Microservice CI/CD on AWS EKS with Argo CD (GitOps), SonarQube, Trivy, Docker Hub — Full, Working Template

This repo is a minimal end-to-end example with **3 microservices** and a **CI/CD pipeline**:

- **CI** (GitHub Actions): `git push` → **SonarQube** static analysis → **unit tests** → **Docker build** → **Trivy vulnerability scan** → **push images to Docker Hub** → **update GitOps manifests** (kustomize) → commit `[skip ci]`
- **CD** (GitOps via Argo CD): **EKS** created by **Terraform** → **Argo CD** installed by Terraform (Helm) → Argo watches the `gitops/` paths → GitHub webhook to Argo `/api/webhook` → **Prometheus + Grafana** installed by Terraform (Helm) for cluster/app monitoring.

  Replace placeholders like `YOUR_GITHUB_REPO_URL` and `YOUR_DOCKERHUB_USERNAME` in the files below (or follow the step-by-step guide in the chat).

Repo Layout

```
.
├── app/
│   ├── service-a/           # Node/Express (port 3000)
│   ├── service-b/           # Python/Flask (port 5000)
│   └── service-c/           # Go net/http (port 8080)
│
├── gitops/
│   ├── apps/
│   │   ├── service-a/
│   │   ├── service-b/
│   │   └── service-c/
│   └── argocd/
│       └── applications.yaml  # Argo CD Applications for the 3 services
│
├── infra/
│   └── terraform/
│       ├── providers.tf
│       ├── variables.tf
│       ├── vpc.tf
│       ├── eks.tf
│       ├── outputs.tf
│       ├── helm-argocd.tf
│       ├── helm-monitoring.tf
│       ├── k8s-namespaces.tf
│       ├── k8s-secrets.tf
│       └── terraform.tfvars.example
```

```
│
└── .github/
    └── workflows/
        └── ci.yml
```

Microservices (3)

`app/service-a` **(Node/Express)**

`app/service-a/package.json`

```
{
  "name": "service-a",
  "version": "1.0.0",
  "main": "src/index.js",
  "type": "commonjs",
  "scripts": {
    "start": "node src/index.js",
    "test": "node --test"
  },
  "dependencies": {
    "express": "^4.19.2"
  }
}
```

`app/service-a/src/index.js`

```
const express = require('express');
const app = express();
const port = process.env.PORT || 3000;

app.get('/', (req, res) => res.json({ service: 'a', message: 'ok' }));
app.get('/health', (req, res) => res.send('ok'));

app.listen(port, () => console.log(`Service A listening on ${port}`));
```

`app/service-a/test/app.test.js`

```
import test from 'node:test';
import assert from 'node:assert/strict';

test('math works', () => {
  assert.equal(1 + 1, 2);
});
```

`app/service-a/Dockerfile`

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install --omit=dev
COPY src ./src
EXPOSE 3000
CMD ["npm", "start"]
```

`app/service-a/.dockerignore`

```
node_modules
npm-cache
.git
```

`app/service-a/sonar-project.properties`

```
sonar.projectKey=service-a
sonar.projectName=service-a
sonar.sources=src
sonar.tests=test
sonar.javascript.lcov.reportPaths=coverage/lcov.info
sonar.host.url=${env.SONAR_HOST_URL}
sonar.login=${env.SONAR_TOKEN}
sonar.organization=${env.SONAR_ORG}
```

`app/service-b` (Python/Flask)

`app/service-b/app.py`

```
from flask import Flask, jsonify
import os

app = Flask(__name__)

@app.route("/")
def home():
    return jsonify(service="b", message="ok")

@app.route("/health")
def health():
    return "ok"
```

```
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=int(os.environ.get("PORT", 5000)))
```

app/service-b/requirements.txt

```
flask==3.0.3
pytest==8.3.2
```

app/service-b/tests/test_app.py

```
def test_math():
    assert 2 + 2 == 4
```

app/service-b/Dockerfile

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py .
ENV PORT=5000
EXPOSE 5000
CMD ["python", "app.py"]
```

app/service-b/.dockerignore

```
__pycache__
.git
```

app/service-b/sonar-project.properties

```
sonar.projectKey=service-b
sonar.projectName=service-b
sonar.sources=.
sonar.exclusions=tests/**
sonar.python.version=3.11
sonar.host.url=${env.SONAR_HOST_URL}
sonar.login=${env.SONAR_TOKEN}
sonar.organization=${env.SONAR_ORG}
```

`app/service-c` **(Go)**

`app/service-c/go.mod`

```
module service-c

go 1.21
```

`app/service-c/main.go`

```
package main

import (
    "fmt"
    "net/http"
    "os"
)

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        w.Header().Set("Content-Type", "application/json")
        w.Write([]byte(`{"service":"c","message":"ok"}`))
    })
    http.HandleFunc("/health", func(w http.ResponseWriter, r *http.Request) {
        fmt.Fprint(w, "ok")
    })

    port := os.Getenv("PORT")
    if port == "" { port = "8080" }
    fmt.Println("Service C listening on", port)
    http.ListenAndServe(":"+port, nil)
}
```

`app/service-c/main_test.go`

```
package main

import "testing"

func TestMath(t *testing.T) {
    if 3*3 != 9 { t.Fatalf("math broke") }
}
```

`app/service-c/Dockerfile`

```
FROM golang:1.21-alpine AS build
WORKDIR /src
COPY go.mod ./
COPY . .
RUN go build -o /bin/service

FROM alpine:3.19
EXPOSE 8080
COPY --from=build /bin/service /service
CMD ["/service"]
```

`app/service-c/.dockerignore`

```
.git
```

`app/service-c/sonar-project.properties`

```
sonar.projectKey=service-c
sonar.projectName=service-c
sonar.sources=.
sonar.exclusions=**/*_test.go
sonar.host.url=${env.SONAR_HOST_URL}
sonar.login=${env.SONAR_TOKEN}
sonar.organization=${env.SONAR_ORG}
```

GitOps Manifests (Kustomize)

> **Note:** The CI workflow will update `newTag` to the latest image SHA automatically and commit with `[skip ci]` to avoid loops.

`gitops/apps/service-a/kustomization.yaml`

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - deployment.yaml
  - service.yaml
images:
  - name: YOUR_DOCKERHUB_USERNAME/service-a
    newTag: latest
namespace: apps
```

`gitops/apps/service-a/deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: service-a
  labels: { app: service-a }
  namespace: apps
spec:
  replicas: 2
  selector:
    matchLabels: { app: service-a }
  template:
    metadata:
      labels: { app: service-a }
    spec:
      imagePullSecrets:
        - name: dockerhub-regcred
      containers:
        - name: service-a
          image: YOUR_DOCKERHUB_USERNAME/service-a:latest
          ports:
            - containerPort: 3000
          readinessProbe:
            httpGet: { path: /health, port: 3000 }
            initialDelaySeconds: 3
          livenessProbe:
            httpGet: { path: /health, port: 3000 }
            initialDelaySeconds: 10
```

`gitops/apps/service-a/service.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: service-a
  namespace: apps
spec:
  type: LoadBalancer
  selector:
    app: service-a
  ports:
    - port: 80
      targetPort: 3000
```

`gitops/apps/service-b/kustomization.yaml`

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
```

```
resources:
  - deployment.yaml
  - service.yaml
images:
  - name: YOUR_DOCKERHUB_USERNAME/service-b
    newTag: latest
namespace: apps
```

`gitops/apps/service-b/deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: service-b
  labels: { app: service-b }
  namespace: apps
spec:
  replicas: 2
  selector:
    matchLabels: { app: service-b }
  template:
    metadata:
      labels: { app: service-b }
    spec:
      imagePullSecrets:
        - name: dockerhub-regcred
      containers:
        - name: service-b
          image: YOUR_DOCKERHUB_USERNAME/service-b:latest
          ports:
            - containerPort: 5000
          readinessProbe:
            httpGet: { path: /health, port: 5000 }
            initialDelaySeconds: 3
          livenessProbe:
            httpGet: { path: /health, port: 5000 }
            initialDelaySeconds: 10
```

`gitops/apps/service-b/service.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: service-b
  namespace: apps
spec:
  type: LoadBalancer
  selector:
```

```
    app: service-b
  ports:
    - port: 80
      targetPort: 5000
```

`gitops/apps/service-c/kustomization.yaml`

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - deployment.yaml
  - service.yaml
images:
  - name: YOUR_DOCKERHUB_USERNAME/service-c
    newTag: latest
namespace: apps
```

`gitops/apps/service-c/deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: service-c
  labels: { app: service-c }
  namespace: apps
spec:
  replicas: 2
  selector:
    matchLabels: { app: service-c }
  template:
    metadata:
      labels: { app: service-c }
    spec:
      imagePullSecrets:
        - name: dockerhub-regcred
      containers:
        - name: service-c
          image: YOUR_DOCKERHUB_USERNAME/service-c:latest
          ports:
            - containerPort: 8080
          readinessProbe:
            httpGet: { path: /health, port: 8080 }
            initialDelaySeconds: 3
          livenessProbe:
            httpGet: { path: /health, port: 8080 }
            initialDelaySeconds: 10
```

`gitops/apps/service-c/service.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: service-c
  namespace: apps
spec:
  type: LoadBalancer
  selector:
    app: service-c
  ports:
    - port: 80
      targetPort: 8080
```

`gitops/argocd/applications.yaml` **(Argo CD watches your repo)**

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: service-a
  namespace: argocd
spec:
  project: default
  source:
    repoURL: YOUR_GITHUB_REPO_URL
    targetRevision: main
    path: gitops/apps/service-a
  destination:
    server: https://kubernetes.default.svc
    namespace: apps
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
---
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: service-b
  namespace: argocd
spec:
  project: default
  source:
    repoURL: YOUR_GITHUB_REPO_URL
    targetRevision: main
    path: gitops/apps/service-b
  destination:
    server: https://kubernetes.default.svc
    namespace: apps
  syncPolicy:
```

```
    automated:
      prune: true
      selfHeal: true
---
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: service-c
  namespace: argocd
spec:
  project: default
  source:
    repoURL: YOUR_GITHUB_REPO_URL
    targetRevision: main
    path: gitops/apps/service-c
  destination:
    server: https://kubernetes.default.svc
    namespace: apps
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

Terraform — AWS + EKS + Argo CD + Monitoring

`infra/terraform/providers.tf`

```
terraform {
  required_version = ">= 1.6.0"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
    kubernetes = {
      source  = "hashicorp/kubernetes"
      version = "~> 2.25"
    }
    helm = {
      source  = "hashicorp/helm"
      version = "~> 2.11"
    }
  }
}

provider "aws" {
  region = var.aws_region
}
```

```
# EKS cluster auth for kubernetes/helm providers
data "aws_eks_cluster" "this" {
  name = module.eks.cluster_name
}

data "aws_eks_cluster_auth" "this" {
  name = module.eks.cluster_name
}

provider "kubernetes" {
  host                   = data.aws_eks_cluster.this.endpoint
  cluster_ca_certificate =
base64decode(data.aws_eks_cluster.this.certificate_authority[0].data)
  token                  = data.aws_eks_cluster_auth.this.token
}

provider "helm" {
  kubernetes {
    host                   = data.aws_eks_cluster.this.endpoint
    cluster_ca_certificate =
base64decode(data.aws_eks_cluster.this.certificate_authority[0].data)
    token                  = data.aws_eks_cluster_auth.this.token
  }
}
```

infra/terraform/variables.tf

```
variable "name" { description = "Name prefix" type = string default =
"demo" }
variable "aws_region" { description = "AWS region" type = string default =
"ap-south-1" }

variable "node_instance_types" { type = list(string) default =
["t3.medium"] }
variable "node_min_size" { type = number default = 1 }
variable "node_desired_size" { type = number default = 2 }
variable "node_max_size" { type = number default = 3 }

variable "git_repository_url" { description = "Git repo URL for Argo CD
(HTTPS)" type = string }

variable "dockerhub_username" { type = string }
variable "dockerhub_password" { type = string }
variable "dockerhub_email" { type = string default = "example@example.com" }

variable "grafana_admin_password" { type = string }
```

`infra/terraform/vpc.tf`

```
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "~> 5.0"

  name = "${var.name}-vpc"
  cidr = "10.0.0.0/16"

  azs             = ["${var.aws_region}a", "${var.aws_region}b", "$
{var.aws_region}c"]
  public_subnets  = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  private_subnets = ["10.0.11.0/24", "10.0.12.0/24", "10.0.13.0/24"]

  enable_nat_gateway = true
  single_nat_gateway = true

  tags = { Project = var.name }
}
```

`infra/terraform/eks.tf`

```
module "eks" {
  source  = "terraform-aws-modules/eks/aws"
  version = "~> 20.0"

  cluster_name    = "${var.name}-eks"
  cluster_version = "1.29"

  vpc_id     = module.vpc.vpc_id
  subnet_ids = module.vpc.private_subnets

  enable_irsa = true

  eks_managed_node_groups = {
    default = {
      ami_type       = "AL2_x86_64"
      instance_types = var.node_instance_types
      min_size       = var.node_min_size
      desired_size   = var.node_desired_size
      max_size       = var.node_max_size
    }
  }

  tags = { Project = var.name }
}
```

infra/terraform/k8s-namespaces.tf

```
resource "kubernetes_namespace" "apps" {
  metadata { name = "apps" }
}
```

infra/terraform/k8s-secrets.tf

```
# Docker Hub pull secret for the apps namespace
resource "kubernetes_secret" "dockerhub" {
  metadata {
    name      = "dockerhub-regcred"
    namespace = kubernetes_namespace.apps.metadata[0].name
  }
  type = "kubernetes.io/dockerconfigjson"

  data = {
    ".dockerconfigjson" = base64encode(jsonencode({
      auths = {
        "https://index.docker.io/v1/" = {
          username = var.dockerhub_username
          password = var.dockerhub_password
          email    = var.dockerhub_email
          auth     = base64encode("${var.dockerhub_username}:$
{var.dockerhub_password}")
        }
      }
    }))
  }
}
```

infra/terraform/helm-argocd.tf

```
# Install Argo CD via Helm and expose server as LoadBalancer
resource "helm_release" "argocd" {
  name             = "argocd"
  repository       = "https://argoproj.github.io/argo-helm"
  chart            = "argo-cd"
  namespace        = "argocd"
  create_namespace = true

  # Optionally pin a chart version, e.g.:
  # version = "7.6.1"

  values = [<<-YAML
    server:
      service:
        type: LoadBalancer
```

```
    dex:
      enabled: false
  YAML]
}

# Apply Argo CD Applications so Argo watches gitops/ paths
resource "kubernetes_manifest" "argocd_apps" {
  manifest = yamldecode(templatefile("${path.module}/templates/argocd-
apps.tmpl.yaml", {
    repo_url = var.git_repository_url
  }))

  depends_on = [helm_release.argocd, kubernetes_namespace.apps]
}
    server:
      service:
        type: LoadBalancer
    dex:
      enabled: false
  YAML]
}

# Apply Argo CD Applications so Argo watches gitops/ paths
resource "kubernetes_manifest" "argocd_apps" {
  manifest = yamldecode(templatefile("${path.module}/templates/argocd-
apps.tmpl.yaml", {
    repo_url = var.git_repository_url
  }))

  depends_on = [helm_release.argocd, kubernetes_namespace.apps]
}
```

infra/terraform/templates/argocd-apps.tmpl.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: argocd
---
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: service-a
  namespace: argocd
spec:
  project: default
  source:
    repoURL: ${repo_url}
    targetRevision: main
    path: gitops/apps/service-a
```

```
  destination:
    server: https://kubernetes.default.svc
    namespace: apps
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
---
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: service-b
  namespace: argocd
spec:
  project: default
  source:
    repoURL: ${repo_url}
    targetRevision: main
    path: gitops/apps/service-b
  destination:
    server: https://kubernetes.default.svc
    namespace: apps
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
---
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: service-c
  namespace: argocd
spec:
  project: default
  source:
    repoURL: ${repo_url}
    targetRevision: main
    path: gitops/apps/service-c
  destination:
    server: https://kubernetes.default.svc
    namespace: apps
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

`infra/terraform/helm-monitoring.tf`

```
# kube-prometheus-stack (Prometheus + Grafana)
resource "helm_release" "kps" {
```

```
  name             = "kube-prometheus-stack"
  repository       = "https://prometheus-community.github.io/helm-charts"
  chart            = "kube-prometheus-stack"
  namespace        = "monitoring"
  create_namespace = true

  values = [<<-YAML
    grafana:
      adminPassword: ${var.grafana_admin_password}
      service:
        type: LoadBalancer
    prometheus:
      service:
        type: LoadBalancer
  YAML]

  depends_on = [module.eks]
}
    grafana:
      adminPassword: ${var.grafana_admin_password}
      service:
        type: LoadBalancer
    prometheus:
      service:
        type: LoadBalancer
  YAML]

  depends_on = [module.eks]
}
```

infra/terraform/outputs.tf

```
output "cluster_name" { value = module.eks.cluster_name }
output "region" { value = var.aws_region }
```

infra/terraform/terraform.tfvars.example

```
name                = "demo"
aws_region          = "ap-south-1"

git_repository_url  = "https://github.com/YOUR_GITHUB_USERNAME/
YOUR_REPO.git"

dockerhub_username  = "YOUR_DOCKERHUB_USERNAME"
dockerhub_password  = "YOUR_DOCKERHUB_TOKEN"
dockerhub_email     = "you@example.com"

grafana_admin_password = "SetAStrongPassword"
```

> **Note:** You must create the folder `infra/terraform/templates/` and place `argocd-apps.tmpl.yaml` in it.

GitHub Actions — CI: SonarQube → Test → Build → Trivy → Push → Bump GitOps

`/.github/workflows/ci.yml`

```
name: CI

on:
  push:
    branches: ["main"]
  pull_request:

permissions:
  contents: write  # to commit kustomize tag bumps

env:
  REGISTRY: docker.io

jobs:
  build-test-scan-push:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        include:
          - svc: service-a
            context: app/service-a
            port: 3000
          - svc: service-b
            context: app/service-b
            port: 5000
          - svc: service-c
            context: app/service-c
            port: 8080

    steps:
      - name: Check out
        uses: actions/checkout@v4

      - name: Set SHORT_SHA
        run: echo "SHORT_SHA=${GITHUB_SHA::7}" >> $GITHUB_ENV

      # ---- Static analysis (SonarQube / SonarCloud) ----
      - name: Sonar Scan ${{ matrix.svc }}
        uses: SonarSource/sonarqube-scan-action@v2.2
        with:
```

```
          projectBaseDir: ${{ matrix.context }}
        env:
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
          SONAR_HOST_URL: ${{ secrets.SONAR_HOST_URL }}
          SONAR_ORG: ${{ secrets.SONAR_ORG }}

      # ---- Unit tests (language-specific lightweight) ----
      - name: Run tests for Node (service-a)
        if: matrix.svc == 'service-a'
        working-directory: app/service-a
        run: |
          npm ci || npm install
          npm test

      - name: Run tests for Python (service-b)
        if: matrix.svc == 'service-b'
        working-directory: app/service-b
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
          pytest -q

      - name: Run tests for Go (service-c)
        if: matrix.svc == 'service-c'
        working-directory: app/service-c
        run: |
          go test ./...

      # ---- Build image (local), scan with Trivy, then push to Docker Hub
----
      - name: Set image name
        run:
echo "IMAGE=${{ secrets.DOCKERHUB_USERNAME }}/${{ matrix.svc }}" >>
$GITHUB_ENV

      - name: Log in to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}

      - name: Build image (load to docker)
        uses: docker/build-push-action@v6
        with:
          context: ${{ matrix.context }}
          push: false
          load: true
          tags: |
            ${{ env.IMAGE }}:${{ env.SHORT_SHA }}
            ${{ env.IMAGE }}:latest
```

```
      - name: Trivy scan (image)
        uses: aquasecurity/trivy-action@0.20.0
        with:
          image-ref: ${{ env.IMAGE }}:${{ env.SHORT_SHA }}
          format: 'table'
          severity: 'CRITICAL,HIGH'
          ignore-unfixed: true
          exit-code: '1'

      - name: Push image tags
        if: success()
        run: |
          docker push $IMAGE:${SHORT_SHA}
          docker push $IMAGE:latest

      # ---- Bump GitOps kustomize image tag and commit [skip ci] ----
      - name: Install yq
        run: |
          sudo wget -qO /usr/local/bin/yq "https://github.com/mikefarah/yq/
releases/download/v4.44.3/yq_linux_amd64" && sudo chmod +x /usr/local/bin/yq

      - name: Update kustomize tag
        run: |
          FILE=gitops/apps/${{ matrix.svc }}/kustomization.yaml
          yq -i
            '(.images[] | select(.name == "'$
{{ secrets.DOCKERHUB_USERNAME }}'/'${{ matrix.svc }}'" ).newTag) =
env(SHORT_SHA)'
            $FILE || true

      - name: Commit and push tag bump [skip ci]
        run: |
          git config user.name "github-actions"
          git config user.email "github-actions@github.com"
          git add gitops/apps/${{ matrix.svc }}/kustomization.yaml
          git commit -m "chore(${{ { matrix.svc } }): update image tag to $
{SHORT_SHA} [skip ci]" || echo "No changes"
          git push
```

Secrets required in your repo: `DOCKERHUB_USERNAME`, `DOCKERHUB_TOKEN`, `SONAR_TOKEN`, `SONAR_HOST_URL` (e.g. `https://sonarcloud.io` or your SonarQube URL), and optionally `SONAR_ORG` (SonarCloud).

Notes on Argo CD Webhook (GitHub → Argo)

After Terraform applies, Argo CD `argocd-server` is exposed as a `LoadBalancer`.

- **Payload URL:** `http://<ARGOCD_ELB_HOSTNAME>/api/webhook`

- **Content type:** `application/json`
- **Events:** "Just the push event" is enough.
- Secret is optional for labs; production should set a secret and configure verification.

Get the ELB hostname:

```
kubectl get svc -n argocd argocd-server -o
jsonpath='{.status.loadBalancer.ingress[0].hostname}'
```

Initial admin password (username: `admin`):

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

Quick Test Commands

After Argo syncs, fetch ELB hostnames for services:

```
kubectl get svc -n apps -o wide
```

Hit each service root/health:

```
curl http://<ELB_A>/            # {"service":"a","message":"ok"}
curl http://<ELB_A>/health      # ok
curl http://<ELB_B>/
curl http://<ELB_C>/
```

Grafana URL:

```
kubectl get svc -n monitoring grafana -o
jsonpath='{.status.loadBalancer.ingress[0].hostname}'
```

Login with `admin` / the `grafana_admin_password` you set.

README Snippet (Optional)

1) Copy these files into your repo. Replace `YOUR_GITHUB_REPO_URL` and `YOUR_DOCKERHUB_USERNAME` in GitOps manifests. 2) Set GitHub secrets: `DOCKERHUB_USERNAME`, `DOCKERHUB_TOKEN`, `SONAR_TOKEN`, `SONAR_HOST_URL`, `SONAR_ORG`. 3) `cd infra/terraform && cp terraform.tfvars.example terraform.tfvars` and fill values. 4) `terraform init && terraform apply`. 5) Configure GitHub → Webhooks → Payload URL

`http://<argocd ELB>/api/webhook`. 6) Push to `main` → CI builds/scans/pushes and bumps tags → Argo syncs → apps live.

Tips & Variations

- Use private repos? Add repo credentials to Argo CD (`argocd repo add` or values in chart) or keep HTTPS public.
- Replace `Service: LoadBalancer` with an Ingress + AWS Load Balancer Controller if you want one public hostname and paths.
- Pin chart versions in Terraform for reproducibility.
- Trivy can be configured to fail on Medium too; adjust `severity`.
- SonarQube Server vs SonarCloud: set `SONAR_HOST_URL` accordingly.