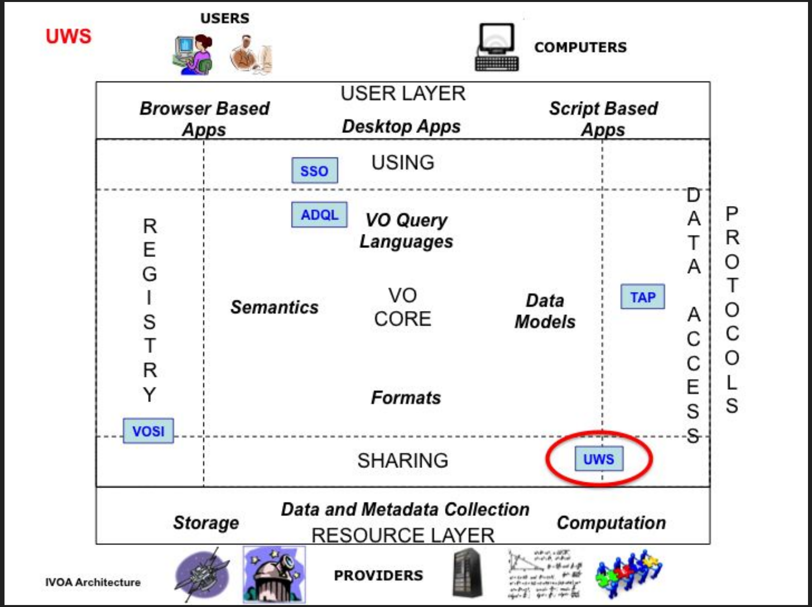
Implementing UWS in Django

Klaas Kliffen - ASTRON

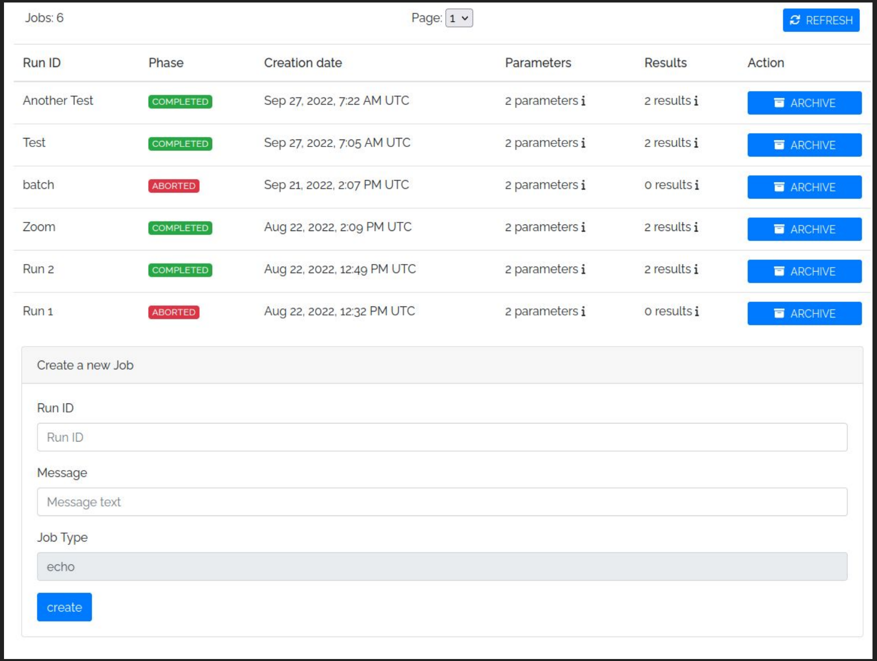
What is the UWS Specification

- A specification for a REST API for asynchronous services
- Scalable interface for long running tasks



Our implementation in ESCAPE ESAP

- ESAP (ESFRI Science Analysis Platform)
- Query and select datasets
- Browse software
- Run on compatible compute infrastructure
- UWS used for:
    - Batch processing
    - Planned: async queries



Jobs: 6 Page: 1

Run ID	Phase	Creation date	Parameters	Results	Action
Another Test	COMPLETED	Sep 27, 2022, 7:22 AM UTC	2 parameters i	2 results i	ARCHIVE
Test	COMPLETED	Sep 27, 2022, 7:05 AM UTC	2 parameters i	2 results i	ARCHIVE
batch	ABORTED	Sep 21, 2022, 2:07 PM UTC	2 parameters i	0 results i	ARCHIVE
Zoom	COMPLETED	Aug 22, 2022, 2:09 PM UTC	2 parameters i	2 results i	ARCHIVE
Run 2	COMPLETED	Aug 22, 2022, 12:49 PM UTC	2 parameters i	2 results i	ARCHIVE
Run 1	ABORTED	Aug 22, 2022, 12:32 PM UTC	2 parameters i	0 results i	ARCHIVE

Create a new Job

Run ID	Run ID
Message	Message text
Job Type	echo

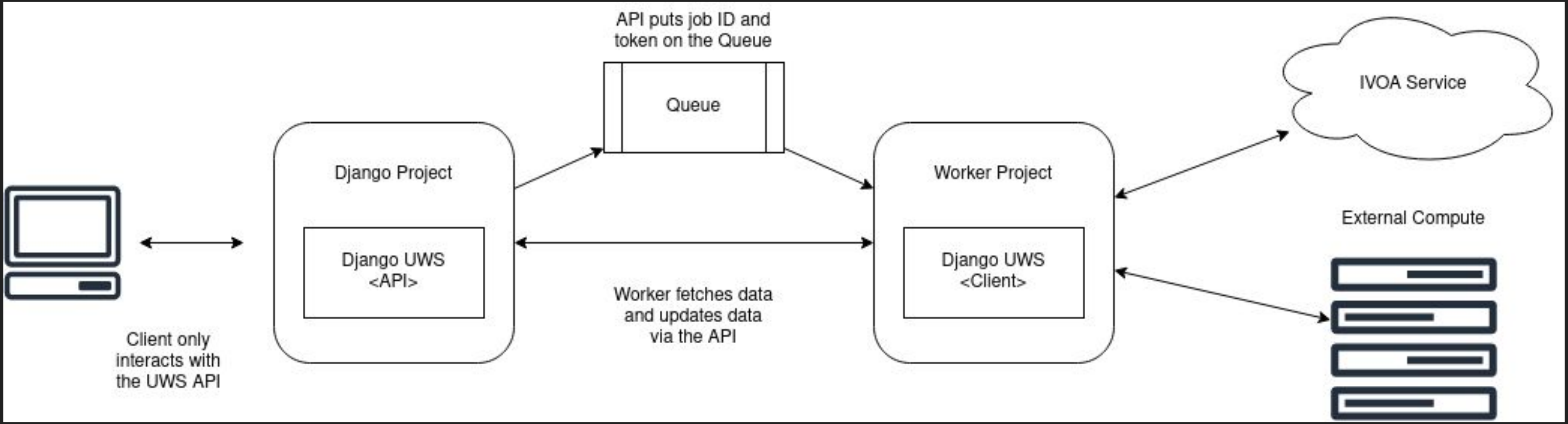
Technology Choices

- Python

- Django - Mature Web Framework
- Django REST Framework - For quickly building REST APIs in Django
- Celery - Mature message brokering software for Python

- RabbitMQ - Message Broker (but easy to switch to others! Such as Redis)

Component Architecture


API puts job ID and token on the Queue
Queue
IVOA Service
Django Project
Worker Project
External Compute
Django UWS <API>
Django UWS <Client>
Worker fetches data and updates data via the API
Client only interacts with the UWS API

Authorization

- Django uses pluggable auth for Clients of the API
    - Multiple options possible: Basic, JSON Web Tokens (JWT), X509 etc.

- Extra auth:
    - Per Job permission instead of per User
    - Plain tokens which can be used from curl (allows calls from external systems)

Lessons Learned - 1

- Test driven development (TDD) really useful!
  - Write tests following the definitions of the specification
  - Confidence in specification compliance and ease of refactoring

- Configure your Message Brokering software correctly!
  - Celery worker types: “pre-fork” vs “eventlet” (cpu vs IO heavy tasks)
  - Failing jobs not getting into an error state
  - Jobs not retried on failures outside the scope of the worker (network failures etc.)

Lessons Learned - 2

- Worker could live in the Django Project repository
    - Still requires (at least) two processes, but direct database access could be preferred over the API

- We chose a single endpoint (**`api/uws/`**), with a type parameter
    - Multiple endpoints might be better (**`api/batch/uws`**, **`api/query/uws`**, etc.)

- Worker does not need to be a Django Project if used as microservice
    - Worker communicates via the API and Django could be dropped as a dependency

Things I missed in the specification

And implemented:

- JSON support (reach out to the JSON working group?)
- Update the Job via the same API (micro service support)

Not implemented:

- A way of describing the accepted parameters via the API (Job Description Language)
  - Self documenting API

Future work

- First release
    - Currently in alpha
    - Reach out if you want to collaborate!
    - Implement the complete specification
        - Actions such as getting a time quote are missing
        - Filtering of Jobs
    - Support for XML
        - Currently the API only supports JSON responses
        - POST requests accept both JSON and URL encoded form data

Thank you for listening!

References

- https://www.ivoa.net/documents/UWS/ - The specification
- https://git.astron.nl/astron-sdc/django-uws - The main repository
- https://pypi.org/project/django-uws/ - Python package published on Pypi
- https://git.astron.nl/astron-sdc/escape-wp5/esap-worker - Example worker implementation
- https://git.astron.nl/astron-sdc/esap-api-gateway - ESAP API using UWS
- https://docs.celeryq.dev/en/stable/ - Celery documentation
- https://www.django-rest-framework.org/ - Django REST Framework