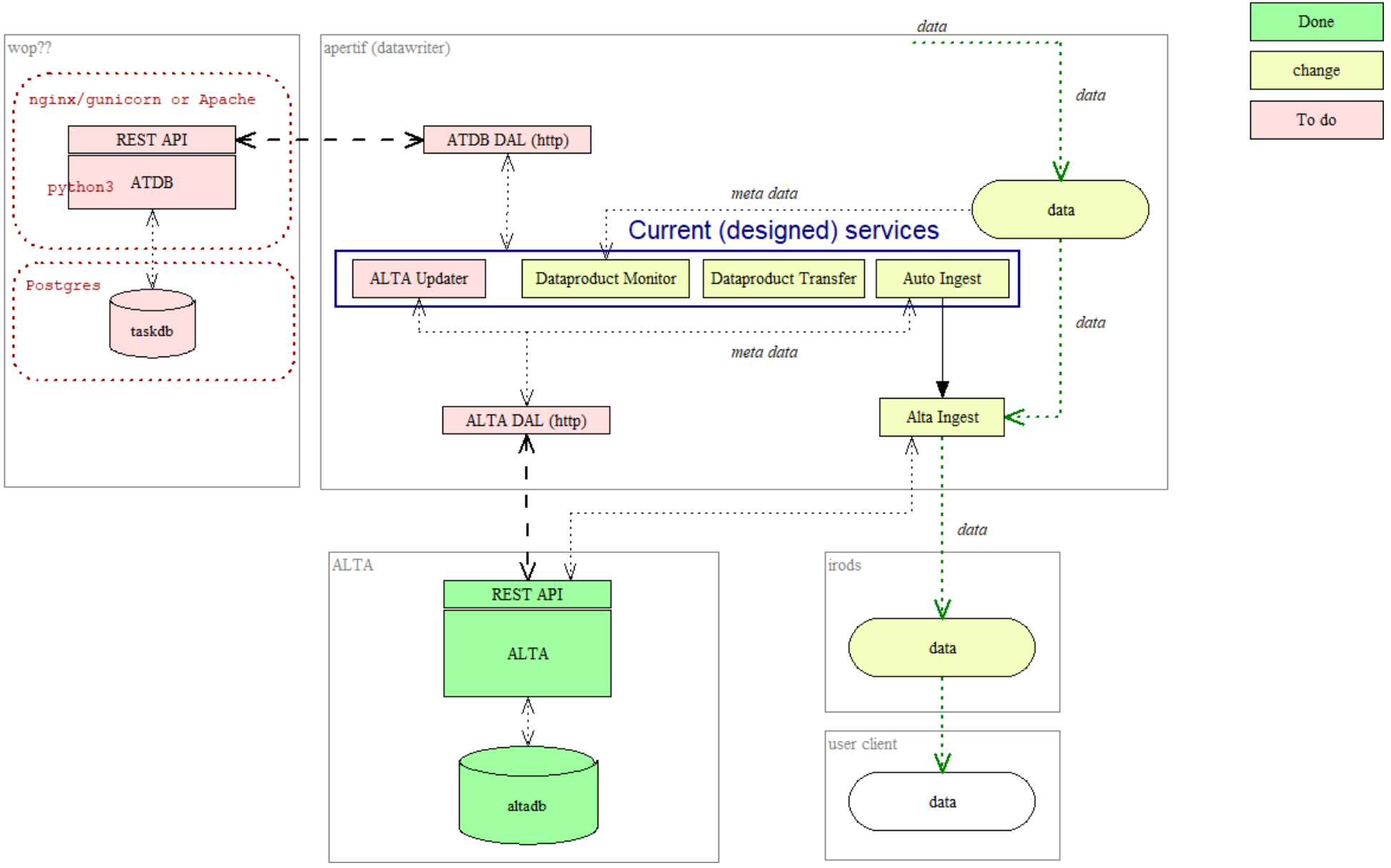
Apertif Task Database (ATDB) - as separate Django project

What are the consequences if the Apertif Task Database becomes a separate (`Django`) application, instead of integrating it with the current ALTA? Note that this analyses is only about a backend (database and application) and a REST API for communication. A frontend (GUI) is not in this scope.

There will be some extra work in setting up the `Django` project and the (web) system environment. But it will be a lot simpler than the ALTA setup and would give more flexibility. It would probably be the most effective to initially keep it as simple as possible and create a 'prototype' to see if it all works together. And only later do the Jenkins build and deploy jobs.

Apertif Task Database (Django option)

(nv: 19 jul 2018 - apertif_taskdatabase_django_option.sdr)



Setup

What needs to be done:

- `Django` project setup, including `Django Rest Framework`.
- Add it to a suitable place in the Apertif code repository (it would be `Python3`, would that cause a problem?)
- Configuration of database settings. (hardcoded password onboard, since we do not use the ALTA `Ansible` provisioning).

Pro's/Con's:

- pro: small and to the point, a fraction of the ALTA source tree.
- con: all this is extra work (1 day).

Environment

What needs to be done:

- Webserver needed that can host a `django/python` web application with wsgi. This can be `Gunicorn` with `Nginx` as a proxy, or `Apache2` which can provide both proxy and wsgi services.
- `Postgres` database server needed.

Pro's/Con's:

- pro: independent from Dwingeloo/ALTA
- con: extra work, also for system admins (1 day).

Development

What needs to be done:

- Development can be done locally in `Pycharm` (or other IDE) within a virtualenv with a local `Postgres` database, with the code in the Apertif `svn`.

Pro's/Con's:

- none, development is straightforward.

Testing

What needs to be done:

- `Django` comes with unittest functionality.
- A test environment can be setup in a virtual machine, using Vagrant and Ansible.

Pro's/Con's:

- pro: less complex than the ALTA test environment
- con: missing the automated (unit & integration) testing facilities of ALTA

Build/Deploy

What needs to be done:

- Building a `Django` application:
  - collect static files (1 command)
  - configure the application for deployment (switch settings file)
  - create database migration file (2 commands)

- Deploy a `Django` application:
  - copy the static files to the 'static' folder on the webserver (if the webserver is configured correctly, this is only 1 copy command).
  - copy the application to its web location (1 copy command)
  - migrate the database (1 command)

This is not complicated, but it will probably become a wish to automate this so it can be done from `Jenkins`. This would require the following:

- a build job to create the artifacts for deployment
- deploy job to place these artifacts on the webserver
- optional, store these artifacts in `Nexus`.

Pro's/Con's:

- pro : separated deploy cycle than ALTA, more flexible.
- con : extra work, because the ALTA option would not have this application at all. Setting up the automated build/deploy may cost days.

Database

What needs to be done:

- `Postgress` installation (once)
- database migration on updates

Pro's/Con's:

- pro : less complexity and less risk with database updates.
- con : ALTA may not be able to reach this database, not directly and not through it's REST API. (no authentication, so it will probably live locally inside the firewall only)

Maintenance

What needs to be done:

- System maintenance of webserver and database.
- Application maintenance (bugs, features)

Pro's/Con's:

- pro : for a novice Django developer it will be easier (and safer) without all the ALTA clutter around.
- con : requires some extra attention (and documentation), because it is a separate system/enviroment.

Documentation

What needs to be done:

- Development documentation. (datamodel, architecture diagram, dataflow diagram).
- Build/Deploy documentation. (how to manually or automated build and deploy)

Pro's/Con's:

- pro : Clearer if described outside of the ALTA docs.
- con : Build/Deploy docs are extra work.

ALTA

What needs to be done:

ALTA needs the pre-ingest information of the Task database also. With the 'integrated' option this would be very easy for Django, because the `taskdb` would conceptually be 'internal' for Django. Now a different mechanism is needed to get that information in ALTA.

This could be a separate `ALTA Update Service` which listens to changes in the ATDB and reports them to ALTA, using the two http interfaces (DAL). This would also require some extra thought about where to put (duplicate) the (uningested) dataproducts in ALTA now that the taskdb is not in ALTA.

Pro's/Con's:

- pro: ALTA not influenced by changes in taskdb.
- con : harder to access the data.
