Environment Variables in GitHub Actions

GitHub Actions allows you to work with environment variables to customize and parameterize your workflows. Environment variables are key-value pairs that can store secrets, configuration settings, or other data.

- You can use environment variables in GitHub Actions workflows to:
    - Store secrets securely.
    - Define parameters or configuration settings.
    - Share information between workflow steps or jobs.
- **Define Workflow Parameters**
- **Best Practices for Using Environment Variables**
- **Exploring Organization, Repository and Environment Variables in GitHub Actions**
- **Testing Environment Variables in Workflow**

Define Workflow Parameters

Environment variables can be used to define parameters or configuration settings for your workflow. This makes it easy to customize workflow behavior:

```
jobs:
  build:
    runs-on: ubuntu-latest
    env:
      APP_ENV: ${{ vars.APP_ENV || "dev" }}
      DEBUG: ${{ vars.DEBUG || "false" }}
    steps:
      - name: Run Workflow with Parameters
        run: |
          # Use environment variables to control workflow behavior
          if [[ "$APP_ENV" == "production" && "$DEBUG" == "false" ]]; then
            echo "Running in production mode."
          else
            echo "Running in a different mode or with debugging enabled."
          fi
```

Best Practices for Using Environment Variables

When working with environment variables in GitHub Actions, consider the following best practices:

Avoid Hardcoding Secrets: Do not hardcode secrets directly in your workflow files. Store secrets securely as encrypted secrets in your repository settings and reference them using the secrets context.

Use Descriptive Names: Choose descriptive and meaningful names for your environment variables to make your workflows more readable and maintainable.

Keep Secrets Secret: Ensure that sensitive information, such as API keys or passwords, is stored and accessed securely. Avoid printing secrets in logs or exposing them unintentionally.

Scope Variables Appropriately: Use environment variables with appropriate scope (workflow-level, job-level) based on your workflow's requirements. Keep in mind that job-level environment variables are accessible only within the specific job.

Document Your Variables: Include comments or documentation in your workflow files to explain the purpose and usage of environment variables, making it easier for collaborators to understand and contribute to your workflows.

Review Access Permissions: Ensure that only authorized users have access to view or edit secrets and environment variables in your repository settings.

Exploring Organization, Repository and Environment Variables in GitHub Actions

To effectively utilize organization, repository, and environment variables in GitHub Actions, it's crucial to understand how to set them up and access them within workflows. This guide walks through the process, using a GitHub organization for Udemy courses as an example.

Organization Variables

1. Navigate to Organization Settings:

- Go to your organization on GitHub.
- Under Settings, find the `"Secrets"` and `"Variables"` section.

2. Create Organization Variables:

- Click on `"Variables"`.
- Create new variables, e.g., `org_var` and `org_overridden_var`.
- Specify values and choose which repositories can access them.
- Note: Private repository access might be restricted based on your plan.

Repository Variables

1. Access Repository Settings:

- In the repository settings, find `"Secrets"` and `"Variables"` under Actions.

2. Define Repository Variables:

- Create variables specific to the repository, e.g., `repository_var`.
- Set values and configure access as needed.

Environment Variables

1. Create Environments:

- Navigate to `"Environments"` under repository settings.
- Define environments such as `"Staging"` and `"Prod"`.
- Add environment variables within each environment.

Implementing Variables in Workflows

1. Accessing Variables in Workflows:

- Use the `vars` context to access organization and repository variables.
- Environment variables are accessed using `vars` or `env` context depending on the scope.

2. Using Default Values:

- Define default values for variables to handle undefined cases.
- Utilize the Or operator (`||`) to provide fallback values when variables are not defined.

Testing Environment Variables in Workflow

1. Create Workflow File:

- Create a new file named `10-01-Using-Variables.yml` in the `.github/workflows` directory of your repository.

```yaml
name: 10-01-Using Variables

on:
  workflow_dispatch:

env:
  WORKFLOW_VAR: 'I am a workflow env var'
  OVERWRITTEN: 'I will be overwritten'
  UNDEFINED_VAR_WITH_DEFAULT: ${{ vars.UNDEFINED_VAR || 'default value' }}

jobs:
  echo:
    runs-on: ubuntu-latest
    env:
      JOB_VAR: 'I am a job env var'
      OVERWRITTEN: 'I have been overwritten at the job level'
    steps:
      - name: Print Env Variables
        env:
          STEP_VAR: 'I am a step env var'
          step_var2: 'I am another step env var'
        run: |
          echo "Step env var: ${{ env.STEP_VAR }}"
          echo "Step env var 2: $step_var2"
          echo "Job env var: ${{ env.JOB_VAR }}"
          echo "Workflow env var: ${{ env.WORKFLOW_VAR }}"
          echo "Overwritten: ${{ env.OVERWRITTEN }}"
```

```
      - name: Overwrite job variable
        env:
          OVERWRITTEN: 'I have been overwritten at the step level'
        run: |
          echo "Step env var: ${{ env.OVERWRITTEN }}"
  echo2:
    runs-on: ubuntu-latest
    steps:
      - name: Print Variables
        run: |
          echo "Org var: ${{ vars.ORG_VAR }}"
          echo "Org overwritten var: ${{ vars.OVERWRITTEN_VAR }}"
          echo "Repo var: ${{ vars.REPOSITORY_VAR }}"
  echo-prod:
    runs-on: ubuntu-latest
    environment: prod
    steps:
      - name: Print Prod Variables
        run: |
          echo "Org var: ${{ vars.ORG_VAR }}"
          echo "Org overwritten var: ${{ vars.OVERWRITTEN_VAR }}"
          echo "Repo var: ${{ vars.REPOSITORY_VAR }}"
          echo "Environment var: ${{ vars.TARGET_VAR }}"
  echo-undefined:
    runs-on: ubuntu-latest
    steps:
      - name: Print Undefined Variables
        run: |
          echo "Org var: ${{ env.UNDEFINED_VAR_WITH_DEFAULT }}"
```

- Copy and paste the provided YAML configuration into this file.

2. Understanding the Workflow:

- This workflow is triggered manually via the GitHub UI (`workflow_dispatch`).
- It defines several environment variables at different levels (workflow, job, and step).
- The variables are then accessed and printed within various steps of different jobs to demonstrate their scope and usage.
- The `echo` job prints environment variables defined at the workflow, job, and step levels, along with demonstrating variable overriding at different levels.
- The `echo2` job demonstrates accessing organization and repository-level variables.
- The `echo-prod` job demonstrates accessing environment-specific variables (assuming a prod environment is defined in your repository settings).
- The `echo-undefined` job demonstrates handling undefined variables by providing a default value.

3. Testing the Workflow:

- Commit and push the workflow file (`10-01-Using-Variables.yml`) to your repository.
- Navigate to the "Actions" tab in your GitHub repository.
- Manually trigger the workflow by clicking on the "Run workflow" button for the `10-01-Using-Variables` workflow.

- Once the workflow run completes, click on each job to view its details.
- Review the logs of each step to ensure that the environment variables are accessed and printed correctly.

4. Observing Output:

Observe the output of each step to see the values of the environment variables at different levels. Pay attention to how variables are overridden at the job and step levels and how undefined variables are handled with default values.