Aim

Upload packages to an OCI registry (**ghcr.io**) alongside **anaconda.org**

Updates

Aug 4, 2023

- Jaime, Hind and Vini met with Matt today and discussed possible ways to implement staging and publication in the OCI mirror. We had met the week earlier with Wolf too for the same purpose (see this comment).
- It looks like we can use a two-organization setup to control staging:
    - Channel-mirrors: production channel. The one conda clients use.
    - Channel-mirrors-staging: only for validation and quality control. Not meant for public consumption.
    - Feedstocks will have a token that allows the upload packages to staging. A successful upload will also call conda-forge-webservices to request validation of that package.
        - If successful, webservices will add the package artifact to a “promotion queue”. This queue can be a file in admin-requests. We need to check that this does not suffer from concurrency issues (even with the turnstyle step?).
        - If failed, the artifact will be removed from the staging channel.
    - We need to ensure that the OCI API can return the hash of the conda artifact in an easy way.
- Assuming repodata.json is accurate (either because it was mirrored from anaconda.org or because we regenerated it), the conda client plugin will be responsible for processing the co-located package URLs into valid OCI API calls to obtain the artifact.

Some documentation

- https://conda-forge.org/docs/_downloads/f55f7fa98f0dc3f3d3684e98163614e0/czi-eoss-5-loi-infra.pdf
- https://conda-forge.org/docs/_downloads/5277103c6f3c8a986da5eccbc5aaf585/czi-eoss-5-full-infra.pdf
  (cf. last page diagram in particular)

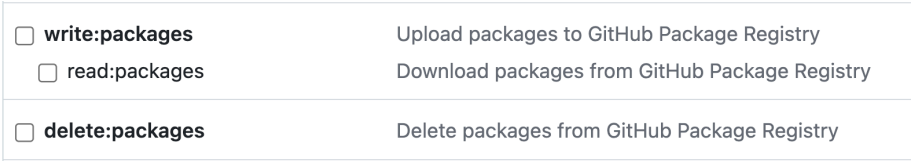
Status

- The upload would be performed a priori using **conda-oci-mirror** implemented by Wolf (https://github.com/channel-mirrors/conda-oci-mirror), which provides additional layers (**index.json** and **info.tar.gz**) compared to **Oras** (https://oras.land/) (only **package.tar.bz2 / .conda**)
- The **conda-oci-mirror** package is working properly so far (with some items that need to be done cf. https://github.com/channel-mirrors/conda-oci-mirror/blob/main/TODO.md)
- First **'upload'** implementation attempt:
    - Install **conda-oci-mirror** (using pip directly from the github repository) within conda-smithy (https://github.com/conda-forge/conda-smithy/pull/1683 - PR outdated after the last changes on **conda-oci-mirror**) and do the upload within **conda-forge-ci-setup-feedstock** (https://github.com/conda-forge/conda-forge-ci-setup-feedstock/pull/208)

  Matthew R. Becker from conda-forge suggests to do this either on the heroku server or using a dispatch to github actions (cf. comments in https://github.com/conda-forge/conda-forge-ci-setup-feedstock/pull/208)
  We would need a staging area and a secured copy to the actual OCI registry after verification (if we proceed similarly to what has been done with anaconda I guess that would be **ghcr.io/cf-staging** and **ghcr.io/conda-forge**) (cf. last page diagram of document)

TODO

- Open a PR for **conda-oci-mirror** recipe
- Discuss with Wolf and Matthew (and maybe others?) how and where the upload would be done
    - Jaime: The main concern right now is how to do staging in a safe way.
        - I read the OCI spec and apparently it supports the notion of "mounting blobs" from other registries. This means it could mimic the cf-staging to conda-forge setup in Anaconda.org. The Github Packages API doesn't seem to support mounting though. There are also some issues online about it, and still open.
        - Permission-wise, GH distinguishes between read, write and delete, which means that a properly scoped token used by feedstocks could maybe just write too many things, but in no way delete existing blobs. Note these tokens are NOT fine-grained:



        - There's also a 30-day restore window if necessary. Deleted packages are available in the Settings UI.

    - We also need to consider the index / repodata.json generation as a way of publishing a package or not, but this offers a few challenges:
        - Conda-index assumes the artifacts are locally available
        - In the OCI mirror, the artifacts are remote, but the metadata is individually queriable. With some CI caching it could be done
        - See how homebrew does this with 15-min scheduled jobs; even the API is pre-generated JSON deployed to GH Pages in an environment. Their biggest payload is 20MB pure JSON though. These point to sha256 headers in GHCR.io
        - Even if “publishing to repodata” is the staging operation, we still need a way to signal if a package is publishable or not. Can a custom label (annotations? tags?) be attached to an artifact to mark it as valid or not, AFTER uploading? If that’s possible, only marked artifacts are part of the repodata.json, and unmarked artifacts (because they didn’t pass validation) will be deleted after X hours. Accidental deletions are still restorable within 30 days.
- Handle/generate appropriate tokens according to the picked solution/implementation