The Journal of Open Source Software

FaaSr: R package for Function-as-a-Service Cloud Computing

Sungjae Park [1*], **Yun-Jung Ku** [1*], **Nan Mu** [1*], **Vahid Daneshmand** [1*], **R. Quinn Thomas** [3*], **Cayelan C. Carey** [2*], **and Renato J. Figueiredo** [1*]

1 Department of Electrical and Computer Engineering, University of Florida, FL, USA **2** Department of Biological Sciences, Virginia Tech, VA, USA **3** Department of Forest Resources and Environmental Conservation, Virginia Tech, VA, USA * These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

The FaaSr software makes it easy for scientists to execute computational workflows developed natively using the R programming language in Function-as-a-Service (FaaS) serverless cloud infrastructures and using S3 cloud object storage(Amazon, 2024b; MinIO, 2024). A key objective of the software is to reduce barriers to entry to cloud computing for scientists in domains such as environmental sciences, where R is widely used(Lai et al., 2019). To this end, FaaSr is designed to hide complexities associated with using cloud Application Programming Interfaces (APIs) for different FaaS and S3 providers, and exposes to the end user a set of simple function interfaces to: 1) register and invoke FaaS functions, 2) compose them to create workflow execution graphs, and 3) access cloud storage at run time. The software supports encapsulation of execution environments in Docker images that can be deployed reproducibly across multiple providers: AWS Lambda(Amazon, 2024a), GitHub Actions(Github, 2024), and OpenWhisk(Apache, 2024), where users are able to leverage a baseline image with the widely-used Rocker/Tidyverse runtime, as well as customize their execution environment if needed. FaaSr is available as a CRAN package to facilitate its installation in R environments.

Statement of need

Scientific research increasingly requires extensive data and computing resources to execute complex workflows that are increasingly event-driven. Cloud computing has emerged as a scalable solution to meet these demands. However, traditional Infrastructure-as-a-Service (IaaS) models often prove to be costly and require server management, presenting challenges to many scientists. In particular, this presents barriers to entry for small to medium teams and in domains where users are not accustomed to cloud server deployment and management and/or cluster and high-performance computing environments. Function-as-a-Service serverless computing has the potential to address these concerns by providing a cost-effective alternative where users are not burdened with server management and can simply focus on writing application logic instead. Nevertheless, today's FaaS platforms still present barriers to entry with respect to usability for scientists, particularly those who heavily rely on the R programming language, because: 1) R is not widely supported by commercial and open-source FaaS platforms as a runtime target, and 2) different FaaS providers use different, non-compatible APIs. While there are systems that enable Python applications to be used in FaaS (such as NumpyWren(Shankar et al., 2018), PyWren(Jonas et al., 2017), and FuncX(Chard et al., 2020)), there is a growing need to support R-native applications. This need is addressed by FaaSr through the use of containers that encapsulate an R-based runtime environment supporting the execution of user-provided functions. In addition, while existing systems are tailored to a specific FaaS

platform, there is a need to support cross-platform execution to avoid vendor lock-in. This need is addressed by FaaSr by hiding provider-specific APIs behind function interfaces that work consistently across multiple serverless providers, including AWS Lambda, GitHub Actions, and OpenWhisk. Furthermore, there is a need to support complex scientific workflows to express the order of execution of functions, as well as parallelism. This need is addressed by FaaSr in a way that remains serverless in nature and does not require dedicated/managed workflow engines.

Design

The FaaSr package consists of server-side and client-side functions. The server-side functions are executed when an action is deployed by a FaaS platform. The FaaSr server-side interfaces perform various operations, on behalf of the user, in stubs that are automatically inserted before and after user function invocation. These include: 1) reading the JSON workflow configuration file payload, 2) validating it against the FaaSr schema, 3) checking for reachability of S3 storage, 4) executing the user-provided function, 5) triggering the invocation of downstream function(s) in the workflow, and 6) storing logs. These functions are invoked at runtime by the containers deployed in an event-driven fashion by FaaS providers; the entry point of the container invokes the FaaSr package. Furthermore, some of the server-side interfaces are exposed to users, and implement functions to: 1) use S3 storage to download (get) and upload (put) full objects as files, 2) use Apache Arrow over S3 to efficiently access objects stored in columnar format using Apache Parquet, and 3) store logs.

The client-side functions are executed iteratively by a user from their desktop environment (e.g. RStudio). The primary client-side functions exposed to users allow them to: 1) register workflows with FaaS providers, 2) invoke workflows as either a one-off or to set timer schedules for triggering workflows at pre-specified intervals, and 3) copy execution logs from S3 storage to their desktop. The client-side interfaces build on the `faasr` function, which creates an object instance in memory in the R session for the user, and which can then be subsequently used to register and invoke functions. This function takes as arguments the name of a JSON-formatted(Pezoa et al., 2016) workflow configuration file, and (optionally) the name of a file storing FaaS/S3 cloud provider credentials. The JSON schema for this file is also stored in the FaaSr-Package repository.

FaaSr supports the execution of workflows that can be expressed as a Directed Acyclic Graph (DAG) of functions. The graph (specifying functions and their dependences) is described in JSON format, which can be generated automatically from a Web-based graphical editor using the FaaSr-JSON-Builder tool(FaaSr, 2024a) . Figure 1 shows an example workflow DAG graph with ten functions for an ecological forecasting application.

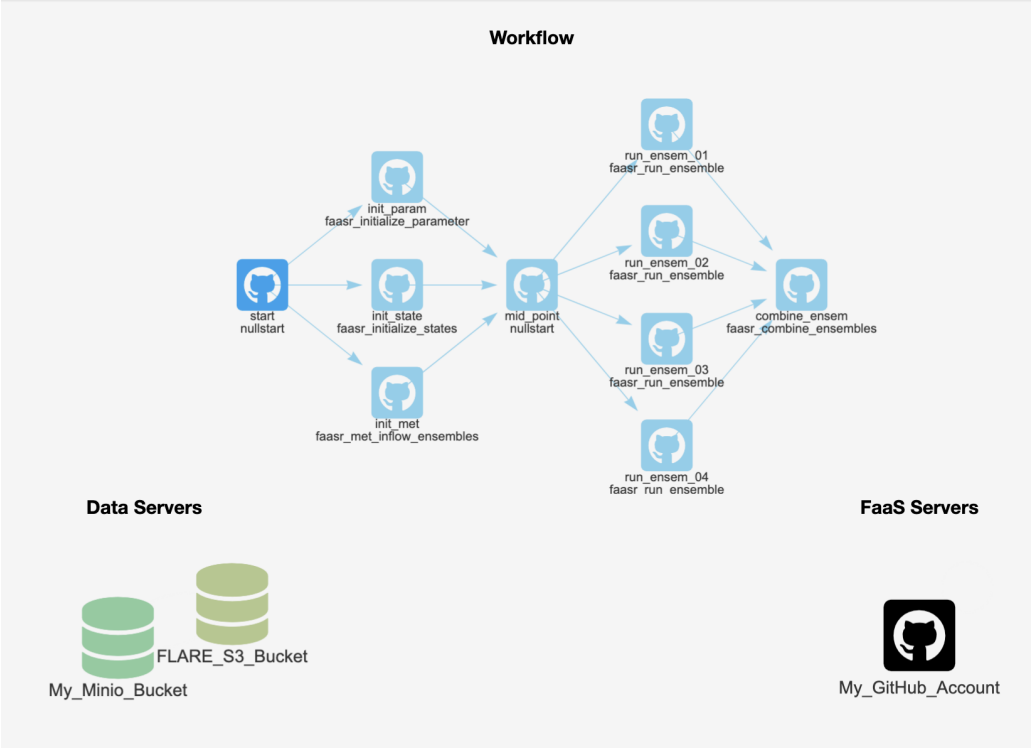

Figure 1: Fig. 1. FaaSr Example Workflow.

Description of Software

The FaaSr software is itself written in R. The main GitHub repository, FaaSr-Package, implements the core functionalities to register and invoke functions and to access data at runtime via S3 as well as via Apache Arrow(Richardson et al., 2024) over S3. FaaSr exposes both a client-side interface (intended for end users interactively using R/RStudio environments) and a server-side interface (intended for runtime invocation once functions are executed on FaaS platforms). These use cURL(Hostetter et al., 1997) and API-based packages httr(Wickham, 2023) and paws(Kretch & Banker, 2023) for sending requests to three supported FaaS providers: GitHub Actions, OpenWhisk, and AWS Lambda. Users are only required to have accounts, keys, and proper access policies for those providers that they wish to utilize.

The client-side interface is available by invoking the FaaSr::faasr() function with a valid payload as argument:

```
faasr_instance <- FaaSr::faasr("payload.json")
```

With the instance `faasr_instance` returned by the `faasr` function, users can register actions in the workflow to the FaaS provider(s) specified in the workflow JSON configuration. For example:

```
faasr_instance$register_workflow()
```

Users can trigger the action in the workflow by using the `invoke_workflow` function. The default action is the first action of the workflow designated in the JSON configuration as `FunctionInvoke`. For example:

```
faasr_instance$invoke_workflow()
```

Users can also call `set_workflow_timer` to establish a timer event that will automatically invoke the workflow. This is based on the cron(Reznick, 1993) specification of time intervals.

For example:

```
faasr_instance$set_workflow_timer("*/5 * * * *")
```

The server-side interface allows functions to interact with storage. For example, to download a file from an S3 server to local storage:

```
faasr_get_file(remote_folder=folder, remote_file=input1, local_file="df0.csv")
```

To upload a file from local storage to an S3 server:

```
faasr_put_file(local_file="df1.csv", remote_folder=folder, remote_file=output1)
```

To read/write from an S3 bucket with Apache Arrow and Parquet:

```
s3 <- faasr_arrow_s3_bucket()
```

To write a log message:

```
faasr_log("Function compute_sum finished")
```

The software also includes a FaaSr-Docker repository(FaaSr, 2024b) with code and actions used to build, configure, and upload container images to the respective container registers for the three platforms currently supported by FaaSr (GitHub's GCR, AWS's ECR, and DockerHub). These are used to build the base and default runtime environment for FaaSr (based on Rocker and TidyVerse) as well as for advanced users who may want to build their custom images starting from the base image.

Finally, the software also includes a FaaSr-JSON-Builder repository(FaaSr, 2024a) with code for an R-native graphical user interface Shiny app that allows users to create and edit workflows interactively and generate FaaSr schema-compliant JSON files.

Documentation

The software has been released on The Comprehensive R Archive Network (CRAN) (https://cran.r-project.org/web/packages/FaaSr/index.html) and the documentation is available on both CRAN and the FaaSr website (https://faasr.io/documentation)

Acknowledgements

FaaSr is funded in part by grants from the National Science Foundation (OAC-2311123 and OAC-2311124). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

Amazon. (2024a). *Lambda*. [Online], Available: https://aws.amazon.com/es/lambda/.

Amazon. (2024b). *S3*. [Online], Available: https://aws.amazon.com/s3/.

Apache. (2024). *Open source serverless cloud platform*. [Online], Available: https://openwhisk.apache.org/.

Chard, R., Babuji, Y., Li, Z., Skluzacek, T., Woodard, A., Blaiszik, B., Foster, I., & Chard, K. (2020). Funcx: A federated function serving fabric for science. *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, 65–76.

FaaSr. (2024a). *FaaSr JSON-builder*. [Online], Available: https://github.com/FaaSr/FaaSr-JSON-Builder.

FaaSr. (2024b). *FaaSr-docker repository*. [Online], Available: https://github.com/FaaSr/FaaSr-Docker.

Github. (2024). *Github actions*. [Online], Available: https://docs.github.com/ko/actions.

Hostetter, M., Kranz, D. A., Seed, C., Terman, C., & Ward, S. (1997). Curl: A gentle slope language for the web. *World Wide Web Journal*, *2*(2), 121–134.

Jonas, E., Pu, Q., Venkataraman, S., Stoica, I., & Recht, B. (2017). Occupy the cloud: Distributed computing for the 99%. *Proceedings of the 2017 Symposium on Cloud Computing*, 445–451.

Kretch, D., & Banker, A. (2023). *Paws: Amazon web services software development kit*. https://CRAN.R-project.org/package=paws

Lai, J., Lortie, C. J., Muenchen, R. A., Yang, J., & Ma, K. (2019). Evaluating the popularity of r in ecology. *Ecosphere*, *10*(1), e02567.

MinIO. (2024). *The object store for AI data infrastructure*. [Online], Available: https://docs.min.io/.

Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, M., & Vrgoč, D. (2016). Foundations of JSON schema. *Proceedings of the 25th International Conference on World Wide Web*, 263–273.

Reznick, L. (1993). Using cron and crontab. *Sys Admin*, *2*(4), 29–32.

Richardson, N., Cook, I., Crane, N., Dunnington, D., François, R., Keane, J., Moldovan-Grünfeld, D., Ooms, J., Wujciak-Jens, J., & Apache Arrow. (2024). *Arrow: Integration to 'apache' 'arrow'*. https://github.com/apache/arrow/

Shankar, V., Krauth, K., Pu, Q., Jonas, E., Venkataraman, S., Stoica, I., Recht, B., & Ragan-Kelley, J. (2018). Numpywren: Serverless linear algebra. *arXiv Preprint arXiv:1810.09679*.

Wickham, H. (2023). *Httr: Tools for working with URLs and HTTP*. https://CRAN.R-project.org/package=httr