1 CGOA
1.1 Table of Contents
1.2 Overview
1.3 Exam Overview
1.4 Exam Domains & Weights
1.5 Key Topics
1.5.1 GitOps Principles
1.5.2 GitOps Patterns
1.5.3 Tooling
1.6 Study Resources
1.7 Navigation
1.8 GitOps Principles
1.9 What is GitOps?
1.10 Core Principles
1.10.1 1. Declarative Configuration
1.10.2 2. Version Controlled
1.10.3 3. Automated Reconciliation
1.10.4 4. Software Agents
1.11 GitOps Workflow
1.12 GitOps Tools
1.12.1 Flux CD
1.12.2 Argo CD
1.13 Repository Structure
1.13.1 Monorepo
1.13.2 Multi-repo
1.14 Flux CD Resources
1.14.1 GitRepository
1.14.2 Kustomization
1.14.3 HelmRelease
1.15 Best Practices
1.15.1 1. Environment Promotion
1.15.2 2. Secrets Management
1.15.3 3. Image Automation
1.15.4 4. Notifications
1.16 GitOps vs Traditional CI/CD
1.17 Sample Practice Questions
1.18 Practice Resources
1.19 GitOps Terminology (20%)
1.19.1 Question 1
1.19.2 Question 2
1.19.3 Question 3
1.20 GitOps Principles (30%)
1.20.1 Question 4
1.20.2 Question 5
1.20.3 Question 6
1.21 Related Practices (16%)
1.21.1 Question 7
1.21.2 Question 8
1.22 GitOps Patterns (20%)
1.22.1 Question 9
1.22.2 Question 10

1.22.3 Question 11
1.23 Tooling (14%)
1.23.1 Question 12
1.23.2 Question 13
1.23.3 Question 14
1.24 Exam Tips

1 CGOA

Generated on: 2026-01-13 15:04:54 **Version:** 1.0

1.1 Table of Contents

1. Overview
2. GitOps Principles
3. Sample Practice Questions

1.2 Overview

The **Certified GitOps Associate (CGOA)** exam demonstrates knowledge of GitOps principles, practices, and related technologies.

1.3 Exam Overview

Detail	Information
Exam Format	Multiple Choice
Number of Questions	60
Duration	90 minutes
Passing Score	75%
Certification Validity	3 years
Cost	$250 USD
Retake Policy	1 free retake

1.4 Exam Domains & Weights

Domain	Weight
GitOps Terminology	20%

Domain	Weight
GitOps Principles	30%
Related Practices	16%
GitOps Patterns	20%
Tooling	14%

1.5 Key Topics

1.5.1 GitOps Principles

- Declarative configuration
- Version controlled
- Automated delivery
- Software agents

1.5.2 GitOps Patterns

- Repository structures
- Environment promotion
- Secret management
- Multi-tenancy

1.5.3 Tooling

- Flux CD
- Argo CD
- Git workflows

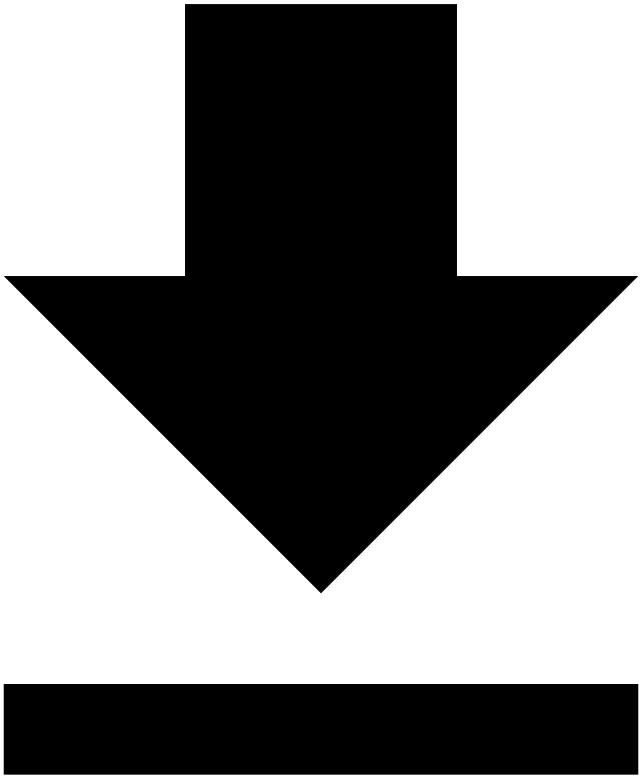
1.6 Study Resources

- OpenGitOps
- GitOps Working Group
- Flux Documentation
- Argo CD Documentation

1.7 Navigation

- Next: Sample Questions →

1.8 GitOps Principles



Download PDF Version

Comprehensive guide to GitOps for CGOA certification.

1.9 What is GitOps?

GitOps is a set of practices that uses Git as the single source of truth for declarative infrastructure and applications:

- **Declarative** - System state described declaratively
- **Versioned** - Desired state stored in Git
- **Automated** - Changes automatically applied
- **Auditable** - Git history provides audit trail

1.10 Core Principles

1.10.1 1. Declarative Configuration

All system configuration is declarative:

```yaml
# Infrastructure as Code
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: myapp:v1.0.0
```

1.10.2 2. Version Controlled

- All configuration stored in Git
- Changes tracked through commits
- Pull requests for review
- Branches for environments

1.10.3 3. Automated Reconciliation

- Continuous sync between Git and cluster
- Drift detection and correction
- Self-healing infrastructure

1.10.4 4. Software Agents

- Operators watch Git repositories
- Pull changes automatically
- Apply to target systems

1.11 GitOps Workflow

```
Developer → Git Commit → Pull Request → Merge → GitOps Agent →
Kubernetes
```

```
                    ↓
              Code Review
                    ↓
              CI Pipeline
                    ↓
              Image Build
```

1.12 GitOps Tools

1.12.1 Flux CD

```
# Install Flux
curl -s https://fluxcd.io/install.sh | sudo bash

# Bootstrap
flux bootstrap github \
  --owner=<github-user> \
  --repository=fleet-infra \
  --branch=main \
  --path=./clusters/my-cluster \
  --personal
```

1.12.2 Argo CD

```
# Install Argo CD
kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/
        argoproj/argo-cd/stable/manifests/install.yaml
```

1.13 Repository Structure

1.13.1 Monorepo

```
fleet-infra/
├── apps/
│   ├── base/
│   │   └── myapp/
│   │       ├── deployment.yaml
│   │       ├── service.yaml
│   │       └── kustomization.yaml
│   ├── dev/
│   │   └── kustomization.yaml
│   ├── staging/
│   │   └── kustomization.yaml
│   └── production/
│       └── kustomization.yaml
├── infrastructure/
```

```
│   ├── controllers/
│   └── configs/
└── clusters/
    ├── dev/
    ├── staging/
    └── production/
```

1.13.2 Multi-repo

```
# App repos
myapp-repo/
├── src/
├── Dockerfile
└── k8s/
    └── base/

# Config repo
gitops-config/
├── apps/
│   └── myapp/
└── clusters/
```

1.14 Flux CD Resources

1.14.1 GitRepository

```yaml
apiVersion: source.toolkit.fluxcd.io/v1
kind: GitRepository
metadata:
  name: myapp
  namespace: flux-system
spec:
  interval: 1m
  url: https://github.com/myorg/myapp
  ref:
    branch: main
  secretRef:
    name: github-token
```

1.14.2 Kustomization

```yaml
apiVersion: kustomize.toolkit.fluxcd.io/v1
kind: Kustomization
metadata:
  name: myapp
  namespace: flux-system
spec:
  interval: 10m
  targetNamespace: default
```

```yaml
  sourceRef:
    kind: GitRepository
    name: myapp
  path: ./k8s/overlays/production
  prune: true
  healthChecks:
  - apiVersion: apps/v1
    kind: Deployment
    name: myapp
    namespace: default
```

1.14.3 HelmRelease

```yaml
apiVersion: helm.toolkit.fluxcd.io/v2beta1
kind: HelmRelease
metadata:
  name: nginx
  namespace: default
spec:
  interval: 5m
  chart:
    spec:
      chart: nginx
      version: '15.x'
      sourceRef:
        kind: HelmRepository
        name: bitnami
        namespace: flux-system
  values:
    replicaCount: 2
```

1.15 Best Practices

1.15.1 1. Environment Promotion

```
dev → staging → production

# Use branches or directories
main branch → dev cluster
release branch → staging cluster
tags → production cluster
```

1.15.2 2. Secrets Management

- Use Sealed Secrets or SOPS
- External secrets operators
- Never commit plain secrets

```yaml
# Sealed Secret
apiVersion: bitnami.com/v1alpha1
kind: SealedSecret
metadata:
  name: mysecret
spec:
  encryptedData:
    password: AgBy8hCi...
```

1.15.3 3. Image Automation

```yaml
apiVersion: image.toolkit.fluxcd.io/v1beta1
kind: ImagePolicy
metadata:
  name: myapp
spec:
  imageRepositoryRef:
    name: myapp
  policy:
    semver:
      range: '>=1.0.0'
```

1.15.4 4. Notifications

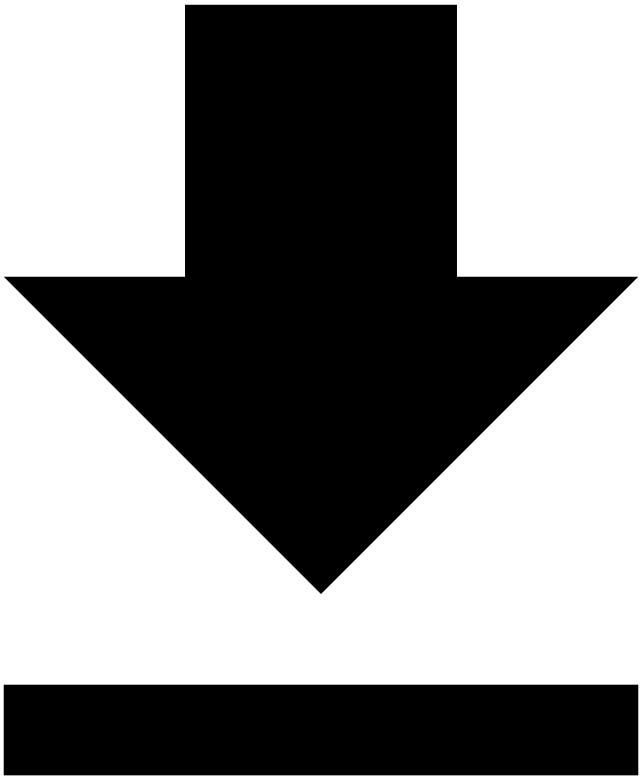
```yaml
apiVersion: notification.toolkit.fluxcd.io/v1beta1
kind: Alert
metadata:
  name: slack-alert
spec:
  providerRef:
    name: slack
  eventSeverity: error
  eventSources:
  - kind: Kustomization
    name: '*'
```

1.16 GitOps vs Traditional CI/CD

Aspect	Traditional CI/CD	GitOps
Deployment	Push-based	Pull-based
Source of Truth	CI/CD pipeline	Git repository
Drift Detection	Manual	Automatic
Rollback	Re-run pipeline	Git revert
Audit	CI/CD logs	Git history

← Back to CGOA Overview

1.17 Sample Practice Questions



Download PDF Version

1.18 Practice Resources

- OpenGitOps
- Flux Documentation
- Argo CD Documentation

1.19 GitOps Terminology (20%)

1.19.1 Question 1

What are the four principles of GitOps according to OpenGitOps?

Show Solution

1. **Declarative** - A system managed by GitOps must have its desired state expressed declaratively
2. **Versioned and Immutable** - Desired state is stored in a way that enforces immutability, versioning, and retains a complete version history
3. **Pulled Automatically** - Software agents automatically pull the desired state declarations from the source
4. **Continuously Reconciled** - Software agents continuously observe actual system state and attempt to apply the desired state

1.19.2 Question 2

What is “drift” in GitOps terminology?

Show Solution

Drift refers to when the actual state of a system diverges from the desired state defined in Git. This can happen when: - Manual changes are made directly to the cluster - External processes modify resources - Failed reconciliation attempts

GitOps tools detect drift and can automatically reconcile to restore the desired state.

1.19.3 Question 3

What is the difference between “push” and “pull” based GitOps?

Show Solution

Push-based: - CI/CD pipeline pushes changes to the cluster - Pipeline needs cluster credentials - Less secure (credentials in CI)

Pull-based (GitOps preferred): - Agent in cluster pulls changes from Git - No external access to cluster needed - More secure (agent has cluster access) - Continuous reconciliation

---

1.20 GitOps Principles (30%)

1.20.1 Question 4

Why is declarative configuration important in GitOps?

Show Solution

Declarative configuration is important because: - **Predictability** - You describe WHAT you want, not HOW to get there - **Idempotency** - Applying the same config multiple times yields the same result - **Version control** - Easy to track changes over time - **Rollback** - Simple to revert to previous states - **Auditability** - Clear record of desired state at any point

1.20.2 Question 5

How does GitOps handle secrets management?

Show Solution

Common approaches: 1. **Sealed Secrets** - Encrypt secrets that can only be decrypted in cluster 2. **SOPS** - Mozilla's Secrets OPerationS for encrypting files 3. **External Secrets Operator** - Sync secrets from external vaults 4. **Vault** - HashiCorp Vault integration 5. **Cloud KMS** - AWS KMS, GCP KMS, Azure Key Vault

Best practice: Never store plain-text secrets in Git.

1.20.3 Question 6

What is continuous reconciliation?

Show Solution

Continuous reconciliation is the process where GitOps agents: 1. Continuously monitor the actual state of the system 2. Compare it with the desired state in Git 3. Automatically apply changes to match desired state 4. Report any drift or failures

This ensures the system always converges to the desired state, even after manual changes or failures.

1.21 Related Practices (16%)

1.21.1 Question 7

How does GitOps relate to Infrastructure as Code (IaC)?

Show Solution

Relationship: - IaC defines infrastructure declaratively (Terraform, Pulumi, CloudFormation) - GitOps applies IaC principles to application deployment - Both use version control as source of truth - GitOps adds continuous reconciliation

Key difference: - IaC typically uses push-based deployment - GitOps uses pull-based with continuous reconciliation

1.21.2 Question 8

What is the relationship between GitOps and CI/CD?

Show Solution

CI (Continuous Integration): - Build and test code - Create artifacts (images, packages) - Update Git with new versions

CD with GitOps: - Git is the source of truth for deployments - GitOps agent pulls and deploys changes - Separation of concerns: CI builds, GitOps deploys

Benefits: - Clear audit trail - Easy rollbacks - Consistent deployments

---

1.22 GitOps Patterns (20%)

1.22.1 Question 9

What are common Git repository structures for GitOps?

Show Solution

Monorepo:

```
repo/
├── apps/
│   ├── app1/
│   └── app2/
├── infrastructure/
└── clusters/
    ├── dev/
    ├── staging/
    └── prod/
```

Polyrepo: - Separate repos for apps, infrastructure, and cluster config - More isolation but harder to manage

Environment branches: - main → production - staging → staging - develop → development

1.22.2 Question 10

How do you handle environment promotion in GitOps?

Show Solution

Common patterns:

1. **Directory-based:**

```
environments/
├── dev/
├── staging/
└── prod/
```

1. **Branch-based:**

• Merge from dev → staging → prod branches

1. **Tag-based:**

• Tag releases for promotion

1. **PR-based promotion:**

• Create PR to promote changes between environments

1.22.3 Question 11

What is the “App of Apps” pattern?

Show Solution

The **App of Apps** pattern uses a parent Application that manages child Applications:

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: apps
spec:
  source:
    path: apps/
  # This directory contains Application manifests
```

Benefits: - Centralized management - Hierarchical organization - Easier bootstrapping - Consistent configuration

---

1.23 Tooling (14%)

1.23.1 Question 12

Compare Flux CD and Argo CD.

Show Solution

Feature	Flux CD	Argo CD
Architecture	Toolkit of controllers	Monolithic with UI
UI	Separate (Weave GitOps)	Built-in
Multi-tenancy	Native	Via Projects

Feature	Flux CD	Argo CD
Helm support	HelmRelease CRD	Native
Kustomize	Native	Native
Image automation	Built-in	Separate (Argo Image Updater)

Both implement GitOps principles effectively.

1.23.2 Question 13

Bootstrap Flux CD in a cluster.

Show Solution

```
# Install Flux CLI
curl -s https://fluxcd.io/install.sh | sudo bash

# Bootstrap with GitHub
flux bootstrap github \
  --owner=my-org \
  --repository=fleet-infra \
  --branch=main \
  --path=clusters/my-cluster \
  --personal

# Check status
flux check
```

1.23.3 Question 14

What is a Flux Kustomization?

Show Solution

```
apiVersion: kustomize.toolkit.fluxcd.io/v1
kind: Kustomization
metadata:
  name: my-app
  namespace: flux-system
spec:
  interval: 10m
  path: ./apps/my-app
  prune: true
  sourceRef:
    kind: GitRepository
    name: flux-system
  healthChecks:
  - apiVersion: apps/v1
    kind: Deployment
    name: my-app
    namespace: default
```

It defines what to deploy, from where, and how often to reconcile.

1.24 Exam Tips

1. **Know the four GitOps principles** - These are fundamental
2. **Understand push vs pull** - Why pull is preferred
3. **Know secret management options** - Sealed Secrets, SOPS, External Secrets
4. **Understand repository patterns** - Monorepo vs polyrepo
5. **Compare Flux and Argo CD** - Key differences and similarities

---

← Back to CGOA Overview
