



Cloud-friendly APT repository publishing

1 Contents

2	Why we need a new APT publisher	2
3	Alternatives to reprepro	2
4	Aptly	3
5	Pulp	4
6	Conclusion	5
7	Implementation plan	6

8 Why we need a new APT publisher

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

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

- 16 • When branching a release `reprepro` has to be invoked manually to initialize
17 the exported repositories
- 18 • When branching a release the OBS publisher has to be manually disabled
19 or it will cause severe lock contention with the manual invocation men-
20 tioned above
- 21 • Removing a package requires manual intervention
- 22 • Snapshots are not supported natively
- 23 • Cloud storage is not supported

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

- 26 • Publishing the whole Apertis release on non-cloud storage
- 27 • Publishing the whole Apertis release on cloud storage
- 28 • Natively supporting snapshots
- 29 • Automatic branching of an Apertis release, not requiring manual interven-
30 tion on the APT publisher
- 31 • Synchronize OBS and APT repositories; as an example, removing a pack-
32 age from OBS should trigger the removal of the package from the APT
33 repositories as well

¹<https://em.pages.apertis.org/apertis-website/architecture/workflow-guide/>

34 Alternatives to reprepro

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

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

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

- 43 • [aptly](#)
- 44 • [pulp](#)

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

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

50 Aptly

51 [Aptly](#)⁴ is a complete solution for Debian repository management, including
52 mirroring, snapshots and publication.

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

59 Package import and repository publication are separate operations:

- 60 • The package is first imported to the internal package pool and associated
61 to the requested repository in a single operation
- 62 • When all required packages are imported, the repository can be published
63 atomically

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

66 Moreover, Aptly identifies each package using the (name, version, architecture)
67 triplet: by doing so, it allows keeping multiple versions of the same package in

²<https://wiki.debian.org/DebianRepository/Setup>

³<https://github.com/lkhq/laniakea>

⁴<https://www.aptly.info/>

68 a single repository, while `reprepro` kept only the latest package version. This
69 requires additional processing for Aptly to replicate the current behavior.

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

75 **Pros**

- 76 • tailored for APT repository management: includes some interesting fea-
77 tures such as multi-component publishing
- 78 • command-line or REST API interface (requires an additional HTTP server
79 for authentication and permissions management)

80 **Cons**

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

86 **Pulp**

87 `Pulp`⁵ is a generic solution for storing and publishing binary artifacts. It uses
88 plugins for managing specific artifact types, and offers a plugin for DEB pack-
89 ages.

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

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

97 Its package publication workflow involves several Pulp objects:

- 98 • the binary artifact (package) itself
- 99 • a Repository
- 100 • a Publication
- 101 • a Distribution

⁵<https://pulpproject.org/>

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

- 105 • add or remove the package from the Repository
- 106 • retrieve the latest Repository version
- 107 • create a new Publication for this repository version
- 108 • update the Distribution to point to the new Publication
- 109 • remove the previous Publication

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

114 **Pros**

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

118 **Cons**

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

122 **Conclusion**

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

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

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

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

- 136 • `aptly` doesn't remove previous versions of an updated package; although
137 this behavior could be implemented in `aptly` itself, it will be less effort to
138 have the publisher handle removing obsoleted packages

- 139 • the package pool will keep growing as new and updated packages are
140 added, it should therefore be cleaned up on a regular basis by triggering
141 database cleanups
- 142 • publishing large repositories with aptly can take a long time; decoupling
143 the action of adding a package from the actual repository publication
144 would be a useful optimization, however it would be outside the scope of
145 the initial implementation

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

149 **Implementation plan**

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