Creating a Custom Action

Kamran Ayub

Helping you prepare for the GitHub Actions exam

www.linkedin.com/in/kamranayub

`my-github-action/`

- A folder that can be the root of a repository

```
my-application/

  my-github-action/
```

- Or a sub-folder in a repository

```
my-application/
  .github/my-github-action/
  src/
  tests/
```

- Consider storing action code in .github folder
- If you also will be storing application code side-by-side with action code

```
my-github-action/

  action.yml
```

- **Must** contain an action.yml file

```
my-javascript-action/
  action.yml
  index.js
  package.json
  package-lock.json
```

- **Must** contain an action.yml file
- **Must** contain an entrypoint Node.js module
- **May** contain additional files to run the code

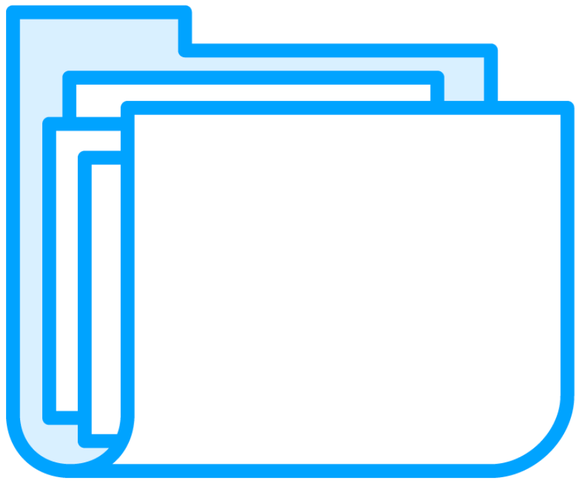
Example: TypeScript-based JavaScript Action

Could use Vite to build the JavaScript module using TypeScript

```
my-docker-action/
  action.yml

  Dockerfile

  entrypoint.sh
```

- **Must** contain an action.yml file
- **May** contain a local Dockerfile
- **May** contain an entrypoint shell script

GitHub Action Directory Structure



MUST contain an action.yml

MAY contain other supporting action files

- You can name them anything you want since they are referenced in the action.yml file

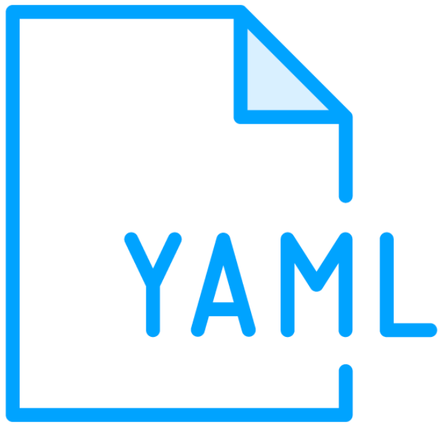
MAY include pre- and post-scripts depending on the action type

- For running logic before or after action

MAY contain files for publishing the action

- Documentation, tests, etc.

Understanding Action Metadata and Syntax



Written in YAML language

- A data serialization format similar to XML and JSON
- File extension ends in `.yml` or `.yaml`
- https://www.freecodecamp.org/news/what-is-yaml-the-yml-file-format/

Defines common metadata fields

- Some are required, others are optional

Defines action-specific metadata fields

Used by runners and GitHub marketplace

Common Metadata for Actions

All actions can use the commonly defined metadata fields

action.yml

```yaml
# Required
name: Demo Pluralsight Action
description: An example GitHub action made for the Pluralsight course

# Optional
author: Kamran Ayub
```

Common Metadata for Actions

Strings should be quoted or escaped if they contain quotes

action.yml

```yaml
# Required
name: Demo Pluralsight Action
description: "An example GitHub action made for Kamran's Pluralsight Course"

# Optional
author: Kamran Ayub
```

Defining Inputs for Actions

Inputs are optional and define parameters the action can accept

action.yml

```yaml
# Required
name: Demo Pluralsight Action
description: An example GitHub action made for the Pluralsight course

# Optional
inputs:
```

Defining Inputs for Actions

Accept parameters that get passed to the action

action.yml

Must start with a letter and may only contain alphanumeric characters, hyphen (-), or underscore (_).

```yaml
inputs:
  greeting:
    description: 'The word or phrase to begin the greeting with'
    required: false
    default: 'Hello '
  github-token:
    description: 'The personal access token to access GitHub resources'
    required: true
```

Defining Inputs for Actions

Description is always required for inputs

action.yml

```yaml
inputs:
  greeting:
    description: 'The word or phrase to begin the greeting with'
    required: false
    default: 'Hello '
  github-token:
    description: 'The personal access token to access GitHub resources'
    required: true
```

Defining Inputs for Actions

You can also deprecate inputs to maintain backwards compatibility

action.yml

```yaml
inputs:
  github-token:
    description: 'The personal access token to access GitHub resources'
    required: true
  github-pat:
    description: 'DEPRECATED: Personal access token for GitHub'
    deprecationMessage: 'Please use the github-token input'
```

Passing Inputs to Code

Inputs are passed as environment variables to action code or workflows

action.yml

```yaml
inputs:
  greeting:
    description: '...'
  github-token:
    description: '...'
```

index.js

```
# For JavaScript and Docker actions
process.env.INPUT_GREETING

process.env.INPUT_GITHUB-TOKEN
```

Gotchas with Inputs

Hyphens are not transformed to underscores for INPUT environment variables

- This makes them inaccessible to shell scripts
- Consider using environment variables or not using hyphens

Composite actions will not create `INPUT` environment variables

- Only JavaScript and Docker actions get `INPUT` environment variables created

The `default` field cannot be dynamic, it must be a plain string

Dynamic Inputs

`default` **field does not support referencing workflow variables**

action.yml

```yaml
inputs:
  github-token:
    description: 'The GitHub token to authenticate with GitHub API'
    required: true
    default: '${{ secrets.GITHUB_TOKEN }}'
```

Defining Outputs for Actions

Outputs are optional and define parameters the action can return

action.yml

```yaml
# Required
name: Demo Pluralsight Action
description: An example GitHub action made for the Pluralsight course

# Optional
outputs:
```

Defining Outputs for Actions

Define variables that are set within the action and exposed outside it

action.yml

```yaml
outputs:
  result:
    description: 'Result of the file processing'
```

Defining Outputs for Composite Actions

Composite action outputs can be assigned to a step's output value

action.yml

```yaml
outputs:
  slack-id:
    description: "Slack channel ID"
    value: ${{ steps.get-slack-channel.outputs.channel-id }}
runs:
  using: "composite"
  steps:
    - id: get-slack-channel
      run: echo "channel-id=$(echo $SLACK_CHANNEL_ID)" >> $GITHUB_OUTPUT
      shell: bash
```

Gotchas with Outputs

Unicode strings

Maximum of 1MB per output and 50MB total for all outputs

- Anything bigger could be stored as an Artifact

You do not need to define an output in the action YAML to set one through code

Referring to Inputs and Outputs

Using context expression syntax

action.yml

```
# Refer to inputs:
${{ inputs.github-token }}

# Refer to step outputs:
${{ steps.get-employee.outputs.slack-user-id }}

# Write outputs in a step using shared job context:
run: echo "slack-user-id=$(echo SLACK_USER_ID)" >> $GITHUB_OUTPUT
```

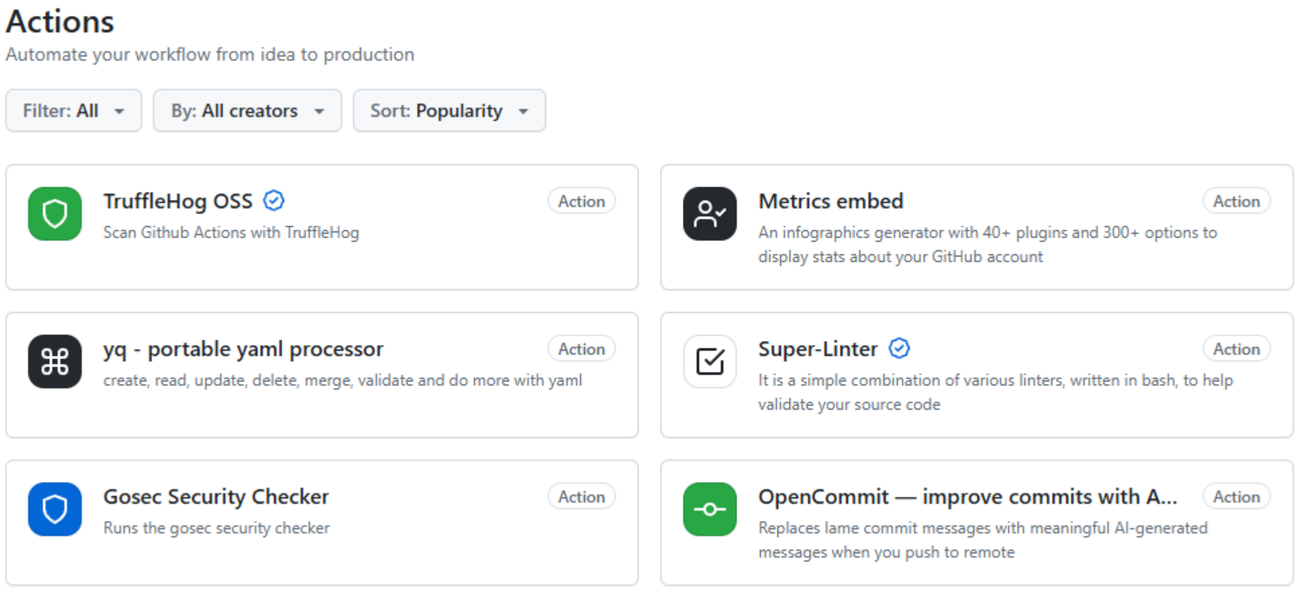
Branding Metadata for Actions

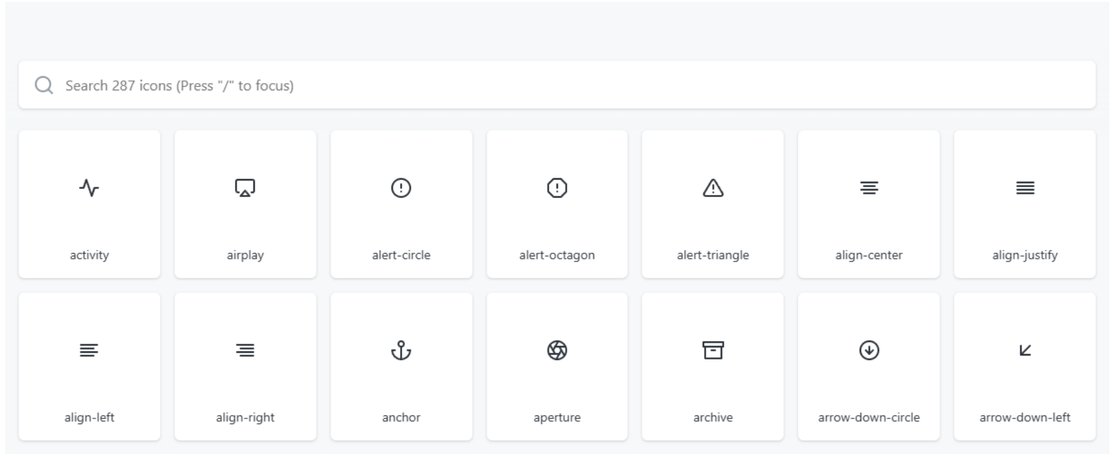
Used for publishing to GitHub Marketplace

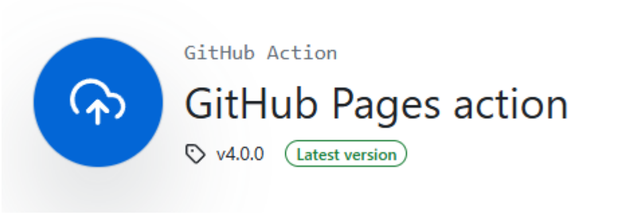
action.yml

```yaml
branding:
  icon: 'award'
  color: 'green'
```

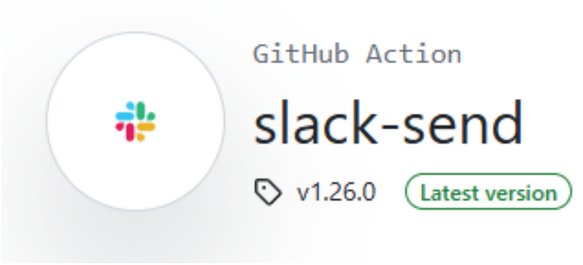
Demo: Feather Icons and GitHub Marketplace








```
branding:
  icon: 'upload-cloud'
  color: 'blue'
```



References

Metadata syntax for GitHub Actions
https://docs.github.com/en/actions/creating-actions/metadata-syntax-for-github-actions

Actions Contexts
https://docs.github.com/en/actions/learn-github-actions/contexts

Metadata Specific to Each Action

The runs field contains metadata specific to each action type

action.yml

```yaml
name: Demo Pluralsight Action
description: An example GitHub action made for the Pluralsight course

# Required for each action type
runs:
```

`runs` Syntax for JavaScript Actions

Specify the Node.js engine to use and the entrypoint module

action.yml

```yaml
runs:

  # Required
  using: 'node20'
  main: 'index.js'
```

`runs` Syntax for JavaScript Actions

You can conditionally run scripts before and after the action

action.yml

```yaml
runs:

  # Required
  using: 'node20'
  main: 'index.js'

  # Optional
  pre: 'setup.js'
  pre-if: runner.os == 'linux'
  post: 'cleanup.js'
  post-if: runner.os == 'linux'
```

`runs` Syntax for JavaScript Actions

You can conditionally run scripts before and after the action

action.yml

```yaml
runs:

  # Required
  using: 'node20'
  main: 'index.js'

  # Optional
  pre: 'setup.js'
  pre-if: always() # This is the default
  post: 'cleanup.js'
  post-if: success() # This is the job status, not the action status
```

`runs` Syntax for Docker Actions

Specify the Docker image which can be a local Dockerfile or public image

action.yml

```yaml
runs:

  # Required
  using: 'docker'
  image: '<relative path to Dockerfile or docker image tag>'
```

`runs` Syntax for Docker Actions

A local Dockerfile must start with a capital D but can be in any relative directory

action.yml

```yaml
runs:

  # Required
  using: 'docker'
  image: 'Dockerfile'
```

`runs` Syntax for Docker Actions

You can specify a public base image name or tag

action.yml

```yaml
runs:

  # Required
  using: 'docker'
  image: 'node:20'
```

`runs` Syntax for Docker Actions

You can also pass any accessible repository

action.yml

```yaml
runs:

  # Required
  using: 'docker'
  image: 'https://docker.globomantics.com/build-apps:latest'
```

`runs` Syntax for Docker Actions

You can override the `ENTRYPOINT` configuration of the Docker image

action.yml

```yaml
runs:

  # Required
  using: 'docker'
  image: '...'

  # Optional
  entrypoint: 'entrypoint.sh'
```

`runs` Syntax for Docker Actions

Conditionally run pre and post actions

action.yml

```yaml
runs:

  # Required
  using: 'docker'
  image: '...'

  # Optional
  entrypoint: 'entrypoint.sh'
  pre-entrypoint: 'setup.sh'
  pre-if: runner.type == 'macos'
  post-entrypoint: 'cleanup.sh'
  post-if: success()
```

`runs` Syntax for Docker Actions

You can pass environment variables to the container environment

action.yml

```yaml
runs:

  # Required
  using: docker
  image: Dockerfile

  # Optional
  env:
    NODE_ENV: 'production'
```

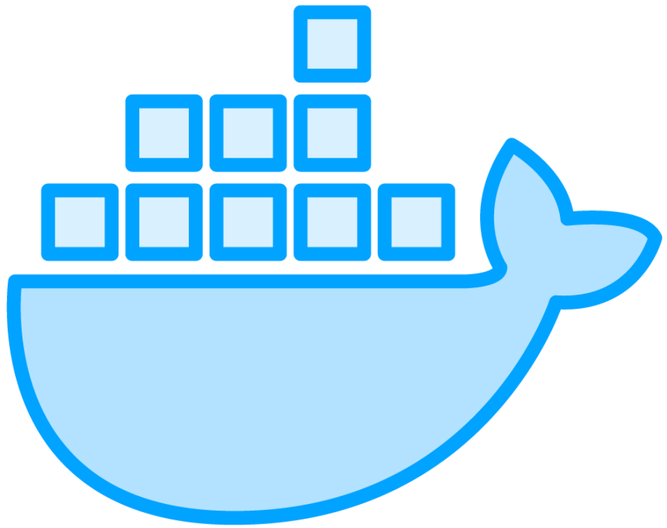
`runs` Syntax for Docker Actions

Pass arguments (or inputs) to the `ENTRYPOINT` command

action.yml

```yaml
runs:
  using: docker
  image: Dockerfile

  inputs:
    slack-base-url:
      description: '...'

  entrypoint: 'entrypoint.sh'
  args:
    - '--apiBase'
    - ${{ inputs.slack-base-url }}
    - '--dry-run'
```

Docker Action Gotchas



Make sure to mark entrypoint as executable

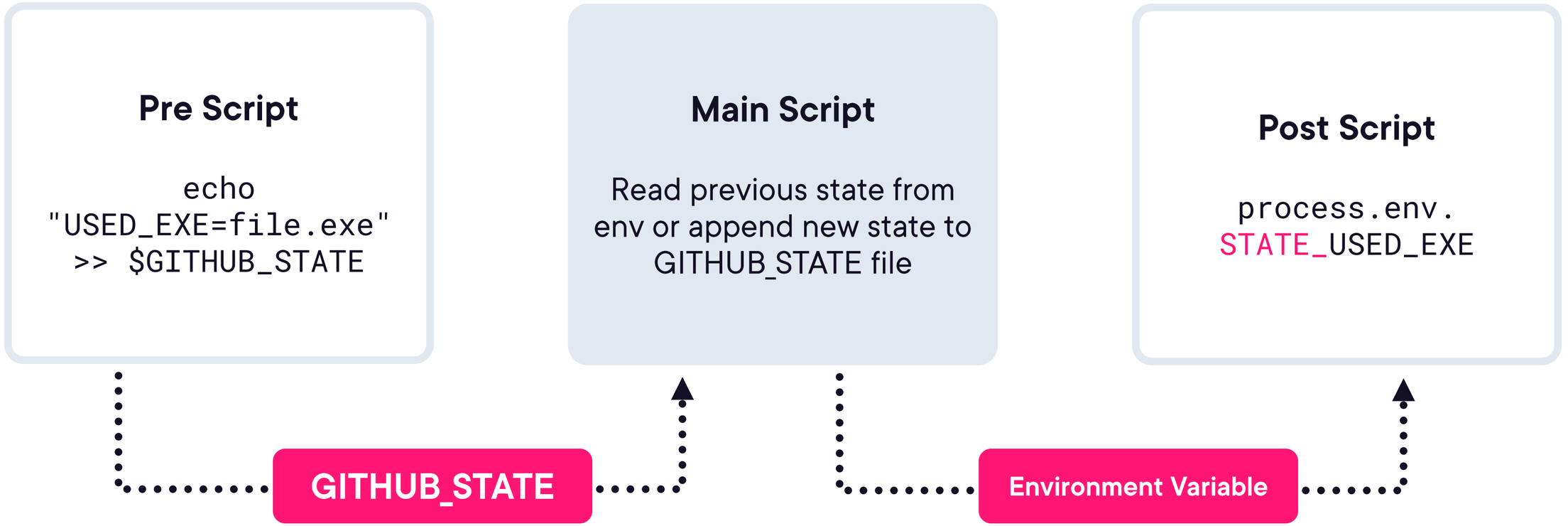
- `chmod +x entrypoint.sh`
- `git update-index --chmod=+x entrypoint.sh`

Some Dockerfile instructions are treated differently

- WORKDIR is set for you automatically
- `args` will override CMD instruction
- Do not use USER instruction

`pre` **and** `post` **are not supported when running local actions**

Passing Values to Pre and Post Actions



https://docs.github.com/en/actions/using-workflows/workflow-commands-for-github-actions#sending-values-to-the-pre-and-post-actions

`runs` Syntax for Composite Actions

Specify the steps to run

action.yml

```yaml
runs:

  # Required
  using: 'composite'
  steps:

  - id: step-1
    name: ...
    uses: ...
    with:
      arg1: ...

  - id: step-2
    name: ...
    uses: ...
```

Composing Actions Using Steps

Example of sending a build status notification

action.yml

```yaml
inputs:
  // ...

runs:
  using: 'composite'
  steps:
  - uses: './get-employee-js-action'
    id: get-employee
    with:
      github-username: ${{ inputs.github-username }}

  - uses: slackapi/slack-github-action@v1.26.0
    with:
      channel-id: '${{ steps.get-employee.outputs.slack-user-id }}'
    env:
      SLACK_BOT_TOKEN: ${{ inputs.slack-bot-token }}
```

`runs` Syntax for Composite Actions

Composite action inputs can be referenced in steps

action.yml

`runs` Syntax for Composite Actions

Local actions can be run relative to the action.yml directory

action.yml

```yaml
inputs:
  // ...

runs:
  using: 'composite'
  steps:
  - uses: './get-employee-js-action'
    id: get-employee
    with:
      github-username: ${{ inputs.github-username }}

  - uses: slackapi/slack-github-action@v1.26.0
    with:
      channel-id: '${{ steps.get-employee.outputs.slack-user-id }}'
    env:
      SLACK_BOT_TOKEN: ${{ inputs.slack-bot-token }}
```

`runs` Syntax for Composite Actions

Public actions can be run the same way as in workflow steps

action.yml

```yaml
inputs:
  // ...

runs:
  using: 'composite'
  steps:
  - uses: './get-employee-js-action'
    id: get-employee
    with:
      github-username: ${{ inputs.github-username }}

  - uses: slackapi/slack-github-action@v1.26.0
    with:
      channel-id: '${{ steps.get-employee.outputs.slack-user-id }}'
    env:
      SLACK_BOT_TOKEN: ${{ inputs.slack-bot-token }}
```

`runs` Syntax for Composite Actions

You can pass outputs between steps using outputs

action.yml

```yaml
inputs:
  // ...

runs:
  using: 'composite'
  steps:
  - uses: './get-employee-js-action'
    id: get-employee
    with:
      github-username: ${{ inputs.github-username }}

  - uses: slackapi/slack-github-action@v1.26.0
    with:
      channel-id: |
        '${{ steps.get-employee.outputs.slack-user-id }}'
    env:
      SLACK_BOT_TOKEN: ${{ inputs.slack-bot-token }}
```

get-employee-js-action/action.yml

```yaml
inputs:
  github-username:
    // ...

outputs:
  slack-user-id:
    description: '...'

runs:
  using: 'node20'
  main: main.js
```

References

Metadata Syntax for GitHub Actions
https://docs.github.com/en/actions/creating-actions/metadata-syntax-for-github-actions

Dockerfile Support for GitHub Actions
https://docs.github.com/en/actions/creating-actions/dockerfile-support-for-github-actions

Workflow Syntax for GitHub Actions
https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions

Coding a JavaScript Action

JavaScript Action

1ST

Look up email address of GitHub actor (GitHub API)

2ND

Look up Slack user ID by email address (Slack API)

3RD

Send Slack DM to user (channel) ID (Slack Action)

Demo: Creating the Action Directory and Metadata

Create directory and initial action.yml

action.yml

```yaml
name: 'Get Employee Slack Action (JavaScript)'
description: "Looks up an employee's Slack ID by their GitHub username"

runs:
  using: node20
  main: main.js
```

```
npm init -y
npm install @actions/core @actions/github @slack/web-api
```

Demo: Creating the Action Code

Initializing a JavaScript project with dependencies

Demo: GitHub Actions Toolkit (JavaScript)

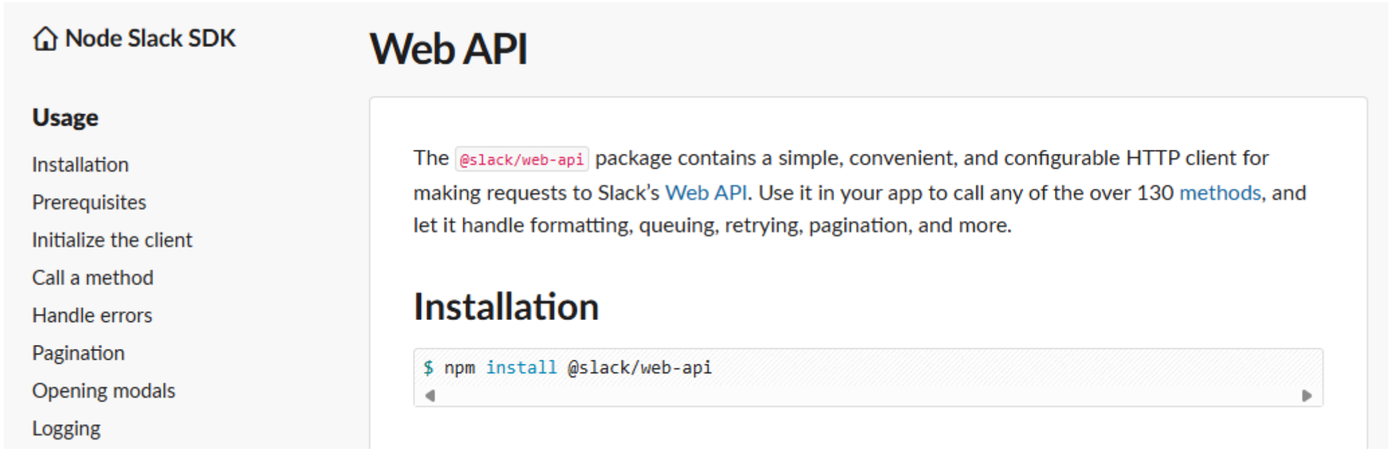
toolkit-unit-tests passing toolkit-audit passing

GitHub Actions Toolkit

The GitHub Actions ToolKit provides a set of packages to make creating actions easier.

Get started with the javascript-action template!

Demo: Slack Web API (JavaScript)



Coding the Action

JavaScript actions have a main entrypoint script

main.js

```
// @ts-check
const core = require("@actions/core");
const github = require("@actions/github");
const { WebClient } = require("@slack/web-api");

run();

async function run() {
  try {
    const githubUsername = core.getInput("github-username");
    const githubToken = core.getInput("github-token");
    const slackToken = core.getInput("slack-token");

    const slackUserId = await lookupSlackUserByGitHubUsername(githubUsername, githubToken, slackToken);
    core.setOutput("slack-user-id", slackUserId);
  } catch (error) {
    core.setFailed(error.message);
  }
}

async function lookupSlackUserByGitHubUsername(githubUsername, githubToken, slackToken) {
  const octokit = github.getOctokit(githubToken);

  const {
    data: { email },
  } = await octokit.rest.users.getByUsername({
    username: githubUsername,
  });

  if (!email) {
    throw new Error(
      `No public email found for GitHub user: ${githubUsername}`
    );
  }

  const web = new WebClient(slackToken);
  const { user: slackUser } = await web.users.lookupByEmail({ email: email });

  if (!slackUser) {
    throw new Error(`No Slack user found with email: ${email}`);
  }

  return slackUser.id
}
```

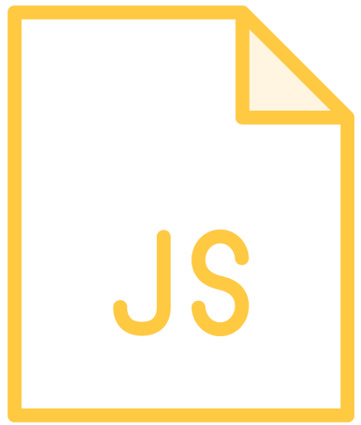
Demo: Creating the Action Directory and Metadata

Create directory and initial action.yml

action.yml

```yaml
inputs:
  github-token:
    description: 'The GitHub token to authenticate with GitHub API'
    required: true
  github-username:
    description: 'The GitHub username of the employee to look up'
    required: true
  slack-token:
    description: 'The organizational Slack access token'
    required: true
outputs:
  slack-user-id:
    description: 'The Slack user ID of the employee, if found'
```

Running a JavaScript Action



Running the action requires a GitHub Actions runner

Easiest method to test is to run the action in a workflow in the repository

- This is also called "integration testing"

You can also use any JavaScript unit testing framework for testing the logic

- Jest, Mocha, vitest, etc.
- You would mock calls to `@actions/` packages

References

JavaScript Action Template
https://github.com/actions/javascript-action

Creating a JavaScript Action
https://docs.github.com/en/actions/creating-actions/creating-a-javascript-action

Implementing Workflow Commands

The Actions Toolkit wraps workflow commands using a Node.js API

main.js

```
const core = require("@actions/core");

const githubUsername = core.getInput("github-username");

core.setOutput("slack-user-id", slackUserId);

core.setFailed(error.message);
```

Workflow commands

Actions can communicate with the runner machine to set environment variables, output values used by other actions, add debug messages to the output logs, and other tasks.

https://docs.github.com/en/actions/using-workflows/workflow-commands-for-github-actions

Implementing Workflow Commands

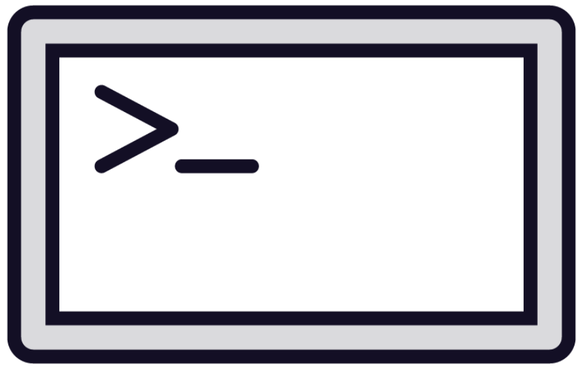
The Actions Toolkit wraps workflow commands using a Node.js API

main.js

```
const core = require("@actions/core");

const githubUsername = core.getInput("github-username");

core.setOutput("slack-user-id", slackUserId);

core.setFailed(error.message);
```

How do you implement workflow commands without the toolkit?

Implementing Workflow Commands



Using standard out (stdout)

- Logging or printing to console
- Commands follow a structured syntax

Appending to environment files

- The paths are stored in environment variables that match the file name (e.g. `GITHUB_OUTPUT`)

Reading environment variables

- GitHub Actions creates standard environment variables using different conventions (e.g. `INPUT_<NAME>`)

```
# Read from shell (doesn't support input names with dashes)
echo "$INPUT_GITHUB_TOKEN"

# Using Node.js
process.env['INPUT_GITHUB-USERNAME'];

# Using C#
System.GetEnvironmentVariable("INPUT_GITHUB-USERNAME");
```

Reading Action Inputs

Inputs are passed as environment variables prefixed with `INPUT_` that you can read from the shell or your programming language of choice. Input names with dashes cannot be read from POSIX shell scripts.

```
# Workflow message syntax
::workflow-command parameter1={data},parameter2={data}::{command value}

# Log an error
::error::Just a message

# Log an error with parameters
::error file=index.js,line=1::Just a message

# Register a secret
::add-mask::My Super Secret
```

Sending Commands via Standard Out

Different commands can be sent through logging to standard out using a structured command syntax. Parameters are usually optional.

Grouping Multiline Commands

```
::group::{title}

# Any output here will be collapsed into a group
# within the GitHub Actions logging UI.

::endgroup::
```

Grouping Multiline Commands

```
::group::ERROR: 'token' is required

::error file=index.js,line=1::'token' is required

# Only include stack trace when debugging is turned on
::debug::Stack Trace:
::debug::  getUser at /github/workspace/index.js:1

::endgroup::
```

```
# Set an output (Bash)
echo "slack-user-id=abc123" >> $GITHUB_OUTPUT

# Export a new environment variable
echo "ORG_CACHE=/usr/.org_cache" >> $GITHUB_ENV

# Append to a file (Node.js)
fs.appendFile(process.env.GITHUB_OUTPUT, `slack-user-id=${slackUserId}`);

# Append to a file (C#)
File.AppendAllLinesAsync(System.GetEnvironmentVariable("GITHUB_OUTPUT", [
  $"slack-user-id={slackUserId}"
]);
```

Sending Commands Through Files

You can append lines to specific environment files. The paths are stored in environment variables, like $GITHUB_OUTPUT.

Setting Multiline String Values

```
{name}<<{delimiter}
{value}
{delimiter}
```

Setting Multiline String Values

Example: Setting an output to a formatted JSON value

index.js

```
const fs = require('fs/promises');
const crypto = require('crypto');

const payload = {
  user: 'A12BC3',
  message: 'Greetings and saluations!'
};

async function setPayloadOutput() {
  const payloadJson = JSON.stringify(payload, null, 2);
  const payloadHash = crypto.createHash('256').update(payloadJson).digest('hex');
  const payloadOutput = `slack-payload<<${payloadHash}
${payloadJson}
${payloadHash}\n`;

  await fs.appendFile(process.env.GITHUB_OUTPUT, payloadOutput);
}
```

Setting Exit Codes

The status of a GitHub Action can be controlled using exit codes

index.js

```
core.setFailed("message");
```

is a shortcut for

```
// Uses error command
console.log("error::message");

// Exits with non-zero code
process.exit(1);
```

Setting Exit Codes

Using the error command will not automatically set the action status to `failed`

script.sh

```
# Even though you printed an error
echo "error::an error message"

# The script will exit with code 0 (success)
```

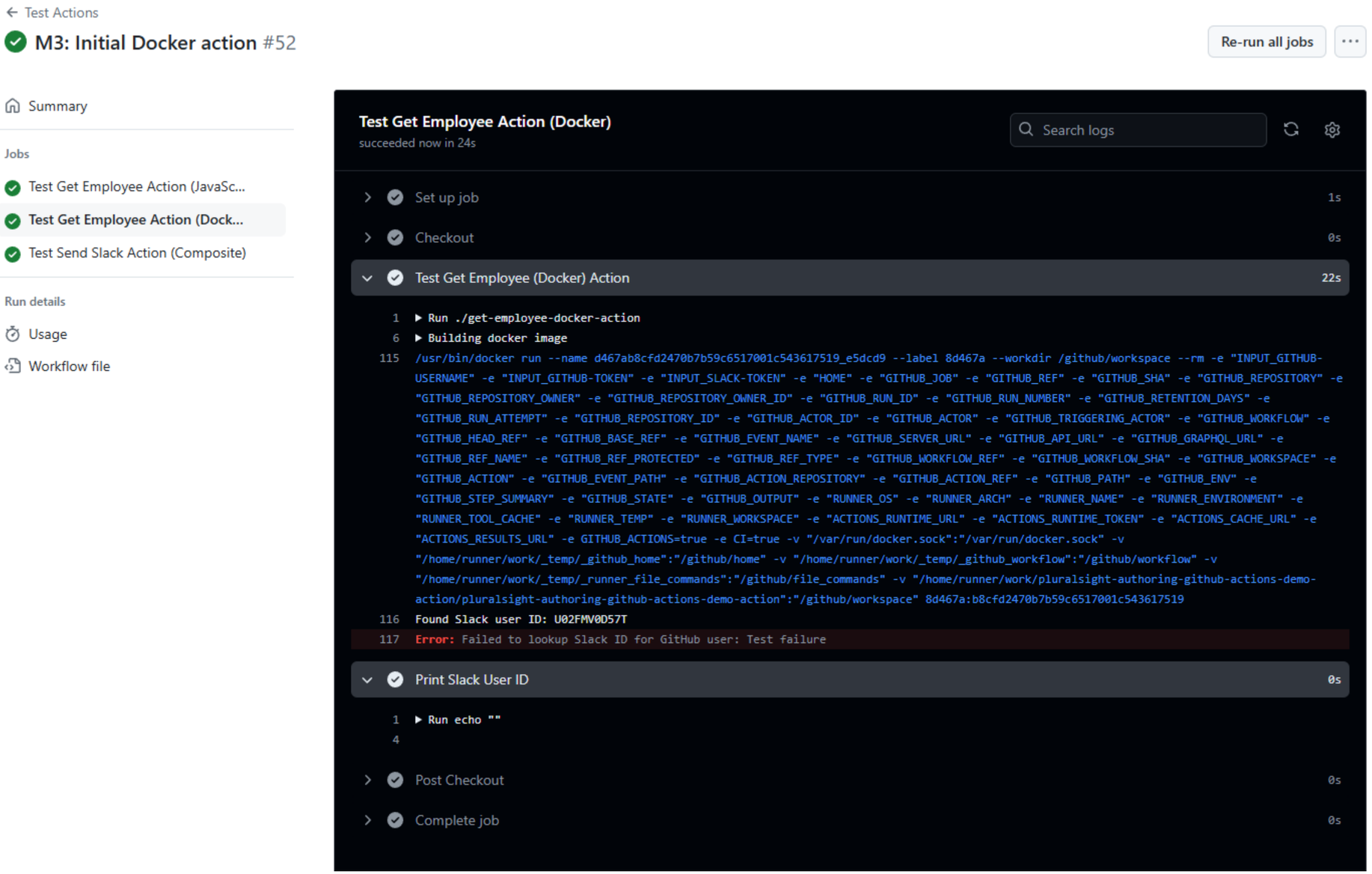
Setting Exit Codes

Using the error command will not automatically set the action status to `failed`

script.sh

```
# Even though you printed an error
echo "error::an error message"

# You need to explicitly exit with a non-zero code
exit 1
```

Demo: GitHub Actions Succeeded with Error



Stopping Actions From Reading Workflow Commands

Useful when printing debug output or user input

```
::stop-commands::{endtoken}

# You can output any workflow commands here, like when debugging,
# and they will not be parsed or executed.

::{endtoken}::
```

Stopping Actions From Reading Workflow Commands

Example: Ignoring commands in the output of a child process

index.js

```
const { spawn } = require('child_process');

// Example stop token based on current timestamp unlikely to occur in output
const commandStopToken = 'STOP_TOKEN_' + Date.now().toString();

// Send stop command before process is started with token
console.log(`::stop-commands::${commandStopToken}`);

// Spawn a child process that inherits main process stdout and stderr streams
const child = spawn('node', ['other-script.js'], {
  stdio: 'inherit'
});

// Send stop token when process is done
child.on('close', (code) => {
  console.log(`::${commandStopToken}::`);
});
```

References

Full List of Workflow Commands
https://docs.github.com/en/actions/using-workflows/workflow-commands-for-github-actions

Setting Exit Codes for Actions
https://docs.github.com/en/actions/creating-actions/setting-exit-codes-for-actions

Coding a Docker Action

.NET Code

1ST

Look up email address of GitHub actor (GitHub API)

2ND

Look up Slack user ID by email address (Slack API)

Docker Action

3RD

Send Slack DM to user (channel) ID (Slack Action)

Demo: Creating the Action Directory and Metadata

Create directory and initial action.yml

action.yml

```yaml
name: 'Get Employee Slack Action (Docker)'
description: "Looks up an employee's Slack ID by their GitHub username"

runs:
  using: docker
  image: Dockerfile
```

Demo: Creating a Local Dockerfile

You can reference a local Dockerfile to build an image

Dockerfile

```
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build-env

COPY ./src /App
RUN cd /App && dotnet restore
RUN cd /App && dotnet publish -c Release -o out

FROM mcr.microsoft.com/dotnet/aspnet:8.0
COPY --from=build-env /App/out .
ENTRYPOINT ["dotnet", "/DotNetDockerAction.dll"]
```

```
dotnet new console -o src -n DotNetDockerAction
dotnet add package Octokit
dotnet add package Slack.NetStandard
```

Demo: Creating the Action Code

Initializing a new .NET console app with dependencies

Demo: GitHub Actions Toolkit (JavaScript)

toolkit-unit-tests passing toolkit-audit passing

GitHub Actions Toolkit

The GitHub Actions ToolKit provides a set of packages to make creating actions easier.

Get started with the javascript-action template!

Coding the Action

Docker actions allow you to use any language of your choice

Program.cs

```
using Octokit;
using Slack.NetStandard;

var githubUsername = Environment.GetEnvironmentVariable("INPUT_GITHUB-USERNAME");
var githubToken = Environment.GetEnvironmentVariable("INPUT_GITHUB-TOKEN");
var slackToken = Environment.GetEnvironmentVariable("INPUT_SLACK-TOKEN");

try
{
    var slackUserId = await LookupSlackUserByGitHubUsername(githubUsername, githubToken, slackToken);

    Console.WriteLine("Found Slack user ID: " + slackUserId);

    await File.AppendAllLinesAsync(Environment.GetEnvironmentVariable("GITHUB_OUTPUT"), [
        $"slack-user-id={slackUserId}"
    ]);
}
catch (Exception ex)
{
    Console.WriteLine($"::error::Failed to lookup Slack ID for GitHub user: {ex.Message}");
    Environment.Exit(1);
}

async Task<string> LookupSlackUserByGitHubUsername(string githubUsername, string githubToken, string slackToken)
{
    var octokit = new GitHubClient(new ProductHeaderValue("PluralsightDemoAction"));
    var tokenAuth = new Credentials(githubToken);
    octokit.Credentials = tokenAuth;

    var githubUser = await octokit.User.Get(githubUsername);
    var email = githubUser?.Email;

    if (email == null)
    {
        throw new ApplicationException($"GitHub user '{githubUsername}' does not have a public email address");
    }

    var client = new SlackWebApiClient(slackToken);
    var slackResponse = await client.Users.LookupByEmail(email);

    if (slackResponse?.User == null)
    {
        throw new ApplicationException($"Slack user with email '{email}' not found");
    }

    return slackResponse.User.ID;
}
```

Coding the Action

Using workflow commands

Program.cs

```
using Octokit;
using Slack.NetStandard;

var githubUsername = Environment.GetEnvironmentVariable("INPUT_GITHUB-USERNAME");
var githubToken = Environment.GetEnvironmentVariable("INPUT_GITHUB-TOKEN");
var slackToken = Environment.GetEnvironmentVariable("INPUT_SLACK-TOKEN");

try
{
    var slackUserId = await LookupSlackUserByGitHubUsername(githubUsername, githubToken, slackToken);

    Console.WriteLine("Found Slack user ID: " + slackUserId);

    await File.AppendAllLinesAsync(Environment.GetEnvironmentVariable("GITHUB_OUTPUT"), [
        $"slack-user-id={slackUserId}"
    ]);
}
catch (Exception ex)
{
    Console.WriteLine($"::error::Failed to lookup Slack ID for GitHub user: {ex.Message}");
    Environment.Exit(1);
}

async Task<string> LookupSlackUserByGitHubUsername(string githubUsername, string githubToken, string slackToken)
{
    var octokit = new GitHubClient(new ProductHeaderValue("PluralsightDemoAction"));
    var tokenAuth = new Credentials(githubToken);
    octokit.Credentials = tokenAuth;

    var githubUser = await octokit.User.Get(githubUsername);
    var email = githubUser?.Email;

    if (email == null)
    {
        throw new ApplicationException($"GitHub user '{githubUsername}' does not have a public email address");
    }

    var client = new SlackWebApiClient(slackToken);
    var slackResponse = await client.Users.LookupByEmail(email);

    if (slackResponse?.User == null)
    {
        throw new ApplicationException($"Slack user with email '{email}' not found");
    }

    return slackResponse.User.ID;
}
```

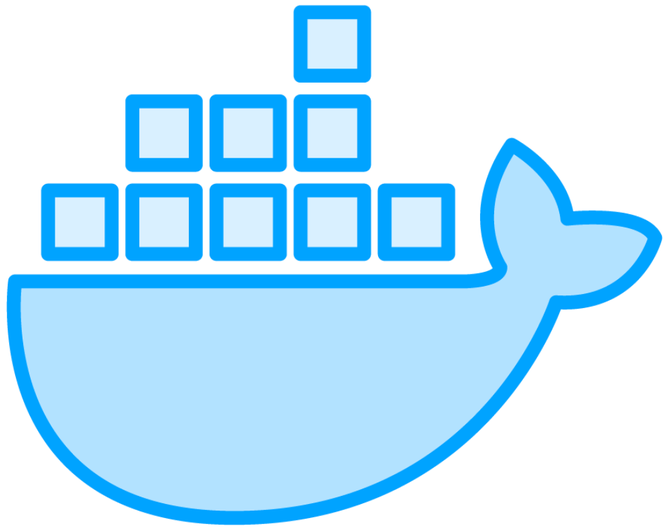
Demo: Creating the Action Directory and Metadata

Create directory and initial action.yml

action.yml

```yaml
name: 'Get Employee Slack Action (Docker)'
description: "Looks up an employee's Slack ID by their GitHub username"

runs:
  using: docker
  image: Dockerfile
```

Running a Docker Action



You could build the Docker image locally and run it

- You would need to pass all the environment configuration needed to simulate the runner

Easiest method to test is to run the action in a workflow in the repository

- This is also called "integration testing"

You can also use a unit testing framework for testing the logic

Running a Docker Action Locally

You can build the image locally and run it

```
#!/bin/sh

# Build the image
docker build -t ps_gh_docker_action:latest .

# Run the built image and pass a subset of the expected environment
# Use `env` to pass hyphenated environment variables to Docker
# Both "GITHUB_TOKEN" and "SLACK_TOKEN" secrets need to be defined in the environment
env "INPUT_GITHUB-USERNAME=kamranayub" \
  "INPUT_GITHUB-TOKEN=$GITHUB_TOKEN" \
  "INPUT_SLACK-TOKEN=$SLACK_TOKEN" \
  "GITHUB_OUTPUT=/github/file_commands/GITHUB_OUTPUT" \
  docker run \
    --workdir /github/workspace \
    -e "INPUT_GITHUB-USERNAME" \
    -e "INPUT_GITHUB-TOKEN" \
    -e "INPUT_SLACK-TOKEN" \
    -e "GITHUB_OUTPUT" \
    -v "$(pwd)/test/_runner_file_commands":"/github/file_commands" \
    -v "$(pwd)":"/github/workspace" \
    ps_gh_docker_action:latest
```

References

Container Action Template
https://github.com/actions/container-action

Creating a Docker Container Action
https://docs.github.com/en/actions/creating-actions/creating-a-docker-container-action

Coding a Composite Action

Inner Action

1ST

Look up email address of GitHub actor (GitHub API)

2ND

Look up Slack user ID by email address (Slack API)

3RD

Send Slack DM to user (channel) ID (Slack Action)

Composite Action

Demo: Creating the Action Directory and Metadata

Create directory and initial action.yml

action.yml

```yaml
name: 'Send Status Notification to Slack (Composite)'
description: "Sends the status of the GitHub Action to the user via Slack"

runs:
  using: composite
  steps:
```

Demo: Defining Steps

Referencing and using a local action

action.yml

```yaml
name: 'Send Status Notification to Slack (Composite)'
description: "Sends the status of the GitHub Action to the user via Slack"

runs:
  using: composite
  steps:

  - uses: './get-employee-js-action'
    with:
      github-username:
      github-token:
      slack-token:
```

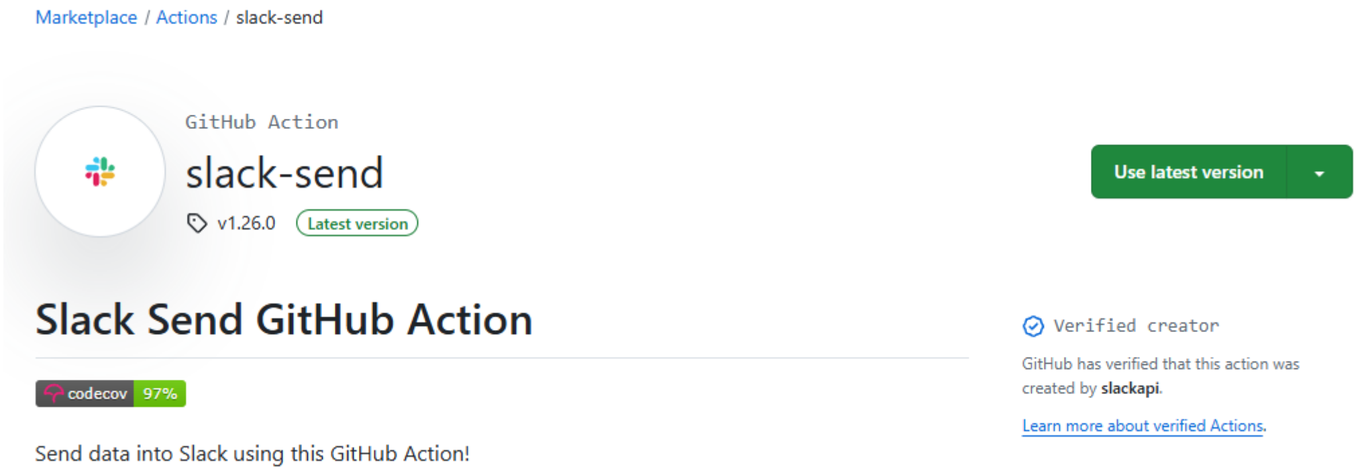
Demo: Defining Inputs

Composite actions can accept inputs to pass to steps

action.yml

```yaml
name: 'Send Status Notification to Slack (Composite)'
description: "Sends the status of the GitHub Action to the user via Slack"

inputs:
  github-token:
    description: 'The GitHub token to authenticate with GitHub API'
    required: true
  github-username:
    description: 'The GitHub username of the employee to look up'
    required: true
  slack-token:
    description: 'The organizational Slack access token to look up users'
    required: true

runs:
  using: composite
  steps:

  - uses: './get-employee-js-action'
    with:
      github-username: ${{ inputs.github-username }}
      github-token: ${{ inputs.github-token }}
      slack-token: ${{ inputs.slack-token }}
```

Demo: Send Slack Action



Demo: Using Published Actions

Composite actions can reference public or private actions

action.yml

```yaml
name: 'Send Status Notification to Slack (Composite)'
description: "Sends the status of the GitHub Action to the user via Slack"

inputs:
  github-token:
    description: 'The GitHub token to authenticate with GitHub API'
    required: true
  github-username:
    description: 'The GitHub username of the employee to look up'
    required: true
  slack-token:
    description: 'The organizational Slack access token to look up users'
    required: true

runs:
  using: composite
  steps:

  - uses: './get-employee-js-action'
    with:
      github-username: ${{ inputs.github-username }}
      github-token: ${{ inputs.github-token }}
      slack-token: ${{ inputs.slack-token }}

  - uses: slackapi/slack-github-action@v1.26.0
    with:
      channel-id: ''
      slack-message: "GitHub build result: ${{ job.status }}\n${{ github.event.pull_request.html_url || github.event.head_commit.url }}"
    env:
      SLACK_BOT_TOKEN: ${{ inputs.slack-bot-token }}
```

A common convention is that variables should be inputs and secrets should be environment variables.

Demo: Using Published Actions

Composite actions can reference public or private actions

action.yml

```yaml
name: 'Send Status Notification to Slack (Composite)'
description: "Sends the status of the GitHub Action to the user via Slack"

inputs:
  github-token:
    description: 'The GitHub token to authenticate with GitHub API'
    required: true
  github-username:
    description: 'The GitHub username of the employee to look up'
    required: true
  slack-token:
    description: 'The organizational Slack access token to look up users'
    required: true

runs:
  using: composite
  steps:

  - uses: './get-employee-js-action'
    id: get-employee-js
    with:
      github-username: ${{ inputs.github-username }}
      github-token: ${{ inputs.github-token }}
      slack-token: ${{ inputs.slack-token }}

  - uses: slackapi/slack-github-action@v1.26.0
    with:
      channel-id: ${{ steps.get-employee-js.outputs.slack-user-id }}
      slack-message: "GitHub build result: ${{ job.status }}\n${{ github.event.pull_request.html_url || github.event.head_commit.url }}"
    env:
      SLACK_BOT_TOKEN: ${{ inputs.slack-bot-token }}
```

Demo: Using Published Actions

Composite actions can reference public or private actions

action.yml

```yaml
name: 'Pluralsight Demo: Send Status Notification to Slack (Composite)'
description: 'Sends the status of the GitHub Action to the user via Slack'

inputs:
  github-token:
    description: 'The GitHub token to authenticate with GitHub API'
    required: true
  github-username:
    description: 'The GitHub username of the employee to look up'
    required: true
  slack-token:
    description: 'The organizational Slack access token to look up users'
    required: true
  slack-bot-token:
    description: 'The organizational Slack access token to post messages (requires chat:write permissions)'
    required: true

runs:
  using: 'composite'
  steps:

  - uses: './get-employee-js-action'
    id: get-employee-js
    with:
      github-username: ${{ inputs.github-username }}
      github-token: ${{ inputs.github-token }}
      slack-token: ${{ inputs.slack-token }}

  - uses: './get-employee-docker-action'
    id: get-employee-docker
    with:
      github-username: ${{ inputs.github-username }}
      github-token: ${{ inputs.github-token }}
      slack-token: ${{ inputs.slack-token }}

  - run: |
      echo "::error::Slack user IDs do not match from JS and Docker"
      exit 1
    if: steps.get-employee-js.outputs.slack-user-id != steps.get-employee-docker.outputs.slack-user-id
    shell: sh

  - uses: slackapi/slack-github-action@v1.26.0
    if: steps.get-employee-js.outputs.slack-user-id == steps.get-employee-docker.outputs.slack-user-id
    with:
      channel-id: '${{ steps.get-employee-js.outputs.slack-user-id }}'
      slack-message: "GitHub build result: ${{ job.status }}\n${{ github.event.pull_request.html_url || github.event.head_commit.url }}"
    env:
      SLACK_BOT_TOKEN: ${{ inputs.slack-bot-token }}
```

Demo: Conditional Step Execution

Composite actions can use workflow conditions to control the flow of execution

action.yml

```yaml
name: 'Pluralsight Demo: Send Status Notification to Slack (Composite)'
description: 'Sends the status of the GitHub Action to the user via Slack'

inputs:
  github-token:
    description: 'The GitHub token to authenticate with GitHub API'
    required: true
  github-username:
    description: 'The GitHub username of the employee to look up'
    required: true
  slack-token:
    description: 'The organizational Slack access token to look up users'
    required: true
  slack-bot-token:
    description: 'The organizational Slack access token to post messages (requires chat:write permissions)'
    required: true

runs:
  using: 'composite'
  steps:

  - uses: './get-employee-js-action'
    id: get-employee-js
    with:
      github-username: ${{ inputs.github-username }}
      github-token: ${{ inputs.github-token }}
      slack-token: ${{ inputs.slack-token }}

  - uses: './get-employee-docker-action'
    id: get-employee-docker
    with:
      github-username: ${{ inputs.github-username }}
      github-token: ${{ inputs.github-token }}
      slack-token: ${{ inputs.slack-token }}

  - run: |
      echo "::error::Slack user IDs do not match from JS and Docker"
      exit 1
    if: steps.get-employee-js.outputs.slack-user-id != steps.get-employee-docker.outputs.slack-user-id
    shell: sh

  - uses: slackapi/slack-github-action@v1.26.0
    if: steps.get-employee-js.outputs.slack-user-id == steps.get-employee-docker.outputs.slack-user-id
    with:
      channel-id: '${{ steps.get-employee-js.outputs.slack-user-id }}'
      slack-message: "GitHub build result: ${{ job.status }}\n${{ github.event.pull_request.html_url || github.event.head_commit.url }}"
    env:
      SLACK_BOT_TOKEN: ${{ inputs.slack-bot-token }}
```

References

Workflow Syntax for GitHub Actions
https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions

Creating a Composite Action
https://docs.github.com/en/actions/creating-actions/creating-a-composite-action

Runs Syntax for Composite Actions
https://docs.github.com/en/actions/creating-actions/metadata-syntax-for-github-actions#runs-for-composite-actions