EPJ Web of Conferences **251**, 02053 (2021) https://doi.org/10.1051/epjconf/202125102053
CHEP 2021

Towards Real-World Applications of ServiceX, an Analysis Data Transformation System

K. Choi[1,*], *A.* Eckart[2], *B.* Galewsky[3], *R.* Gardner[2], *M.* Neubauer[3], *P.* Onyisi[1], *M.* Proffitt[4], *I.* Vukotic[2], and *G.* Watts[4]

[1]University of Texas at Austin
[2]The University of Chicago
[3]University of Illinois at Urbana-Champaign
[4]University of Washington

Abstract. One of the biggest challenges in the High-Luminosity LHC (HL-LHC) era will be the significantly increased data size to be recorded and analyzed from the collisions at the ATLAS and CMS experiments. ServiceX is a software R&D project in the area of Data Organization, Management and Access of the IRIS- HEP to investigate new computational models for the HL-LHC era. ServiceX is an experiment-agnostic service to enable on-demand data delivery specifically tailored for nearly-interactive vectorized analyses. It is capable of retrieving data from grid sites, on-the-fly data transformation, and delivering user-selected data in a variety of different formats. New features will be presented that make the service ready for public use. An ongoing effort to integrate ServiceX with a popular statistical analysis framework in ATLAS will be described with an emphasis of a practical implementation of ServiceX into the physics analysis pipeline.

1 Introduction

ServiceX is a scalable HEP event data location, extraction, filtering, and transformation system that has been developed as part of the Institute for Research and Innovation in Software for High Energy Physics (IRIS-HEP). It runs on any Kubernetes cluster and can be offered as a public service or hosted on an institution's private cluster.

ServiceX accepts requests via a REST interface. The requests include a dataset identifier (DID) that resolves to a number of input data files along with a columnar event data selection statement expressed in an elemental expression language called Query AST Language Expressions (Qastle) [1]. The service relies on Rucio [2] to lookup file replicas for the requested DID. It will attempt to select the file replicas that are most efficiently accessible by the host Kubernetes cluster. The Qastle syntax is designed to be translated into code that describes operations on input data, with the transformer providing the data handling libraries. There is currently support for C++ code that is run in a transformer based on the ATLAS Event-Loop framework. Additionally, there is a python code generator that produces a script to drive the python Uproot [3] library. It is suited for reading flat ntuples such as CMS NanoAOD [4] files and analysis group generated files. The results from a ServiceX transformation are either flat

*e-mail: kyungeonchoi@utexas.edu

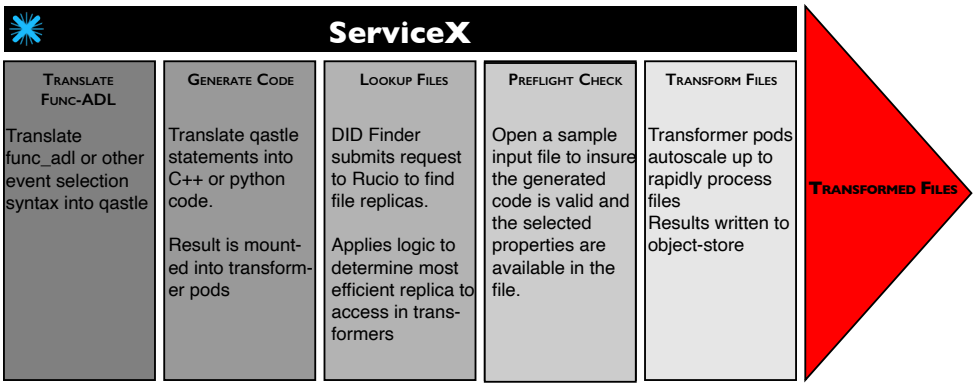

Figure 1. End-to-end workflow of a ServiceX transformation run. The analysis description language, func-adl [5], or ROOT TCut [6] syntax are supported for a transformation query.

ROOT files or Parquet columnar data files persisted to an object store which can be accessed via HTTP protocols or with the Amazon S3 API. Figure 1 shows how each of these steps is assembled to produce the desired set of transformed files.

This paper introduces the latest developments in ServiceX that allow the system to be readily implemented into real-world applications. Analysis pipelines that employ ServiceX for the currently available transformers are outlined. In this paper, an analysis pipeline refers to a chain from the reconstructed object to the final result. The primary goal of this paper follows to establish a practical implementation of ServiceX into the analysis pipeline that can be utilized in physics analysis.

2 New features in ServiceX

ServiceX was first presented at CHEP 2019 [7]. Since that conference, the system has evolved to make it useful as a production system to solve real world problems. The main enhancements were:

2.1 Public access

While ServiceX can be deployed in an experiment group's private Kubernetes cluster, users will still wish to connect to it from remote locations. As soon as the service is exposed to the internet, careful consideration must be given to securing it. In version 1.0, ServiceX uses Globus Auth [8] to authenticate users. Administrators are notified of user signups in a private Slack channel and can approve new accounts from there. Approved users are given an API token for making requests to the ServiceX deployment.

2.2 Auto-scaling of Transformer Pods

Initial versions of ServiceX required the user to specify the number of workers to launch to process the transformation job. This was simple to implement, but potentially wasteful in its use of resources, since CPUs could be sitting around idly. The service now makes use of Kubernetes auto-scaling capabilities to only launch new pods if files becoming available from Rucio exceed the existing set.

2.3 Support for Reading ATLAS xAOD Files

Reading of flat ntuples using Uproot libraries and Awkward arrays lends itself quite easily to the columnar style of analysis advanced by ServiceX. The ATLAS xAOD [9] files are another matter all together as xAOD files use custom ROOT objects and require a C++ framework to fully utilize. Research into the analysis description languages, func-adl, provided ServiceX with the ability to translate high level event selection queries into C++ code that is executed by an experiment approved framework inside the transformer pods.

2.4 Support for High-Level Expressions

ServiceX uses an elemental LISP-inspired expression language called Qastle to represent event selection and transformation request. It is not intended for end-users to author selections in this representation. Instead, it is hoped that researchers of analysis description languages will create transpilers to generate Qastle queries from a higher level language. The ServiceX backend will immediately allow them experiment with their language using the full scale resources of the server. Initial work was done using the func-adl language from the Watts lab. As described in more detail in section 4.2, it is equally important to work with popular event selection languages to encourage analysts to move their research over to this new environment with minimal disruption.

3 ServiceX in analysis pipeline

Figure 2 shows a schema of the current ATLAS analysis pipeline starting from the AOD (Analysis Object Data) to the final analysis formats in black lines. Many individual and group-based derivations, which reduce the size of AOD by removing unnecessary information, exist and even smaller ROOT ntuples for the final analysis. ServiceX, on the other hand, can access directly to the upstream of the analysis pipeline as it supports different types of transformers for different input file formats.

The transformers available today are developed for those file formats that are primary in the analysis pipeline at present: xAOD transformer for the ATLAS xAOD or Derived AOD format (red lines), and Uproot transformer for flat ROOT ntuples (orange line). The

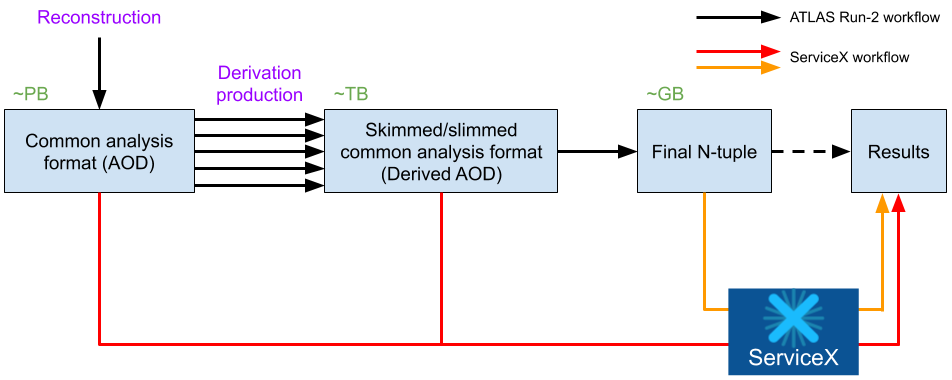

Figure 2. A schema of ATLAS derivation production and subsequent analysis pipeline (black) and ServiceX pipelines (red/orange).

latter is also compatible with the CMS NanoAOD format, which also features a flat ROOT TTree structure. A transformer for the CMS MiniAOD format, which also requires dedicated libraries similar to the ATLAS xAOD/DAOD format, is currently being developed.

The strong point of ServiceX is the flexibility of the transformer. The architecture implemented in ServiceX relies on a number of containerized microservices which include a transformer. Thus ServiceX is adaptable to future data formats with new transformers, and it is also feasible to have dedicated transformers for specific purposes.

4 ServiceX for statistical analysis framework

Given that ServiceX is relatively new to the community and the public release of the service became available recently, there are not many practical implementations of ServiceX into the analysis pipeline up to the present time. It is also because of the fact that most analyzers are accustomed to the traditional analysis pipeline based on ROOT and grid jobs. Therefore, it is helpful to lower the barrier to allow them to experience the new analysis ecosystem in Python. We have been developed a tool that can be implemented into a practical physics analysis with a minimum effort from analyzers.

4.1 TRExFitter

TRExFitter [10] is a framework used by many ATLAS physics analyses for statistical inference via profile likelihood fits. It is designed to handle everything that analysers need for the statistical part of their analysis. It produces RooFit workspaces, perform fits on them, and interfaces with RooStats macros for limit and significance. It also generates publication-level pre-fit and post-fit plots and tables. Many more additional features are also supported to understand the fit behavior.

TRExFitter takes ROOT ntuples or histograms as input format, and a configuration file to steer the framework. A configuration file includes high-level physics choices: specification of signal/validation/control regions (the channels in HistFactory schema), observables to be used for the statistical analysis, Monte Carlo samples for signal and background and data samples, sources of systematic uncertainties, details of fit model, parameter of interest, and other general settings.

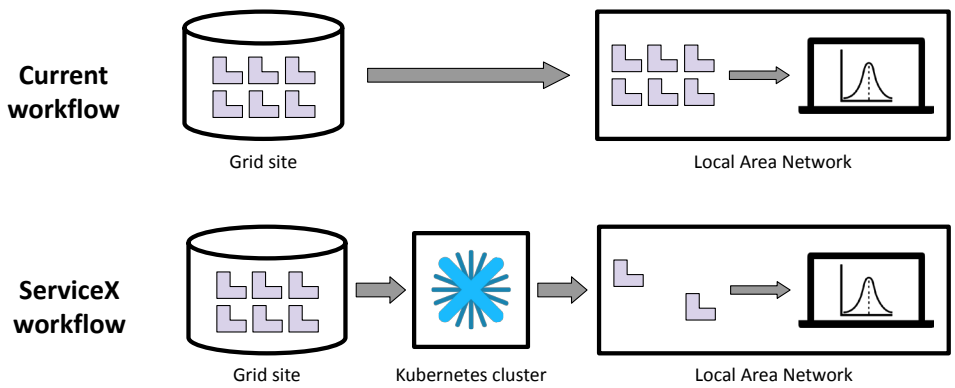

Figure 3. Current workflow of TRExFitter (top) and alternative ServiceX workflow (bottom) to generate histograms from ROOT ntuples that are produced on the grid.

TRExFitter provides the feature to generate histograms for statistical analysis from ROOT ntuples based on the provided configuration file. Hence, it is a typical workflow to download whole ROOT ntuples from the grid to a local cluster or directly accessible machines to generate histograms as shown in the top of Figure 3. On the other hand, ServiceX can deliver only necessary branches or columns with event filtering as shown in the bottom of Figure 3. The advantages of the ServiceX workflow are as follows:

- Local storage: It takes up less storage space as ServiceX delivers minimal information to generate histograms.
- Download time: Data transferred over WAN is smaller for the ServiceX workflow. The network speed between a grid site and Kubernetes cluster is sufficiently fast as they are usually co-located.

4.2 TCut translator for ServiceX

ServiceX utilizes func-adl, a python-based declarative analysis description language, to filter events and request branches from the input data file. It is an intuitive language to extract data directly from ROOT ntuples, but TRExFitter relies on the ROOT `TTree::Draw` method, which uses TCut syntax for TTree selections. Since TCut syntax is not directly readable by the func-adl, a python package for TCut to func-adl translation is developed. The package further converts func-adl to Qastle language, which is the language that ServiceX transformers understand.

The package supports arithmetic operators (`+, -, *, /`), logical operators (`!, &&, ||`), and relational and comparison operators (`==, !=, >, <, >=, <=`). Listing 1 shows an example of translating a ROOT-based query into a ServiceX query. The first argument is the name of the TTree object in an input file. The second is the list of selected branches for delivery. The last argument is the TCut object for TTree selection. Only events that pass the selection will be delivered for the requested branches. Thus, the second argument effectively removes branches from the input ROOT tree, and the third drops events. The package, `tcut-to-qastle` [11], is published at PyPI for a convenient access.

```
import tcut_to_qastle as tq

# Get ServiceX query
query = tq.translate("nominal", "A,B,D", "(A && !B) || (C > 0.1)")
```

Listing 1. An example translation of ROOT-based query into ServiceX query

4.3 `servicex-for-trexfitter`

The `servicex-for-trexfitter` is a python package, which has been developed to provide seamless integration of ServiceX into the TRExFitter framework. It makes use of the building blocks that are described above: Uproot ServiceX to read input ROOT ntuples from the grid and perform transformations; ServiceX frontend library to access ServiceX backend and manage ServiceX delivery requests; the `tcut-to-qastle` package to translate ROOT-based query into the ServiceX query.

Each of the following steps runs within the `servicex-for-trexfitter` package: prepares ServiceX requests by analyzing a TRExFitter configuration file to deliver a minimum amount of data from the grid, makes ServiceX requests simultaneously, downloads output of finished transformations asynchronously from the object store of Kubernetes cluster, and converts downloaded output files into a ROOT file for each sample.

It takes a TRExFitter configuration file, which has an identical structure with the traditional ROOT ntuple input. The only addition is a new field for grid dataset ID for each sample since ServiceX reads input ROOT ntuples from the grid. The caching feature of the ServiceX frontend library allows only modified or added part of the TRExFitter configuration creates new ServiceX requests.

4.4 Example

The following prerequisites are needed to run the `servicex-for-trexfitter` package. It is written for Python version equal to or higher than 3.6. Access to an Uproot ServiceX endpoint is also required. Any running Uproot ServiceX instance should work, or access to the centrally-managed ServiceX instance can be granted as described in the ServiceX documentation [12]. To convert the outputs from ServiceX into ROOT TTree, PyROOT [13] has to be installed. Lastly, the input ROOT ntuples need to be organized in a way that each sample in the TRExFitter configuration file corresponds to a single Rucio dataset ID.

Listing 2 shows an example of how to use the `servicex-for-trexfitter`. An instance can be created with an argument of TRExFitter configuration file. Data transformation and delivery status can be interactively monitored just after the method `get_ntuples()` is called, and the path to the slimmed/skimmed output ROOT ntuples will be printed once the delivery is completed.

```
from servicex_for_trexfitter import ServiceXTRExFitter
sx_trex = ServiceXTRExFitter("example.config")
sx_trex.get_ntuples()
```

Listing 2. An example of running `servicex-for-trexfitter` to get slimmed/skimmed ROOT ntuples using ServiceX

4.5 Benchmark results

The performance of `servicex-for-trexfitter` is measured and compared with the current workflow using the same TRExFitter configuration file. A practical TRExFitter configuration file which contains 17 Samples and 34 Systematics is used for the benchmark. The total size of ROOT ntuples is 650 Gigabytes stored at the MidWest Tier-2 Center. The benchmark for ServiceX workflow utilizes the Uproot ServiceX deployed at the University of Chicago SSL-River Kubernetes cluster.

The results are shown in Table 1. The current workflow takes up the same amount of local disk space with the total size of ROOT ntuples on the grid, whereas the workflow using `servicex-for-trexfitter` takes up a lot smaller disk space as it delivers only information that is needed to generate histograms defined in the TRExFitter configuration

	Current workflow	`servicex-for-trexfitter`
Local disk space	650 GB	0.4 GB
Download ROOT ntuples	>hours	7 mins
Process ROOT ntuples	138 mins	2.5 mins

Table 1. Benchmark results from the current TRExFitter workflow and the workflow using `servicex-for-trexfitter`.

file. The wall time to download ROOT ntuples for the subsequent step, generating histograms from downloaded ROOT ntuples, shows a much shorter time for the workflow using `servicex-for-trexfitter`. This is due to the fact that ServiceX scales the workers to parallelize transformations and delivers only a subset of ROOT ntuples over WAN. In addition, the time spent on processing downloaded ROOT ntuples is significantly shorter for the workflow using `servicex-for-trexfitter` as it needs to process only 0.4 GB than 650 GB.

5 Conclusion and outlook

The recent developments in ServiceX put the service in a state of a production system that allows practical implementations into a traditional analysis workflow. The authentication system which enables public access over the internet is particularly beneficial as it opens the door to more users. The auto-scaling of transformer pods makes the system more robust, and the new C++ transformer for the ATLAS xAOD improves the performance significantly. The support of TCut expressions to the func-adl language lowers the threshold to try the service.

The real-world application which employs the new features of ServiceX has been developed to provide a novel data delivery method to the popular statistical analysis framework in ATLAS. Accessing input ROOT ntuples directly from the grid saves significant amount of local disk space. Scaling up the transformer pods in a Kubernetes cluster can remarkably reduce a turnaround time by extracting only necessary information from input ROOT ntuples. The primary goal of the `servicex-for-trexfitter` package is also fulfilled by requiring a minimal addition to the traditional approach.

This paper describes the implementation of ServiceX to interface with the existing ROOT-based statistical analysis framework. The HEP tools in Python are rapidly evolving, and ServiceX can nicely align with those tools to achieve a complete analysis workflow within the Python ecosystem.

References

[1] *Qastle*, https://github.com/iris-hep/qastle (2021), accessed: 2021-02-23

[2] *Rucio's documentation*, https://rucio.cern.ch/documentation/ (2021), accessed: 2021-05-11

[3] *Uproot documentation*, https://uproot.readthedocs.io/en/latest/ (2021), accessed: 2021-05-11

[4] K. Ehatäht for the CMS collaboration, EPJ Web of Conferences 245, 06002 (2020)

[5] *Functional ADL*, https://iris-hep.org/projects/func-adl.html (2021), accessed: 2021-02-22

[6] *ROOT TCut Class Reference*, https://root.cern.ch/doc/master/classTCut.html (2021), accessed: 2021-05-12

[7] B. Galewsky, R. Gardner, L. Gray, M. Neubauer, J. Pivarski, M. Proffitt, I. Vukotic, G. Watts, and M. Weinberg, EPJ Web of Conferences 245, 04043 (2020)

[8] S. Tuecke et al., "Globus auth: A research identity and access management platform," 2016 IEEE 12th International Conference on e-Science (e-Science), Baltimore, MD, USA, 2016, pp. 203-212, doi: 10.1109/eScience.2016.7870901

[9] A Buckley et al 2015 J. Phys.: Conf. Ser. 664 072045

[10] *TRExFitter documentation*, https://trexfitter-docs.web.cern.ch/trexfitter-docs/ (2021), accessed: 2021-02-19

[11] *TCutToQastleWrapper*, https://github.com/ssl-hep/TCutToQastleWrapper (2021), accessed: 2021-02-23

[12] *ServiceX documentation*, https://servicex.readthedocs.io/en/latest/ (2021), accessed: 2021-02-24

[13] *Python interface: PyROOT*, https://root.cern/manual/python/ (2021), accessed: 2021-02-24

[14] Vohra D. (2016) Apache Parquet. In: Practical Hadoop Ecosystem. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-2199-0_8