The *Portals Framework*

Enabling Flexible Stateful Serverless Applications

Jonas Spenger (RISE, KTH), *Aleksey Veresov (KTH), Maximilian Kurzawski (KTH), Paris Carbone (RISE, KTH), Philipp Haller (KTH)*

Project Information

- Framework for stateful serverless applications.
- **Unifying stateful dataflow streaming and actor programming.**
- Developed at KTH and RISE since 2022.
- Open source; Apache 2.0 License.

Key Features

- **Atomic streams & exactly-once processing guarantees**.
- **Multi-dataflow applications**, cyclic dependencies.
- **Portal services:** inter-dataflow services with actor-like comm.
- **Dynamic topology**, **decentralized** cloud/edge execution.
- Data and task parallelism.

Portals Highlights

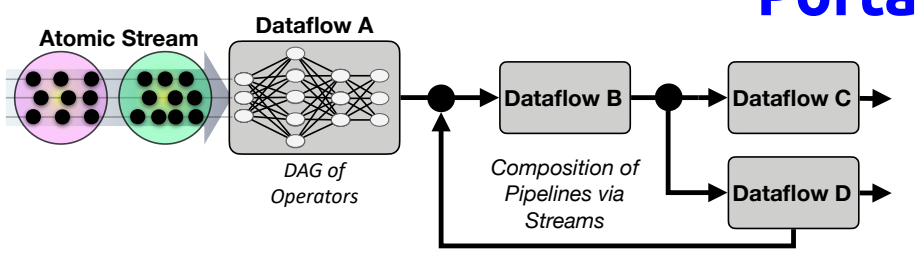


Figure from *[2]*

Multi-dataflow composition

- **Dataflows as microservices**.
- Composition using **atomic streams**.
- Direct communication between operators with portals.

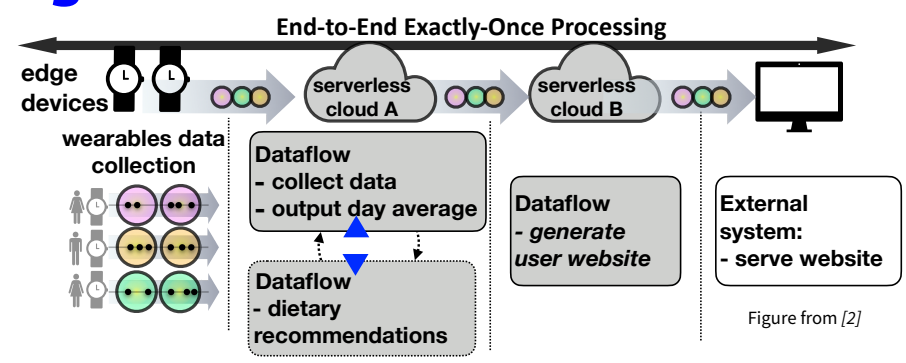


Figure from *[2]*

Decentralized, dynamic topology

- Applications spanning multiple deployments, cloud and edge devices.
- **Topology may change over time.**
- A runtime for cloud and edge.

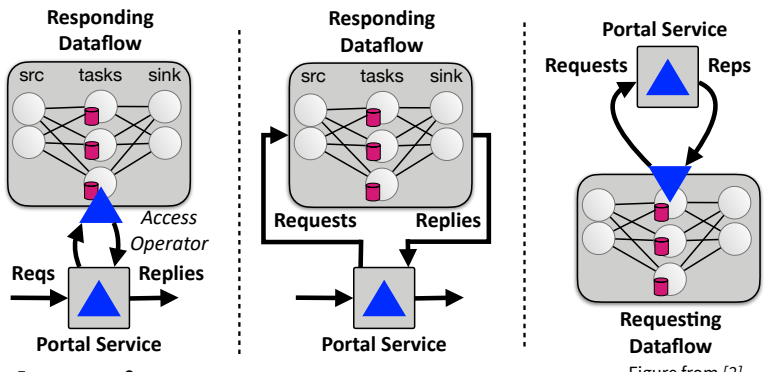


Figure from *[2]*

Portal services

- **A Portal exposes a service**, implemented with task operators**.**
- Enables **request/reply communication** between task operators.

Shopping cart example

- **Inventory exposed through a Portal**; cart connects to inventory's portal to get/return items to the inventory; analytics service started dynamically.

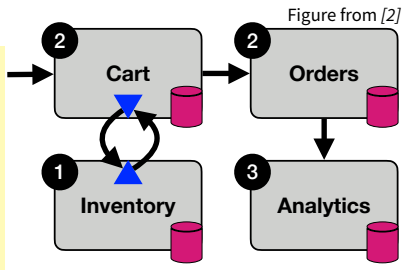


Figure from *[2]*

Cart App:

```
// Ref to Inventory's Portal Service
val inventory = Portal[...](...)

// Cart
Dataflows("cart").source(...)
  .taskWithRequester(inventory):
    case AddToCart(item) =>
      req = GetItem(item)
      future = Request(inventory)(req)
      Await(future):
        future.value match
          case GetItemSuccess =>
            state.update(item, state.get(item) + 1)
            ...
    ... // truncated
```

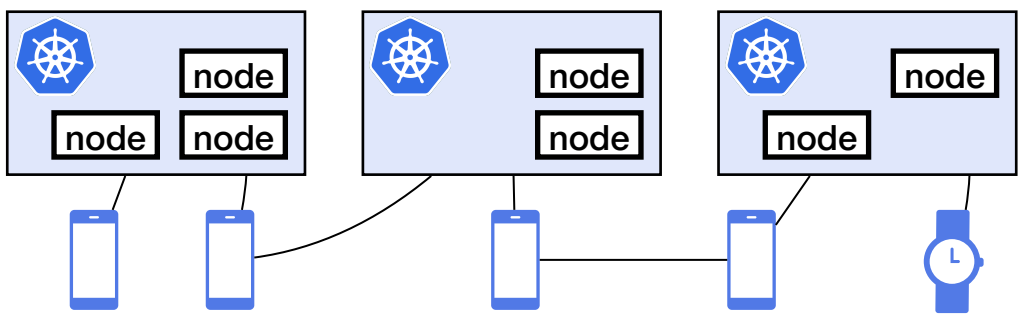
Powered by atomic streams

- **Enforces the exactly-once processing guarantees.**
- Provides interface for the atomic processing contract.

Atomic Processing Contract
i) Take atom
ii) Process atom until completion
iv) Commit to output
v) Repeat

Portals Distributed Runtime

- Distributed execution, leveraging **Atomic Streams, Reply Streams**.
- Serverless deployment environment using Docker/Kubernetes.
- Decentralized, support for connecting to remote deployments.



Distributed, decentralized runtime leveraging Atomic Streams and Reply Streams; edge-cloud.

Formally Verified Fault-Tolerance

- Formalization of Portals i) high-level model and ii) implementation.
- Mechanised, machine checked, in *Coq*.
- **Rigorous proof of *exactly-once processing*.**

A faultless high level program should be behaviourally indistinguishable from its implementation in a faulty low-level.

$$\frac{\Pi(p) \vdash \Sigma(p) \xrightarrow{\{x_i\}_i} \sigma \qquad \overline{X} = \{p\, x_i\}_i \qquad \overline{X}(B) = B'}{\Pi \vdash \langle \Sigma, B \rangle \overset{\overline{X}}{\Longrightarrow} \langle \Sigma[p \mapsto \sigma], B' \rangle}$$

$$\frac{B(p\, s \triangleleft) = \overline{m} : m \qquad \overline{X} = p\, s \triangleleft -m : \{q_i\, s \triangleright +m \mid q_i\, s \triangleright \in \mathrm{dom}(B)\}_i}{\Pi \vdash \langle \Sigma, B \rangle \overset{\overline{X}}{\Longrightarrow} \langle \Sigma, \overline{X}(B) \rangle}$$

Further Reading & References

- [1] **Spenger et al., "*Portals: An extension of dataflow streaming for stateful serverless*.", Onward'22.**
- [2] Spenger et al., "*Portals: A Showcase of Multi-Dataflow Stateful Serverless*", PVLDB'23.
- https://www.portals-project.org/; https://github.com/portals-project/portals

Acknowledgements

- We would like to thank previous contributors to the Portals Project: Chengyang Huang, Gabriele Morello, Siyao Liu.
- *This work was partially funded by Digital Futures, the Swedish Foundation for Strategic Research under Grant No.: BD15-0006, as well as RISE AI.*