PassQueen
a.k.a dumpsqueen.com

Certified GitOps Associate Exam

Linux Foundation CGOA

Version Demo

Total Demo Questions: 8

Total Premium Questions: 84

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, GitOps Terminology	12
Topic 2, GitOps Principles	28
Topic 3, Related Practices	14
Topic 4, GitOps Patterns	20
Topic 5, Tooling	10
Total	84

QUESTION NO: 1

Why is the feedback loop important for reconciliation?

A. To determine if a reconciliation is needed and whether a sync should be partial or complete.

B. To analyze state-sync logging information and perform a sync.

C. To trigger an alert if a change is detected, and log the event to the log aggregation service.

D. Feedback loop is not important for reconciliation.

ANSWER: A

Explanation:

To determine if a reconciliation is needed and whether a sync should be partial or complete is correct because reconciliation depends on continuous feedback about the running system's actual state. In GitOps, the desired state is declaratively defined in a source of truth, typically a Git repository. A reconciliation agent continually observes the environment, compares the observed state with that declared desired state, and detects drift or pending changes. This comparison supplies the feedback needed to decide whether corrective action is necessary.

When a difference is identified, the controller can initiate synchronization to bring the environment back toward the declared configuration. The scope of that synchronization is informed by what the feedback loop observes: a controller may apply only the resources that differ, or perform a broader synchronization when needed by the tool's configuration and operational model. The essential point is that reconciliation is not a one-time deployment action; it is an ongoing control loop that observes, compares, and acts to maintain convergence. The OpenGitOps principles describe this as continuously reconciling actual state toward desired state, while the Flux documentation explains reconciliation as ensuring a managed resource is kept aligned with its declarative definition. See the OpenGitOps Principles and Flux Kustomization reconciliation documentation.

QUESTION NO: 2

In GitOps, which option describesState Storemanagement?

A. Storing state information in a relational database.

B. Storing state information in a centralized database.

C. Storing state information in a distributed file system.

D. Storing state information in a version control system.

ANSWER: D

Explanation:

Storing state information in a version control system is the defining approach for State Store management in GitOps. GitOps uses a Git repository, or another version-control system, as the source of truth for the desired state of infrastructure and applications. Declarative configuration files, such as Kubernetes manifests, Helm chart values, or Kustomize configurations, are committed to the repository. Each proposed change is reviewed and recorded as a commit, producing a traceable history of who changed the desired state, what changed, and when it changed.

This versioned history supports the core GitOps capabilities of auditability, collaboration through pull requests, and straightforward rollback to a previously known desired state. A GitOps reconciliation agent continuously compares the state declared in the repository with the actual state of the target environment and works to bring the environment into alignment. The State Store therefore records the intended configuration, rather than functioning merely as a runtime data store. The OpenGitOps principles explicitly identify a version-controlled source of truth as a central GitOps principle. See the OpenGitOps principles and the Linux Foundation's Certified GitOps Associate certification page.

QUESTION NO: 3

In the context of GitOps, what does Continuous mean?

A. Reconciliation happens only during instantiation.

B. Reconciliation must happen instantaneously.

C. Reconciliation continues to happen.

D. Reconciliation only happens once.

ANSWER: C

Explanation:

Reconciliation continues to happen. In GitOps, the desired state of infrastructure and applications is declared in version-controlled source repositories, while an agent running in the target environment repeatedly observes the real, running state. The agent compares that observed state with the declared state and takes corrective action when it detects drift. This is an ongoing control loop rather than an activity limited to initial deployment, a single execution, or a requirement for literally zero-delay synchronization.

The OpenGitOps principles explicitly describe this as continuous reconciliation: software agents must continuously compare actual and desired state, then automatically attempt to correct differences. This enables Git to remain the authoritative source for intended system configuration throughout the workload lifecycle. For example, if someone makes an unapproved manual change to a Kubernetes resource, the reconciliation agent will detect that the live resource no longer matches the repository and work to restore the declared configuration. The repeated process is what makes the environment reliably converge toward the desired state and provides the operational meaning of “Continuous” in GitOps. See the OpenGitOps project and its GitOps principles for the formal definition.

QUESTION NO: 4

A team uses a GitOps controller to manage a production Kubernetes cluster. An engineer changes a Deployment image directly with `kubectl` during an incident, and the controller restores the image defined in Git a few minutes later. What should the team do after confirming that the emergency image is the required fix?

A. Update the Deployment manifest in Git and allow the controller to reconcile the approved image.

B. Pause reconciliation permanently so that the live Deployment remains authoritative.

C. Continue applying the image change with kubectl whenever reconciliation restores the prior image.

D. Delete the Deployment from Git so that the controller no longer observes it.

ANSWER: A

Explanation:

The team should update the declarative Deployment configuration in Git through the normal reviewed workflow. Git remains the source of truth for the desired state, so a live change that is not represented in Git will be treated as drift and can be reverted by reconciliation.

Pausing reconciliation may leave the cluster unmanaged and does not create an auditable desired-state change. Repeatedly applying the change with `kubectl` only creates another temporary divergence. Updating Git allows the controller to converge the cluster on the approved emergency fix and preserves history for later review.

QUESTION NO: 5

What is an example of how GitOps helps DevSecOps?

A. You must sign into your GitHub account before running kubectl commands.

B. The entire version history of Desired State changes is available for auditing.

C. Store all access tokens in Git.

D. Unit testing during CD limits the bugs introduced into deployed code.

ANSWER: B

Explanation:

The entire version history of Desired State changes is available for auditing. This is a core GitOps benefit for DevSecOps because Git is the source of truth for the desired configuration of infrastructure and applications. Changes to that desired state are proposed, reviewed, approved, and recorded as commits or pull requests. The resulting history provides a durable record of what changed, who authorized it, when it was changed, and the associated review context. That traceability supports security investigations, compliance evidence, change-control requirements, and accountability without relying on undocumented manual changes to a live environment.

GitOps also uses automated reconciliation to compare the deployed environment with the versioned desired state. As a result, approved security-related configuration—such as policies, workload settings, and deployment manifests—can be consistently applied from the Git repository. The OpenGitOps principles identify a declarative system, versioned and immutable desired state, and automated reconciliation as foundational practices. An immutable version history makes it possible to inspect and audit the evolution of the intended system state, which directly strengthens DevSecOps governance and operational visibility. See the OpenGitOps principles and the GitOps Working Group for further guidance.

QUESTION NO: 6

A production GitOps repository allows engineers to commit directly to the branch watched by the reconciliation controller. The organization requires every production configuration change to be reviewed, validated, and auditable before it can affect the cluster. Which change best meets this requirement?

A. Protect the production branch and require reviewed pull requests with successful validation checks before merging.

B. Allow direct commits but require engineers to document each change in the commit message.

C. Allow the controller to reconcile first and request approval only if an error occurs.

D. Give every engineer cluster administrator access so that urgent changes can be verified directly.

ANSWER: A

Explanation:

Protecting the production branch and requiring pull requests with required checks is correct because the GitOps controller should reconcile only approved desired-state changes. Branch protection can prevent direct commits, require review, and ensure that validation checks complete before a change is merged.

Git history alone provides traceability, but it does not guarantee that a change received review or passed automated validation. Manual cluster approval after reconciliation occurs too late, while allowing direct commits continues to bypass the required change-control process.

QUESTION NO: 7

What does Pulled Automatically refer to?

A. A GET request to a relational database.

B. It always refers to Git pull.

C. Webhooks informing the system about new commits.

D. Accessing the Desired State from the State Store.

ANSWER: D

Explanation:

Accessing the Desired State from the State Store is correct because “Pulled Automatically” describes the GitOps reconciliation model in which software agents running in or alongside the managed environment retrieve the declared desired state from an authoritative state store. The agent compares that declaration with the environment’s observed, actual state and takes the actions needed to continuously converge the environment toward the declared configuration.

This principle is about the direction and automation of reconciliation: the agent obtains the desired-state declaration itself, rather than relying on a person to manually apply configuration. The state store is commonly a Git repository, which provides version history, reviewable changes, and an auditable source of truth, but the key concept is automated retrieval of desired state from the state store followed by reconciliation. This pull-based model enables drift detection and correction because the agent repeatedly checks the authoritative declaration and the current environment state. The OpenGitOps principles define this explicitly: software agents should automatically pull the desired state declarations from the source of truth and continuously reconcile the system to match it. See the OpenGitOps Principles and the Flux reconciliation documentation for practical examples of this operating model.

QUESTION NO: 8

You are working on a GitOps project and need to understand the similarities and differences between pull-based messaging systems and event-driven systems. What is a key difference between these two types of systems?

A. Pull-based systems require a constant network connection to receive updates.

B. Event-driven systems are less flexible and scalable compared to pull-based systems.

C. Pull-based systems are more efficient in handling real-time events.

D. When only events trigger reconciliation, the system is more vulnerable to drift caused by other things.

ANSWER: D

Explanation:

When only events trigger reconciliation, the system is more vulnerable to drift caused by other things. This captures the important operational distinction in GitOps: a pull-based reconciler periodically retrieves the declared state from Git and compares it with the live state, then applies changes needed to converge on that desired state. Because this loop runs on an interval, it can detect and correct configuration drift even when the drift was caused by a manual cluster change, a failed prior action, or another change that did not generate a delivery event.

Event notifications, such as webhooks, can make a system react quickly when a repository changes, but an event alone is not a complete ongoing drift-detection mechanism. If reconciliation happens only in response to those notifications, changes outside the event path can remain undetected until another relevant event occurs. GitOps guidance emphasizes continuous reconciliation as a core principle, and Flux documents interval-based reconciliation for Git sources and managed resources. See the OpenGitOps principles and the Flux GitRepository documentation.