Portals: A Showcase of Multi-Dataflow Stateful Serverless

*†**Jonas Spenger**, †Chengyang Huang, †Philipp Haller, *†Paris Carbone

*RISE Research Institutes of Sweden, Stockholm, Sweden; †KTH Royal Institute of Technology, Stockholm, Sweden

https://www.portals-project.org/vldb2023demo/

demo info

Context

- **Serverless functions** have made it easy to write and deploy distributed applications on fully managed runtimes.
- Recent developments have been on including **state management** and **compositional patterns**, enabling a wider range of applications; yet, finding the right abstractions and implementation methods remains an open problem.
- Our work is on a decentralized programming framework for **stateful serverless applications in the cloud-edge continuum**.

Summary

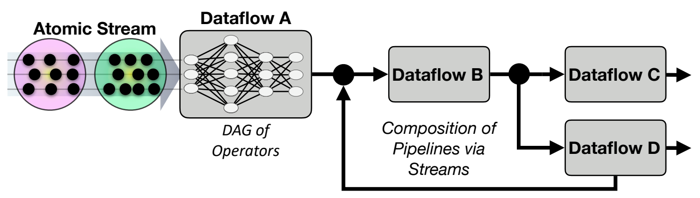
This demonstration presents Portals [1, 2], a programming framework for stateful serverless applications, with the following highlights:

- **Multi-dataflow applications**. Dynamically composing multiple dataflow pipelines together via **atomic streams**.
- **Inter-dataflow services / Portal services**. Exposing dataflow pipelines (operators, state) as inter-dataflow services.
- **Decentralized cloud and local execution**. Decentralized API and runtime, with end-to-end processing guarantees.

- Portals Vision -

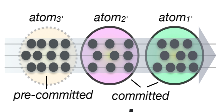
Key Features

*// Programming framework for stateful serverless // **Atomic streams & exactly-once processing guarantees (WIP)** // Dynamic topology // Decentralized execution // Flexible API //*



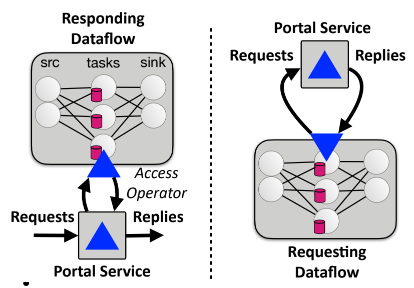
Multi-dataflow composition

- **Dataflows as microservices.**
- Composition using **atomic streams**.



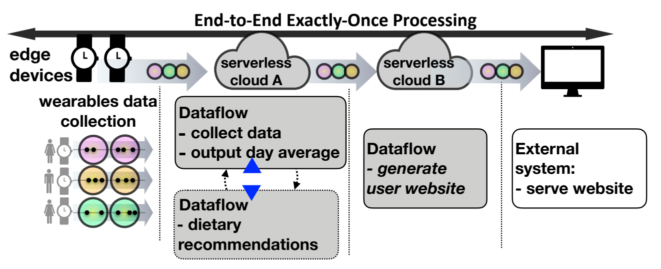
Powered by Atomic Streams

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



Portal services

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

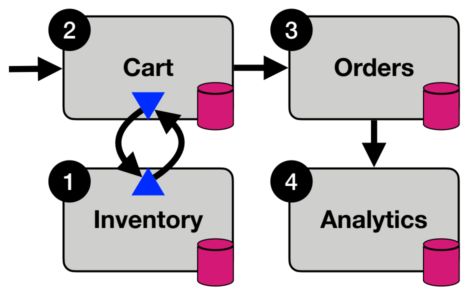


End-to-End Exactly-Once Processing
edge devices
serverless cloud A
serverless cloud B
wearables data collection
Dataflow - collect data - output day average
Dataflow - dietary recommendations
Dataflow - generate user website
External system: - serve website

Decentralized, dynamic topology

- Applications spanning multiple deployments.
- **Topology may change over time.**
- A runtime for cloud and edge (WIP).

Scenario 1: Shopping Cart Pipeline

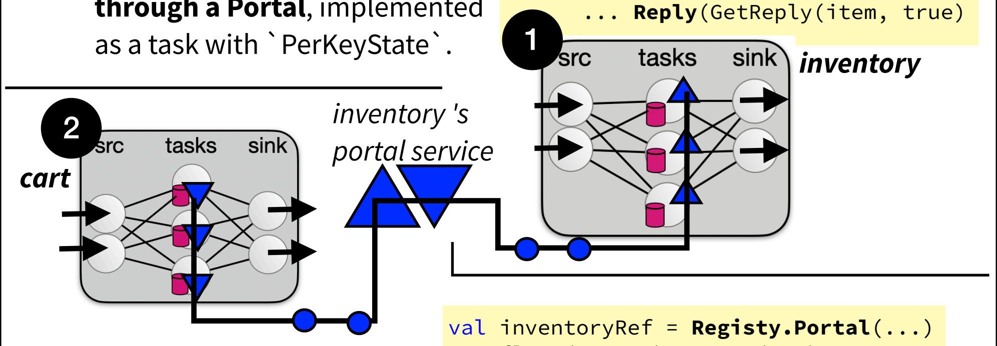


Demo overview

- **Four services**: cart; orders; inventory; analytics.
- Services launched dynamically, connected.
- Portal service exposes the inventory, analytics.
- Starting new services that use the analytics, on demand.

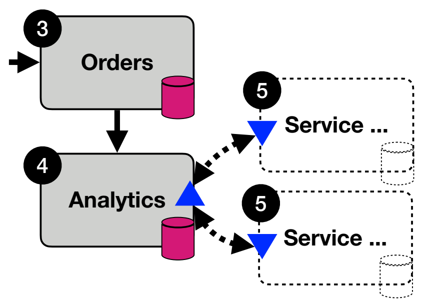
(1) The inventory exposed through a Portal, implemented as a task with `PerKeyState`.

```
val inventory = Portal(...)
Dataflows("inventory").source(...)
  .taskWithReplier(inventory){...}{
   ... if State.get() > 0 then
     ... Reply(GetReply(item, true)
```



(2) The **cart** consumes user requests, and **interacts with the inventory through a portal** to add/remove items to the cart.

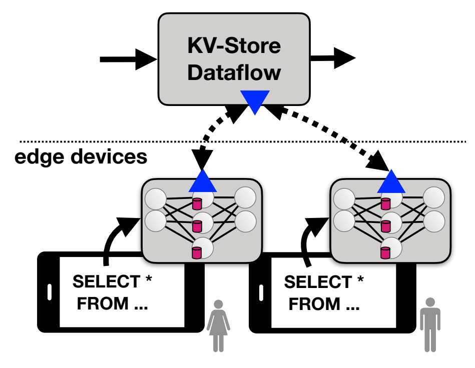
```
val inventoryRef = Registy.Portal(...)
Dataflows("cart").source(...)
  .taskWithRequester(inventoryRef):
   ... Request(inventoryRef)
(GetItem(item)).onComplete:
       case Success(rep) =>
         State.update(item, ... + 1)
```



(3) The orders app consumes the checked out orders.
(4) The analytics service produces a Top100 list of the orders, exposed through a portal service.
(5) New services dyn. connect to the the analytics portal.

Scenario 2: SQL to Dataflow

- **SQL API based on portal services with state managed by a dataflow [3].**
- Supports multi-table SQL queries and transactions.
- The SQL engine uses Apache Calcite.



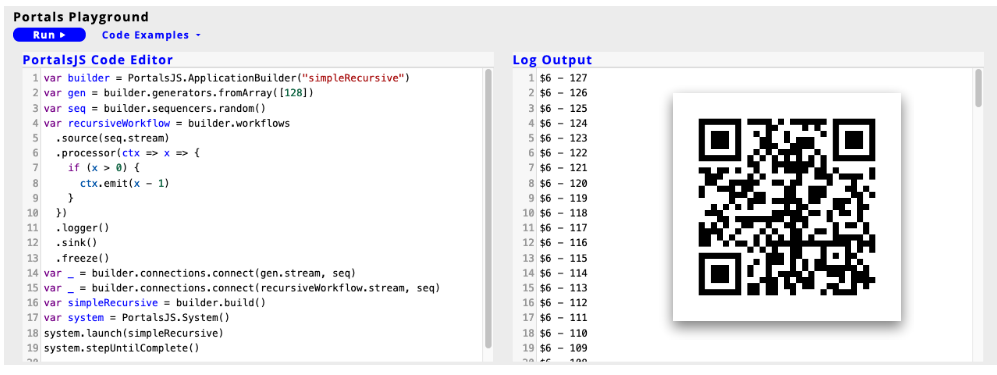
```
val table =
  TableWorkflow[Types.KV]("KVTable")
```

Multiple querying apps connecting to the same shared KeyValue store, implemented as a library, leveraging the decentralized execution.

```
Dataflows[String, String](...)
  .source(...)
  .query(table)
```

Scenario 3: Playground

- The Portals Playground is a Javascript-based sandbox capable of **running Portals apps in the browser**. Check out the examples!
- The JS runtime can be used for web apps and edge devices.



PortalsJS Code Editor

```
var builder = PortalsJS.ApplicationBuilder("simpleRecursive")
var gen = builder.generators.fromArray([128])
var seq = builder.sequencers.random()
var recursiveWorkflow = builder.workflows
  .source(seq.stream)
  .processor(ctx => x => {
    if (x > 0) {
      ctx.emit(x - 1)
    }
  })
  .logger()
  .sink()
  .freeze()
var _ = builder.connections.connect(gen.stream, seq)
var _ = builder.connections.connect(recursiveWorkflow.stream, seq)
var simpleRecursive = builder.build()
var system = PortalsJS.System()
system.launch(simpleRecursive)
system.stepUntilComplete()
```

Log Output

```
$6 - 127
$6 - 126
$6 - 125
$6 - 124
$6 - 123
$6 - 122
$6 - 121
$6 - 120
$6 - 119
$6 - 118
$6 - 117
$6 - 116
$6 - 115
$6 - 114
$6 - 113
$6 - 112
$6 - 111
$6 - 110
$6 - 109
```

https://www.portals-project.org/playground/

Ongoing / Future Work

- Impl. of the distributed serverless runtime.
- Dataflow optimiser exploiting the global view; improving the performance of cyclic dependencies across pipelines.
- Multi-dataflow ACID transactions.

References

[1] Spenger et al., "*Portals: An extension of dataflow streaming for stateful serverless.*", Onward'22.

[2] https://github.com/portals-project/portals

[3] Chengyang Huang. "Queryable Workflows: Extending Dataflow Streaming with Dynamic Request/Reply Communication." Dissertation, 2023.

Acknowledgements

We would like to thank other contributors to the Portals project: Maximilian Kurzawski; Aleksey Veresov; 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.