ServiceX

iris hep

Institute for Research & Innovation
in Software for High Energy Physics

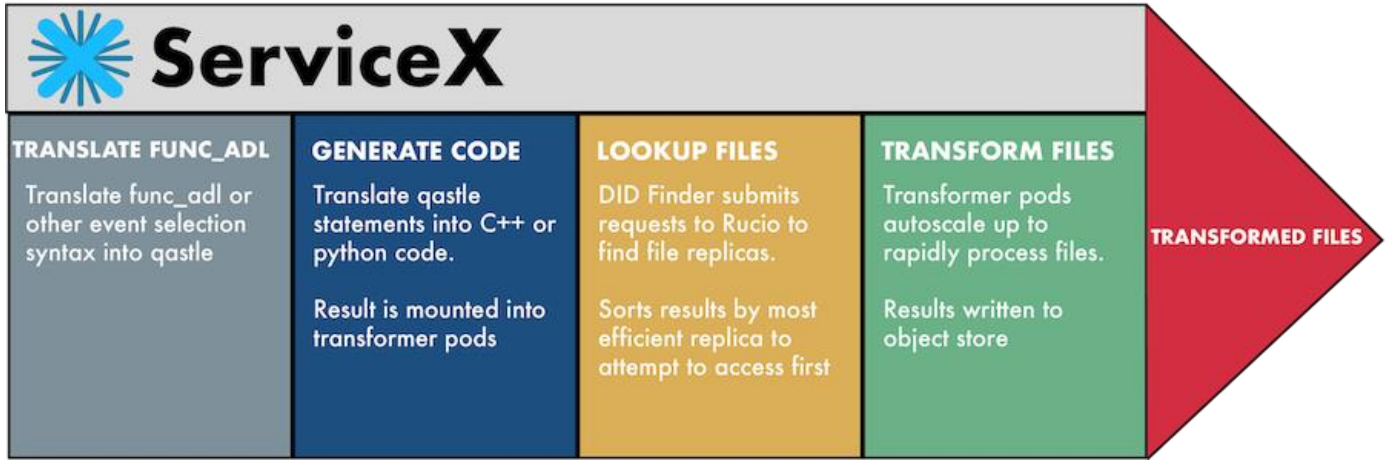
3.0

G. Watts (UW/Seattle)

For the ServiceX Team

Nov 1, 2024

ServiceX Architecture Reminder



TRANSLATE FUNC_ADL

Translate func_adl or other event selection syntax into qastle

GENERATE CODE

Translate qastle statements into C++ or python code.

Result is mounted into transformer pods

LOOKUP FILES

DID Finder submits requests to Rucio to find file replicas.

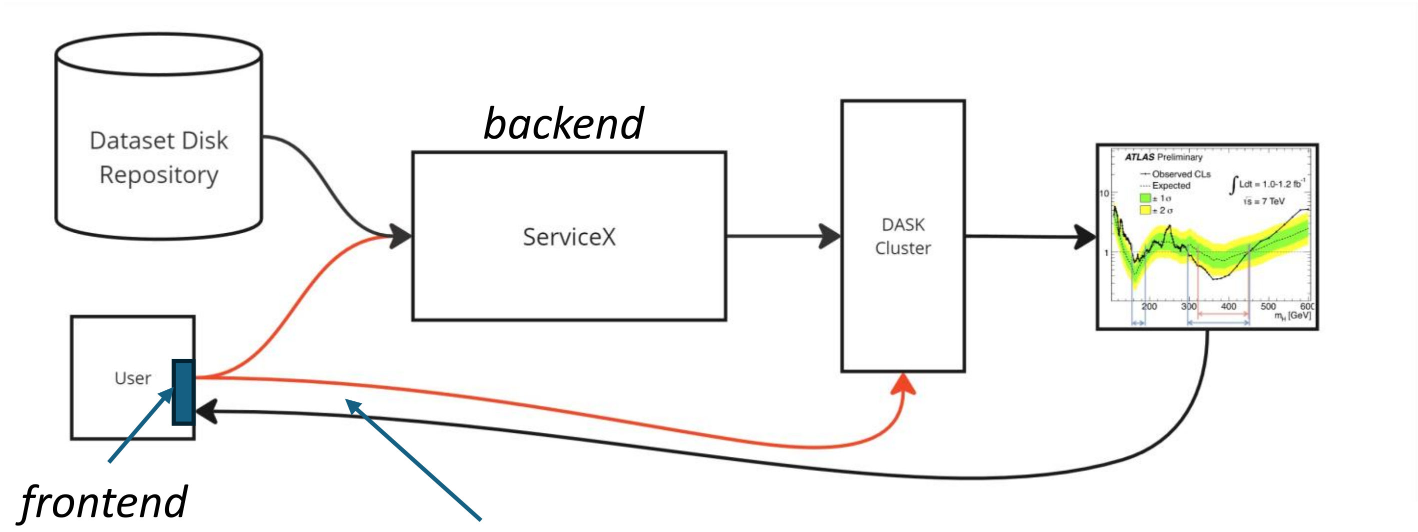
Sorts results by most efficient replica to attempt to access first

TRANSFORM FILES

Transformer pods autoscale up to rapidly process files.

Results written to object store

TRANSFORMED FILES



Web API

Recent Developments in ServiceX

- Backend Developments (ServiceX itself)
  - 1.4.1 (August 7th) -> 1.5.X (Sept 20th)
  - Lessons from the IDAP 200 Gpbs test: reduce lost internal scaling messages, small updates to how we transform the data, lots of small stability improvements.
  - New transformers/codegenerators to run plain-old-python
  - The WebAPI did not change, however!
  - Won’t really discuss further
- Front End (Library to enable user interaction)
  - 3.0 released Sept 20th (after 5 months of development)
  - Big (breaking) change in how the user interacts with ServiceX
  - Pulled several ideas from the community into the central library (e.g. datasets)
  - Queries can be coded with typed classes, dictionaries, or yaml text files now!
  - DOCUMENTATION!!

Getting Started: Help!

Find our new documentation on *readthedocs*:

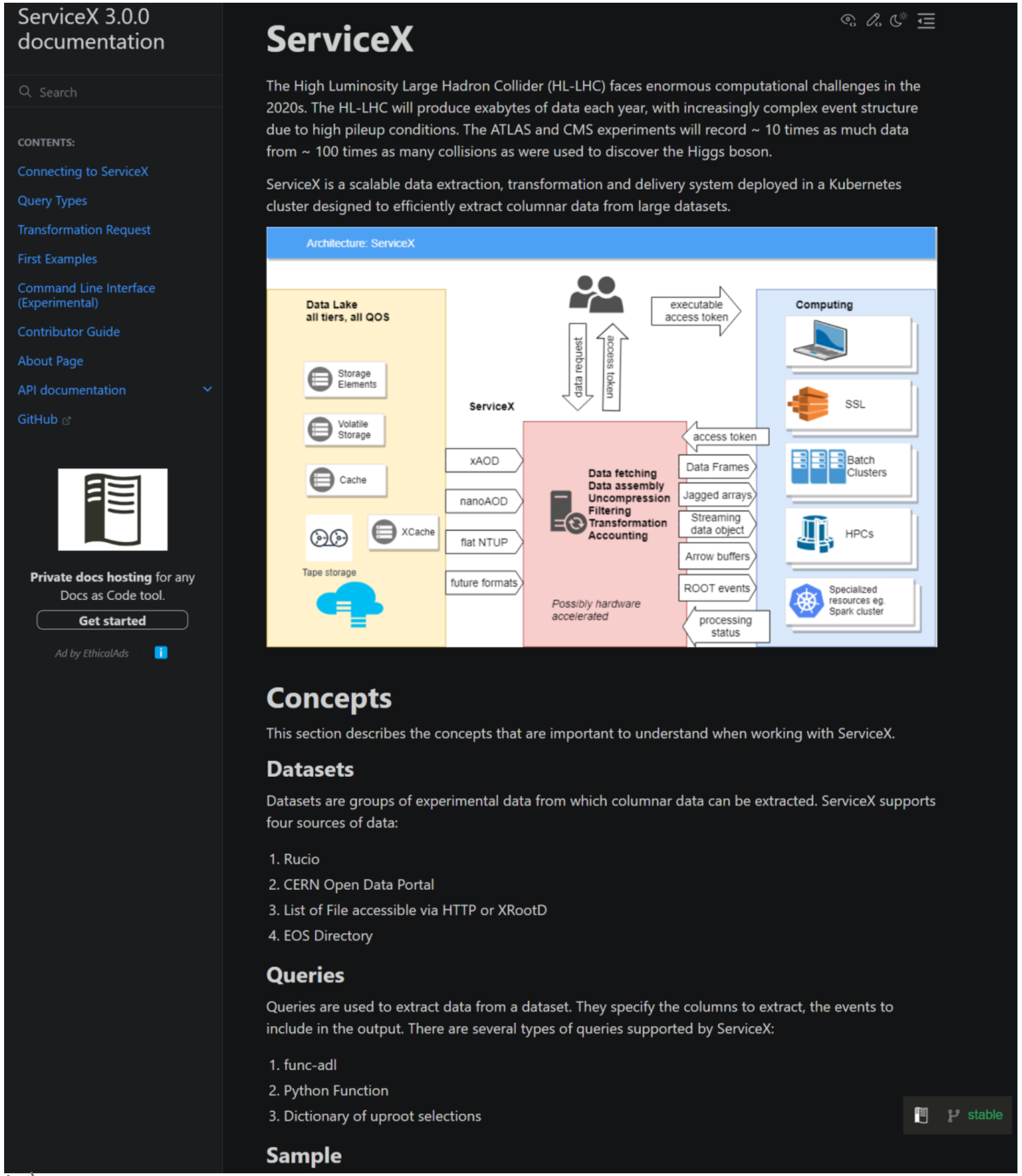
[ServiceX 3.0.0 documentation]

Missing something? Spot an error?

[Create an issue]

Or… the source for the documentation is in the repo

[Submit a MR]



ServiceX 3.0.0 documentation

Search

CONTENTS:

ServiceX

The High Luminosity Large Hadron Collider (HL-LHC) faces enormous computational challenges in the 2020s. The HL-LHC will produce exabytes of data each year, with increasingly complex event structure due to high pileup conditions. The ATLAS and CMS experiments will record ~ 10 times as much data from ~ 100 times as many collisions as were used to discover the Higgs boson.

ServiceX is a scalable data extraction, transformation and delivery system deployed in a Kubernetes cluster designed to efficiently extract columnar data from large datasets.

Concepts

This section describes the concepts that are important to understand when working with ServiceX.

Datasets

Datasets are groups of experimental data from which columnar data can be extracted. ServiceX supports four sources of data:

1. Rucio
2. CERN Open Data Portal
3. List of File accessible via HTTP or XRootD
4. EOS Directory

Queries

Queries are used to extract data from a dataset. They specify the columns to extract, the events to include in the output. There are several types of queries supported by ServiceX:

1. func-adl
2. Python Function
3. Dictionary of uproot selections

Sample

stable

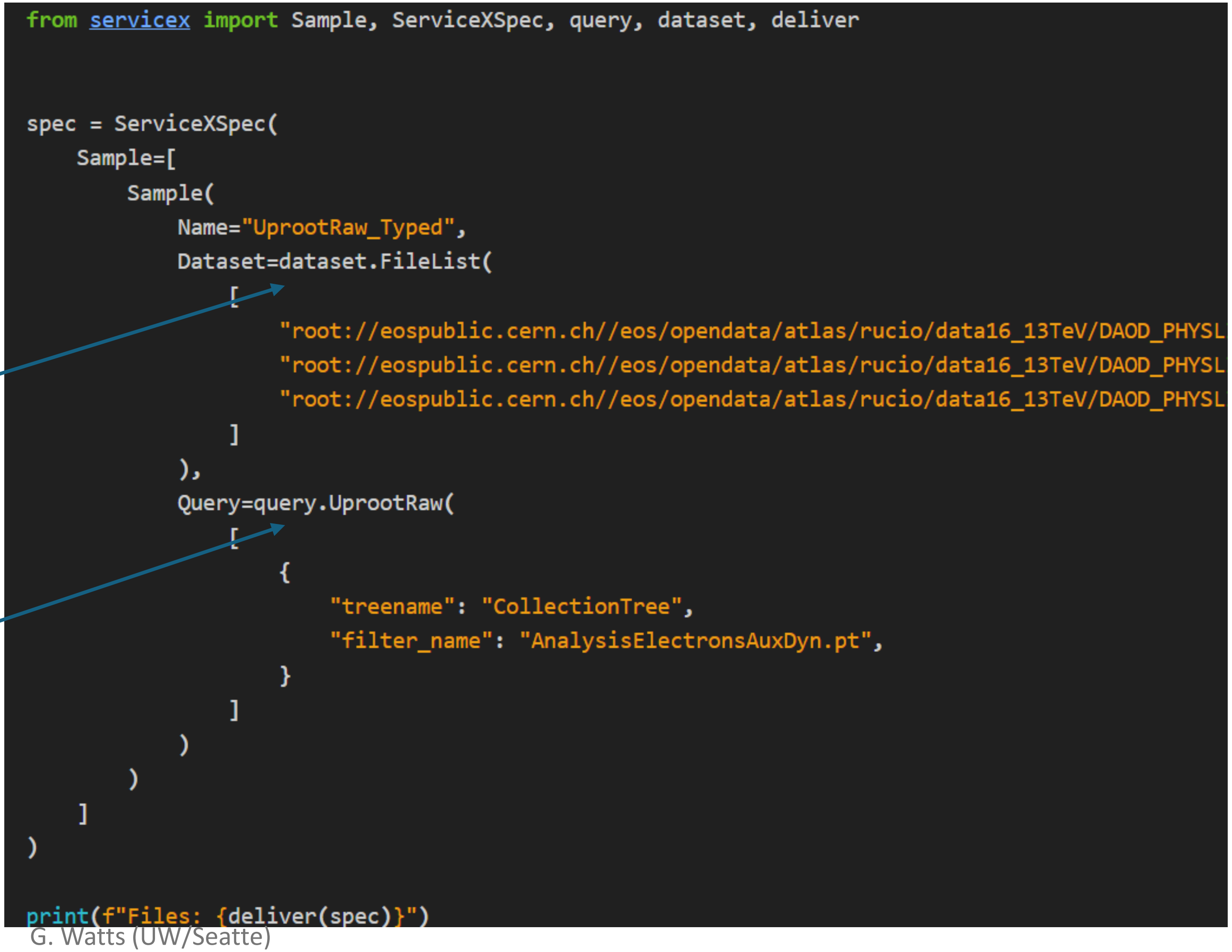
How does it work?

Data Source (FileList, Rucio, etc.)

- Where to get the data from

Query

- How to transform the data
- Raw, func_adl, etc.

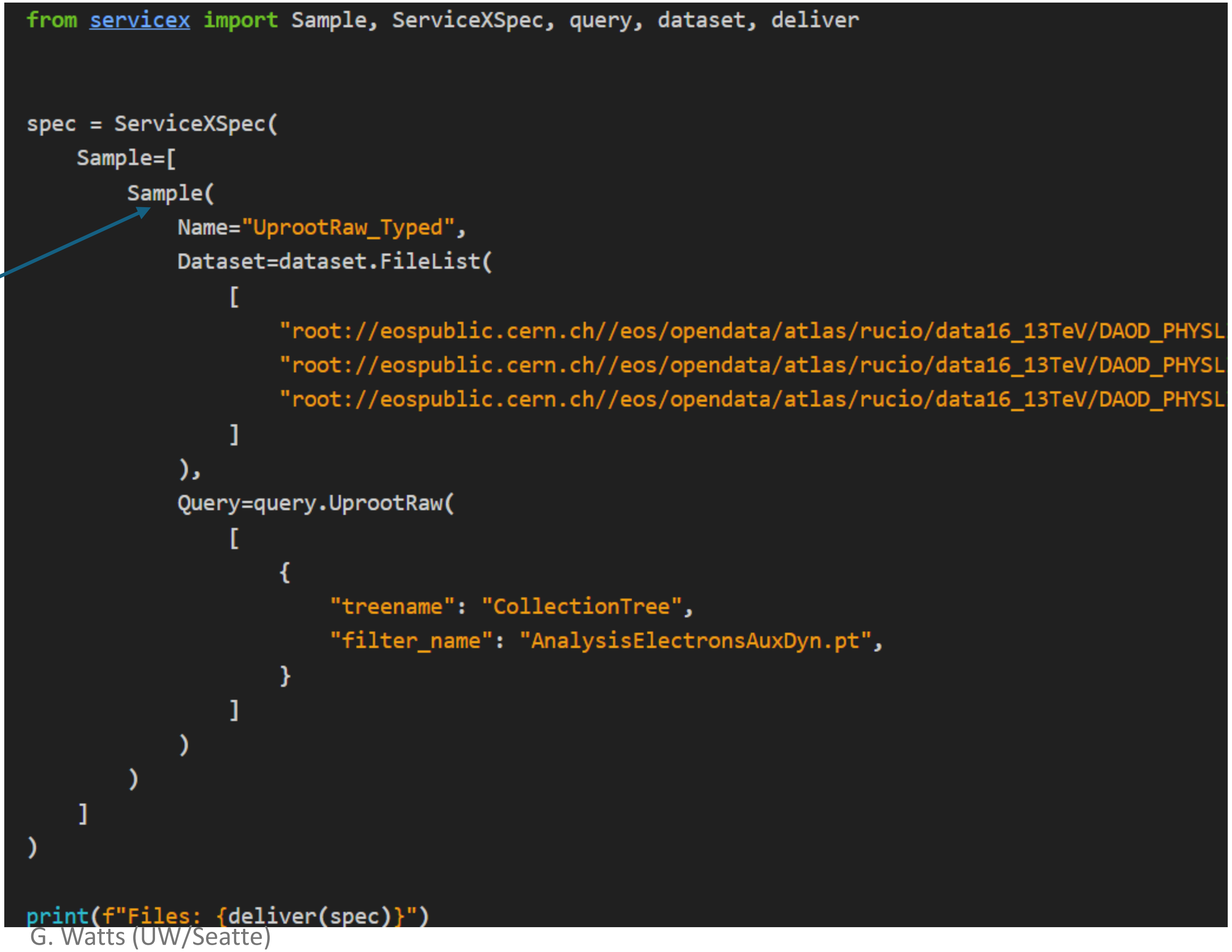


```
from servicex import Sample, ServiceXSpec, query, dataset, deliver


spec = ServiceXSpec(
    Sample=[
        Sample(
            Name="UprootRaw_Typed",
            Dataset=dataset.FileList(
                [
                    "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSL
                    "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSL
                    "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSL
                ]
            ),
            Query=query.UprootRaw(
                [
                    {
                        "treename": "CollectionTree",
                        "filter_name": "AnalysisElectronsAuxDyn.pt",
                    }
                ]
            )
        )
    ]
)


print(f"Files: {deliver(spec)}")
```

How does it work?

Sample

- Consists of the data set and query together
- Produces a set of files
- Labeled with the Name



```
from servicex import Sample, ServiceXSpec, query, dataset, deliver


spec = ServiceXSpec(
    Sample=[
        Sample(
            Name="UprootRaw_Typed",
            Dataset=dataset.FileList(
                [
                    "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSL
                    "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSL
                    "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSL
                ]
            ),
            Query=query.UprootRaw(
                [
                    {
                        "treename": "CollectionTree",
                        "filter_name": "AnalysisElectronsAuxDyn.pt",
                    }
                ]
            )
        )
    ]
)

print(f"Files: {deliver(spec)}")
```

How does it work?

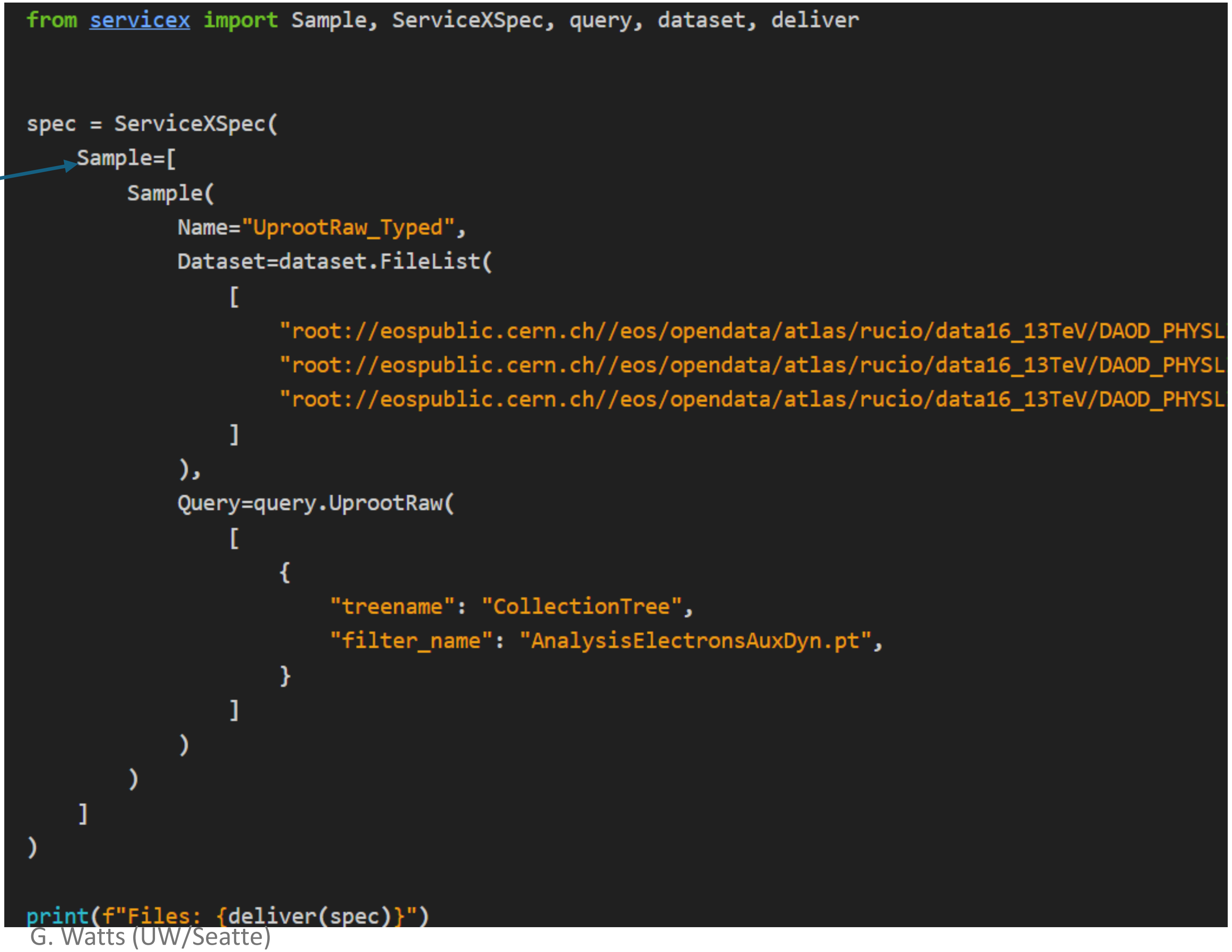
Samples

- You can submit 1 or 100 samples at a time
- Output will always be a dict of everything

All of these objects take lots of extra parameter that allow you to specify explicitly:

- Code generator
- Backend ServiceX location
- Transformer image, etc.

The servicex.yaml file is still required!!

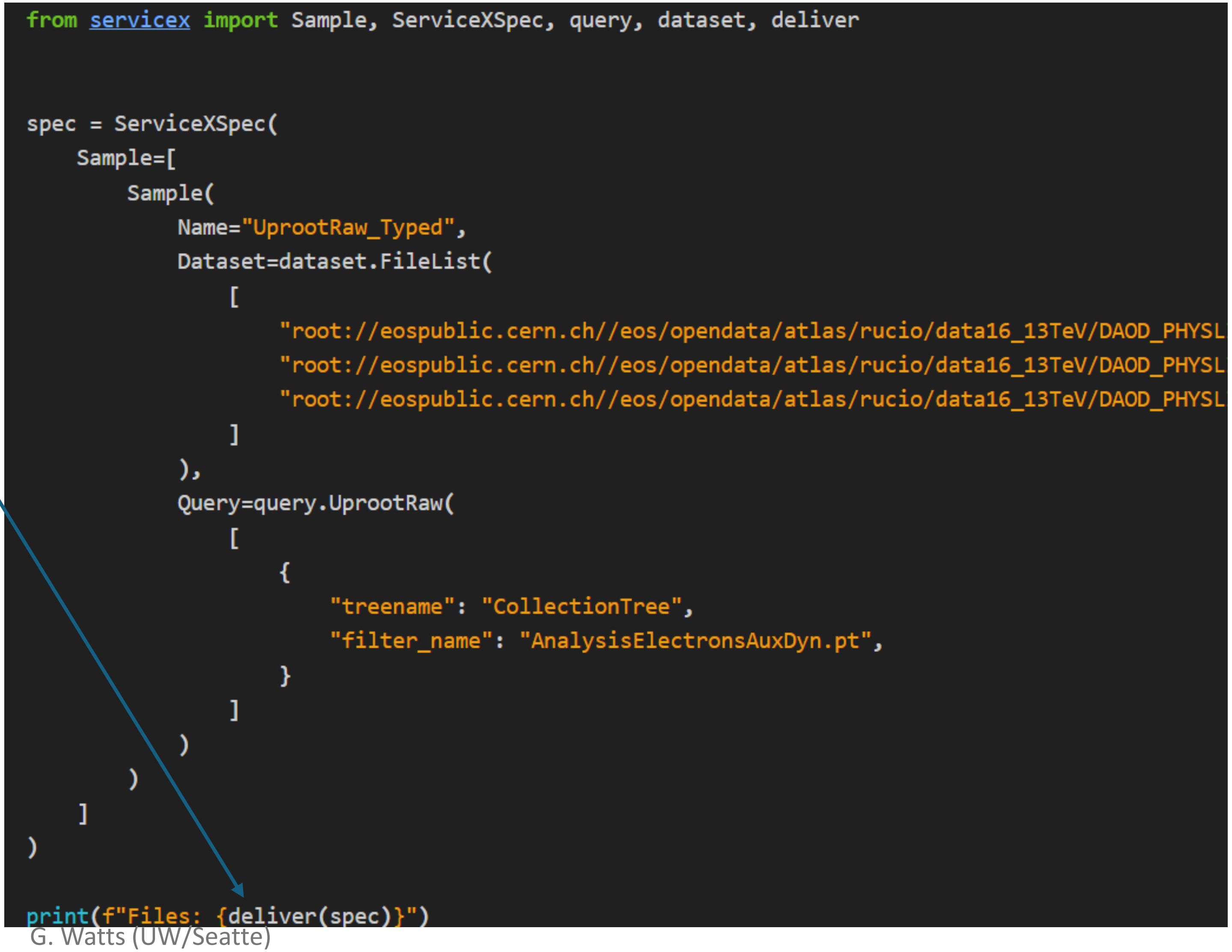


```
from servicex import Sample, ServiceXSpec, query, dataset, deliver


spec = ServiceXSpec(
    Sample=[
        Sample(
            Name="UprootRaw_Typed",
            Dataset=dataset.FileList(
                [
                    "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSL
                    "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSL
                    "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSL
                ]
            ),
            Query=query.UprootRaw(
                [
                    {
                        "treename": "CollectionTree",
                        "filter_name": "AnalysisElectronsAuxDyn.pt",
                    }
                ]
            )
        )
    ]
)

print(f"Files: {deliver(spec)}")
```

How does it work?

deliver changes the spec with the Samples into files.

- There is an async version of this coming soon!
- It will handle all interaction with the ServiceX backend via the WebAPI



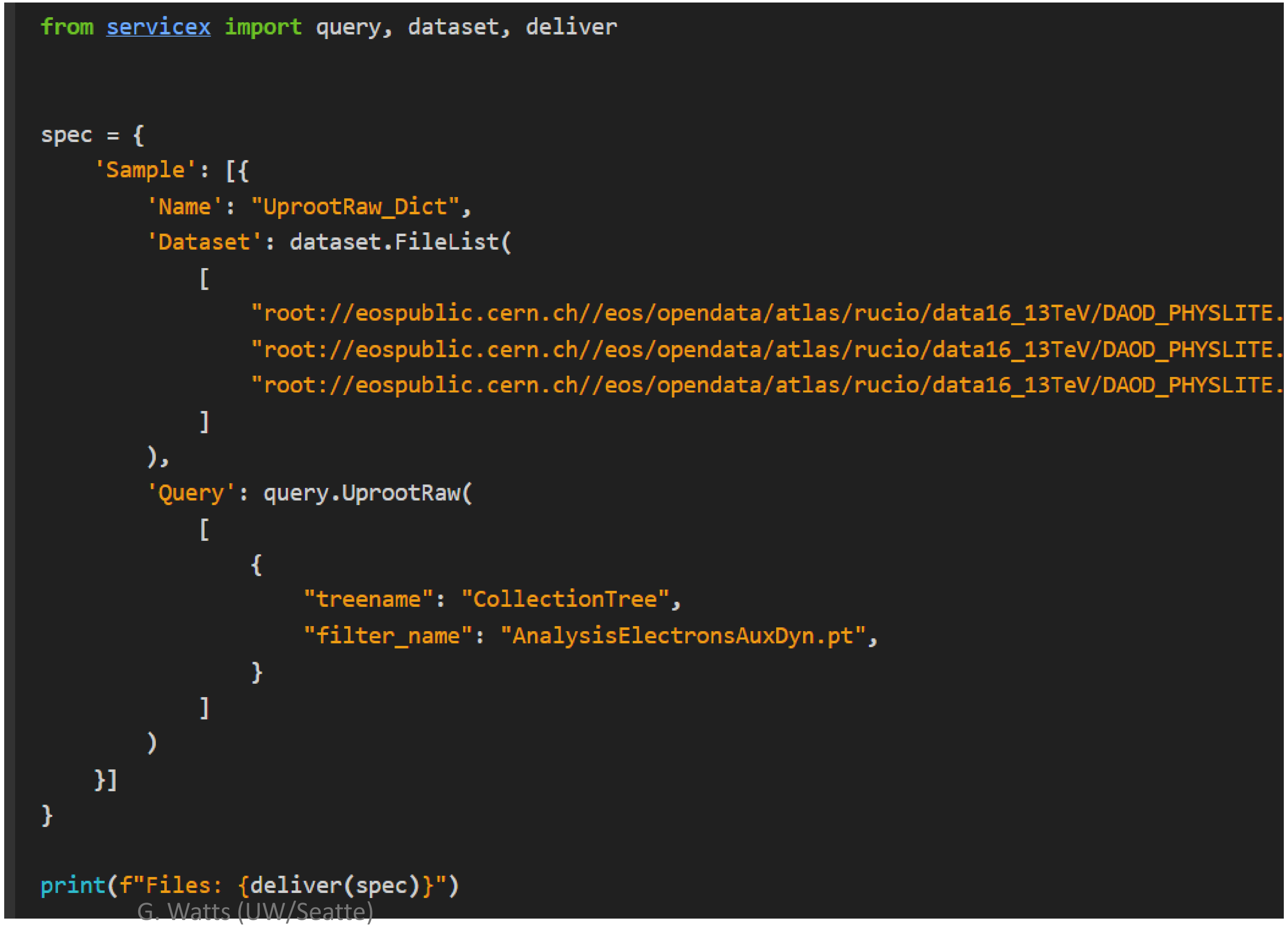
```
from servicex import Sample, ServiceXSpec, query, dataset, deliver


spec = ServiceXSpec(
    Sample=[
        Sample(
            Name="UprootRaw_Typed",
            Dataset=dataset.FileList(
                [
                    "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSL
                    "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSL
                    "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSL
                ]
            ),
            Query=query.UprootRaw(
                [
                    {
                        "treename": "CollectionTree",
                        "filter_name": "AnalysisElectronsAuxDyn.pt",
                    }
                ]
            )
        )
    ]
)

print(f"Files: {deliver(spec)}")
```

Using a Dictionary instead…

We expect:

- Dictionary is the easiest to use for quick oneoffs
- Typed classes will be used by libraries and frameworks
- The YAML is attractive because it can be checked into git directly!

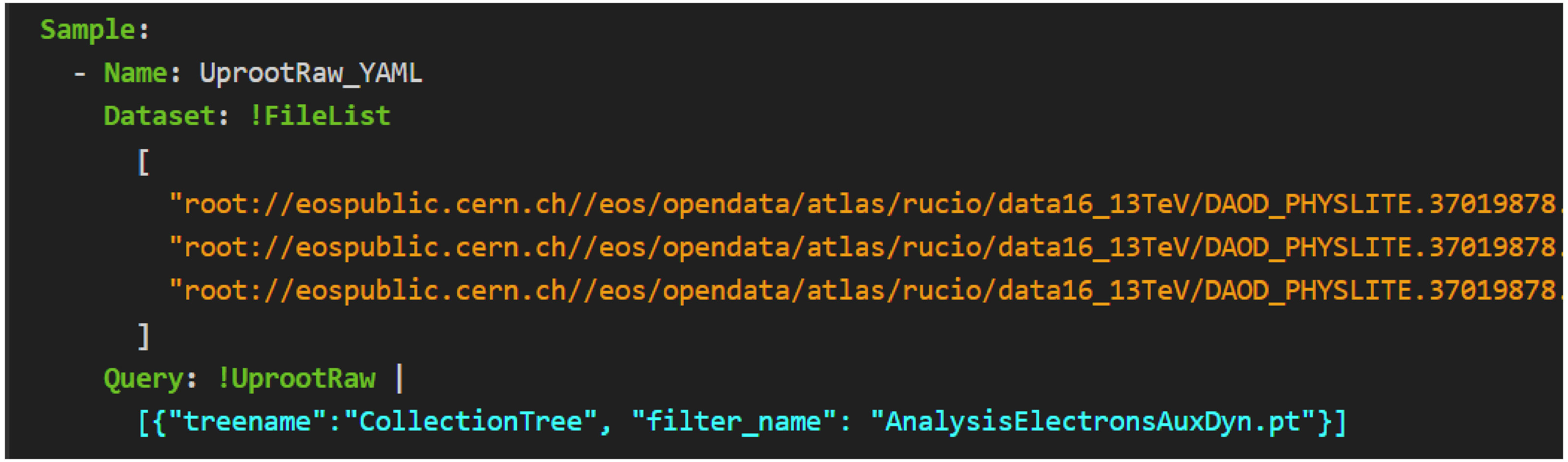
At its core, the frontend only uses the typed classes – everything is translated into that!



```
from servicex import query, dataset, deliver


spec = {
    'Sample': [{
        'Name': "UprootRaw_Dict",
        'Dataset': dataset.FileList(
            [
                "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSLITE.
                "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSLITE.
                "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSLITE.
            ]
        ),
        'Query': query.UprootRaw(
            [
                {
                    "treename": "CollectionTree",
                    "filter_name": "AnalysisElectronsAuxDyn.pt",
                }
            ]
        )
    }]
}

print(f"Files: {deliver(spec)}")
```

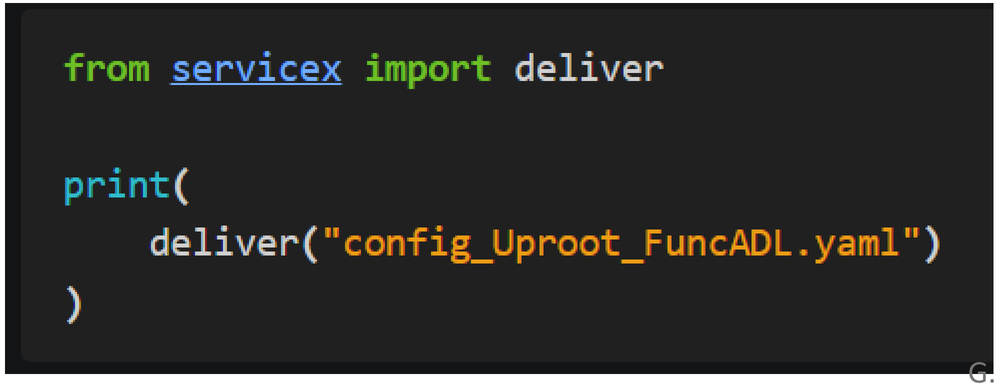
Using YAML

The YAML file itself:



```
Sample:
  - Name: UprootRaw_YAML
    Dataset: !FileList
      [
        "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSLITE.37019878.
        "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSLITE.37019878.
        "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSLITE.37019878.
      ]
    Query: !UprootRaw |
      [{"treename":"CollectionTree", "filter_name": "AnalysisElectronsAuxDyn.pt"}]
```

Use in SX:



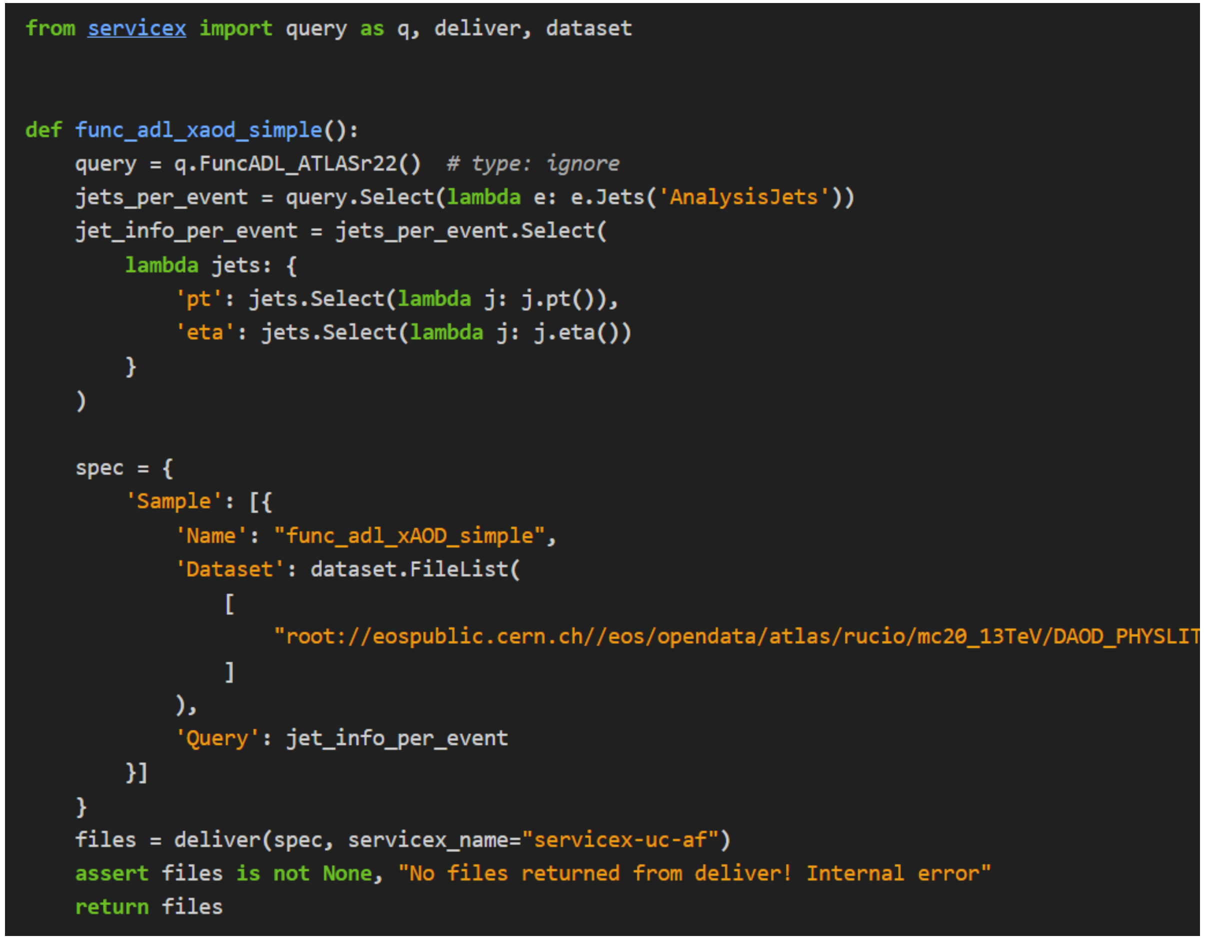
```
from servicex import deliver

print(
    deliver("config_Uproot_FuncADL.yaml")
)
```

There is not yet a full text representation for all query languages!

Other Queries: Func_ADL xAOD (Fully Typed)

Func_adl

ServiceX Spec and Delivery

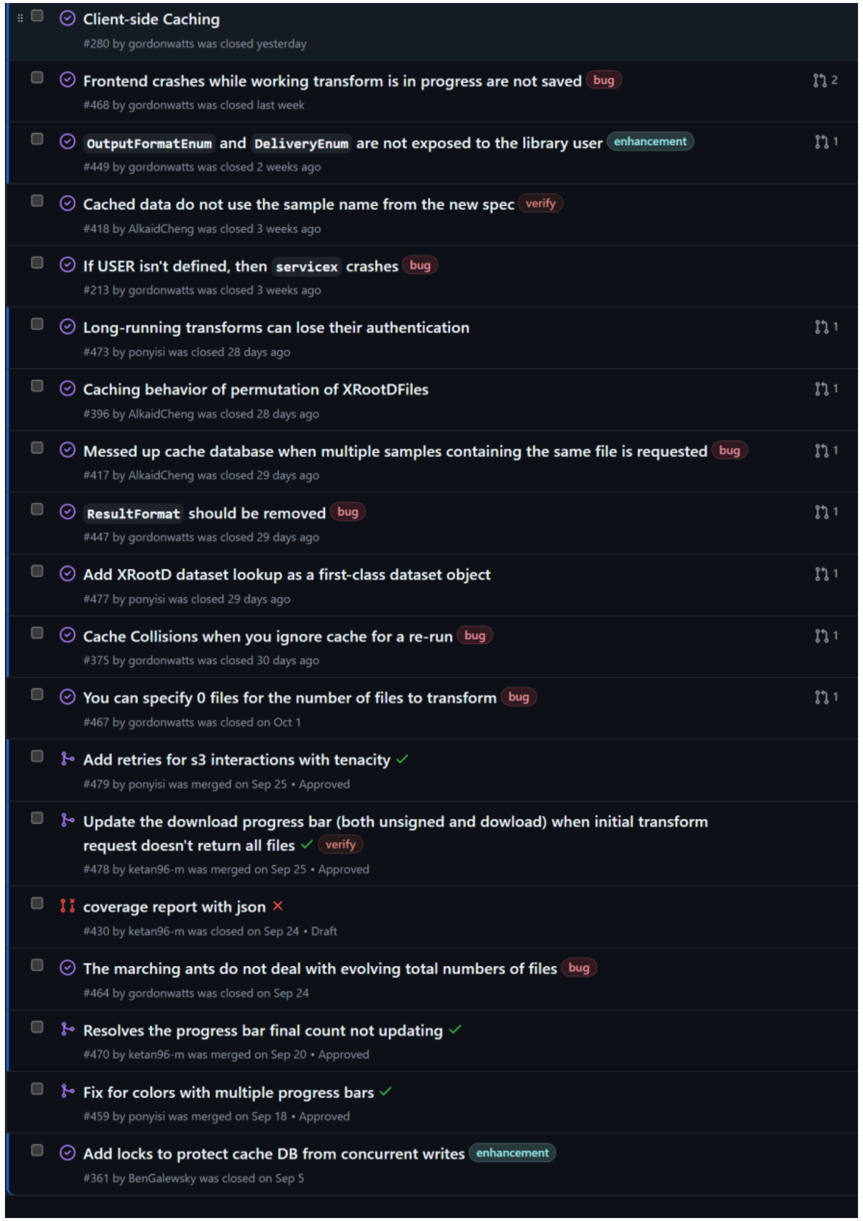


```python
from servicex import query as q, deliver, dataset


def func_adl_xaod_simple():
    query = q.FuncADL_ATLASr22()  # type: ignore
    jets_per_event = query.Select(lambda e: e.Jets('AnalysisJets'))
    jet_info_per_event = jets_per_event.Select(
        lambda jets: {
            'pt': jets.Select(lambda j: j.pt()),
            'eta': jets.Select(lambda j: j.eta())
        }
    )

    spec = {
        'Sample': [{
            'Name': "func_adl_xAOD_simple",
            'Dataset': dataset.FileList(
                [
                    "root://eospublic.cern.ch//eos/opendata/atlas/rucio/mc20_13TeV/DAOD_PHYSLIT
                ]
            ),
            'Query': jet_info_per_event
        }]
    }
    files = deliver(spec, servicex_name="servicex-uc-af")
    assert files is not None, "No files returned from deliver! Internal error"
    return files
```

ssl-hep/ServiceX_frontend: Client access library for ServiceX

Status

- The Analysis Grand Challenge and the IDAP 200 Gbps code has been converted to use 3.0
- Being used in the wild as well (been out about 1.5 months).
- However, there are some rough edges
  - 3.0.1's branch already has a significant number of updates
  - Will release "soon"
- We also have a backlog of new features where we are planning for 3.1



□ Client-side Caching
#280 by gordonwatts was closed yesterday

□ Frontend crashes while working transform is in progress are not saved `bug`
#468 by gordonwatts was closed last week

□ `OutputFormatEnum` and `DeliveryEnum` are not exposed to the library user `enhancement`
#449 by gordonwatts was closed 2 weeks ago

□ Cached data do not use the sample name from the new spec `verify`
#418 by AlkaidCheng was closed 3 weeks ago

□ If USER isn't defined, then `servicex` crashes `bug`
#213 by gordonwatts was closed 3 weeks ago

□ Long-running transforms can lose their authentication
#473 by ponyisi was closed 28 days ago

□ Caching behavior of permutation of XRootDFiles
#396 by AlkaidCheng was closed 28 days ago

□ Messed up cache database when multiple samples containing the same file is requested `bug`
#417 by AlkaidCheng was closed 29 days ago

□ `ResultFormat` should be removed `bug`
#447 by gordonwatts was closed 29 days ago

□ Add XRootD dataset lookup as a first-class dataset object
#477 by ponyisi was closed 29 days ago

□ Cache Collisions when you ignore cache for a re-run `bug`
#375 by gordonwatts was closed 30 days ago

□ You can specify 0 files for the number of files to transform `bug`
#467 by gordonwatts was closed on Oct 1

□ Add retries for s3 interactions with tenacity ✓
#479 by ponyisi was merged on Sep 25 • Approved

□ Update the download progress bar (both unsigned and dowload) when initial transform request doesn't return all files ✓ `verify`
#478 by ketan96-m was merged on Sep 25 • Approved

□ coverage report with json ✗
#430 by ketan96-m was closed on Sep 24 • Draft

□ The marching ants do not deal with evolving total numbers of files `bug`
#464 by gordonwatts was closed on Sep 24

□ Resolves the progress bar final count not updating ✓
#470 by ketan96-m was merged on Sep 20 • Approved

□ Fix for colors with multiple progress bars ✓
#459 by ponyisi was merged on Sep 18 • Approved

□ Add locks to protect cache DB from concurrent writes `enhancement`
#361 by BenGalewsky was closed on Sep 5

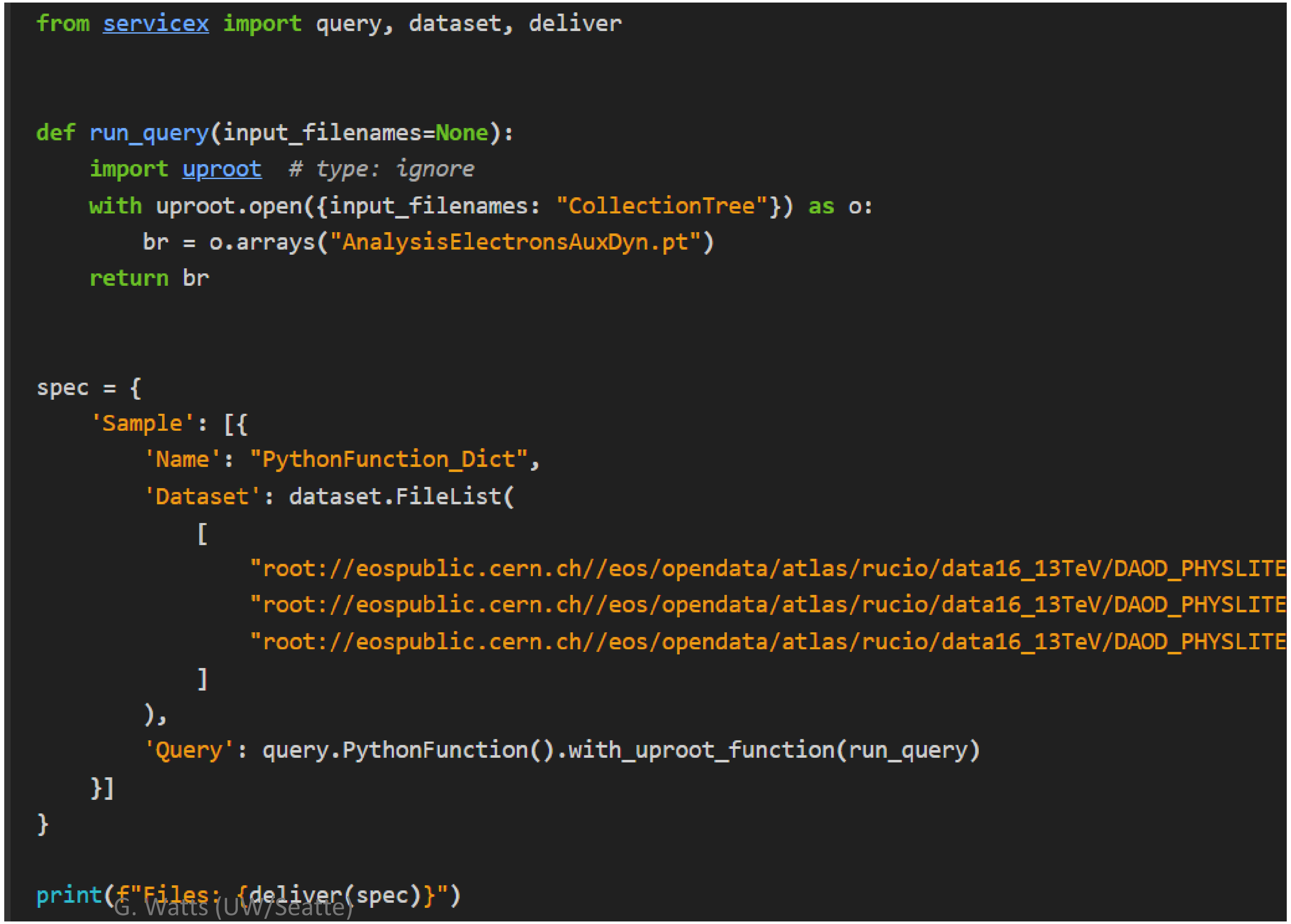
Backup

Full Python Function Transformer!!

Python Function

This allows one to extract non-ntuple-like objects from the file (e.g. cutflow, etc.)

ServiceX Spec and Delivery



```python
from servicex import query, dataset, deliver


def run_query(input_filenames=None):
    import uproot  # type: ignore
    with uproot.open({input_filenames: "CollectionTree"}) as o:
        br = o.arrays("AnalysisElectronsAuxDyn.pt")
    return br


spec = {
    'Sample': [{
        'Name': "PythonFunction_Dict",
        'Dataset': dataset.FileList(
            [
                "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSLITE
                "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSLITE
                "root://eospublic.cern.ch//eos/opendata/atlas/rucio/data16_13TeV/DAOD_PHYSLITE
            ]
        ),
        'Query': query.PythonFunction().with_uproot_function(run_query)
    }]
}

print(f"Files: {deliver(spec)}")
```