Creating a new service with Flask 3 and Connexion 3

What to use for a new (Sphinx) service?

- OTS is an async app, built on aiohttp and Connexion 2
- Created from a service template in Gitlab
- We need to create a new service: Directmail
- Connexion 3 no longer supports aiohttp

Which frameworks should Gigaclear dev support in production?

We currently have in production:

- aiohttp + connexion 2 (async)
- Flask 2 + connexion 2 (sync)
- More?

aiohttp and Connexion

- See Redesigning Connexion as an ASGI middleware stack

https://medium.com/@robbe.sneyders/redesigning-connexion-as-an-asgi-middleware-stack-a5dc17e81ff8

> *Connexion has been rewritten as an asgi (*) middleware stack. aiohttp isn't supported as it is not asgi compatible.*

We shouldn't create new services using aiohttp. Flask 2 is also deprecated in favour of Flask 3. (Should we officially retire the aiohttp template?)

(*) asgi: Asynchronous Server Gateway Interface
https://asgi.readthedocs.io/en/latest/

What a template provides

- async programming
- a capable web app framework (e.g. aiohttp)
- Spec-first, open api yaml specification (Connexion)
- Serialization/deserialization and validation using dataclasses-jsonschema (also Connexion)
- Swagger UI plus client for our API (Connexion)
- Infrastructure integration: e.g. gcauth and logging
- Persistence layer with SQLAlchemy and alembic
- Standard test patterns and fixtures with pytest
- Configuration and tooling integration
- CI/CD, test and release stuff, etc...

Makes a bunch of technology choices and saves a bunch of work.

Provides familiar technologies that solve specific problems.

Options for a new service

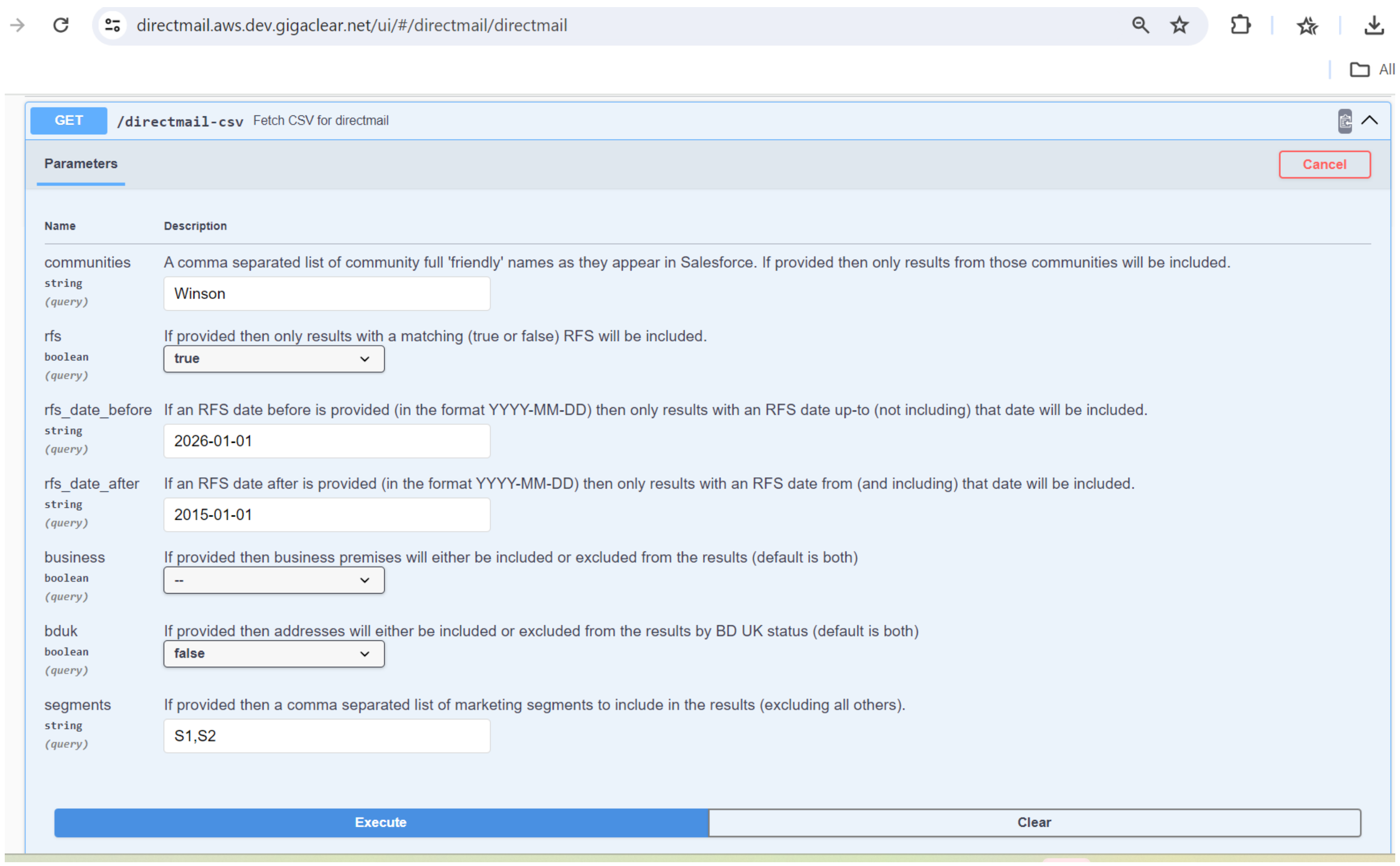
Major backend API frameworks within the Python ecosystem (roughly in order of size):

- Django REST Framework (full Django monolith)
- FastAPI (provides Swagger)
- Flask
- Connexion 3 (provides Swagger)

Advantages of using a "popular" framework:

- Likely to last, so ongoing bugfixes and security fixes
- Community resources like StackOverflow for finding information

Directmail



/directmail-csv Fetch CSV for directmail

Name	Description
communities string (query)	A comma separated list of community full 'friendly' names as they appear in Salesforce. If provided then only results from those communities will be included. Winson
rfs boolean (query)	If provided then only results with a matching (true or false) RFS will be included. true
rfs_date_before string (query)	If an RFS date before is provided (in the format YYYY-MM-DD) then only results with an RFS date up-to (not including) that date will be included. 2026-01-01
rfs_date_after string (query)	If an RFS date after is provided (in the format YYYY-MM-DD) then only results with an RFS date from (and including) that date will be included. 2015-01-01
business boolean (query)	If provided then business premises will either be included or excluded from the results (default is both) --
bduk boolean (query)	If provided then addresses will either be included or excluded from the results by BD UK status (default is both) false
segments string (query)	If provided then a comma separated list of marketing segments to include in the results (excluding all others). S1,S2

- Greenfield, low traffic
- Ideal opportunity to experiment

Requirements

- Hosted in Kubernetes (Amazon eks) - therefore Docker
- Ideally async
- Straightforward to work with
- Provides a Swagger UI and client
- Secure, reasonably performant, likely to get security upgrades for some years
- Spec first with request validation using dataclasses (of some kind)
- Compatible with our infrastructure like gcauth (especially OAuth 2)
- Easy to test with pytest (has a test client)

Flask 3 & Connexion 3

- Meets our requirements with familiar technologies
- Enthusiasm in the team to try it

Downside: Flask 3 is not natively async (*) but runs an event loop per request in a thread. This adds complexity and may have performance implications. Directmail is low traffic, performance is not a priority.

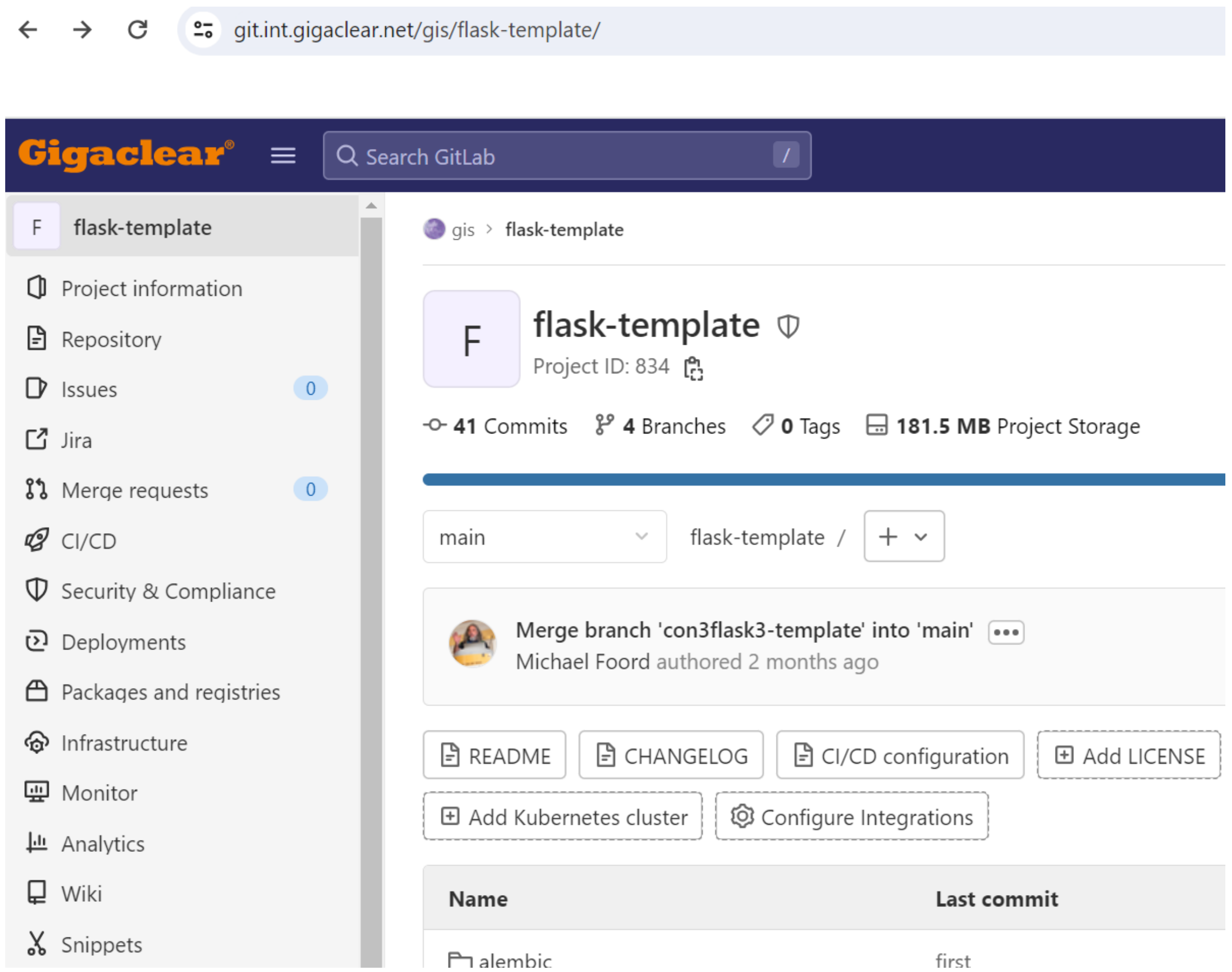
Approach:

- Develop a service template as Proof of Concept / way of testing viability
- Fork the working template to create directmail

Forking allows changes to the template to be easily merged to services.

(*) Quart exists as a native async implementation of Flask.
https://github.com/pallets/quart

Flask 3 & Connexion 3 Template



https://git.int.gigaclear.net/gis/flask-template/

Flask 3 & Connexion 3 Template

flask-template

A minimal template for an async Flask 3 with Connexion 3 app, using oauth 2 for authentication and docker compose for testing and local running. Tests are synchronous but use the connexion 3 test client and run the app async. There are tests with authentication.

The template does not have a configured database or models.

Using Docker

You need a .env file with the minimum values filled out.

Docker login

Prior to building you'll need a docker login:

```
docker login git.int.gigaclear.net:5005 --username=$
{GITLAB_USERNAME} --password=${GITLAB_TOKEN}
```

The Template: Standard Service Layout

A standard pattern for service structure/layout:

- `run.py` entrypoint along with the `create_app` function (in app)
- `client-gen-config.json` (and app/clients directory) to generate Swagger clients
- `app/data_queries.py` for code that makes queries
- `app/errors.py` for standard error responses
- `app/models.py` for SQLAlchemy model definitions
- `app/data_types.py` the request/response dataclass definitions
- An outline `api-spec.yml` with endpoint functions in `app/api.py` (`/healthz` and an authenticated endpoint)

Doesn't mandate project architecture. Further design/evolution driven by DDD (Domain Driven Design), etc.

The Template: Configuration & Integration

Standard configuration and integration with our infrastructure:

- Logging setup and configured (using structlog, integrated with elastic apm)
- Response compression setup and configured (brotli)
- Faster serialization put in place (using orjson, `data_access.py`)
- CORS (Cross Origin Resource Sharing) middleware
- `gcauth` for login functionality and authorization, roles etc
- A git submodule for our custom Swagger UI package (and the configuration/setup to use it)

The Template: Tooling Support

Plus tooling support:

- pytest tests package: example tests, `test_client` and token fixtures in `conftest.py`
- `Dockerfile`, `Pipfile`, `scripts/qa_commands.sh (*)`
- `config.py` and `.env.example` files - for both `pipenv shell` and docker
- docker compose files for running the service locally and for running tests (VSCode integration for debugging)
- alembic migration directory and config (**)
- `.gitlab-ci.yml` for pipelines: linting & mypy step, run tests, create build artefact (tags and releases)

(*) config for Black, mypy, ruff and pip-audit
(**) Not used by directmail

Template Downside

Downside 1: the template is not especially "clean" or "barebones". There's quite a lot of files, plus a bunch of code (237 lines in app/__init__.py, connexion_utils.py, utils.py, data_access.py, etc). Some of this *could* be moved into a package and out of the template/projects themself.

Downside 2: because Flask 3 is not natively async the test client is not async. It is possible to wrap the test client and write async tests but writing synchronous tests is the most straightforward way.

[*contd.*]

Template Downside

Downside 3: although spec first Connexion with Swagger, Flask 3 and async programming are all widely used within the Python community - using them in that combination is less common.

Connexion documentation is barebones and there isn't a lot of information available (StackOverflow/Google/etc) on working with this tech stack. We've had to dive down into the Connexion source code at times.

Creating a New Service

Creating a new service is done in Azure AD with scripts from `auth-service`. It also requires an MR against `auth-service` (and possibly `gcservices`) to define the new service and any roles.

With new style helm charts deployments can be done without requiring a change to the helm repo (deployment will track tags for the project, a step closer to CD). Just tag and release!

Conclusion

The Flask 3 & Connexion 3 template is suitable for creating new services. Where performance is an issue I would consider alternatives.

FastAPI is the industry standard (after the monolith of Django REST Framework which doesn't play well with SQLAlchemy) and FastAPI does provide spec first and support for open api yaml spec along with Swagger UI and client. Serialization and validation are done with pydantic.

I'd like to try FastAPI as a Gigaclear standard (with template) for new services.