# Custom Action Challenge

## A Drop-In Starter that Actually Runs

Your Name

November 2, 2025

**How to compile this PDF:** This document uses the `minted` package. Build with:

```
latexmk -pdf -shell-escape custom-action-challenge-starter.tex
```

## Overview

This starter turns raw notes into a working Docker-based GitHub Action that prints the GitHub actor, commit SHA, event type, and the repository visibility. It also includes a self-test workflow so you can run it immediately after pushing to a repository.

## What You Get

- A minimal yet production-ready Docker action: `action.yml` + `Dockerfile`.
- A safe shell entrypoint with strict mode and JSON parsing via `jq`.
- A CI workflow to prove it works on `push` and `workflow_dispatch`.
- Clear instructions for local (same-repo) use and publishing to the Marketplace.

## Repository Layout

Place these files in a new repository (public preferred for Marketplace later):

- `entrypoint.sh`
- `Dockerfile`
- `action.yml`
- `.github/workflows/test-custom-action.yml`

# 1 Files & Code

## 1.1 entrypoint.sh

**Purpose:** Prints actor, commit SHA, event type, and repo visibility using the event payload.

```bash
#!/usr/bin/env bash
set -euo pipefail

echo "Actor      : ${GITHUB_ACTOR:-unknown}"
echo "Commit SHA : ${GITHUB_SHA:-unknown}"
echo "Event type : ${GITHUB_EVENT_NAME:-unknown}"

visibility="unknown"
if [[ -n "${GITHUB_EVENT_PATH:-}" && -f "$GITHUB_EVENT_PATH" ]]; then
  # Attempt to read .repository.visibility from the event JSON
  visibility="$(jq -r '.repository.visibility // empty' "$GITHUB_EVENT_PATH" || true)"
fi

case "$visibility" in
  private) echo "Visibility : PRIVATE";;
  public)  echo "Visibility : PUBLIC";;
  internal) echo "Visibility : INTERNAL";;
  *)       echo "Visibility : ${visibility}";;
esac
```

## 1.2 Dockerfile

**Purpose:** Provides a tiny container with `bash` and `jq` for JSON parsing.

```dockerfile
FROM ubuntu:22.04

RUN apt-get update && \
    apt-get install -y --no-install-recommends bash jq ca-certificates && \
    rm -rf /var/lib/apt/lists/*

COPY entrypoint.sh /usr/local/bin/entrypoint.sh
RUN chmod +x /usr/local/bin/entrypoint.sh

ENTRYPOINT ["/usr/local/bin/entrypoint.sh"]
```

### 1.3 action.yml

> **Purpose:** Declares the action metadata and points to the Dockerfile.

```yaml
name: "Print GitHub Actor and Repo Details"
description: "Shows who triggered the run, the commit SHA, event type, and repo
↪  visibility."
author: "Your Name"
runs:
  using: "docker"
  image: "Dockerfile"
branding:
  icon: info
  color: blue
```

### 1.4 .github/workflows/test-custom-action.yml

> **Purpose:** Validates the action by running it on push and manual dispatch.

```yaml
name: Test Custom Action
on: [push, workflow_dispatch]

jobs:
  run-custom-action:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      # Option A: If testing within the same repository (no release needed)
      - name: Run custom action from local path
        uses: ./

      # Option B: If using a published repo (replace with your org/user and branch/tag)
      # - name: Run custom action from GitHub
      #   uses: your-username/your-repo@main
```

## 2 Quick Start

1. Create a new repository, e.g., `custom-action-challenge`.
2. Add the four files from this document and commit to `main`.
3. Push to GitHub and open the **Actions** tab.
4. Select **Test Custom Action** and run it (or push another commit).
5. Inspect job logs. You should see lines similar to:

```
Actor      : your-handle
Commit SHA : abcdef123456...
Event type : workflow_dispatch
Visibility : PUBLIC
```

## 3 Using the Action in Other Repos

In any workflow where you want to call this action, add a step like:

```yaml
- name: Who triggered this?
  uses: your-username/custom-action-challenge@v1
```

Create a release tag (`v1`) or pin to a specific commit SHA for reproducibility.

## 4 Publish to GitHub Marketplace (Optional)

1. Ensure the repository is public and has a clear `README.md`.
2. Create a release: **Releases** → **Draft a new release** (e.g., `v1.0.0`).
3. Check "Publish this action to the GitHub Marketplace" and follow the prompts.
4. Consumers can then use `your-username/custom-action-challenge@v1`.

## 5 Troubleshooting

- **No output or `jq` not found:** Confirm the Docker image built and `entrypoint.sh` is executable.
- **Visibility shows `unknown`:** Some events may not include `repository.visibility`. Try on `push` or `workflow_dispatch` in a normal repository, not a fork.
- **Action fails to run:** If using Option B, ensure `uses: your-username/your-repo@main` points to a repo that contains `action.yml` at the repository root.
- **Marketplace validation errors:** Add fields like `author`, `branding`, and ensure your `README.md` describes usage.

## Security Notes

- Docker actions run in containers you control; keep the base image minimal and up to date.
- Avoid echoing secrets. If you later add inputs/outputs, mark sensitive outputs and respect `ACTIONS_STEP_DEBUG`.
- Pin external actions you consume (`@vX` or commit SHA) to reduce supply-chain risk.

## Next Steps

- Add `inputs` to accept options (e.g., a custom message) and read them via env vars.
- Add `outputs` for downstream steps.
- Write unit tests for the entrypoint script (e.g., using `bats`).