Second Qualification task

David Woo

Analysis:

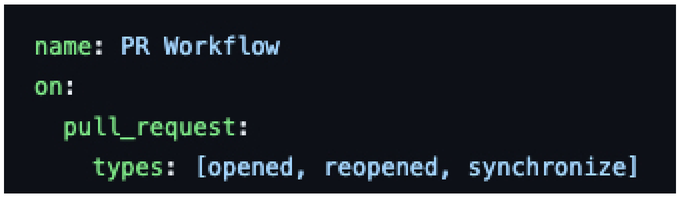
The current JSON-schema-org/website has many GitHub action workflow

- Request Review from Team
- Assign Team to PR workflow
- Build and check Links
- Check PR Dependencies
- Code Checker
- Greet on User First PR Merge
- Greeting the new contributor
- Issue Labeler
- Linting on PR
- New Implementation Commentor
- PR workflow(2)

Here, I mainly focus on PR workflow

Workflow Triggers:

- Activate pull requests, covering newly opened, reopened, and updated PRs.



```
name: PR Workflow
on:
  pull_request:
    types: [opened, reopened, synchronize]
```

Jobs:

1. Linting_and_type-checking:

    - Environment: Runs-on the latest Ubuntu

- Steps:
    - Checks out the repo
    - Caches Node dependencies using Yarn, reducing installation time for subsequent runs
    - Sets up Node.js(version 20), adhering to the project’s tech stack
    - Executes linting and type checking, ensuring code qualify and consistency.

2. Build:

- Dependencies: Requires the successful completion of the linting and type-checking job, ensuring only qualified code proceeds
- Environment: Similar to the first job, it runs on the latest Ubuntu.
- Steps:
    - Repeats the setup from the first job(checks out repo, cache dependencies)
    - Specifically caches the Next.js build, optimizing build times
    - Finally, executes the build process, verifying that the application can be built without errors.

Strategy for Scalability and Quality

1. Expand Test Coverage:

   Beyond linting and type checking, integrate additional automated tests like unit tests, integration tests, and end-to-end tests using tools compatible with the tech stacks(e.g MSW, jest for unit testing, integration testing, cypress for EC2 tests). This ensures high code quality and reduces bugs in production.

2. Environment Variables & Secrets:

   Utilize GitHub secrets to manage sensitive data and environment variables for different deployment environments(staging, production), enhancing security and making it easier to manage configuration changes.

3. Deployment Integration:

   Extend the CI/CD pipeline to include automated deployment steps once all checks pass.

For a web project, this might involve deploying to a staging environment first, followed by manual approval for production deployment.

4. Documentation & contribution guidelines

   Keep the CI/CD process well-documented and update contribution guidelines to help new contributors understand the workflow and expectations. This fosters a positive open-source community and smoothens the contribution process.

Conclusion

The current Github Actions workflow is a solid foundation for ensuring code quality. By expanding testing coverage, and integrating the deployment process, the CI/CD strategy can evolve to support scalability and maintain high quality as the project grows. Additionally, embracing best practices for security, documentation, and community engagement will boost the project's success and sustainability in the open-source community.

Reference:

Automate Your Development Workflow With Github Actions
GitHub action workflow