Managing GitHub Actions Workflows

Mr FOMAZOU TCHINDA

November 12, 2024

Managing Your Workflow Environments

In this section, we will cover:

- Naming Your Workflow and Workflow Runs
- Contexts
- Environment Variables
- Default Environment Variables
- Secrets and Configuration Variables
- Managing Permissions for Your Workflow
- Deployment Environments

Naming Your Workflow and Workflow Runs: Workflows in GitHub Actions are named using the "name" keyword:

```
name: CI Workflow
```

Workflow runs can be named dynamically using the "run-name" field. This is especially useful in multi-step workflows.

Contexts

Contexts provide valuable information about the workflow environment. They are accessed in steps and can be used to retrieve important data such as the commit SHA, branch names, etc.
Example:

```
steps:
- name: Checkout code
uses: actions/checkout@v2

- name: Print branch name
run: echo "Branch: \${{ github.ref }}"
```

Environment Variables

GitHub Actions allows the use of environment variables that are either set by GitHub or by you. These variables are available throughout your workflow.
Default environment variables, such as "GITHUB_SHA", provide context on the current repository and commit.
Example:

```
steps:
- name: Print GitHub SHA
run: echo "GitHub SHA: \${{ github.sha }}"
```

Managing Secrets and Configuration Variables

Secrets are encrypted environment variables stored in GitHub, ideal for storing sensitive information like API tokens. Secrets are made available in workflows using the “$ secrets.NAME ” syntax.
Example:

```
steps:
- name: Set up API Key
run: echo "API_KEY=\${{ secrets.API_KEY }}"
```

Configuration variables are typically used for storing non-sensitive values like URLs or paths.

Managing Permissions for Your Workflow

You can control the permissions granted to workflows through GitHub's built-in actions settings. For example, you can restrict write permissions or prevent pushing to branches in sensitive repositories.
Example:

```
permissions:
contents: read
```

Deployment Environments

Deployment environments in GitHub Actions allow you to manage workflows that deploy code. This is particularly useful in CI/CD pipelines.
Example:

```
jobs:
deploy:
runs-on: ubuntu-latest
environment:
name: production
url: \${{ steps.deploy.outputs.page_url }}
```

Managing Data Within Workflows

In this section, we will explore:

- Working with Inputs and Outputs in Workflows
- Defining and Referencing Workflow Inputs
- Capturing Output from a Step
- Defining Artifacts
- Using Caches in GitHub Actions

Working with Inputs and Outputs in Workflows

Inputs are values that are passed to actions and jobs. Outputs are values returned from jobs and steps.

Example for defining and referencing inputs:

```
inputs:
path:
description: 'Path to the file'
required: true
```

Example for capturing outputs:

```
jobs:
build:
runs-on: ubuntu-latest
steps:
- name: Checkout code
uses: actions/checkout@v2

- name: Capture output
```

Defining Artifacts

Artifacts can be used to store files between jobs or for later access. Example:

```
- name: Upload artifact
uses: actions/upload-artifact@v2
with:
name: example-artifact
path: ./build-output/
```

Using Caches in GitHub Actions

Caching can be used to store dependencies between workflow runs, making subsequent runs faster.
Example of using the cache action:

```
- name: Cache dependencies
uses: actions/cache@v2
with:
path: ~/.npm
key: \${{ runner.os }}-node-\${{ hashFiles('**/package-l
```

Managing Workflow Execution

In this section, we will discuss advanced triggering options and how to manage workflow concurrency.
Topics include:

- Advanced Triggering
- Filters for Refining Triggers
- Triggering Without a Change
- Dealing with Concurrency

Advanced Triggering

Advanced triggering allows workflows to run based on specific events or activities. You can filter events based on branches, paths, and types of actions.
Example:

```
on:
push:
branches:
- main
```

Running a Workflow with a Matrix

A matrix allows you to run jobs across different configurations (e.g., multiple OS versions).
Example:

```
jobs:
test:
strategy:
matrix:
os: [ubuntu-latest, windows-latest, macos-latest]
runs-on: \${{ matrix.os }}
steps:
- uses: actions/checkout@v2
- run: npm test
```

Dealing with Concurrency

To avoid multiple workflows running simultaneously on the same branch, you can set concurrency keys.
Example:

```
concurrency:
group: \${{ github.ref }}
cancel-in-progress: true
```

Conclusion

In this lecture, we've covered managing environments, data, and execution within GitHub Actions workflows. We've also looked at examples for setting up variables, secrets, triggers, and caching. This forms the foundation for automating tasks and managing CI/CD pipelines effectively.