Caching Dependencies to Speed Up Workflows in GitHub Actions

Caching dependencies is an effective way to make your GitHub Actions workflows faster and more efficient. GitHub Actions provides a caching feature that allows you to create and use caches for dependencies and other commonly reused files. This README provides an overview of key concepts and best practices for caching in GitHub Actions.

About Caching Workflow Dependencies

Workflow runs often reuse the same outputs or downloaded dependencies from one run to another. For instance, package and dependency management tools like Maven, Gradle, npm, and Yarn keep a local cache of downloaded dependencies. To help speed up the time it takes to recreate files like dependencies, GitHub Actions can cache files you frequently use in workflows.

Caching Strategies

To cache dependencies for a job, you can use GitHub Actions' `cache` action. This action creates and restores a cache identified by a unique key. Alternatively, if you are caching package managers like npm, Yarn, pip, or Gradle, consider using their respective `setup-*` actions, which require minimal configuration and automatically create and restore dependency caches for you. The `setup-*` actions not always cache the folders containing the installed packages (for example, the `setup-node` action will not cache the `node_modules` folder after an `npm install` command).

Best Practices and Considerations

- **Sensitive Information**: Be cautious about what you store in the cache. Do not include sensitive information like access tokens or login credentials, as anyone with read access can access the cache contents.

- **Caching Across Branches**: Workflow runs can restore caches created in the current branch or the default branch (usually `main`). Pull requests can also access caches created in the base branch. However, caches cannot be restored across child branches or sibling branches.

- **Matching Cache Keys**: The `cache` action uses a cache key to restore a cache. It is essential to understand how cache keys work, including using expressions and restore keys. Cache keys should be specific enough to avoid unnecessary cache misses, but not too broad as to lead to incorrect hits.

- **Cache Hits and Misses**: When a cache key exactly matches an existing cache, it's called a cache hit. When there is no exact match, it's a cache miss, and GitHub Actions automatically creates a new cache if the job completes successfully.

Managing Caches

- **Cache Eviction**: GitHub Actions will remove caches not accessed for over 7 days. Be aware of usage limits and the eviction policy. If necessary, workflows can be set up to delete specific caches.

Usage Limits and Storage Quotas

GitHub Actions has limits on the number of caches you can store and the total size of all caches in a repository. The default limit is 10 GB per repository, but this limit may vary depending on your organization's policies or repository administrators' settings.