Using Actions

Use an action from the marketplace

->Just copy the given code from the marketplace

->When searching, be as specific as possible

Use an action from a repository

->Actions in the workflows’s repo

->Actions in any public repository

->Docker images from an image registry

Using actions in the same repo

->Specify a path relative to the repository root

e.g.

“./.github/action1”

Using an action in a different repo

->Specify the repo owner’s user ID, the repo, and a reference

->Reference can be a branch, tag, or SHA

e.g.

“user/repo/path@ref”

Use an action from an image repo

->Specify the “docker://” path to the image and tag

e.g.:

“docker://image:tag”

Passing arguments to an action

->Step use the with attribute to pass arguments

->Creates a new block for mapping arguments to inputs

e.g.

uses: {github account}/{action name}

with:

key:value

key:value

e.g.

- name: Checkout the code

uses: actions/checkout@v2

with:

repository: apache/tomcat

ref: master

path: ./tomcat

Using Environment Variables

->Dynamic key value pairs stored in memory

->injected at runtime

->case sensitive

A white background with black text

Description automatically generated

Defining Custom Environment Variable

->Use the env attribute

Define in:

->Workflows

->Jobs

->Steps

A close-up of a text

Description automatically generated

Accessing Environment Variables

->Shell variable syntax

-Bash (Linux/macos): $Variable\_name

-Powershell (windows): $Env:variable\_name

->Variable is read from the shell

->Yaml syntax

${{ env.Variable\_name }}

->Variable is read from the worklfow

Using secrets

->stored as encrypted environment variables

->Can’t be viewed or edited

->Limited to 100 secrets per workflow

->Secrets limited to 64 kb

Accessing Secrets

->Use the secrets worjflow context

${{ secrets.SECRET\_NAME }}

->Must be explicitly passed to a step or action.

->You can define secrets in settings

Using artifacts

Artifact

->Data preserved from a workflow

->Files or collection of files

->compiled binaries

->archives

->test results

->log files

->Pass data between workflow jobs

Job 1 – create and upload artifact

Job 2 – wait for job1 to complete

download and use artifact

->Can only be uploaded by a workflow – actions/upload-artifact

->Can only be downloaded by the uploading workflow - actions/download-artifact

->Manual downloads

->Free accounts get 500 MB for storage

->Stored for 90 days

Managing pull requests

->Pull facilitated conversations around code

->Merge code form one branch to another branch

Automating Pull-request merging

->Automatically approve and merge PRs based on criteria

->Run automated tests to check the code in the PR

->Check the username that submitted the PR

->Approve and merge the PR

->Delete the branch associated with the PR

e.g.

name: Auto Merge Owner PR

on:

pull\_request:

types: [opened, reopened]

jobs:

lint:

runs-on: ubuntu-latest

steps:

* uses actions/checkout@v1
* uses: actions/setup-python@1

with:

python-version: 3.8

* name : flake\* and pyint

run: |

pip install -r requirements.txt

flake8 –ignore=E501,E231 \*.py tests/\*.py

* name: unit test

run: python -m unittest -verbose -failtest

merge:

if: github.actor == ‘automate6500’

needs: [lint]

runs-on: ubuntu-latest

steps:

* uses: actions/checkout@v1
* uses: [hmarr/auto-approve-action@v2.0.0](mailto:hmarr/auto-approve-action@v2.0.0)

with:

github-toke: ${{ secrets.GITHUB\_TOKEN }}

* uses: [managedkaos/merge-piull-request@v1.2](mailto:managedkaos/merge-piull-request@v1.2)

env:

GITHUB\_TOKEN: @{{ secrets.HITHUB\_TOKEN }}