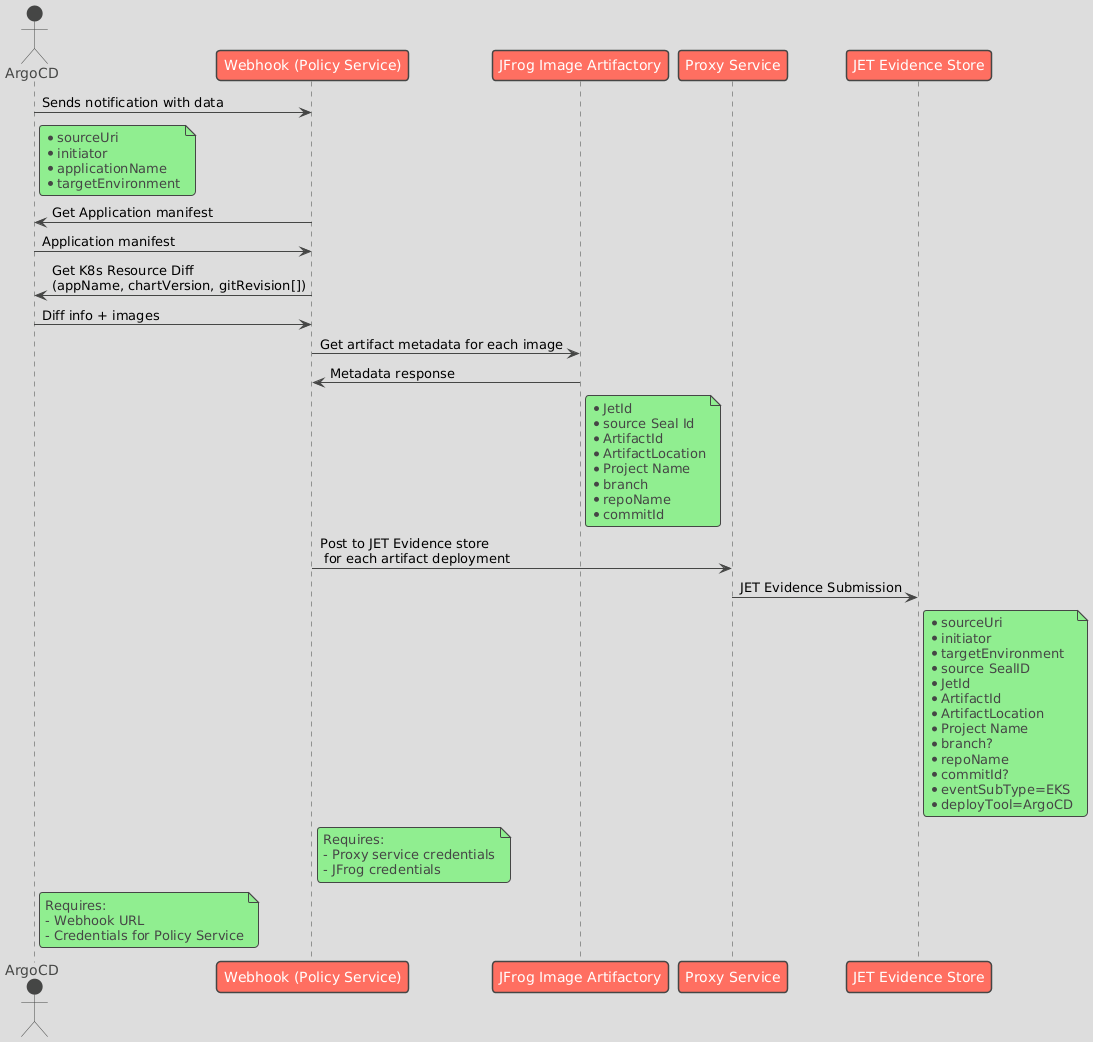
# Post-Deployment JET Evidence Submission​​

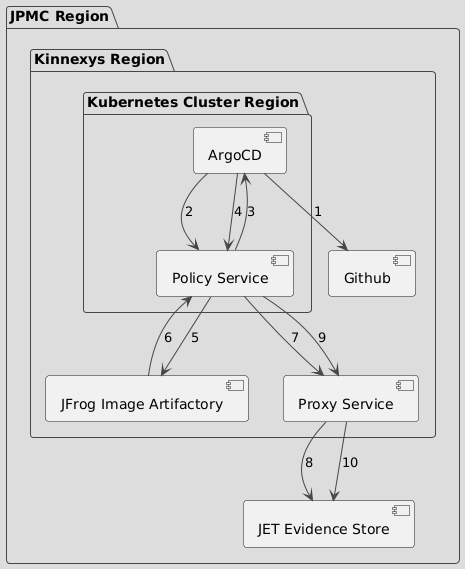
### Problem Statement:

Each deployment triggered by an ArgoCD Sync action must result in the submission of deployment evidence to the JET Evidence Store. The design outlines the automated workflow that ensures evidence is captured and submitted immediately following every ArgoCD sync operation.

The production environment additionally has the changewindow custom resource present in the cluster which should be used to submit release event evidence to JET Evidence.

### Staging:





1. ArgoCD checks for sync status with Application manifest in Git repo and the actual application deployed and accordingly deploys the application.
2. Argo CD sends a notification to the Policy Service webhook, with the policy service residing in same namespace. The notification includes: application name, sourceUri, initiator, targetEnvironment. The following config maps and secrets related to ArgoCD are to be edited as follows:

apiVersion: v1

kind: Secret

metadata:

name: argocd-notifications-secret

stringData:

notifiers.yaml: |

sample-webhook:

- name: policy-service-webhook

url: https://utkarsh858.free.beeceptor.com

type: Opaque



apiVersion: v1

kind: ConfigMap

metadata:

name: argocd-notifications-cm

namespace: argocd

uid: ab5299d0-d946-4171-bf8a-63055c424218

resourceVersion: '119240651'

creationTimestamp: '2024-12-05T17:11:18Z'

labels:

app.kubernetes.io/component: notifications-controller

app.kubernetes.io/name: argocd-notifications-controller

app.kubernetes.io/part-of: argocd

selfLink: /api/v1/namespaces/argocd/configmaps/argocd-notifications-cm

data:

trigger.sync-operation-change: |

- when: app.status.operationState.phase in ['Succeeded']

send: [sync-operation-completed]

- when: app.status.operationState.phase in ['Error', 'Failed']

send: [sync-operation-completed]

context: |

environment: dev

service.webhook.sample-webhook: |

url: https://utkarsh858.free.beeceptor.com/

headers: #optional headers

- name: Content-type

value: application/json

subscriptions: |

- recipients:

- sample-webhook

triggers:

- sync-operation-change

template.sync-operation-completed: |

webhook:

sample-webhook:

method: POST

path: '/submitDeployment'

body: |

{

{{if eq .app.status.operationState.phase "Succeeded"}} "eventStatus": "SUCCESS",{{end}}

{{if eq .app.status.operationState.phase "Error"}} "eventStatus": "FAILED",{{end}}

{{if eq .app.status.operationState.phase "Failed"}} "eventStatus": "FAILED",{{end}}

"sources":"{{.app.sync.sources}}",

"revisions":"{{.app.sync.revisions}}"

"sourceUri":"http://argocd.xyz.abc/"

}



1. Policy service login to ArgoCD via CLi binary by using ‘argocd login <argocd\_host> -–core’ option, it logins via Kubernetes API to ArgoCD.   
   The policy service must have the following role with required permissions bound to make the needed commands work. Or we can say that it can have the service account of application controller for it to have suitable permissions to get the data:

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

name: <role-name>

namespace: <argocd-namespace>

rules:

- verbs:

- get

- list

- watch

apiGroups:

- ''

resources:

- secrets

- configmaps

- verbs:

- create

- get

- list

- watch

- update

- patch

apiGroups:

- argoproj.io

resources:

- applications

- appprojects

- verbs:

- create

- list

apiGroups:

- ''

resources:

- events

- verbs:

- get

- list

- watch

apiGroups:

- apps

resources:

- deployments



1. The service then gets the application manifest by using ‘argocd get app <app\_name>’ and extracts the chartVersion and revision from the manifest. The application manifest are usually of two or more sources with first source being the helm chart and remaining sources being the git repositories containing the values.yaml files. The values inside revisions array in yaml correspond to those sources.

...

status:

history:

- ...

revisions:

- chart\_version (eg- 0.1.0)

- revision id (eg- ad76de8f9rf76g8rg8rg9bb)

- revision id 2 (eg- ijof9r4ji39rj4oir4r94gj)

...



Then the latest images deployed can be extracted with help of ‘argocd app diff <app-name> –source-positions 1 –revisions <chart\_version> –source-positions 2 –revisions <revision\_id> –source-positions 3 –revisions <revision\_id 2>

The output format is:

===== apps/Deployment hello-world/my-multi-source-app-hello-world ======

138c138

< replicas: 2

---

> replicas: 3

157c157

< - image: nginx:1.16.0

---

> - image: nginx:1.14.2



1. Policy Service get JFrog credentials and host from ‘REGCRED’ kubernetes secret.
2. Policy Service queries JFrog Image Artifactory for artifact metadata corresponding to each image.
3. JFrog Image Artifactory responds with metadata including: JetId, source sealID, ArtifactId, ArtifactLocation, Project Name, branch, repoName, and commitId.
4. Policy Service posts a record to the JET Evidence Store via Proxy Service for each artifact deployment. To connect to Proxy Service policy service will have endpoint and resource id details in its configuration.

JET Evidence:

POST /api/v0.0/deploy/generic/publish



Request payload: (verify git details)

{

"eventStatus": "SUCCESS",

"deployTool": "deploy system, DEFAULT TO JET",

"sealId": 107647,

"jetId": "70F27854ACAF95997AA30130B7A150AE8591C4AF05EE3B5389DC6EC057F624BB",

*"repoName": "ABCDRepo"*,

"projectName": "randomUnitProjectName",

*"branch": "feature/ABCD-1234",*

"eventSubType": "ANDROID",

*"commitId": "377560d4717b1d4456119fbf0793e4dab9a10ed8"*,

"sourceUri": "---identifier for evidance upload system-----",

"artifactId": "image123:sha256@1324654321354513521",

"artifactLocation": "artifactory/registry/image123",

"targetEnvironment": "dev",

"initiator": "user who uploaded the evidence",

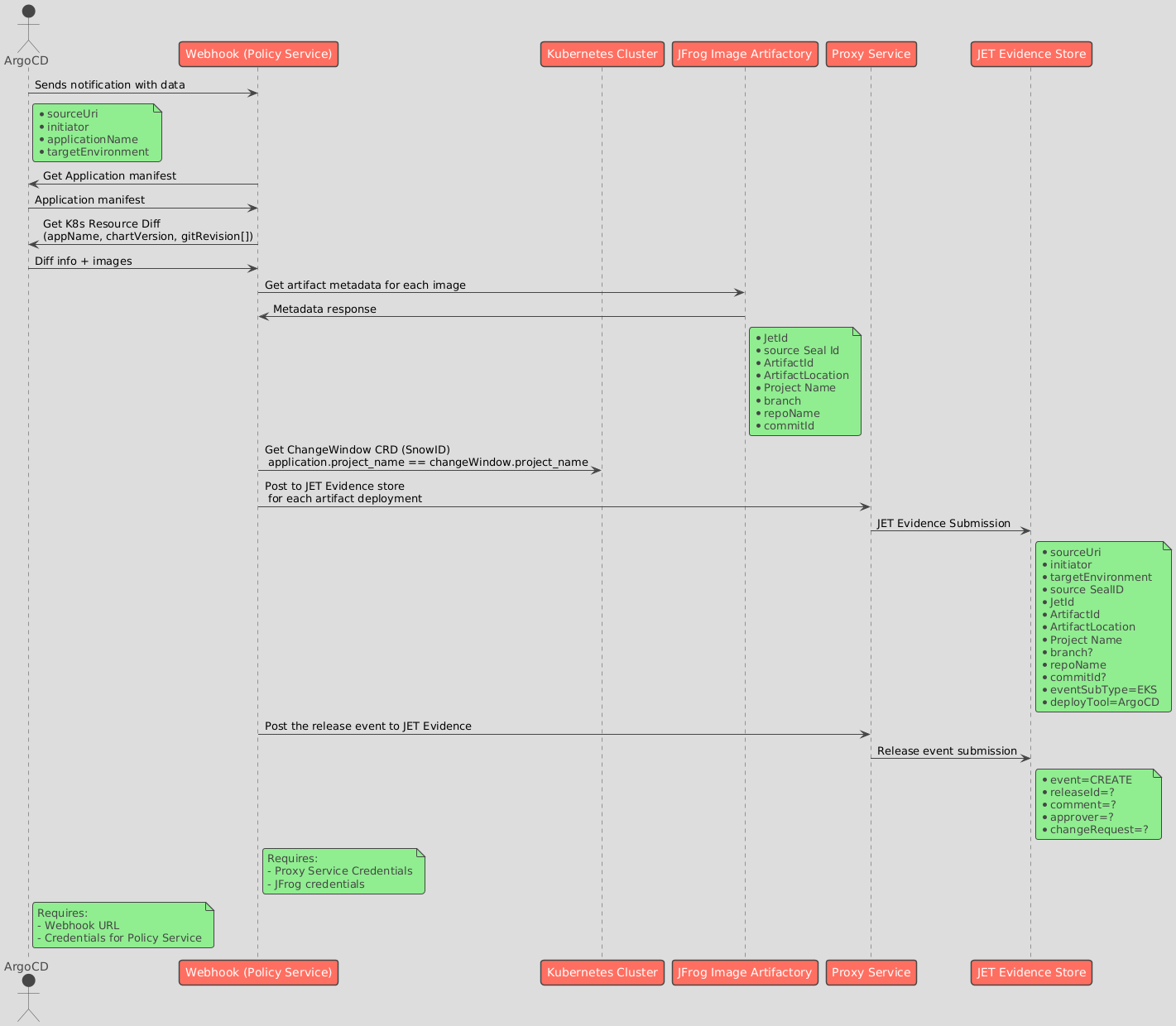
"extPayload":"----rawPayload RESERVED KEYWORD CAN NOT BE USED AS CUSTOM FIELD BY CLIENT----"

}



1. Each record includes: sourceUri, initiator, targetEnvironment, source SealID, JetId, ArtifactId, ArtifactLocation, Project Name, branch, repoName, commitId, eventSubType=EKS, and deployTool=ArgoCD.
2. Argo CD requires the webhook URL and credentials for Policy Service.
3. Any failure during evidence submission in policy-service is logged and alerted via production support system.

### Prod:

​​

There are some additional steps required in production environment, the policy service can submit evidence related to release to JET Evidence along with evidence of individual artifacts. To submit it will need a changeRequest ID i.e. snowID which it gets from changeWindowCRD.

The ChangeWindow CRD is present in same namespace as that of application.

POST /api/v0.0/releases/event



{

“Event”:”CREATE”,

“releaseID”:0,

“Comment”:””,

“Approver”:”<ArgoCD sync initiator”,

“changeRequest”:”<snow id>”

}



#### Requirements

* PolicyService must remain in same namespace as that of ArgoCD and applications
* Webhook of Policy Service along with suitable configmaps with message template must be configured in ArgoCD.
* PolicyService must have resources url of Proxy service so as to connect to JET Evidence store to access APIs.
* PolicyService must have JFrog creds in the cluster secrets.
* PolicyService must be configured with appropriate roles to be able to login to ArgoCD via Kubernetes API and to be able to get snowId from changewindowCRD.