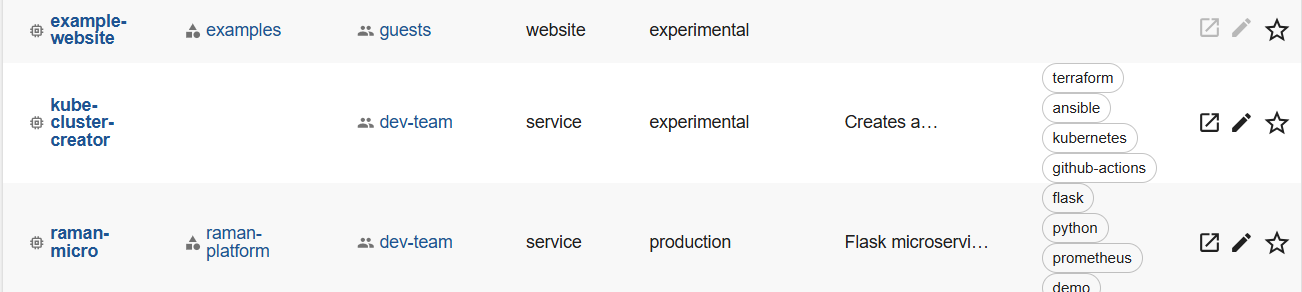
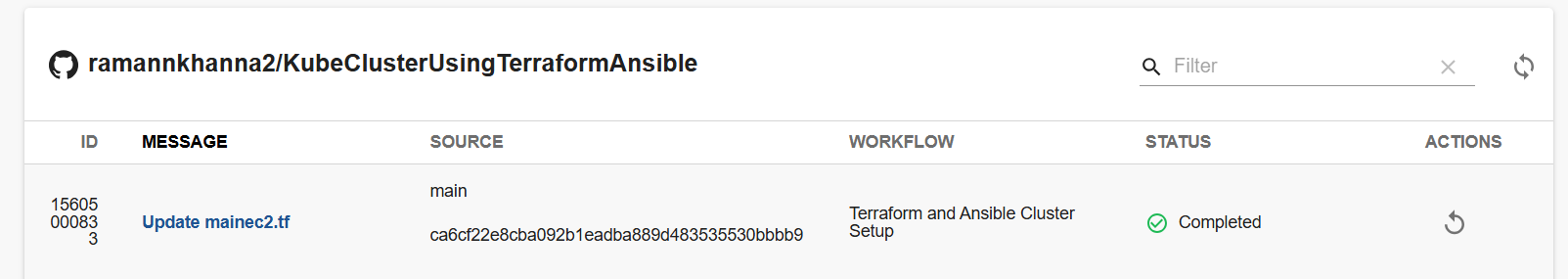
**Lab 16 : Integrate KubeClusterUsingTerraformAnsible into Backstage for One-Click Cluster Creation**

* Already have the setup of kubeadm cluster creation in <https://github.com/ramannkhanna2/KubeClusterUsingTerraformAnsible.git>
* Thers the catalog-info.yaml inside .
* Register it as a component in backstage with below url :
* <https://github.com/ramannkhanna2/KubeClusterUsingTerraformAnsible/blob/main/catalog-info.yaml>
* No u shud see your github actions one click workflow to setup kubeadm 3 node Kubernetes cluster .





**Lab 17 : Adding one click deploy of our ‘terraform-secure-infra’ to backstage.**

* **Upload all content of terraform-secure-infra to remote github repo**

(venv) root@ip-172-31-14-172:~/tf-infra# cat .github/workflows/provision.yml

name: Terraform Infra Provision via Root

on:

workflow\_dispatch:

jobs:

terraform:

runs-on: ubuntu-latest

name: Run Terraform from Remote Server as Root

steps:

- name: Checkout (this repo contains only workflow)

uses: actions/checkout@v3

- name: SSH into Server and Run Terraform as root

uses: appleboy/ssh-action@v1.0.0

with:

host: ${{ secrets.SSH\_HOST }}

username: ${{ secrets.SSH\_USER }} # typically 'ubuntu'

key: ${{ secrets.SSH\_PRIVATE\_KEY }}

script: |

echo "[+] Switching to root user..."

sudo -i <<EOF

echo "[+] Moved to root. Running Terraform..."

cd /root/terraform-secure-infra

terraform init

terraform plan -out=tfplan

terraform apply -auto-approve tfplan

EOF

Add github secrets :

SSH\_HOST :54.39.45.76

SSH\_USER : ubuntu

SSH\_PRIVATE\_KEY :

* Than go to github actions and test.
* Than add destroy automation as well :

(venv) root@ip-172-31-14-172:~/tf-infra# cat .github/workflows/destroy.yml

name: Terraform Infra Destroy

on:

workflow\_dispatch: # Manual trigger only

jobs:

destroy:

runs-on: ubuntu-latest

name: Destroy Infra on Remote via Terraform

steps:

- name: Checkout (optional if needed)

uses: actions/checkout@v3

- name: SSH into server and run Terraform destroy

uses: appleboy/ssh-action@v1.0.0

with:

host: ${{ secrets.SSH\_HOST }}

username: ${{ secrets.SSH\_USER }} # e.g., 'ubuntu'

key: ${{ secrets.SSH\_PRIVATE\_KEY }}

script: |

echo "[+] Switching to root to destroy infra..."

sudo -i <<EOF

cd /root/terraform-secure-infra

terraform init

terraform destroy -auto-approve

EOF

* Test destroy as well
* Now lets add this automation to backstage :

(venv) root@ip-172-31-14-172:~/tf-infra# cat catalog-info.yml

apiVersion: backstage.io/v1alpha1

kind: Component

metadata:

name: tf-infra

description: |

Terraform-based secure infrastructure automation with GitHub Actions.

Provision and destroy infrastructure.

tags:

- terraform

- infrastructure

- github-actions

annotations:

github.com/project-slug: ramannkhanna2/tf-infra

backstage.io/techdocs-ref: dir:.

# Enables GitHub Actions plugin integration

github.com/actions: ramannkhanna2/tf-infra

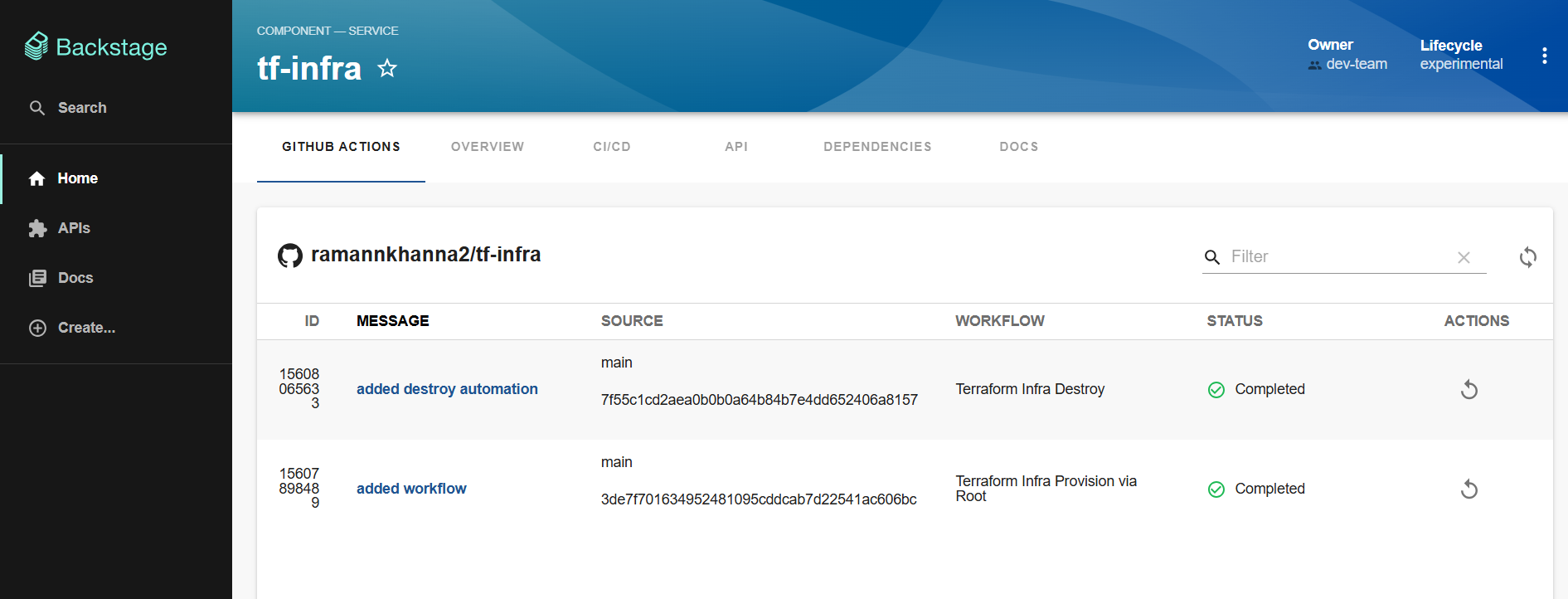
spec:

type: service

lifecycle: experimental

owner: dev-team

* Add above and push to tf-infra and than register to backstage .
* <https://github.com/ramannkhanna2/tf-infra/blob/main/catalog-info.yml>



**Lab 18 : Gatekeeper + GitOps(argoCD) + Backstage Integration**

**📦 Pre-Requirements**

| **Component** | **Status** | **Notes** |
| --- | --- | --- |
| 🧠 Kubeadm cluster | ✅ Ready | Already provisioned by you |
| 🧪 Helm | ✅ Installed | helm version |
| 🔐 OPA Gatekeeper | ✅ We'll install it | Policy enforcement |
| 📁 ArgoCD | ⏳ To be installed | GitOps sync engine |
| 🐙 GitHub Repo | ✅ You have it | Your raman-micro or Helm app |
| 🎛️ Backstage | ✅ Running | We’ll add ArgoCD plugin |

**✅ Phase 1: Enforce Gatekeeper Policies (Cluster Level)**

* Increase capacity of all nodes to 2 vcpus , 4 gb ram atlease for all nodes

**🧱 Step 1.1: Install Gatekeeper**

kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper/release-3.12/deploy/gatekeeper.yaml

Verify:

bash

CopyEdit

kubectl get pods -n gatekeeper-system

Wait for all pods (controller, audit) to be Running.

* Create a github repo named kube+gatekeeper+argo and clone it to add all files in it

**🚫 Step 1.2: Create ConstraintTemplate: Block Untrusted Image Registry**

root@ip-172-31-27-88:~/kube-gatekeeper-argo# cat template-image-policy.yaml

apiVersion: templates.gatekeeper.sh/v1beta1

kind: ConstraintTemplate

metadata:

name: k8sallowedrepos

spec:

crd:

spec:

names:

kind: K8sAllowedRepos

targets:

- target: admission.k8s.gatekeeper.sh

rego: |

package k8sallowedrepos

violation[{"msg": msg}] {

container := input.review.object.spec.containers[\_]

not startswith(container.image, "docker.io/raman")

msg := sprintf("container image '%v' is not from allowed registry", [container.image])

}

Apply:

bash

CopyEdit

kubectl apply -f template-image-policy.yaml

|  |  |
| --- | --- |
| template-image-policy.yaml | Defines *how* to validate container images |

|  |  |
| --- | --- |
| constraint-image-policy.yaml | Activates enforcement for real objects like Pods |

**⚙️ Step 1.3: Apply Constraint: Only Allow docker.io/raman**

# constraint-image-policy.yaml

apiVersion: constraints.gatekeeper.sh/v1beta1

kind: K8sAllowedRepos

metadata:

name: allowed-docker-raman

spec:

match:

kinds:

- apiGroups: [""]

kinds: ["Pod"]

parameters: {}

bash

CopyEdit

kubectl apply -f constraint-image-policy.yaml

**🔐 You now have:**

**✅ A real-time Kubernetes admission control policy:**

❌ Blocks **all Pods** that do **not** use images from docker.io/raman

**💡 Step 1.4: Add Another Constraint — Require CPU & Memory Limits**

# template-limits-required.yaml

apiVersion: templates.gatekeeper.sh/v1beta1

kind: ConstraintTemplate

metadata:

name: k8srequiredresources

spec:

crd:

spec:

names:

kind: K8sRequiredResources

targets:

- target: admission.k8s.gatekeeper.sh

rego: |

package k8srequiredresources

violation[{"msg": msg}] {

container := input.review.object.spec.containers[\_]

not container.resources.limits.cpu

msg := "Missing CPU limit"

}

violation[{"msg": msg}] {

container := input.review.object.spec.containers[\_]

not container.resources.limits.memory

msg := "Missing memory limit"

}

bash

CopyEdit

kubectl apply -f template-limits-required.yaml

You’ve now successfully created a **second Gatekeeper ConstraintTemplate**:

bash

CopyEdit

✅ constrainttemplate.templates.gatekeeper.sh/k8srequiredresources created

This policy will enforce that **all Pods must define both cpu and memory limits**, which is a **crucial best practice** for Kubernetes multi-tenancy and resource fairness.

Constraint:

yaml

CopyEdit

# constraint-limits.yaml

apiVersion: constraints.gatekeeper.sh/v1beta1

kind: K8sRequiredResources

metadata:

name: require-cpu-memory-limits

spec:

match:

kinds:

- apiGroups: [""]

kinds: ["Pod"]

bash

CopyEdit

kubectl apply -f constraint-limits.yaml

✅ This means your cluster will now **block any Pod that doesn’t define both cpu and memory limits** for every container.

**🧪 Optional Test: Validate Enforcement**

Here’s a **negative test** to confirm Gatekeeper is enforcing the rule:

yaml

CopyEdit

# no-limits.yaml

apiVersion: v1

kind: Pod

metadata:

name: no-limits

spec:

containers:

- name: busybox

image: docker.io/raman/busybox:latest

command: ["sleep", "3600"]

Try to apply:

bash

CopyEdit

kubectl apply -f no-limits.yaml

You should see:

pgsql

CopyEdit

Error from server (Forbidden): admission webhook "validation.gatekeeper.sh" denied the request: [Missing CPU limit, Missing memory limit]

**✅ Current Setup Already Ensures**

| **Layer** | **Tool** | **Enforces Policy?** |
| --- | --- | --- |
| Kubernetes | OPA Gatekeeper | ✅ Denies at admission |

Your cluster will now:

* ❌ Reject any pod without resources.limits
* ❌ Reject any image not starting with docker.io/raman

Which means:

**No manifest can ever be deployed to your cluster if it violates the rules.**

**✅ Goal: GitOps + Backstage Integration for raman-micro**

We’ll now:

1. 🔁 Use **ArgoCD** to GitOps-deploy your k8s/\*.yaml to the cluster
2. 🔍 Hook that into **Backstage** so devs can visually see:
   * Sync status
   * Health
   * Drift

**🚀 Phase 3: GitOps with ArgoCD**

**🔧 Prerequisites Recap**

| **Requirement** | **Status** |
| --- | --- |
| ArgoCD installed (namespace argocd) | ✅ TobeDone |
| raman-micro GitHub repo | ✅ Exists |
| Manifests under k8s/ | ✅ TobeDone |
| Gatekeeper enforcing image + limits | ✅ Active |

Let's create a proper k8s/ directory in your raman-micro GitHub repo with fully compliant Kubernetes manifests that:

* ✅ Use docker.io/raman/raman-micro image
* ✅ Include CPU & memory limits
* ✅ Pass Gatekeeper constraints
* ✅ Are deployable via ArgoCD

**📁 Folder Structure to Add to Your Repo**

markdown

CopyEdit

raman-micro/

└── k8s/

├── deployment.yaml

└── service.yaml

**📄 k8s/deployment.yaml**

yaml

CopyEdit

apiVersion: apps/v1

kind: Deployment

metadata:

name: raman-micro

labels:

app: raman-micro

spec:

replicas: 1

selector:

matchLabels:

app: raman-micro

template:

metadata:

labels:

app: raman-micro

spec:

containers:

- name: raman-micro

image: docker.io/raman/raman-micro:latest

ports:

- containerPort: 5000

resources:

limits:

cpu: "250m"

memory: "256Mi"

**📄 k8s/service.yaml**

yaml

CopyEdit

apiVersion: v1

kind: Service

metadata:

name: raman-micro

spec:

selector:

app: raman-micro

ports:

- protocol: TCP

port: 80

targetPort: 5000

type: NodePort

**✅ What to Do Next**

**🔧 Step 1: Add These Files to Your GitHub Repo raman-micro**

git add .

git commit -m "Add k8s manifests for ArgoCD sync"

git push origin main

* **Install argocd CLI**

Run the following on your Ubuntu (or Amazon Linux) system:

bash

CopyEdit

# Download latest ArgoCD CLI (v2.11.3 as of June 2025)

VERSION=$(curl -s https://api.github.com/repos/argoproj/argo-cd/releases/latest | grep tag\_name | cut -d '"' -f 4)

curl -sSL -o argocd "https://github.com/argoproj/argo-cd/releases/download/${VERSION}/argocd-linux-amd64"

chmod +x argocd

sudo mv argocd /usr/local/bin/

Verify install:

bash

CopyEdit

argocd version

**✅ Yes, the ArgoCD UI runs as a pod in your Kubernetes cluster.**

When you installed ArgoCD using:

kubectl create namespace argocd

kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml

This deployed all ArgoCD components — including the **UI (argocd-server)** — into your cluster.

**📦 ArgoCD Core Components (All Run as Pods)**

You can check with:

bash

CopyEdit

kubectl get pods -n argocd

* You will notice pods will not get created due to the gatekeeper policy

❌ Problem: Gatekeeper is Blocking ArgoCD Pods

**✅ Disable the policy temporarily ; will activate again after argo cd installation.**

Only if you're still testing.

bash

CopyEdit

Kubectl get constraint

kubectl delete k8srequiredresources.constraints.gatekeeper.sh require-cpu-memory-limits

kubectl delete k8sallowedrepos.constraints.gatekeeper.sh allowed-docker-raman

* Again install argocd :

kubectl delete -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml

kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml

* Kubectl get pods -n argocd
* Now again enable gatekeeper constraints :

root@ip-172-31-27-88:~/kube-gatekeeper-argo# k apply -f constraint-image-policy.yaml

k8sallowedrepos.constraints.gatekeeper.sh/allowed-docker-raman created

root@ip-172-31-27-88:~/kube-gatekeeper-argo# k apply -f constraint-limit.yaml

k8srequiredresources.constraints.gatekeeper.sh/require-cpu-memory-limits created

root@ip-172-31-27-88:~/kube-gatekeeper-argo# k get constraints

NAME ENFORCEMENT-ACTION TOTAL-VIOLATIONS

k8sallowedrepos.constraints.gatekeeper.sh/allowed-docker-raman

NAME ENFORCEMENT-ACTION TOTAL-VIOLATIONS

k8srequiredresources.constraints.gatekeeper.sh/require-cpu-memory-limits

* To expose argocd :

root@ip-172-31-27-88:~/kube-gatekeeper-argo# k edit -n argocd svc argocd-server

service/argocd-server edited

root@ip-172-31-27-88:~/kube-gatekeeper-argo# k get svc -n argocd argocd-server

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

argocd-server NodePort 10.99.164.110 <none> 80:30786/TCP,443:31194/TCP 4m51s

Let’s now help you **log into the UI** — ArgoCD uses:

* **Username:** admin
* **Password:** stored in a Kubernetes **secret** (base64-encoded)

**✅ Step-by-Step: Get ArgoCD Admin Credentials**

**Get the admin password :**

kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d && echo

* **Login argocd using cli as well :**

root@ip-172-31-27-88:~/kube-gatekeeper-argo# argocd login 15.228.172.48:30786 --username admin --password V9kGvXX0SH1Fw7Fn --insecure

'admin:login' logged in successfully

Context '15.228.172.48:30786' updated

**📦 Step 3.1: Create ArgoCD App for raman-micro**

Replace YOUR\_GITHUB\_URL if needed.

bash

CopyEdit

argocd app create raman-micro \

--repo https://github.com/ramannkhanna2/raman-micro.git \

--path k8s \

--dest-server https://kubernetes.default.svc \

--dest-namespace default \

--sync-policy automated \

--self-heal \

--auto-prune

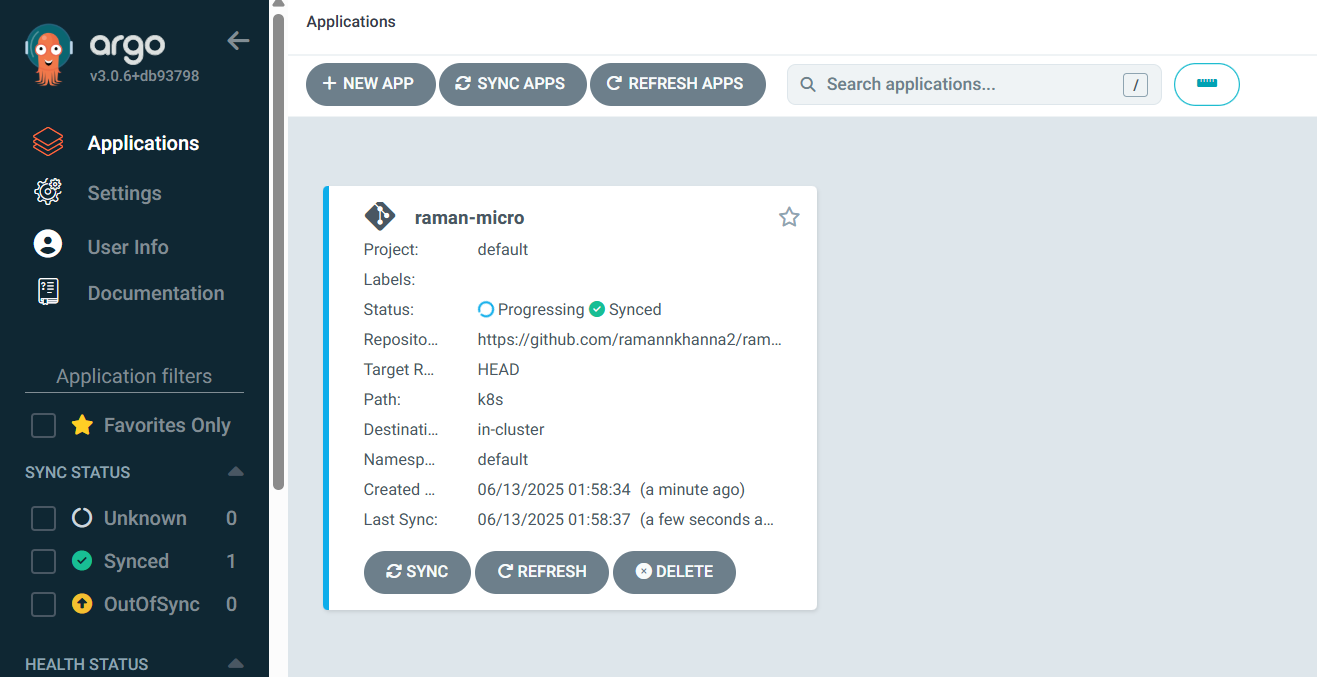
Explanation:

* --path k8s: we’re using plain manifests, not Helm
* --sync-policy automated: auto-deploys when Git changes
* --self-heal: ensures drifted resources are reconciled
* --auto-prune: deletes removed resources from Git

root@ip-172-31-27-88:~/kube-gatekeeper-argo# k get application -n argocd

NAME SYNC STATUS HEALTH STATUS

raman-micro Synced Progressing



**🔄 Step 3.2: Trigger a Sync (optional)**

bash

CopyEdit

argocd app sync raman-micro

* **Check if argo cd setup is working fine by making changes on raman-micro/k8s github repo and se eif its automatically replicating on the kubecluster using argi cd ..**
* **If fails check for the right image from dockerhub and update in raman-micro github repo i.e : ramann123/raman-micro:latest**

**🧭 Phase 4: Backstage + ArgoCD Plugin Integration**

* <https://backstage.io/plugins/>
* https://roadie.io/backstage/plugins/argo-cd/?utm\_source=backstage.io&utm\_medium=marketplace&utm\_campaign=argo-cd

Install the plugin into Backstage.

cd /root/backstage-app/ramanapp/packages/app

yarn add @roadiehq/backstage-plugin-argo-cd

Add proxy config to the app-config.yaml file

proxy:

'/argocd/api':

target: https://<your-argocd-instance>/api/v1/

changeOrigin: **true**

# only if your argocd api has self-signed cert

secure: **false**

headers:

Cookie:

$env: ARGOCD\_AUTH\_TOKEN

For above generate token and save as env :

bash

CopyEdit

argocd account generate-token

Add argoCD widget to your overview page

// packages/app/src/components/catalog/EntityPage.tsx

import {

EntityArgoCDOverviewCard,

isArgocdAvailable

} from '@roadiehq/backstage-plugin-argo-cd';

const overviewContent = (

<Grid container spacing={3} alignItems="stretch">

...

<EntitySwitch>

<EntitySwitch.Case if={e => Boolean(isArgocdAvailable(e))}>

<Grid item sm={4}>

<EntityArgoCDOverviewCard />

</Grid>

</EntitySwitch.Case>

</EntitySwitch>

...

</Grid>

);

Add annotation to the yaml config file of a component

metadata:

annotations:

argocd/app-name: <your-app-name>

Get and provide **ARGOCD\_AUTH\_TOKEN** as env variable in following format

ARGOCD\_AUTH\_TOKEN='argocd.token=<token>'

**🔄 Step 4.5: Restart Backstage**

bash

CopyEdit

cd ~/backstage-app/raman-app

yarn start

Go to:  
🧭 Backstage → raman-micro → ArgoCD tab

✅ You should now see:

* Sync Status
* Drift
* Health
* Last sync timestamp
* Commit SHA

**🎉 You Now Have:**

| **Capability** | **Tool** | **Status** |
| --- | --- | --- |
| GitOps deployment | ArgoCD | ✅ |
| Admission policy enforcement | Gatekeeper | ✅ |
| Visual deployment status | Backstage | ✅ |
| Real-time sync & drift detection | ArgoCD UI | ✅ |