kitahara/sample-app-manifest:dev -k ~/.sigstore/cosign.pub
NAME          SIGNED   SIGNER   PROVENANCE
ghcr.io/hirokuni-kitahara/sample-app-manifest:dev  true      N/A       found

IMAGE DIGEST
sha256:64442cdc308687261d60e03929dbeac25e7bc9748eccf323648c0ad573022fdb

SOURCE REPO                                COMMIT ID
https://github.com/hirokuni-kitahara/sample-kustomize-app.git  5c6172b89473fbf15d8df8e4e61203cd06b87b49
https://github.com/hirokuni-kitahara/sample-kustomize-base.git  bb8adffb03c7d8be5b1ca0af5eb5ee1c27182db6

BUILD TYPE        BUILT AT           ADDITIONAL INFO
argocd-interlace 2022-10-13T00:01:21  manifest generated for Application "sample-application-cluster1"
$
```

Get provenance from provenance store

Source materials (url, commit id, etc.)

```
,"  
  "materials": [  
    {  
      "uri": "https://github.com/hirokuni-kitahara/sample-kustomize-app.git",  
      "digest": {  
        "commit": "8af6b72bb2232d500f6b8ca291fc221f89c522a6",  
        "path": "./",  
        "revision": "main"  
      }  
    },  
    {  
      "uri": "https://github.com/hirokuni-kitahara/sample-kustomize-base.git",  
      "digest": {  
        "commit": "bb8adfffb03c7d8be5b1ca0af5eb5ee1c27182db6",  
        "path": "",  
        "revision": ""  
      }  
    }  
  ]
```

Snapshot of Application CR (including status)

```
,  
  "parameters": {  
    "applicationSnapshot": "{\"kind\":\"Application\",  
  }
```

OCI Image reference, digest

```
  "subject": [  
    {  
      "name": "ghcr.io/hirokuni-kitahara/sample-app-manifest:dev",  
      "digest": {  
        "sha256": "b9b7ccdb6c10d7f9cf156f8b7257baf431e2524257b3ae51"  
      }  
    }  
  ],
```

Summary

- Modern CD GitOps engines dynamically generate YAML manifests, but it causes a problem of supply chain integrity
- YAML manifest signing improves the level of protection in delivery time.
- Interlace controller extends ArgoCD to fill the gap in manifest signing in CD GitOps.
- Trust but verify approach by Interlace makes application deployment transparent and accountable.

Thank you!

Q & A

Yuji Watanabe

twitter.com/ywatan1

linkedin.com/in/yuji-watanabe-0108322/

github.com/yuji-watanabe-jp

Hirokuni Kitahara

twitter.com/HirokuniKitaha1

linkedin.com/in/hirokuni-kitahara-a7450a165

github.com/hirokuni-kitahara

Resources

Interlace - <https://github.com/argoproj-labs/argocd-interlace>

Manifest Signing - <https://github.com/sigstore/k8s-manifest-sigstore>

Integrity Shield - <https://github.com/stolostron/integrity-shield>

Kyverno - <https://github.com/kyverno/kyverno>

DETROIT 2022



<https://sched.co/182HF>