Cloud-friendly APT repository publishing

Contents

Why we need a new APT publisher

Apertis relies on OBS[1] for building and publishing binary packages. However, upstream OBS provides an APT publisher based on `dpkg-scanpackages`, which is not suitable for a project the scale of Apertis, where a single OBS project contains a lot of packages.

Therefore, our OBS instance uses a custom publisher based on `reprepro`, but it is still subject to some limitations that are now more noticeable as the scale of Apertis has grown considerably:

- It purely acts on the events created by OBS, which means that if they do not get successfully processed immediately the repository will go out-of-sync
- The repositories are hosted on the same machine as OBS, and there is no way for external tools to interact with them; for instance there is no way to request the creation of snapshots with a different strategy than the current one of creating a snapshot for every single change
- When branching a release `reprepro` has to be invoked manually to initialize the exported repositories
- When branching a release the OBS publisher has to be manually disabled or it will cause severe lock contention with the manual invocation mentioned above
- Removing a package requires manual intervention
- Snapshots are not supported natively
- Cloud storage is not supported

In order to address these shortcomings, we need to develop a new APT publisher (based on a backend other than `reprepro`) which should be capable of:

- Publishing the whole Apertis release on non-cloud storage
- Publishing the whole Apertis release on cloud storage
- Natively supporting snapshots
- Automatic branching of an Apertis release, not requiring manual intervention on the APT publisher

[1] https://www.apertis.org/architecture/distribution/workflow-guide/

- Using a synchronization strategy to ensure that OBS and APT repositories automatically tend to consistent state:
  - removing a package from OBS should trigger the removal of the package from the APT repositories as well
  - once a publishing failure is resolved (network issues, etc.) the publisher should recover automatically

Alternatives to `reprepro`

The Debian wiki includes a page[2] listing most of the software currently available for managing APT repositories. However, a significant portion of those tools cover only one of the following use-cases:

- managing a small repository, containing only a few packages
- replicating a (sometimes simplified) official Debian infrastructure

A few of the mentioned tools, however, are aimed at managing large-scale repositories within a custom infrastructure, and offer more advanced features which could be of interest to Apertis. Those are:

- aptly
- pulp

Laniakea[3] was also considered, but as it's meant to work within a full Debian-like infrastructure and doesn't offer any cloud-based storage option, it was dismissed as well.

Extended search did not point to other alternative solutions covering our use-case.

Aptly

Aptly[4] is a complete solution for Debian repository management, including mirroring, snapshots and publication.

It uses an internal, locally-stored package pool and database, and provides cloud storage options for publishing ready-to-serve repositories. Aptly also provides a full-featured CLI client and an almost complete REST API. It could therefore run either directly on the same server as OBS, or on a different one. The REST API misses mirroring support for now, so these features can only be used from the command-line client.

Package import and repository publication are separate operations:

- The package is first imported to the internal package pool and associated to the requested repository in a single operation

[2] https://wiki.debian.org/DebianRepository/Setup
[3] https://github.com/lkhq/laniakea
[4] https://www.aptly.info/

- When all required packages are imported, the repository can be published atomically

Repositories can be published both to the local filesystem and to a cloud-based storage service (Amazon S3 or OpenStack Swift).

Moreover, Aptly identifies each package using the (name, version, architecture) triplet: by doing so, it allows keeping multiple versions of the same package in a single repository, while `reprepro` kept only the latest package version. This requires additional processing for Aptly to replicate the current behavior.

Finally, attention should be paid to regularly cleaning up the database and package pool: unused packages are kept in the pool, even when obsoleted by a newer version and/or removed from all repositories, until a database cleanup is triggered. A daily cleanup job should be sufficient to make sure the internal pool doesn't carry unused packages over time.

Pros

- tailored for APT repository management: includes some interesting features such as multi-component publishing
- command-line or REST API interface (requires an additional HTTP server for authentication and permissions management)

Cons

- uses a local package pool which can grow large if a lot of packages and versions are used simultaneously
- requires additional processing to keep only the latest version of each package
- needs regular database cleanups

Pulp

Pulp[5] is a generic solution for storing and publishing binary artifacts. It uses plugins for managing specific artifact types, and offers a plugin for DEB packages.

It offers flexible storage options, including S3 and Azure, which can also be extended as the storage backend is built on top of `django-storages`, which provides a number of additional options.

Pulp can be used through a REST API, and provides a command-line client for wrapping a significant portion of the API calls. Unfortunately, the DEB plugin isn't handled by this client, meaning only the REST API is available for managing those packages.

Its package publication workflow involves several Pulp objects:

[5] https://pulpproject.org/

- the binary artifact (package) itself
- a Repository
- a Publication
- a Distribution

Each Distribution is tied to a single Publication, which is itself tied to a specific Repository version. As each Repository modification increments the Repository version, adding or removing a package involves the following steps:

- add or remove the package from the Repository
- retrieve the latest Repository version
- create a new Publication for this repository version
- update the Distribution to point to the new Publication
- remove the previous Publication

This workflow feels too heavy and error-prone when working with a distribution the scale of Apertis, where lots of packages are often added or updated. Additionally, each Distribution must have its own base URL, preventing publishing multiple Apertis versions and components in the same repository.

Pros

- generic artifacts management solution: can be re-used for storing non-package artifacts too
- flexible storage options

Cons

- complex workflow for publishing/removing packages
- unable to store multiple repositories on the same base URL
- can only be used through REST API

Conclusion

Based on the above software evaluation, `aptly` seems to be the more appropriate choice:

- supports snapshots
- can make use of both local and cloud-based storage for publishing repositories
- provides useful features aimed specifically at APT repository management
- allow publishing several repositories and components to a single endpoint

Its main shortcoming (locally-stored package pool) can be addressed by implementing an option for storing the pool on cloud-based storage. This would be the most efficient approach when compared to the alternative (hosting `aptly` on a remote server and using it through the REST API).

Moreover, the following points must be kept in mind when implementing the publisher:

- `aptly` doesn't remove previous versions of an updated package; although this behavior could be implemented in `aptly` itself, it will be less effort to have the publisher handle removing obsoleted packages
- the package pool will keep growing as new and updated packages are added, it should therefore be cleaned up on a regular basis by triggering database cleanups
- publishing large repositories with aptly can take a long time; decoupling the action of adding a package from the actual repository publication would be a useful optimization, however it would be outside the scope of the initial implementation

Finally, `aptly` is actively maintained upstream, with a new team of developers having taken over its development last year. The chances of it being abandoned and/or replaced with a different project are therefore very low.

Implementation plan

- Update OBS to a more recent upstream version: this will provide a more up-to-date base on which we can develop and upstream the new APT publisher
- Start with a prototype, local-only version capable of:
  - adding a package to a (manually created) local repository
  - publishing the repository to local storage
  - deleting a package from the repository when removing it from OBS
- Implement automated branching and repository creation for new OBS projects
- Automate periodic database cleanups
- Add configuration options for publishing to cloud-based storage
- Implement cloud-based storage options for `aptly`'s internal package pool