1. Question

You need to store a sensitive database password for your organization‘s GitHub Actions workflows. As an organization owner, how can you create a secure secret accessible only to specific repositories within your organization?

* **create an organization-level secret and configure a policy to limit access to only the specific repositories that can use it**
* create a repository secret in the main project repository and manually share it with other needed repositories
* hardcode the password directly within your workflows for ease of access
* allow public access to the main repository and utilize private workflow permissions for individual runs

**Correct**

Create an organization-level secret and configure a policy to limit access to only the specific repositories that can use it offers the best balance of security and efficiency since it stores the password securely and centrally, reducing redundancy and simplifying management. It also provides granular control by restricting access to only the authorized repositories using the password.

2. Question

Which keyword within a GitHub workflow configuration is used to match a triggering event?

* event:
* run:
* when:
* **on:**

**Correct**

A workflow run is triggered for any workflows that have on: values that match the triggering event. For example, you might use on: push to trigger the workflow when a push event occurs or on: pull\_request to trigger the workflow when a pull request is opened or updated.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3. Question

What is the primary purpose of service containers in a GitHub Actions workflow?

* **create Docker images**
* host the workflow steps
* **provide a simple and portable way to host services needed for testing or operating applications**
* define workflow triggers

**Correct**

Service containers are Docker containers that offer a simple and portable way to host services needed for testing or operating applications  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

4. Question

A new self-hosted runner was recently registered with your organization, but you don‘t see it in the runner group assigned to your team. Why can‘t you use the new runner?

* there might be network connectivity issues preventing the runner from appearing in the assigned group
* the new runner is still undergoing initialization and configuration processes
* **new runners are automatically assigned to a default group, therefore it needs to be moved to the group used by your team**
* the permissions for accessing the runner group might not be granted to your team

**Correct**

If a newly registered self-hosted runner is not visible in the assigned runner group, it is likely because the runner has not yet been assigned to any runner group. Runner visibility in groups is determined by assignment, so if the runner has not been explicitly assigned to a group, it will not appear in any group, even if it has been successfully registered with the organization. Therefore, ensuring that the new runner is properly assigned to the appropriate runner group will resolve the issue of its visibility within the team‘s designated group.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Ref link: <https://docs.github.com/en/actions/hosting-your-own-runners/managing-self-hosted-runners/managing-access-to-self-hosted-runners-using-groups#moving-a-self-hosted-runner-to-a-group>   
  
5. Question

Which configuration is appropriate for triggering a workflow on a commit to a feature branch?

* on: push
* on:  
  push:  
  types:  
  - ‘feature‘
* on:  
  commit:  
  branches:  
  - ‘feature/\*‘
* **on:  
  push:  
  branches:  
  - ‘feature/\*‘**

**Correct**

on:  
push:  
branches:  
– ‘feature/\*‘  
With this configuration, the workflow will run whenever a commit is pushed to any branch that starts with feature/. Adjust the branch pattern (feature/\*) as needed to match your specific naming conventions for feature branches.  
on: Specifies the events that trigger the workflow.  
push: Indicates that the workflow should trigger on a push event.  
branches: Specifies the branches that should trigger the workflow. In this case, it‘s set to trigger on any push to branches starting with feature/.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link: <https://docs.github.com/en/actions/writing-workflows/choosing-when-your-workflow-runs/triggering-a-workflow>

6. Question

How can you customize the schedule of a GitHub actions workflow to run on weekdays only?

* add a condition in the workflow YAML for weekdays
* specify the schedule in the repository settings
* utilize the on: schedule: weekdays configuration
* **use the on: schedule: cron syntax**

**Correct**

You can schedule a workflow to run at specific UTC times using POSIX cron syntax. So using the on: schedule: cron then the correct expression, you can customize your workflow to run on weekdays only.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
Ref Link: <https://docs.github.com/en/actions/writing-workflows/choosing-when-your-workflow-runs/events-that-trigger-workflows#schedule>  
  
  
  
7. Question

What is the primary purpose of using workflow commands as a run step in a GitHub Actions workflow?

* **to communicate instructions and information to the runner environment**
* to define environment variables for the entire workflow
* to trigger the workflow to move to the next step
* to execute custom scripts on the runner

**Correct**

Workflow commands are used to interact with the runner during the execution of a step, providing a way to pass information, set environment variables, or perform other actions related to the workflow

Ref link: <https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/workflow-commands-for-github-actions>

8. Question

What action should be taken if you want to find the expiration date of a specific artifact?

* use the Artifacts tab on the GitHub repository page
* navigate to the Actions tab, click on the workflow, and check the summary for expiration information
* the expiration date of an artifact cannot be determined in GitHub Actions
* **execute a specific API call to retrieve the expiration date**

**Correct**

The expiration date of an artifact can be confirmed using the API, specifically by checking the expires\_at value returned by the Actions API  
  
ref link: <https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/storing-and-sharing-data-from-a-workflow>  
  
9. Question

What is the purpose of the jobs section in a GitHub Actions workflow?

* to define environment variables
* **to declare the workflows triggering events**
* **to organize and define the steps to be executed in parallel or sequentially**
* to specify the workflow name

**Correct**

The jobs section in a GitHub Actions workflow defines individual jobs, and within each job, you can specify a series of steps to be executed. Jobs can run in parallel, and you can control the flow of execution using dependencies between jobs.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref link: <https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/using-jobs-in-a-workflow>

10. Question

Your organization uses self-hosted runners for GitHub Actions and wants to implement security best practices. How can you control access to specific runners for different repositories across teams?

* limit public access to repositories and rely on individual user permissions
* **assign runners to groups and grant repository access permissions at the group level**
* configure individual access policies for each runner within each repository
* use workflow labels to specify which runners each workflow can utilize

**Correct**

Runner groups are used to collect sets of runners and create a security boundary around them. You can then decide which organizations or repositories are permitted to run jobs on those sets of machines. Organization owners can configure access policies that control which repositories in an organization have access to the runner group.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref link: <https://docs.github.com/en/actions/hosting-your-own-runners/managing-self-hosted-runners/managing-access-to-self-hosted-runners-using-groups>  
  
11. Question

You are working on a project within your organization that requires a custom GitHub Action. However, due to the sensitive nature of the project, you cannot make the action public. What approach should you take to ensure you can still utilize this action within your project?

* copy the action code into your project‘s repository and include it directly in your workflow files
* **allow GitHub Actions workflows in your private repository to access another private repository containing the custom action**
* utilize a separate public repository to host the custom action and grant access to specific users or organizations
* convert the custom action into a Docker container and store it in a private container registry.

**Correct** :To maintain the privacy of sensitive projects while still leveraging custom GitHub Actions, you can allow GitHub Actions workflows in your private repository to access another private repository containing the custom action or reusable workflow. This approach ensures that actions or workflows stored in private repositories can be used within workflows defined in other private repositories owned by the same organization or user, preserving security and privacy.  
Ref link: <https://docs.github.com/en/actions/sharing-automations/sharing-actions-and-workflows-from-your-private-repository>  
12. Question

While executing a GitHub Actions workflow, you encounter an issue where one of the actions fails unexpectedly. How does GitHub interpret the exit code of an action?

* **GitHub interprets a zero exit code as success, indicating that the action was completed successfully and other tasks can proceed**
* GitHub interprets a nonzero exit code as success, allowing the workflow to continue without interruption
* GitHub ignores the exit code of actions and relies solely on manual intervention to determine the success or failure of a workflow run
* GitHub treats all exit codes, regardless of their value, as failures, leading to the termination of the workflow run

**Correct**

In GitHub Actions, the exit code of an action determines its success or failure status. A zero exit code indicates success, allowing the workflow to execute subsequent tasks. Conversely, a nonzero exit code signifies failure, leading to canceling concurrent actions and skipping future actions in the workflow. This distinction enables GitHub to accurately assess the outcome of each action and manage workflow execution accordingly.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref link: <https://docs.github.com/en/actions/sharing-automations/creating-actions/setting-exit-codes-for-actions>

13. Question

How long does the GITHUB\_TOKEN last, and when does it expire?

* 12 hours, regardless of job completion
* **after a job finishes or after a maximum of 24 hours**
* it does not expire, and its validity is unlimited
* 48 hours, regardless of workflow events

**Correct**

The GITHUB\_TOKEN expires either when a job finishes or after a maximum of 24 hours.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Ref Link: <https://docs.github.com/en/actions/security-for-github-actions/security-guides/automatic-token-authentication>

14. Question

Your organization requires IP allowlists to protect internal resources accessed by GitHub Actions workflows. Most of your workflows run on GitHub-hosted runners, with both Windows and macOS needs. How can you achieve this desired security while ensuring workflow reliability?

* implement self-hosted runners on-premises with specific IP addresses added to the allowlist
* **utilize large runners with static IP address ranges and add these ranges to the allowlist**
* create separate workflows for accessing internal resources, using only self-hosted runners and bypassing IP allowlists entirely
* configure IP allowlists on internal resources to include the entire Azure IP address range, as GitHub-hosted Windows and Ubuntu runners operate within Azure.

**Correct**

Utilizing large runners with static IP address ranges and adding these ranges to the allowlist offers a secure and manageable solution because larger runners provide increased flexibility and isolation compared to standard hosted runners. Their static IP address ranges simplify allowlist configuration, enabling secure access to internal resources. This approach balances security and efficiency, ensuring reliable workflows with appropriate protections.  
Ref Link: <https://docs.github.com/en/actions/using-github-hosted-runners/using-github-hosted-runners/about-github-hosted-runners#ip-addresses>  
  
  
15. Question

How can you specify dependencies between jobs in a workflow?

* by using the depends-on keyword in each job definition
* by setting environment variables between jobs
* by organizing jobs into separate workflow files
* **by defining dependencies in the workflow YAML file**

**Correct**

Dependencies are specified using the needs keyword in the YAML file, where you list the names of the jobs that a particular job depends on.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Ref Link: <https://docs.github.com/en/actions/writing-workflows/workflow-syntax-for-github-actions#jobsjob_idneeds>

16. Question

On GitHub-hosted runners, what is recorded in the “Set up job“ step of a given job? (select three)

* **runner image**
* **operating system**
* **GITHUB\_TOKEN permissions**
* code vulnerability scan

**Correct**

For jobs run on GitHub-hosted runners, “Set up job“ records details of the runner image, and includes a link to the list of preinstalled tools that were present on the runner machine.  Below is an example of what is included in the “Set up job“ step  
Current runner version: ‘2.313.0‘  
Operating System  
  Ubuntu  
  22.04.4  
  LTS  
Runner Image  
  Image: ubuntu-22.04  
  Version: 20240218.1.0  
  Included Software: <https://github.com/actions/runner-images/blob/ubuntu22/20240218.1/images/ubuntu/Ubuntu2204-Readme.md>  
  Image Release: <https://github.com/actions/runner-images/releases/tag/ubuntu22%2F20240218.1>  
Runner Image Provisioner  
  2.0.341.1  
GITHUB\_TOKEN Permissions  
  Contents: write  
  Metadata: read  
Secret source: Actions  
Prepare workflow directory  
Prepare all required actions  
Getting action download info  
Download action repository ‘actions/checkout@v4‘ (SHA:b4ffde65f46336ab88eb53be808477a3936bae11)  
Complete job name: Check current step number  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link: <https://docs.github.com/en/actions/writing-workflows/choosing-when-your-workflow-runs/triggering-a-workflow>

17. Question

Which context property can be used to access information about the event that triggered a workflow run?

* github.repository
* github.job
* jobs..result
* **github.event**

**Correct**

Information about the event that triggered a workflow run is available in the github.event context. The properties in the github.event context depend on the type of event that triggered the workflow.  You can use the github.event context in your workflow.  The github context contains information about the workflow run and the event that triggered the run  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref link: <https://docs.github.com/en/actions/writing-workflows/choosing-when-your-workflow-runs/triggering-a-workflow#using-event-information>  
  
18. Question

What is the purpose of the continue-on-error keyword in a GitHub Actions step?

* it allows the workflow to continue even if the step fails
* it defines conditions under which a step should be retried
* it indicates that the step should always be executed, regardless of errors
* **it specifies the steps to run only on error conditions**

**Correct**

The continue-on-error keyword in a step allows the workflow to continue even if a step fails. The user would have to set this to true to allow this to happen. This can be useful in scenarios where you want to proceed with the workflow despite non-critical step failures.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Ref Link: <https://docs.github.com/en/actions/writing-workflows/workflow-syntax-for-github-actions#jobsjob_idcontinue-on-error>

19. Question

What is one of the main reasons for hosting a GitHub Action in a separate repository when making it public?

* it reduces the need for version control
* **it makes it easier for developers to extend and fix issues with the action**
* it limits the visibility of the action to the repository where it‘s hosted
* it increases the complexity of managing the action

**Correct**

The explanation of this topic revolves around the decision-making process when creating a GitHub Action, particularly regarding where to host the action and its visibility, whether it will be public or private.  
For public actions, it‘s recommended to host them in their own separate repository rather than bundling them with other application code. This approach allows for better versioning, tracking, and release management, similar to any other software project. By having its own repository, it becomes easier for the GitHub community to discover the action, aids developers in extending and fixing issues, and separates the action‘s versioning from the application code‘s versioning.  
On the other hand, for private actions, they can be stored in any location within the repository. However, only workflows within the same repository can use these actions. When combining action, workflow, and application code in a single repository, it‘s recommended to store the action in the .github directory. This helps in organizing the repository and clarifying the purpose of each component.  
In summary, the decision of whether to make an action public or private and where to host it depends on factors such as visibility, collaboration, and version control needs.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link : <https://docs.github.com/en/actions/sharing-automations/creating-actions/publishing-actions-in-github-marketplace>  
  
20. Question

In a workflow requiring review, what action is taken if a job is rejected?

* **the workflow fails**
* the job is put on hold until further notice
* the job is automatically re-submitted for review
* the workflow proceeds as normal

**Correct**

Once a job is approved, the job will proceed. But, if a job is rejected, the workflow will fail.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

21. Question

The workflow logs do not provide enough detail to diagnose the problem with the recent Javascript action you completed. What steps should you take to continue troubleshooting the issue?

* convert the JavaScript action to a Docker container action for more detailed logging and isolation
* review the JavaScript action‘s code for potential errors or misconfigurations
* use the GitHub Actions API to retrieve additional diagnostic information about the failed job
* **enable debug logging to increase the verbosity of the job‘s logs**

Correct

Enabling debug logging in the GitHub repository settings increases the verbosity of the job‘s logs, providing more detailed information that can help troubleshoot issues with the JavaScript action. This is a common first step to gather additional diagnostic information about the failed job.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link: <https://docs.github.com/en/actions/monitoring-and-troubleshooting-workflows/troubleshooting-workflows/enabling-debug-logging>  
  
22. Question

What essential step is involved in deploying a release to a cloud provider using a GitHub Actions workflow?

* excluding the workflow YAML file from the repository
* **defining deployment steps in the GitHub Actions workflow YAML file**
* avoiding the use of environment variables during the deployment process
* configuring multiple workflows in separate repositories

**Correct**

Deployment steps are typically defined in the YAML file associated with the workflow. So, to deploy a release to a cloud provider, you would have to configure the YAML file to this task  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link: <https://docs.github.com/en/actions/use-cases-and-examples/deploying/deploying-to-google-kubernetes-engine>

23. Question

Which of the following statements accurately describes the behavior of workflow jobs referencing an environment‘s protection rules?

* **workflow jobs won‘t start until all of the environment‘s protection rules pass**
* workflow jobs will start immediately, regardless of the environment‘s protection rules
* workflow jobs will only start if some of the environment‘s protection rules pass
* workflow jobs will never start if the environment has protection rules

**Correct**

When a workflow job references an environment, the job won‘t start until all of the environment‘s protection rules pass  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link: <https://docs.github.com/en/actions/managing-workflow-runs-and-deployments/managing-deployments/managing-environments-for-deployment>  
  
24. Question

What is the recommended approach for storing secrets larger than 48 KB?

* secrets larger than 48 KB cannot be stored
* **use a workaround involving encryption with GPG and storing the decryption passphrase as a secret**
* store large secrets directly as repository secrets to avoid limitations
* avoid storing large secrets entirely to ensure security

**Correct**

To store secrets larger than 48 KB, you have to use encryption with GPG. The encrypted file is stored in the repository, and the decryption passphrase is stored as a secret on GitHub.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref link: https://docs.github.com/en/actions/security-for-github-actions/security-guides/using-secrets-in-github-actions#limits-for-secrets

25. Question

You have committed the required files to your GitHub repo for the Docker container action, but the action is failing to run. What could be the issue?  
prd-app-repo/  
|–dockerfile  
|–action.yml  
|–script.sh  
|–README

* **the file names are case-sensitive, therefore change dockerfile to Dockerfile**
* a Docker container action doesn‘t use the action.yml file, therefore causing it to fail
* the README file should be outside of the directory supporting the Docker container action
* the script.sh file is not referenced in the README file

**correct**

Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Make sure that your filename is capitalized correctly (use a capital D but not a capital f) if you‘re having issues.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Ref link: <https://docs.github.com/en/actions/sharing-automations/creating-actions/creating-a-docker-container-action>  
  
26. Question

Heather is troubleshooting a failure on one of her GitHub Actions workflow runs for recent pull request.  Where can she view the logs to determine why the run may have failed? (select two)

* **in the “Checks“ tab of a pull request**
* in the “Issues“ tab of the repository
* in the “Insights“ tab of the repository
* **in the “Actions“ tab of the repository**

**correct**

Heather can view the logs for the GitHub Action workflow runs in the “Actions“ tab of her repository. This tab provides detailed logs for each workflow run, including information about each step and any errors encountered.  
Since the workflow was triggered by a pull request, Heather can also view the logs on the “Checks“ tab of the pull request. This tab displays the status of the workflow checks for that specific pull request, including any logs generated during the workflow run.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref link: <https://docs.github.com/en/actions/monitoring-and-troubleshooting-workflows/monitoring-workflows/using-workflow-run-logs>  
  
27. Question

You need to create a custom Javascript action for your organization, but there are problems in the actions.yml file in the code snippet shown below. What is the problem with the code?  
runs:  
using: ‘node12‘  
steps: ‘main.js‘

* **The Javascript action does not use the steps argument. Change it and use the main argument instead**
* The Javascript action requires node20, and the code above references node12
* The Javascript action does not require Node.js and therefore this action will fail
* The Javascript action does not use the action.yml file, it requires a index.js file instead

**Correct**

JavaScript actions can run directly on the runner machine and separate the action code from the environment used to run the action. Because of this, the action code is simplified and can execute faster than actions within a Docker container.  
As a prerequisite for creating and using packaged JavaScript actions, you need to download Node.js, which includes npm. As an optional step (but one that we recommend), use GitHub Actions Toolkit Node.js, which is a collection of Node.js packages that allows you to quickly build JavaScript actions with more consistency.  
JavaScript actions require that the runs statement takes the following two arguments:  
using: Application used to execute the code as defined in main  
main: File that contains the action code; the application defined in using executes this file  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Links: <https://docs.github.com/en/actions/sharing-automations/creating-actions/creating-a-javascript-action>  
  
28. Question

How do actions, workflows, jobs, steps, and runs collaborate in a typical scenario?

* steps initiate actions, which are executed within workflows and result in runs within jobs
* **actions initiate runs, which consist of jobs that execute workflows with individual steps**
* runs initiate actions, which consist of steps executed within jobs and workflows
* jobs initiate runs, which consist of actions with individual steps in a collaborative workflow

**correct**

Actions initiate runs, which consist of jobs that execute workflows with individual steps. This reflects the typical flow where actions initiate the execution of runs, which are composed of jobs that, in turn, contain steps defining specific tasks within the workflow  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Ref Link: <https://docs.github.com/en/actions/writing-workflows/about-workflows>  
  
29. Question

Jess is looking to programmatically access a set of workflow logs for a public repository.  What pieces of information are required for accessing a set of workflow logs?

* owner, repo, and authentication token
* **owner, repo and run\_id**
* repo, authentication token, and run\_id
* owner, repo and job\_id

**Correct**

To access workflow logs programmatically, Jess needs to provide the owner (username or organization name) and the repository name, which specify the repository containing the workflows. Additionally, the run\_id uniquely identifies a particular workflow run, allowing Jess to target the specific logs she wants to access.  Here is a sample curl request to programmatically access logs using these pieces of information:  
curl -L \  
  -H “Accept: application/vnd.github+json“ \  
  -H “X-GitHub-Api-Version: 2022-11-28“ \  
  <https://api.github.com/repos/OWNER/REPO/actions/runs/RUN_ID/logs>  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref link: <https://docs.github.com/en/rest/actions/workflow-runs?apiVersion=2022-11-28#download-workflow-run-logs>  
  
0. Question

You are a developer working on a project hosted on GitHub, and you‘ve created a custom action to automate the process of deploying your application to a staging environment. The action is designed to run in a Docker container and requires several input variables to function correctly. This action could benefit other developers in the GitHub community and want to share it.  
What steps should you take to share your custom action with the GitHub community?

* ensure your repository is public, define the action‘s inputs, outputs, and environment variables, and publish the action as a Docker container
* confirm that your repository is public, define the action‘s inputs, outputs, and environment variables, and manually distribute the action‘s Docker image to interested developers
* **verify that your repository is public, define the action‘s inputs, outputs, and environment variables, and publish the action to the GitHub Marketplace**
* make sure your repository is public, define the action‘s inputs, outputs, and environment variables, and share the action‘s code directly in the repository README

**Correct**

**Answer: C. Verify that your repository is public, define the action’s inputs, outputs, and environment variables, and publish the action to the GitHub Marketplace.**

**Explanation:**

**Correct Option:**

* **C. Publish the action to the GitHub Marketplace:** This is the most effective way to share your custom action with the GitHub community. By publishing it on the Marketplace, it becomes easily discoverable, installable, and reusable by other developers. This option outlines the proper steps to make your action accessible to the community. By publishing to the GitHub Marketplace, your action becomes easily searchable and can be utilized by other developers, enhancing its visibility and usage.

31. Question

What is the recommended practice for treating environment variables in GitHub Actions, regardless of the operating system and shell used?

* use only uppercase letters for environment variable names
* ignore case sensitivity as GitHub Actions handles it automatically
* **treat environment variables as case-sensitive**
* depend on the behavior of the operating system in use

**correct**

Treating environment variables as case-sensitive is good practice to avoid issues and maintain consistency in the workflow  
Ref Link: <https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/workflow-commands-for-github-actions>  
  
32. Question

John is troubleshooting a failed workflow run and would like to view the workflow file associated with the failed run.  What option can he select within the failed run‘s menu to easily view the workflow file?

* Create status badge
* View raw logs
* Download log archive
* **View workflow file**

**Correct**

To easily view the workflow file associated with a failed run, John can select the “View workflow file“ option in the failed run‘s menu. This option allows him to quickly access and inspect the workflow file that was used for the failed run, helping him troubleshoot the issue more effectively.  
Ref Link: <https://docs.github.com/en/actions/monitoring-and-troubleshooting-workflows/monitoring-workflows/viewing-workflow-run-history>  
  
33. Question

How can you add the action you‘ve created to the GitHub Marketplace?

* **tagging it as a new release and then publishing it**
* merging a pull request from another user
* creating a new branch in your repository
* renaming the repository to match an existing GitHub feature

**Correct**

You can add the action you‘ve created to the GitHub Marketplace by tagging it as a new release and then publishing it. There are a few guided steps in GitHub that allow you to publish a release of your action. You can find more information on these steps in the Summary section at the end of this module.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Ref Link: <https://docs.github.com/en/actions/sharing-automations/creating-actions/publishing-actions-in-github-marketplace>  
  
34. Question

Steve would like to use an action in his GitHub project but wants to validate that it is trustworthy before using it.  What steps can Steve take to verify the integrity of a GitHub action before deciding to use it? (select three)

* Check the action‘s GitHub stars
* **Review the action‘s action.yml file to make sure the code does what it says it does**
* **Check if the action is verified in the GitHub Marketplace**
* **Check if the action is in the GitHub Marketplace**

**Correct**

Review the action‘s action.yml file for inputs, outputs, and to make sure the code does what it says it does.  
Check if the action is in the GitHub Marketplace. This is a good check, even if an action does not have to be on the GitHub Marketplace to be valid.  
Check if the action is verified in the GitHub Marketplace. This means that GitHub has approved the use of this action. However, you should still review it before using it.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link: <https://learn.microsoft.com/en-us/training/modules/github-actions-automate-tasks/2-github-actions-automate-development-tasks>  
  
  
35. Question

Jeff is troubleshooting an error within his workflow configuration causing it not to run.  What is wrong with this workflow configuration?  
name: learn-github-actions  
run-name: ${{ github.actor }} is testing out GitHub Actions ??  
jobs:  
Explore-GitHub-Actions:  
runs-on: ubuntu-latest  
steps:  
– run: echo “?? The job was automatically triggered by a ${{ github.event\_name }} event.“  
– run: echo “?? This job is now running on a ${{ runner.os }} server hosted by GitHub!“  
– run: echo “?? The name of your branch is ${{ github.ref }} and your repository is ${{ github.repository }}.“  
– name: List files in the repository  
run: |  
ls ${{ github.workspace }}

* each run: keyword must have an unqiue step: keyword before it
* the indentation of the configuration is invalid
* **missing the on: keyword to specify what events will trigger the workflow**
* the name: keyword is not allowed under the steps section of the configuration

**Correct**

In this configuration, there is no  on: key specified, which means the workflow will not be triggered by any events. Without the on: key, the workflow will not have any triggers defined, and therefore, it won‘t run automatically in response to any events such as pushes, pull requests, or other GitHub Actions events.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link: <https://docs.github.com/en/actions/writing-workflows/workflow-syntax-for-github-actions>  
  
36. Question

In GitHub Actions, if you define both branches and paths filter, what is the effect on the workflow execution?

* the workflow will run when either branches or paths are satisfied
* **the workflow will only run when both branches and paths are satisfied**
* the workflow will not run when both branches and paths are satisfied
* the workflow will run when either branches or paths are satisfied, but will only apply the matching filter

**Correct**

Use the branches filter when you want to include branch name patterns or when you want to both include and exclude branch name patterns. Use the branches-ignore filter when you only want to exclude branch name patterns. You cannot use both the branches and branches-ignore filters for the same event in a workflow.  
If you define both branches/branches-ignore and paths/paths-ignore, the workflow will only run when both filters are satisfied.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link <https://docs.github.com/en/actions/writing-workflows/workflow-syntax-for-github-actions#onpull_requestpull_request_targetbranchesbranches-ignore>  
  
37. Question

You need to create a new action metadata file. What syntax should you use?

* JSON
* Python
* **YAML**
* Javascript

**Correct**

All actions require a metadata file. The metadata filename must be either action.yml or action.yaml. The data in the metadata file defines the inputs, outputs, and runs configuration for your action.  
Action metadata files use YAML syntax.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link: <https://docs.github.com/en/actions/sharing-automations/creating-actions/metadata-syntax-for-github-actions#about-yaml-syntax-for-github-actions>

38. Question

What are valid custom actions types within GitHub? (select three)

* composable actions
* **composite actions**
* **JavaScript actions**
* **docker container actions**

**Correct**

You can build Docker container, JavaScript, and composite actions.  
Docker container actions: These are actions packaged as Docker containers, allowing developers to encapsulate their actions and dependencies within a container. Docker container actions provide a consistent and reproducible environment for executing actions across different platforms and environments.  
JavaScript actions: JavaScript actions are written in JavaScript and run within the Node.js runtime environment. They provide flexibility for developers who are comfortable with JavaScript and want to leverage its ecosystem and libraries to create custom actions for their GitHub workflows.  
Composite actions: Composite actions are reusable actions composed of other actions, allowing developers to encapsulate common patterns or sequences of steps into a single action. They promote code reusability and maintainability by enabling developers to define complex workflows as composable units that can be easily shared and reused across projects.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link : <https://docs.github.com/en/actions/sharing-automations/creating-actions/about-custom-actions>

39. Question

How does the cache action in a GitHub Actions handle a cache miss?

* by searching for a cache in other repositories
* by terminating the workflow if a cache miss occurs
* **by automatically creating a new cache if the job is completed successfully**
* by requiring manual intervention to create a new cache

**Correct**

If there is a cache miss and the job completes successfully, the cache action automatically creates a new cache with the specified key and contents  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Ref link: <https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/caching-dependencies-to-speed-up-workflows#managing-caches>

40. Question

What level of access is required to download workflow artifacts?

* owner
* admin
* **read**
* write

**Correct**

People who are signed into GitHub and have read access to a repository can download workflow artifacts.

Ref link: <https://docs.github.com/en/actions/managing-workflow-runs-and-deployments/managing-workflow-runs/downloading-workflow-artifacts>  
  
41. Question

What is the purpose of a job and its associated steps when using job steps for actions and shell commands?

* define the overall workflow structure
* specify the repository owner and collaborators
* **define a specific task or unit of work with a sequence of steps**
* configure the webhook events triggering the workflow

**Correct**

A job is a unit of work that can contain a sequence of steps. These steps include a combination of actions and shell commands. The primary purpose of a job is to define and execute a specific task or unit of work within the workflow.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link: <https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/workflow-commands-for-github-actions>

42. Question

Rather than using code to create an error annotation, what can you use to send commands to the runner to create the same error annotation?

* **workflow commands provided by the actions/toolkit**
* the set-env command
* environment variables prepended with the RUNNER\_\* annotation
* python scripts that create the specific error messages required

**Correct**

In GitHub Actions, you can use workflow commands to send instructions to the runner environment. These commands allow you to perform various tasks, including creating error annotations, setting environment variables, printing debug messages, and more. By using workflow commands, you can create error annotations without directly modifying the workflow code.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link: <https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/workflow-commands-for-github-actions#using-workflow-commands-to-access-toolkit-functions>  
  
43. Question

If an organization‘s templated workflow contains secret information such as ${{ secrets.token }}, what needs to be configured before using the workflow?

* configure the workflow file to remove the secret information
* skip configuring secrets and rely on default values
* replace the ${{ secrets.token }} with the token‘s value
* **create a secret named token in your repository**

**Correct**

Because the templated workflow contains secret information $ {{ secrets.token }}, the organization needs to configure the secrets before using the templated workflow. These secrets should be added to the organization‘s secrets store in GitHub.  Create a secret named token and store its value as a secret in your repository.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link: <https://docs.github.com/en/actions/writing-workflows/using-workflow-templates>  
  
44. Question

You are planning to manage reusable workflows for your organization within GitHub Actions. Which approach is recommended for optimal organization and maintainability?

* utilize the workflow\_call trigger within workflows to call reusable workflows from other repositories
* reuse workflows directly from individual project repositories.
* implement version control for reusable workflows using branches and tags
* **create a dedicated repository to store and manage all reusable workflows**

**Correct**

While setting up reusable workflows in your target repository and reusing them in other workflows are standard ways supported by GitHub Actions, it also supports reusing workflows from other repositories in the same user account or organization. You can make use of this feature to better organize your reusable workflows.  
If you plan to set up multiple reusable workflows in your organization, it might be a good idea to set up a common workflow repository. This allows you to track the development of reusable workflows better, reuse them in a standard fashion across your organization, and version them for easier access.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
45. Question

Where can you set custom environment variables in a workflow?

* in your profile settings
* in the main code file
* only in command-line arguments
* **in the workflow file**

**Correct**

To set a custom environment variable for a single workflow, you can define it using the env key in the workflow file. The scope of a custom variable set by this method is limited to the element in which it is defined.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link: <https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/store-information-in-variables#defining-environment-variables-for-a-single-workflow>  
  
  
46. Question

As you prepare to distribute your custom GitHub Action, what best practice should you follow to enhance its visibility and utility for potential users?

* **offer a clear description of what the action accomplishes and select the most relevant category to accurately represent its utility**
* keep the description minimal to encourage exploration and experimentation by users
* include an extensive list of features in the description to attract a wider audience.
* provide a brief description of the action‘s functionality and select multiple categories to maximize its exposure

**Correct**

Providing a clear and concise description of the action‘s functionality enables users to quickly understand its purpose and potential benefits. Selecting the most relevant category ensures that the action is appropriately categorized, making it easier for users to discover and incorporate into their workflows.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
Ref Link: https://resources.github.com/learn/pathways/automation/advanced/building-your-first-custom-github-action/  
  
47. Question

How can you define a matrix for a job in a GitHub Actions workflow?

* **use the matrix keyword within the strategy configuration of the job**
* use the matrix keyword within the runs-on parameter
* use the matrix keyword within the workflow configuration
* use the variables section within the job definition

**Correct**

The matrix keyword is used within the strategy configuration of the job to define a matrix of different job configuration  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link: <https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/running-variations-of-jobs-in-a-workflow>  
  
48. Question

What will the value of the NAME variable be when this workflow is triggered?  
name: Java CI with Maven  
env:  
NAME: ‘My Action‘  
on:  
push:  
branches: [ “main“ ]  
jobs:  
build:  
env:  
NAME: ‘My Action 2‘  
runs-on: ubuntu-latest  
  
– name: Print name  
run: echo “$NAME“  
env:  
NAME: ‘My Action 3‘

* this run will error because of correct syntax
* My Action 2
* **My Action 3**
* My Action

**Correct**

My Action 3  
The workflow defines a top-level environment variable NAME with the value ‘My Action‘.  
Within the build job, a job-level environment variable NAME with the value ‘My Action 2‘ is defined. This overrides the top-level environment variable for the scope of the build job.  
Inside the “Print name“ step, another environment variable NAME with the value ‘My Action 3‘ is defined. This overrides both the top-level and job-level environment variables for the scope of the step.  
Therefore, when the “Print name“ step executes the command echo “$NAME“, it will print the value of the NAME variable defined within the step‘s environment, which is ‘My Action 3‘.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

49. Question

Laura would like to add a step to her workflow configuration that adds the /tmp directory to the PATH on an Ubuntu runner.  Which of the following configurations is valid?

* **name: my-workflow  
  on:  
  push:  
  branches: [ main ]  
  jobs:  
  Set-path-Ubuntu:  
  name: Add user‘s local bin to Ubuntu PATH  
  runs-on: ubuntu-latest  
  steps:  
  - uses: actions/checkout@v4  
  - run: echo “/tmp“ >> $GITHUB\_PATH**
* name: my-workflow  
  on:  
  push:  
  branches: [ main ]  
  jobs:  
  Set-path-Ubuntu:  
  name: Add user‘s local bin to Ubuntu PATH  
  runs-on: ubuntu-latest  
  steps:  
  - uses: actions/checkout@v4  
  - run: echo “/tmp“ >> $GITHUB\_ENV
* name: my-workflow  
  on:  
  push:  
  branches: [ main ]  
  jobs:  
  Set-path-Ubuntu:  
  name: Add user‘s local bin to Ubuntu PATH  
  runs-on: ubuntu-latest  
  steps:  
  - uses: actions/checkout@v4  
  - run: echo “/tmp“ >> $GITHUB\_STEP\_SUMMARY
* name: my-workflow  
  on:  
  push:  
  branches: [ main ]  
  jobs:  
  Set-path-Ubuntu:  
  name: Add user‘s local bin to Ubuntu PATH  
  runs-on: ubuntu-latest  
  steps:  
  - uses: actions/checkout@v4  
  - run: echo “/tmp“ >> $GITHUB\_OUTPUT

**Correct**

name: my-workflow  
on:  
push:  
branches: [ main ]  
jobs:  
Set-path-Ubuntu:  
name: Add user‘s local bin to Ubuntu PATH  
runs-on: ubuntu-latest  
steps:  
– uses: actions/checkout@v4  
– run: echo “/tmp“ >> $GITHUB\_PATH  
The run directive uses the echo “/tmp“ >> $GITHUB\_PATH command to add /tmp to the PATH on the Ubuntu runner.

50. Question

What distinguishes JavaScript actions from traditional Node.js projects regarding their development and distribution?

* JavaScript actions are limited to using only GitHub‘s APIs and do not integrate with third-party APIs
* JavaScript actions do not support dependent packages and tagged releases
* JavaScript actions do not require end-to-end testing due to their simplicity
* **JavaScript actions involve committing dependent packages alongside the code and can be published as tagged releases to the GitHub Marketplace**

**Correct**

JavaScript actions in GitHub repositories differ from traditional Node.js projects in their development and distribution processes. They include dependent packages alongside the code and support tagged releases for direct publication to GitHub Marketplace. Due to their integration with various APIs, robust end-to-end testing is encouraged to ensure functionality and security.

51. Question

You have created a YAML workflow file for a common task for your team‘s testing. However, you are unsure where to store it in your organization. Where do the workflow file and associated metadata file need to be placed?

* in a separate dedicated repository specifically for workflow files
* in the root directory of the repository where the task will be executed
* **in the .github/workflows directory within the repository where the task will be executed**
* in the organization‘s settings, under the Workflows section

**Correct**

Reusable workflows are YAML-formatted files, very similar to any other workflow file. As with other workflow files, you locate reusable workflows in the .github/workflows directory of a repository. Subdirectories of the workflows directory are not supported.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link: <https://docs.github.com/en/actions/sharing-automations/reusing-workflows>  
  
52. Question

When setting up a GitHub Actions workflow, which of the following components are required? (select three)

* directory for storing artifacts
* **trigger events to initiate the workflow**
* **workflow file in YAML format**
* **action metadata file**
* environment variables for configuring the workflow environment

**Correct**

A workflow file in YAML format is required to define the steps and actions to be executed in the workflow. An action metadata file (action.yml or action.yaml) is required for creating custom actions. Trigger events are necessary to initiate the workflow, specifying when it should be triggered, such as on pushes, pull requests, or scheduled intervals.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

53. Question

Your company uses GitHub Actions for internal projects. You want to share custom actions and scripts across teams but keep them private. How should you distribute these components?

* store them in a public repository with access control
* use publicly available actions even if they need modifications
* have developers store them directly in project repositories
* **store them in a private repository with access control**

**Correct**

Storing the components in a private repository ensures that only authorized team members can access them. Implementing access control further refines usage by granting permissions based on specific project needs. Therefore, utilizing a private repository with access control offers a secure and efficient solution for sharing internal components within the organization while maintaining privacy and facilitating efficient project development across teams.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

54. Question

Which of the following default environment variables contains the operating system of the runner executing the job?

* RUNNER\_ARCH
* GITHUB\_RUNNER\_OS
* RUNNER\_DEBUG
* **RUNNER\_OS**

**Correct**

The RUNNER\_OS environment variable provides the operating system of the runner executing the job. Possible values are Linux, Windows, or macOS.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

55. Question

Which of the following practices are recommended when managing releases for a GitHub Action using tags? (select three)

* **introduce a new major version tag for changes that will break existing workflows**
* **move the major version tag to point to the Git ref of the current release**
* use non-semantic versioning to name the release tags for better clarity
* create and validate a release directly on the main branch before creating the release tag
* **release major versions with a beta tag to indicate their status**

**Correct**

Tags can be a good way to manage releases for an action. By using tags, users can easily distinguish between major and minor versions. The following is a list of helpful practices to consider when creating releases:  
Create and validate a release on a release branch (such as release/v1) before creating the release tag (for example, v1.0.2).  
Use semantic versioning.  
Move the major version tag (such as v1, v2) to point to the Git ref of the current release.  
Introduce a new major version tag (v2) for changes that will break existing workflows.  
Release major versions with a beta tag to indicate their status; for example, v2-beta. You can remove the -beta tag when the release is ready.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

56. Question

April is in charge of auditing the operations team. While conducting a review, she noticed that many workflows are accessing secrets to carry out deployment and testing functions and is concerned that these secrets may appear in logs. What information can you provide to alleviate April‘s concerns about workflow logs?

* GitHub automatically encrypts secrets before printing them to workflow logs
* GitHub prohibits the printing of workflow logs entirely to ensure security
* GitHub prompts users to manually confirm before printing secrets to workflow logs
* **GitHub automatically redacts secrets printed to workflow logs, replacing them with placeholders**

**Correct**

GitHub ensures the security of secrets printed to workflow logs within GitHub Actions by automatically redacting them. When secrets are printed to workflow logs, GitHub replaces them with placeholders to prevent their exposure. This security measure helps protect sensitive information from being unintentionally leaked or accessed by unauthorized users, ensuring the confidentiality and integrity of secrets used within GitHub Actions workflows.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link: <https://docs.github.com/en/actions/security-for-github-actions/security-guides/using-secrets-in-github-actions#redacting-secrets-from-workflow-run-logs>  
  
  
57. Question

Your organization primarily runs its workloads on the Windows operating system and wants to start using Docker for building and testing jobs as it rearchitects its primary applications. What requirements must be met before using Docker container actions?

* the Windows-based runners must have Docker installed
* **the organization must use runners with a Linux operating system and have Docker installed**
* install and configure Docker on the developer‘s local machines
* upgrade all GitHub repositories to use the Windows operating system as the primary environment

**Correct**

Docker container actions can only be executed on runners with a Linux operating system. Self-hosted runners must use a Linux operating system and have Docker installed to run Docker container actions.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Ref Link: https://docs.github.com/en/actions/sharing-automations/creating-actions/about-custom-actions#docker-container-actions

58. Question

When selecting a runner for GitHub Actions that is cost-effective and requires minimal management, which option would be most suitable?

* **GitHub-hosted runners provided by GitHub**
* Self-hosted runners on physical servers maintained by the organization
* Cloud-hosted runners from a third-party provider
* Self-hosted runners on virtual machines managed by the organization

**Correct**

In scenarios where cost-effectiveness and minimal management are priorities, GitHub-hosted runners provided by GitHub are the most suitable option for GitHub Actions. These runners are maintained and managed by GitHub, eliminating the need for organizations to handle infrastructure setup, maintenance, and costs, thereby providing a hassle-free and economical solution for CI/CD workflows.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link: <https://docs.github.com/en/actions/using-github-hosted-runners/using-github-hosted-runners/about-github-hosted-runners>  
  
What is a valid workflow configuration to automatically publish a Node.js package to GitHub Packages when a release is published?

* name: Node.js Package  
  on:  
  release:  
  types: [published]  
  jobs:  
  publish:  
  runs-on: ubuntu-latest  
  steps:  
  - uses: actions/checkout@v4  
  - uses: actions/setup-node@v4  
  with:  
  node-version: ‘20.x‘  
  registry-url: ‘https://npm.pkg.github.com/‘  
  - run: npm ci  
  - run: npm publish
* name: Node.js Package  
  on:  
  release:  
  types: [published]  
  jobs:  
  publish:  
  runs-on: ubuntu-latest  
  steps:  
  - uses: actions/checkout@v4  
  - run: npm ci  
  - run: npm publish  
  env:  
  NODE\_AUTH\_TOKEN: ${{secrets.GITHUB\_TOKEN}}
* **name: Node.js Package  
  on:  
  release:  
  types: [published]  
  jobs:  
  publish:  
  runs-on: ubuntu-latest  
  steps:  
  - uses: actions/checkout@v4  
  - uses: actions/setup-node@v4  
  with:  
  node-version: ‘20.x‘  
  registry-url: ‘https://npm.pkg.github.com/‘  
  - run: npm ci  
  - run: npm publish  
  env:  
  NODE\_AUTH\_TOKEN: ${{secrets.GITHUB\_TOKEN}}**
* name: Node.js Package  
  on:  
  release:  
  types: [published]  
  jobs:  
  publish:  
  runs-on: ubuntu-latest  
  steps:  
  - uses: actions/checkout@v4  
  - uses: actions/setup-node@v4  
  with:  
  node-version: ‘20.x‘  
  - run: npm ci  
  - run: npm publish  
  env:  
  NODE\_AUTH\_TOKEN: ${{secrets.GITHUB\_TOKEN}}

**Correct**

name: Node.js Package  
on:  
release:  
types: [published]  
jobs:  
publish:  
runs-on: ubuntu-latest  
steps:  
– uses: actions/checkout@v4  
– uses: actions/setup-node@v4  
with:  
node-version: ‘20.x‘  
registry-url: ‘<https://npm.pkg.github.com/&#8216>;  
– run: npm ci  
– run: npm publish  
env:  
NODE\_AUTH\_TOKEN: ${{secrets.GITHUB\_TOKEN}}  
This configuration checks out the code repository, sets up the node configuration to specify the the GitHub Packages registry URL – <https://npm.pkg.github.com/&#8216>;, authenticates to registry using a GITHUB\_TOKEN and publishes the Node.js package via npm.  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

60. Question

How do custom labels determine the eligibility of a self-hosted runner to process a job?

* labels are mutually exclusive, and only one matching label is required
* labels are automatically assigned based on runner characteristics
* **labels are cumulative, and the runner must have all assigned labels to be eligible**
* labels operate independently, and any matching label makes the runner eligible

**Correct**

Custom labels in GitHub Actions operate cumulatively, meaning a self-hosted runner must have all assigned labels to be eligible to process a job  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ref Link: https://docs.github.com/en/actions/hosting-your-own-runners/managing-self-hosted-runners/using-self-hosted-runners-in-a-workflow#using-custom-labels-to-route-jobs