Build and test your snaps automatically using GitHub actions

Ubuntu
Summit 2024

Introduction!

Please download these slides!

You will find them on the **Ubuntu Summit website**, find this workshop on the timetable, open its description page and go to **"Presentation Materials"** (bottom) or Click Here!

This way you can browse the slides in your own pace while setting up and doing the exercises.

And you can copy and paste command lines and example code.

You can click the numerous link!

You can also read the advanced topics which will not get necessarily presented here.

May the source (and these slides) always be with you!

Contents:

- Keep your snaps up-to-date regularly
- Build snaps with each PR or Commit
- ghvmctl
  - About the tool
  - Using it for testing GUI apps inside github runners

Pre-Requisites

There are some prerequisites needed so that you can use this CI setup in your snap project repo.

1. Basic knowledge about Github Actions and workflows
2. Explicitly mentioning the architectures that'll be supported
3. Hardcoding the version of the snap
4. Getting an account in launchpad
5. Grab this script
    a. `wget https://raw.githubusercontent.com/snapcrafters/.github/refs/heads/main/creds_script.py`
    b. Or use snapcraft remote-build
6. Use one of the above scripts to get your launchpad credentials

The Complete Flow

There are basically two flows. The first is a set of workflows to release new updated versions of your snap, using the following actions.

The first workflow

1. Checking for update every night and updating it if there is
2. Releasing the snap to the store and create call for testing and screenshots
3. Promoting the snap to the stable channel

The second workflow

Get the PRs built everyday

For this workshop

We’re using the discord snap repo as example

https://github.com/snapcrafters/discord

Sync with upstream

First we'll look into how we can integrate the action to sync with upstream

- Add this part in a github workflow
- You can name that file update.yaml

```yaml
name: Update

on:
  # Runs at 10:00 UTC every day
  schedule:
    - cron: "0 10 * * *"
  # Allows you to run this workflow manually from the Actions tab
  workflow_dispatch:

concurrency:
  group: ${{ github.workflow }}-${{ github.ref }}
  cancel-in-progress: true

jobs:
  sync:
    name: 🔄 Sync version with upstream
    environment: "Candidate Branch"
    runs-on: ubuntu-latest
    steps:
      - name: 🔄 Sync version with upstream
        uses: snapcrafters/ci/sync-version@main
        with:
          token: ${{ secrets.SNAPCRAFTERS_BOT_COMMIT }}
          update-script: |
            VERSION=$(
              curl -sL https://api.github.com/repos/helm/helm/releases |
              jq -r '.[].tag_name' | grep -m1 -vE 'alpha|beta|rc' | tr -d 'v'
            )
            sed -i 's/^\(version: \).*$/\1'"$VERSION"'/' snap/snapcraft.yaml
```

Sync with upstream

Questions you might have been thinking...

- What’s the update script for my snap?
- Is it same for all snaps?

Sync with upstream

The update script is just a shell script. From the docs, “A script that checks for version updates and updates snapcraft.yaml and other files if required.”

How to create the update script for your snap?

1. Check the parts you have and what are their source-type.
2. Look into the ways you can check for a new release of that source.
3. Create a bash script to update it, using tools like curl, wget, yq, jq etc
4. Add it in the update-script section

Sync with upstream

Common ways to fetch updates for some source-types

- **Git:** Using git ls-remote command
- **Tar/Deb:**
  - **From releases:** If the tar files are released in places like Github, Gitlab etc, then you can use their api to check about new releases. Read more about it here (A quick tip, often the release tarball names are static and follows a tag or a branch, so, you can simply look for a new tag and replace it with the old one)
  - **From Websites:** If the release tarball is in a scrape, you can simply scrape the html to look for a new release using regex. Like it's done here

Sync with upstream

Common ways to fetch updates for some source-types

- **Deb repositories:** If it's a deb repository then you can follow this to fetch the updates. Some examples are here

```
DEB_API="https://discord.com/api/download?platform=linux&format=deb"
DEB_URL=$(curl -w "%{url_effective}\n" -I -L -s -S "${DEB_API}" -o /dev/null)
VERSION=$(echo "${DEB_URL}" | cut -d'/' -f6)
```

Sync with upstream

Wanna give it a try?

Why not try it with codespell or marktext

Or, try it with your project and we'll help you out!

Releasing the snap

Now we’ll have to release the snap

This part revolves around multiple jobs, let’s look at them one by one

Releasing the snap

Getting the architectures

Your snap should specify for which architectures it should be built.

The action automatically gets this info and builds for those archs

```yaml
name: Release

on:
  # Run the workflow each time new commits are pushed to the candidate branch.
  push:
    branches: [ "candidate" ]
  workflow_dispatch:

concurrency:
  group: ${{ github.workflow }}-${{ github.ref }}
  cancel-in-progress: true

permissions:
  contents: read
  issues: write

jobs:
  get-architectures:
    name: 📱 Get snap architectures
    runs-on: ubuntu-latest
    outputs:
      architectures: ${{ steps.get-architectures.outputs.architectures }}
      architectures-list: ${{ steps.get-architectures.outputs.architectures-list }}
    steps:
      - name: 📱 Get snap architectures
        id: get-architectures
        uses: snapcrafters/ci/get-architectures@main
```

Releasing the snap

Building and Releasing

Next, we'll build the snap in launchpad using remote-build feature of snapcraft. This will be done in a matrix strategy, where the architectures will create the matrix.

```yaml
name: Release

on:
  # Run the workflow each time new commits are pushed to the candidate branch.
  push:
    branches: [ "candidate" ]
  workflow_dispatch:

concurrency:
  group: ${{ github.workflow }}-${{ github.ref }}
  cancel-in-progress: true

permissions:
  contents: read
  issues: write

jobs:
  get-architectures:
    name: 🖥 Get snap architectures
    runs-on: ubuntu-latest
    outputs:
      architectures: ${{ steps.get-architectures.outputs.architectures }}
      architectures-list: ${{ steps.get-architectures.outputs.architectures-list }}
    steps:
      - name: 🖥 Get snap architectures
        id: get-architectures
        uses: snapcrafters/ci/get-architectures@main
```

Releasing the snap

Building and Releasing

We'll get a snap file from the build as an output. Then the action will manually upload this snap in the store and store the revision number and other details in a release.

```yaml
name: Release

on:
  # Run the workflow each time new commits are pushed to the candidate branch.
  push:
    branches: [ "candidate" ]
  workflow_dispatch:

concurrency:
  group: ${{ github.workflow }}-${{ github.ref }}
  cancel-in-progress: true

permissions:
  contents: read
  issues: write

jobs:
  get-architectures:
    name: 🖥 Get snap architectures
    runs-on: ubuntu-latest
    outputs:
      architectures: ${{ steps.get-architectures.outputs.architectures }}
      architectures-list: ${{ steps.get-architectures.outputs.architectures-list }}
    steps:
      - name: 🖥 Get snap architectures
        id: get-architectures
        uses: snapcrafters/ci/get-architectures@main
```

Releasing the snap

Were you successful releasing your snap in the store?

Next Steps...

- Getting a call for testing issue
- Taking screenshots of the snaps from a github vm
- Promoting the snap to stable
- Poking around with parameters
- Customizing the Call for Testing message

Call for Testing

Now, in the same file which you setup for releasing the snap, add this.

```
46      call-for-testing:
47        name: 📣 Create call for testing
48        needs: [release, get-architectures]
49        environment: "Candidate Branch"
50        runs-on: ubuntu-latest
51        outputs:
52          issue-number: ${{ steps.issue.outputs.issue-number }}
53        steps:
54          - name: 📣 Create call for testing
55            id: issue
56            uses: snapcrafters/ci/call-for-testing@main
57            with:
58              architectures: ${{ needs.get-architectures.outputs.architectures }}
59              github-token: ${{ secrets.GITHUB_TOKEN }}
60
```

Call for Testing

Now, in the same file which you setup for releasing the snap, add this.

```
46      call-for-testing:
47        name: 📣 Create call for testing
48        needs: [release, get-architectures]
49        environment: "Candidate Branch"
50        runs-on: ubuntu-latest
51        outputs:
52          issue-number: ${{ steps.issue.outputs.issue-number }}
53        steps:
54          - name: 📣 Create call for testing
55            id: issue
56            uses: snapcrafters/ci/call-for-testing@main
57            with:
58              architectures: ${{ needs.get-architectures.outputs.architectures }}
59              github-token: ${{ secrets.GITHUB_TOKEN }}
60
```

Promote to stable

Now, the call for testing we added previously, will create an issue in the repo. This will