Artifacts in GitHub Actions

Artifacts in GitHub Actions allow you to persist data between workflow jobs and share data with other jobs in the same workflow. They are helpful for storing build outputs, test results, logs, and more.

About Workflow Artifacts

- Artifacts are files or collections of files produced during a workflow run.
- Common use cases include saving build and test outputs.
- Artifacts can be uploaded during a workflow run and downloaded later.

Uploading Artifacts

You can use the `upload-artifact` action to upload artifacts. These artifacts can then be retrieved later in the same workflow with the `download-artifact` action. Retrieving artifacts from other workflows is, as of the date of this document, not supported out of the box and requires custom logic (there are actions in the GitHub Actions Marketplace that serve this purpose). Here's an example of uploading a build artifact:

```yaml
- name: Upload Artifact
  uses: actions/upload-artifact@v4
  with:
    name: my-artifact
    path: target/
```

Three aspects can be highlighted:

- We have to specify a name for the artifact (e.g., `dist-without-markdown`).
- We also have to define the path or files we want to include.
- We can also specify files or directories to exclude.
- Glob patterns can be used to specify both include and exclude collections of files.

Managing Artifacts

You can set a custom retention period for an artifact using the retention-days option:

```yaml
- name: Upload Artifact
  uses: actions/upload-artifact@v4
  with:
    name: my-artifact
```

```
      path: target/
      retention-days: 5
```

Once artifacts are uploaded, they can be downloaded or deleted via the GitHub API or via the UI.

Passing Data Between Jobs

Outputs offer a way to share data between jobs, but they are not well-suited for sharing large amounts of data nor for sharing entire directories.

You can use the `upload-artifact` and `download-artifact` actions as an alternative to share data between jobs in a workflow. Ensure that dependent jobs wait for the previous job to complete.

Testing Artifacts in Workflow

1. Create Workflow File:

- Create a new file named `16-01-Artifacts.yml` in the `.github/workflows` directory of your repository.

```
name: 16-01-Artifacts

on:
  workflow_dispatch

jobs:
  upload-artifact:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Setup Java
        uses: actions/setup-java@v4
        with:
          distribution: 'adopt'
          java-version: '11'
          cache: 'maven'

      - name: Build with Maven
        run: mvn clean install

      - name: Upload jar to folder
        uses: actions/upload-artifact@v4
        with:
          name: my-artifact
          path: target/

  download-artifact:
    runs-on: ubuntu-latest
    needs: upload-artifact
```

```
    steps:

    - name: Checkout repository
      uses: actions/checkout@v2

    - name: Download web-app content
      uses: actions/download-artifact@v4
      with:
        name: my-artifact
        path: target/

    - name: View content
      run: ls -R
```

- 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 contains two jobs: `upload-artifact` and `download-artifact`.
- Each job runs on an `ubuntu-latest` runner.
- The `upload-artifact` job checks out the code, sets up Java, builds the project with Maven (`mvn clean install`), and uploads the resulting JAR artifact to the `target/` directory using the `actions/upload-artifact@v4` action.
- The `download-artifact` job depends on the `upload-artifact` job and downloads the previously uploaded JAR artifact using the `actions/download-artifact@v4` action.
- After downloading the artifact, the job prints its content by listing the files in the `target/` directory.

3. Testing the Workflow:

- Commit and push the workflow file (`16-01-Artifacts.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 `16-01-Artifacts` workflow.
- Monitor the workflow run and verify that both the `upload-artifact` and `download-artifact` jobs complete successfully.
- Check the logs of the `download-artifact` job to ensure that the JAR artifact is successfully downloaded and its content is displayed.

4. Observing Artifacts:

- After the workflow run completes, navigate to the "Artifacts" tab in the GitHub Actions interface.
- Verify that the `my-artifact` artifact is listed and contains the expected JAR file.
- Download the artifact and inspect its contents to ensure that it matches the build output from the `upload-artifact` job.