GitLab Certified Associate Practice Exam (Sample)

Study Guide

Everything you need from our exam experts!

Questions

1. Which of the following variables can indicate that scans should be disabled in GitLab workflows?

A. TESTING_SCANNING_DISABLED

B. DEPENDENCY_SCANNING_DISABLED

C. SCAN_DISABLE

D. CHECK_SCANNING_DISABLED

2. What does forking a project in GitLab allow you to do?

A. Create a shared repository for all users

B. Make independent changes to someone else's project

C. Track task enhancements

D. Avoid copying files to your local machine

3. What is the purpose of the .gitlab-ci.yml file in GitLab CI/CD?

A. It serves as a configuration guide for GitLab Runner.

B. It defines the pipeline stages, jobs, and actions to perform.

C. It contains all project documentation and guidelines.

D. It is used to deploy applications to Kubernetes clusters.

4. During which stage can feature flags be used to deploy applications?

A. Configure

B. Release

C. Secure

D. Verify

5. Where can you enable two-factor authentication in GitLab?

A. Under project settings

B. In the user profile settings within the Security section

C. In the repository settings

D. In the CI/CD settings

6. What type of content can be stored using GitLab 'Snippets'?

- **A. Only images and videos**
- **B. Project management plans**
- **C. Reusable pieces of code or text**
- **D. Static HTML pages only**

7. What is a Git Workspace?

- **A. A directory where you store the repository on your computer**
- **B. A remote server managing your code**
- **C. A temporary storage area for uncommitted changes**
- **D. A version control system for multiple teams**

8. What role does versioning play in a system like Git?

- **A. It helps eliminate the need for collaboration**
- **B. It maintains a complete history of changes made**
- **C. It only tracks final versions of files**
- **D. It is unnecessary in a distributed system**

9. How do you resolve a merge conflict in GitLab?

- **A. By deleting the conflicting files**
- **B. By merging the branches without changes**
- **C. By manually editing conflicting files, staging them, and committing**
- **D. By undoing the last commit**

10. If the DOCKER_IMAGE CI/CD variable is not specified, what does CI_APPLICATION_REPOSITORY default to?

- **A. $CI_COMMIT_SHA**
- **B. $CI_REGISTRY_IMAGE/$CI_COMMIT_REF_SLUG**
- **C. $CI_PROJECT_DIR**
- **D. docker.io/default**

Answers

1. B
2. B
3. B
4. B
5. B
6. C
7. A
8. B
9. C
10. B

Explanations

1. Which of the following variables can indicate that scans should be disabled in GitLab workflows?

A. TESTING_SCANNING_DISABLED

B. DEPENDENCY_SCANNING_DISABLED

C. SCAN_DISABLE

D. CHECK_SCANNING_DISABLED

The variable that indicates scans should be disabled in GitLab workflows is commonly associated with a specific type of scanning function. In this context, "DEPENDENCY_SCANNING_DISABLED" is specifically tailored for managing dependency scanning within GitLab CI/CD pipelines. When this variable is set, it signals the system to skip the dependency scanning process during a job run, which is crucial for ensuring that potential vulnerabilities in dependencies are not scanned. This is particularly important when teams are looking to adjust their CI/CD processes for reasons such as reducing execution time, managing resources, or temporarily disabling specific checks while working on other aspects of the project. Understanding this variable is vital for DevOps teams to maintain flexibility in their workflows while ensuring they still can enforce necessary checks when required. Other variables might exist within GitLab, but their naming conventions and specific purposes do not align directly with disabling scans in the context of dependency scanning, making "DEPENDENCY_SCANNING_DISABLED" the correct choice for indicating that dependent scans should no longer occur in the workflow.

2. What does forking a project in GitLab allow you to do?

A. Create a shared repository for all users

B. Make independent changes to someone else’s project

C. Track task enhancements

D. Avoid copying files to your local machine

Forking a project in GitLab allows you to create a personal copy of someone else's project that you can modify independently. This process is essential for collaborative development, particularly in open-source projects, where you might want to introduce changes to a project without affecting the original repository directly. When you fork a project, it is essentially your own version of the project, enabling you to make changes, add features, or fix bugs without needing permission from the original project maintainers. Once you've made your changes in your fork, you can then submit a merge request to propose your changes back to the original repository. Making independent changes is paramount in scenarios where direct contributions to the original project are not possible or when you wish to experiment with new ideas or configurations. This independence supports innovation while maintaining the integrity of the source project, fostering a collaborative and structured workflow. Other options do not capture the full essence of forking. Creating a shared repository implies collaboration from the start rather than independent modification, tracking task enhancements does not specifically relate to the concept of forking, and avoiding copying files locally does not reflect what forking accomplishes, as forking typically involves an online operation where a full copy of repository data is created on GitLab's servers.

3. What is the purpose of the .gitlab-ci.yml file in GitLab CI/CD?

A. It serves as a configuration guide for GitLab Runner.

B. It defines the pipeline stages, jobs, and actions to perform.

C. It contains all project documentation and guidelines.

D. It is used to deploy applications to Kubernetes clusters.

The .gitlab-ci.yml file plays a crucial role in GitLab CI/CD by defining the pipeline stages, jobs, and the specific actions to execute during the CI/CD process. This file acts as the blueprint for automation in the continuous integration and continuous deployment workflows. Within it, you can specify single or multiple jobs that detail how to build, test, and deploy your applications, as well as configure various stages in the pipeline, such as build, test, and deploy. By structuring the CI/CD pipeline in this manner, teams can automate their development processes efficiently, ensuring that code changes are tested and deployed consistently and reliably. This not only enhances the quality of the code but also speeds up the release cycles, making it essential for DevOps practices. Other choices refer to different aspects: while the GitLab Runner does rely on configurations, it is not specifically outlined in the .gitlab-ci.yml. Project documentation can be maintained separately and is not the primary purpose of this file, and deploying applications to Kubernetes is a specific action that can be defined within the .gitlab-ci.yml but is not the overarching purpose of the file itself.

4. During which stage can feature flags be used to deploy applications?

A. Configure

B. Release

C. Secure

D. Verify

Feature flags are a powerful technique used in software development to enable or disable features in a live application without deploying new code. They facilitate incremental feature rollouts, testing, and experimentation without risking the stability of the entire application. The most appropriate stage for deploying applications with feature flags is the release stage. In the release phase, features can be turned on or off for specific user segments or environments. This allows teams to manage risk effectively by controlling how and when new features are made available to users. By using feature flags during the release stage, teams can gather feedback, observe behavior in real-time, and make quick adjustments as necessary based on user interaction with the new features. Using feature flags during other stages such as configure, secure, or verify is less effective since those stages focus on configurations, security checks, and testing, respectively. While feature flags may play a role in configurations and might be considered during verification, their main utility is recognized predominantly during the release phase, where they have the most significant impact on user experience and production systems.

5. Where can you enable two-factor authentication in GitLab?

A. Under project settings

B. In the user profile settings within the Security section

C. In the repository settings

D. In the CI/CD settings

Two-factor authentication (2FA) in GitLab enhances account security by requiring not only a password but also a second form of verification, such as a mobile app code or text message. To enable 2FA, users must navigate to their profile settings, specifically in the Security section. This is the designated area where users can set up their authentication methods, manage recovery codes, and manage their security features. Project settings, repository settings, and CI/CD settings are geared towards managing different aspects of projects and deployment processes rather than user authentication. Therefore, while they are essential for other administrative functions within GitLab, they do not provide options for enabling two-factor authentication. The user profile settings within the Security section is specifically designed for managing personal security configurations, making it the correct choice for enabling 2FA.

6. What type of content can be stored using GitLab 'Snippets'?

A. Only images and videos

B. Project management plans

C. Reusable pieces of code or text

D. Static HTML pages only

GitLab 'Snippets' are designed for storing reusable pieces of code or text, making them a valuable resource for developers and teams. This feature allows users to save and share small snippets of code or notes that can be utilized across different projects or by different team members, promoting efficiency and consistency in coding practices. Snippets can capture everything from simple code examples and configuration files to detailed documentation or reusable scripts, encompassing a range of content that is particularly relevant in software development. The other options do not accurately represent the primary purpose of GitLab Snippets. Images and videos are not the focus of this tool, nor are project management plans or static HTML pages typically stored there. Snippets are specifically tailored for code and text, reinforcing their role as a tool for developers to streamline workflows and enhance collaboration.

7. What is a Git Workspace?

A. A directory where you store the repository on your computer

B. A remote server managing your code

C. A temporary storage area for uncommitted changes

D. A version control system for multiple teams

A Git Workspace refers to the environment where a user interacts with a Git repository on their local machine. It is essentially a directory on your computer where the Git repository is stored. This directory contains the complete history of the project and all its files, allowing users to perform various Git operations such as adding, committing, and checking out branches. By working within a Git Workspace, developers can easily change and manage their project without affecting the original codebase until they decide to push their changes back to a remote repository. In contrast to other options, a Git Workspace is specifically connected to the local storage of a user's project, as opposed to a remote server, temporary storage for uncommitted changes, or a system for coordinating work among multiple teams. Each of these alternatives serves different purposes related to code management but does not define the primary function of a Git Workspace.

8. What role does versioning play in a system like Git?

A. It helps eliminate the need for collaboration

B. It maintains a complete history of changes made

C. It only tracks final versions of files

D. It is unnecessary in a distributed system

Versioning is a fundamental aspect of a system like Git as it maintains a complete history of changes made to the files within a repository. This capability allows users to track modifications over time, making it possible to see who made specific changes, when they were made, and what those changes entailed. The history created by versioning supports various workflows, like reverting to previous states, comparing changes between different versions, and facilitating collaboration among multiple contributors by allowing them to understand the evolution of the project. By maintaining a detailed log of changes, Git provides transparency and accountability, enabling teams to work more effectively together. Each commit in Git serves as a snapshot of the project at a certain point in time, and users can navigate through these snapshots to recover older versions of files, investigate bugs, or understand the rationale behind certain changes. This is essential in collaborative environments where multiple developers may work on the same codebase simultaneously. The other options suggest misunderstandings about versioning's significance in Git. For instance, the idea that it eliminates the need for collaboration ignores that versioning enhances collaboration by providing a clear record of contributions. Tracking only final versions or deeming it unnecessary in a distributed system also misrepresents the essential role that versioning plays in ensuring that all contributions

9. How do you resolve a merge conflict in GitLab?

A. By deleting the conflicting files

B. By merging the branches without changes

C. By manually editing conflicting files, staging them, and committing

D. By undoing the last commit

To resolve a merge conflict in GitLab, the most effective approach is to manually edit the conflicting files, stage them, and commit the changes. When two branches are merged and edits to the same lines of code conflict, Git will mark these sections in the code. To resolve this, you need to open the files, review the conflicting changes, make appropriate modifications to achieve the desired integration, and then stage these files to signal that you have resolved the conflicts. Once staged, committing the changes finalizes the resolution of the merge conflict. This process is essential as it allows for careful consideration of the different changes made in each branch, ensuring that the final merged result accurately reflects what you want to be included in the project. It highlights the collaborative nature of working in Git, where team members may have contributions that need thoughtful integration.

10. If the DOCKER_IMAGE CI/CD variable is not specified, what does CI_APPLICATION_REPOSITORY default to?

A. $CI_COMMIT_SHA

B. $CI_REGISTRY_IMAGE/$CI_COMMIT_REF_SLUG

C. $CI_PROJECT_DIR

D. docker.io/default

When the DOCKER_IMAGE CI/CD variable is not specified, the CI_APPLICATION_REPOSITORY defaults to the value of $CI_REGISTRY_IMAGE/$CI_COMMIT_REF_SLUG. This default behavior provides a structured naming convention for Docker images built within the CI/CD process, ensuring that the resulting images are easily identifiable and linked to specific commits and branches. The $CI_REGISTRY_IMAGE variable typically contains the path to the project's container registry that is set in the GitLab project settings. This allows for the images to be stored in the correct registry associated with the project. Meanwhile, $CI_COMMIT_REF_SLUG generates a URL-friendly version of the branch or tag name, which helps in organizing images according to their associated source code version. By combining these two variables, the system creates a clear, standardized format for the image repository that reflects both the project's registration in the container registry and the precise context of the code version being built, which aids in traceability and organization of docker images in the CI/CD pipeline.