Building Data Processing Pipelines Out of Microservices

Richard Seroter
SENIOR DIRECTOR OF PRODUCT

@rseroter

Overview

The role of orchestration in microservices

Problems with the status quo

About Spring Cloud Data Flow

Comparing Streams and Tasks

Installing Spring Cloud Data Flow

Creating Streams and Tasks

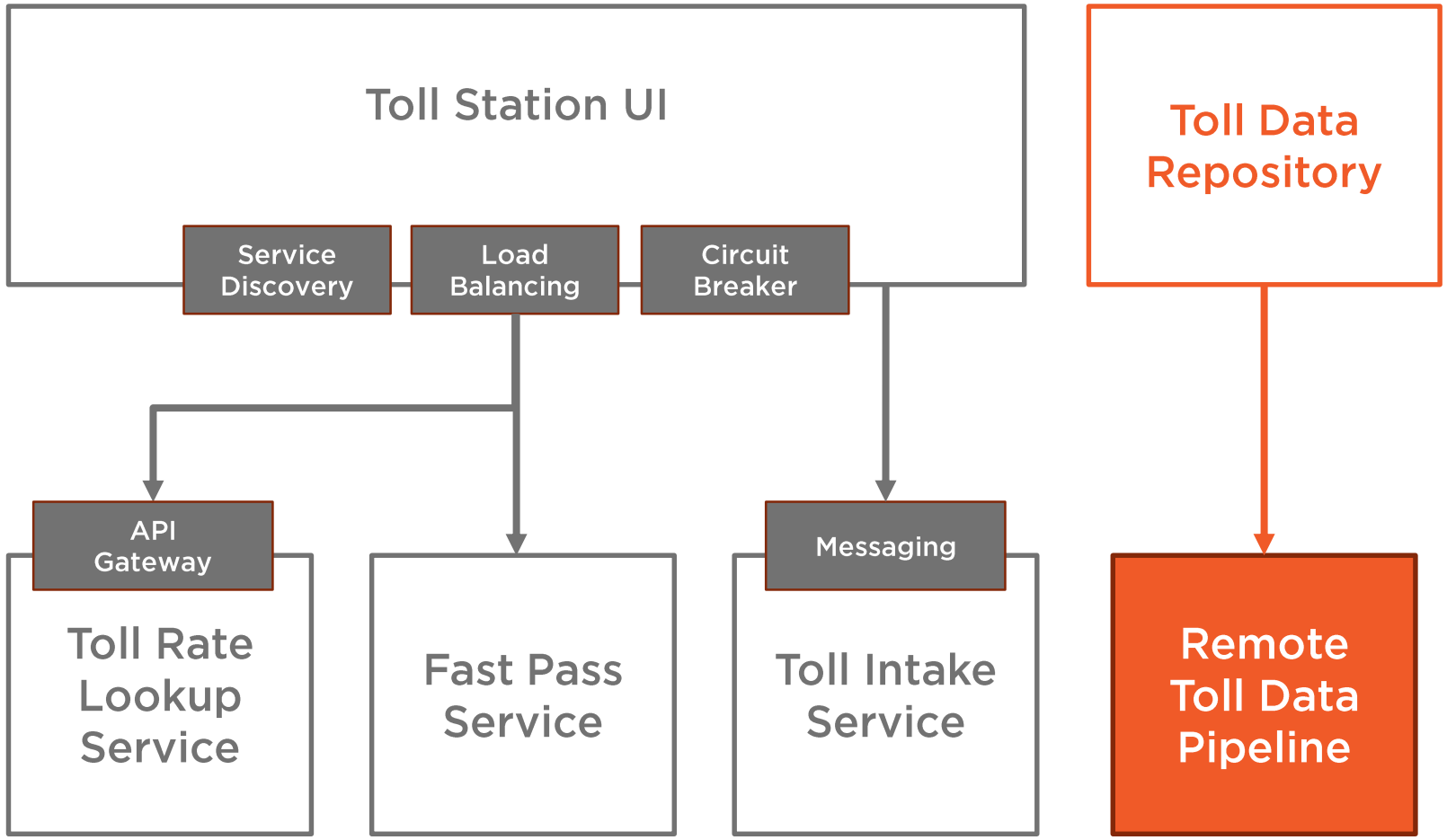
Using the Dashboard and Flo

Creating Composed Tasks

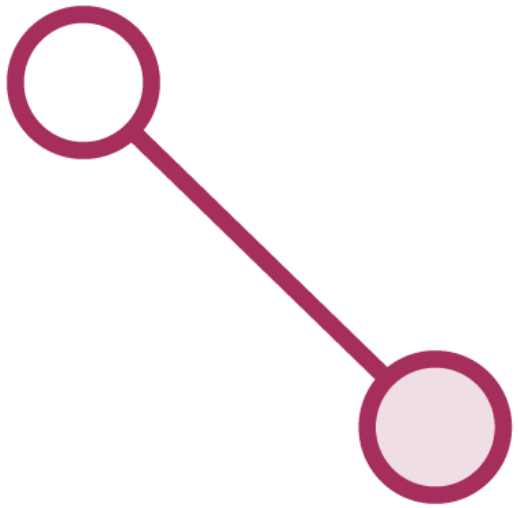
Monitoring and updating pipelines

Summary

Capabilities That We Will Add in This Module

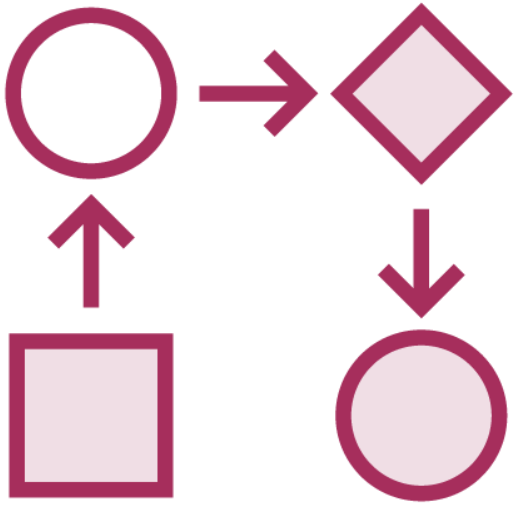


The Role of Orchestration in Microservices

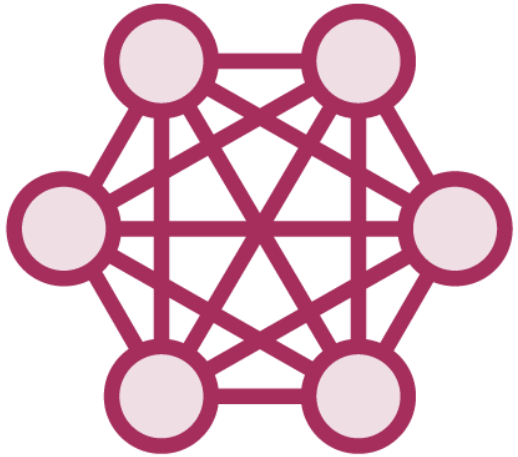


Integrate systems, apps, and data sources

Do batch and real-time processing or analytics



Group services and quickly adjust processes



Establish an event-driven architecture

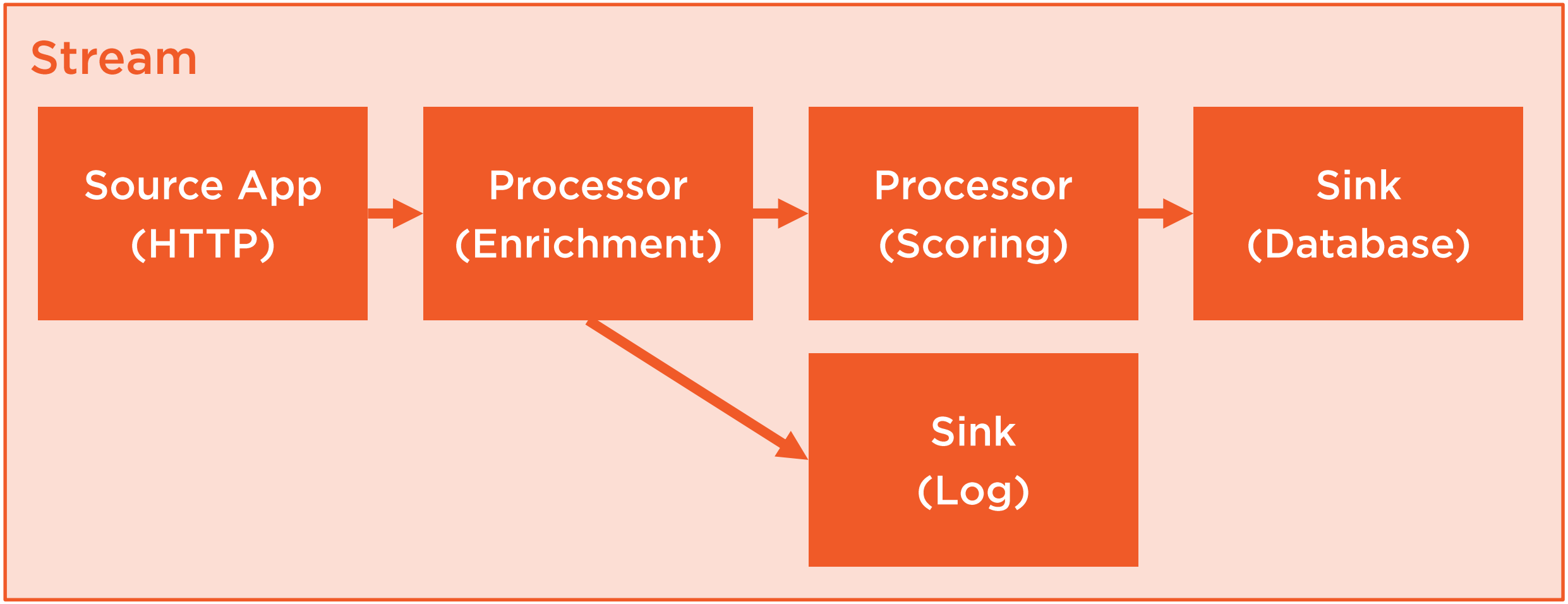
Problems with the Status Quo

- Existing solutions are monolithic in nature
- Hard to quickly change processes
- Scaling is done in all-or-nothing fashion
- Any changes are delivered in bulk
- Specialized skills require silos of expertise in the company

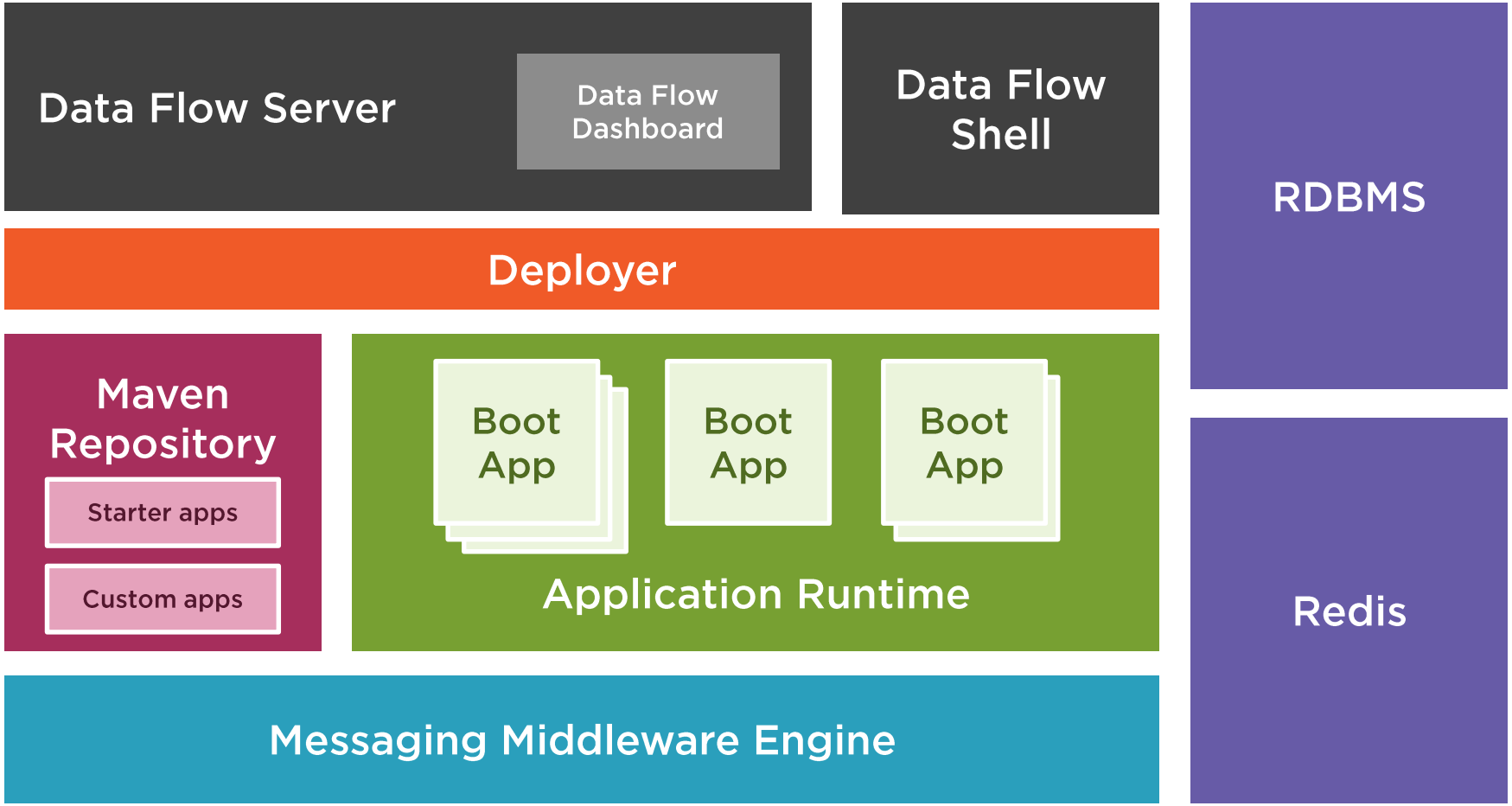
Spring Cloud Data Flow

Toolkit for building data pipelines.

Logical View of a Streaming Pipeline



Core Components of Spring Cloud Data Flow



About the Data Flow Server

Exposes REST endpoints

Executable jars for each target runtime

Change functionality via startup parameters

All features enabled by default

Multiple security options

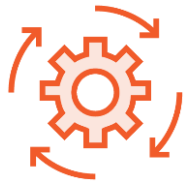
Provides schemas for popular relational databases

Consider Streams vs. Tasks

Streaming pipelines	Task pipelines
Process “ceaseless” workloads	Process irregular workloads
Built with Spring Cloud Stream	Built with Spring Cloud Task
Instances always running	Instances started on-demand
Run as event-driven, sequential process	Branching and conditional operations
Tap into streams	Lifecycle hooks and task events emitted

Where “Apps” Come From

Import Spring Cloud Stream App Starters and Spring Cloud Task App Starters.



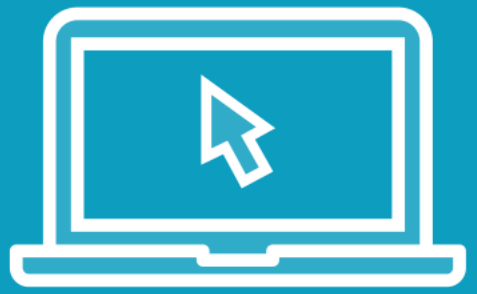
Build custom stream or task apps.

All apps are standalone and can be created, unit-tested, and debugged in isolation.

Installing Spring Cloud Data Flow

1. Ensure Java and Maven installed
2. Setup RDBMS and Redis
3. Stand up messaging middleware
4. Download and launch Server and Shell apps

Demo



- Configure RabbitMQ instance
- Set up MySQL storage
- Set up Redis
- Download local Data Flow Server binaries
- Download Data Flow shell
- Start server and log in with shell

About Stream Starter Apps

Stream offers sources, processor, and sinks

Either Maven or Docker artifacts

Example sources: file, http, JDBC, Amazon S3, MongoDB, Gemfire, and Twitter

Example processors: filter, http client, PMML, transform, and splitter

Example sinks: counter, file, HDFS, log, Amazon S3, WebSocket, and JDBC

Regularly updated list of apps

Creating Streams with Spring Cloud Data Flow

Develop using DSL via REST API, shell, or Flo

DSL describes flow of data

Register one or more apps for use in streams

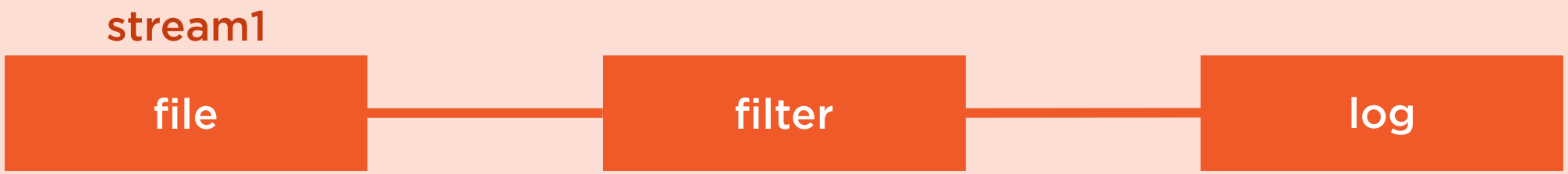
Pipe symbol (“|”) connects apps

Alter app behavior through whitelisted properties

Labels make it easy to tap, extend streams

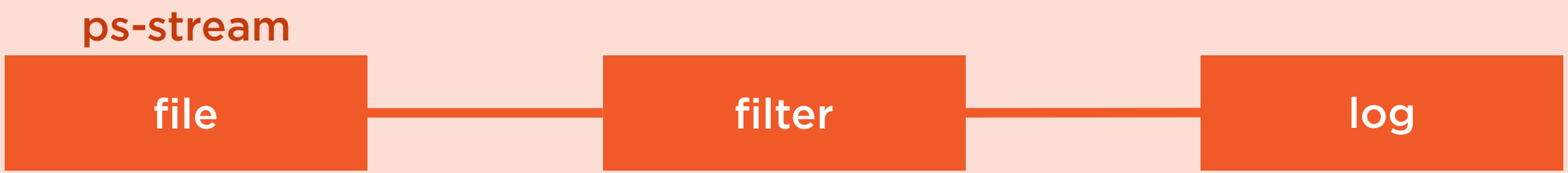
Stream DSL Examples

```
dataflow:> stream create --definition "file | filter | log" --
name stream1
```



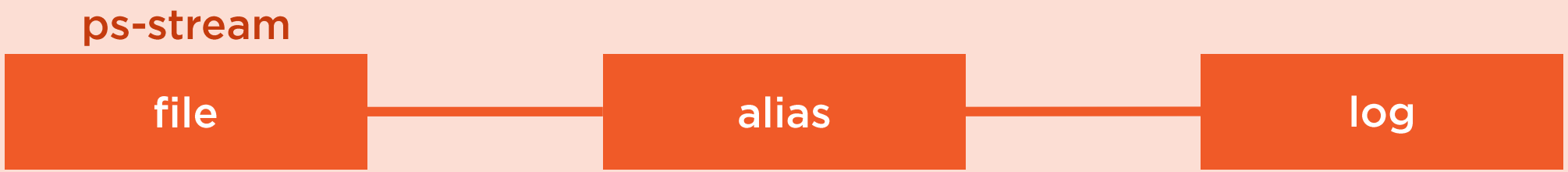
Stream DSL Examples

```
dataflow:> stream create --definition ”file --directory=/temp |
filter | log" --name ps-stream
```



Stream DSL Examples

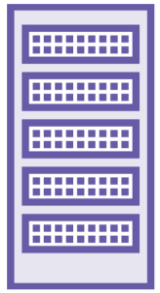
```
dataflow:> stream create --definition ”file --directory=/temp |
alias: filter | log" --name ps-stream
```



Stream DSL Examples

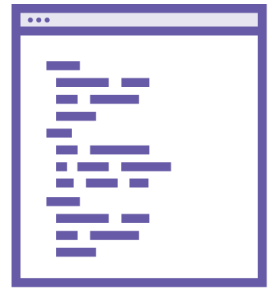
```
dataflow:> stream create --definition ”file --directory=/temp |
alias: filter | log" --name ps-stream

dataflow:> stream create --definition ”:ps-stream.alias > log" --
name ps-stream2
```

Deploying Data Pipelines

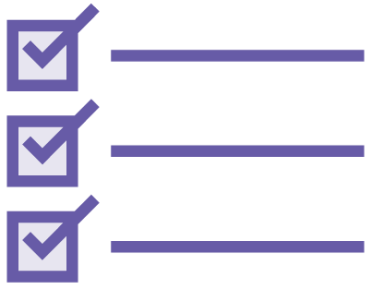


Runtime drives resource utilization

Set instance count for apps



Apply application and deployer properties



Data Flow sets required SCSt properties

Configure partitioning support

Stop and delete streams too

Demo

Load Stream app starters into server

Create a streaming pipeline that takes in an HTTP request and sends data to a file

Test it!

Create a second pipeline that takes data from a file system, transforms it, and writes the result to another file

Creating Tasks with Spring Cloud Data Flow

Develop using DSL via REST API, shell, or Flo

Register one or more apps for use in tasks

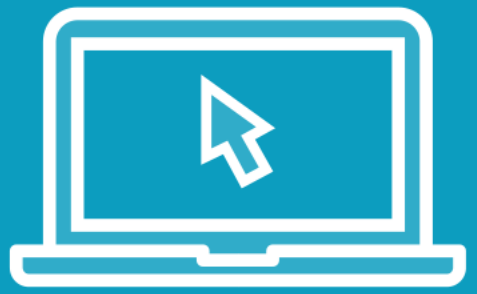
Create a Task Definition from a Task app

When launched, can send properties as arguments

Task execution stored in repository

Options for launching tasks from streams

Demo



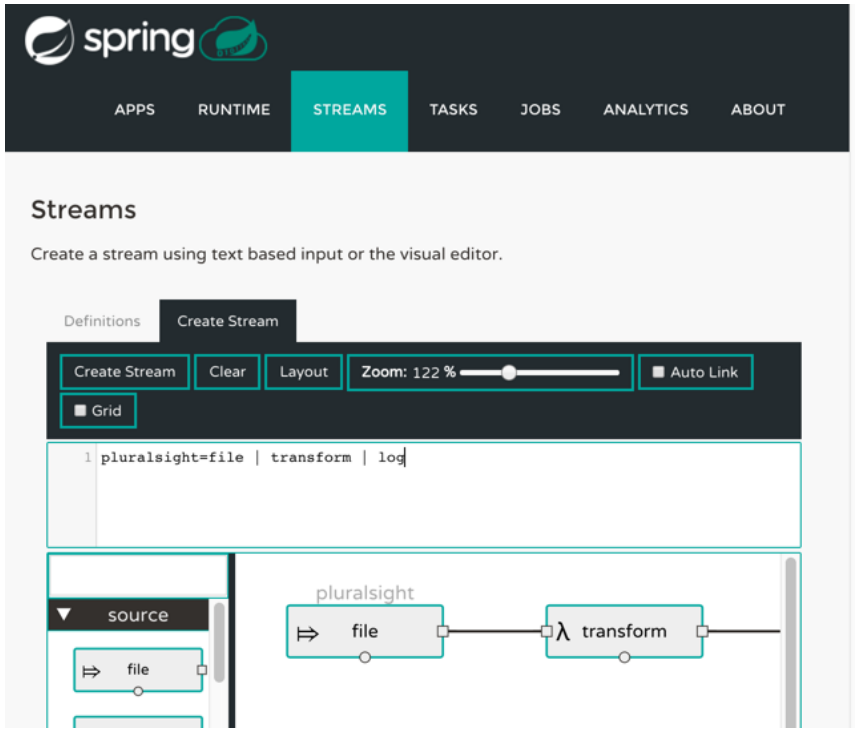
Load Task app starters into the server

Create a new task

Launch the task

View the output, execution history

Using the Dashboard and Flo



- Browser-based GUI
- Tabs for apps, runtime, streams, tasks, jobs, and analytics
- Import apps in bulk
- Visualize existing streams
- Use Flo to visually create data pipelines
- Get data visualization capabilities for apps that support it

Demo

Browse Flo interface

Create a new stream pipeline that receives JSON from HTTP, splits, routes to log

Deploy with logs written to console

Add a tap to the stream

Creating Custom Streaming or Task Apps

Create “regular” Spring Cloud Stream or Spring Cloud Task apps

Whitelist properties so that shell and UI can show them as options

Install module into local Maven repository

Demo

Create a custom Stream application

Register the custom application

Build a streaming pipeline that uses the custom component

Deploy and test the application

Create an updated stream with partition processing

Creating Composed Tasks

A directed graph where each node is a task app

Build via REST API, shell, Flo

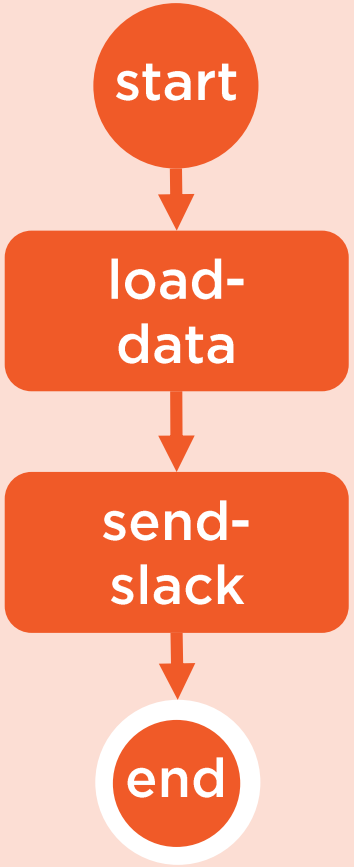
Data Flow Server generates app for composed task

Executed by the Composed Task Runner app

Rich DSL with conditionals, parallel execution

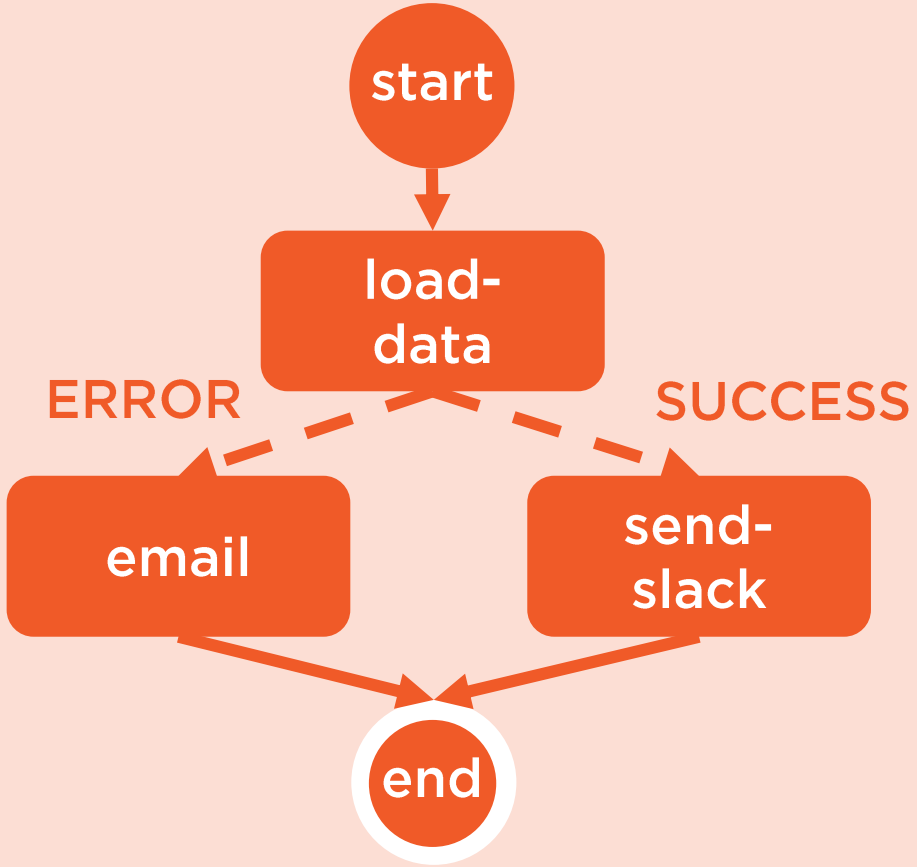
Composed Task Examples

```
dataflow:> task create
mytask --definition
”load-data && send-
slack”
```



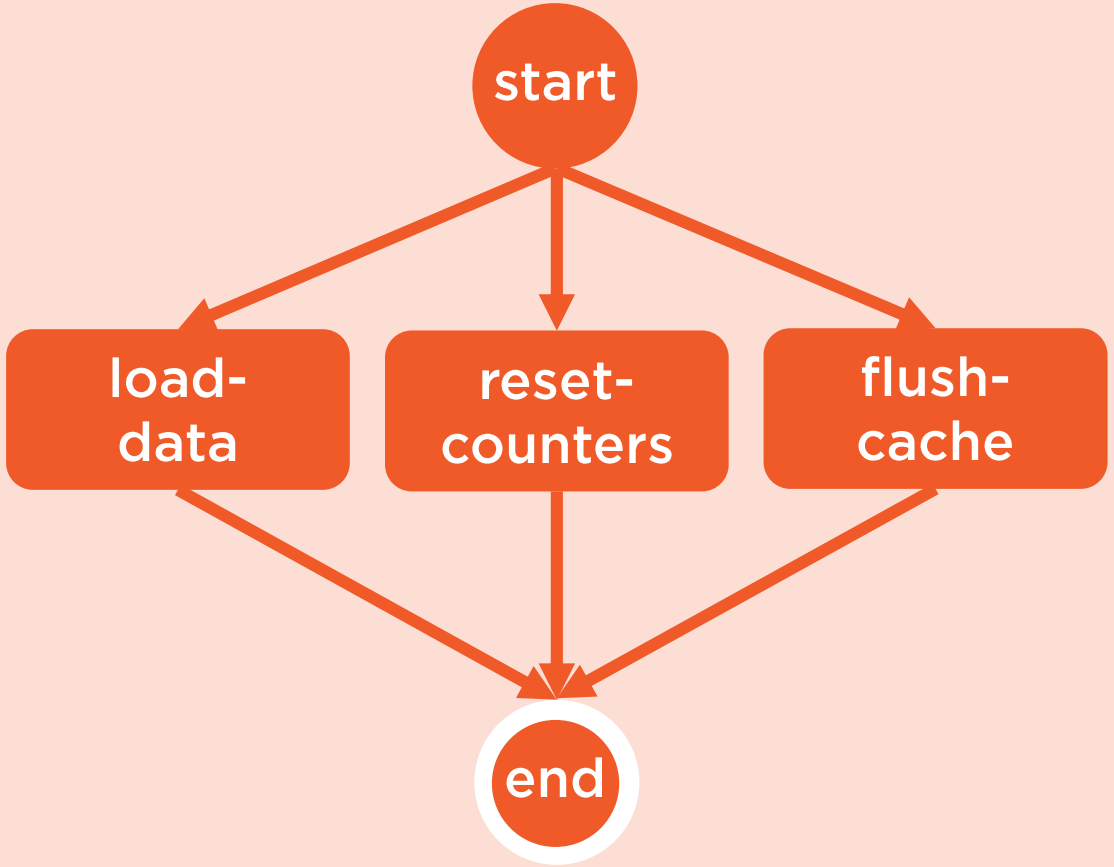
Composed Task Examples

```
dataflow:> task create
mytask --definition
”load-data ‘ERROR’ ->
email ‘SUCCESS’ ->
send-slack”
```

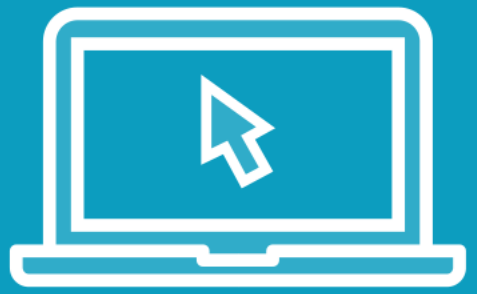


Composed Task Examples

```
dataflow:> task create
mytask --definition
”<load-data || reset-
counters || flush-
cache>”
```

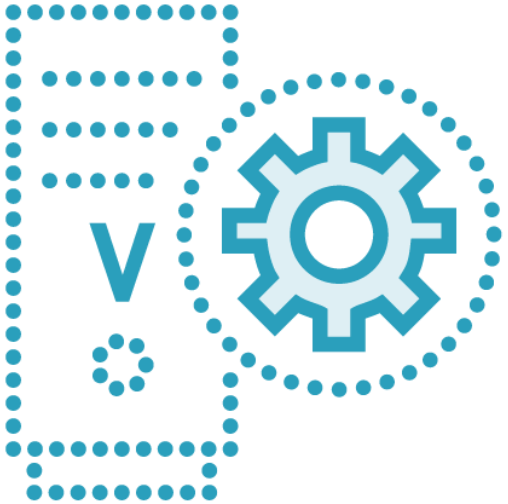


Demo



- Create a custom task application
- Register and execute that individual task
- Register two pre-built task applications
- Create composed tasks
- Execute and observe the results

Monitoring Spring Cloud Data Flow Pipelines

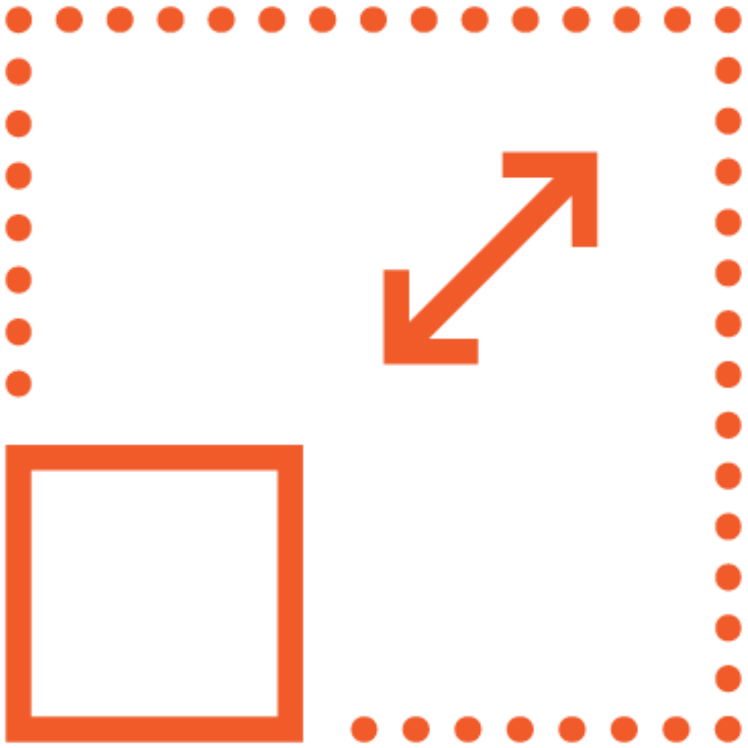


Standard monitoring through Actuators

Sink applications that write counter data

Create Data Flow Streams to collect metrics

Updating Apps, Streams, and Tasks



- Scale standard streams via target runtime
- Cannot scale partitioned streams
- Cannot scale apps that use Kafka 0.8 binder
- Versioning not yet baked in
- Many improvements coming in this area!

Summary

- Overview
- The role of orchestration in microservices
- Problems with the status quo
- About Spring Cloud Data Flow
- Comparing Streams and Tasks
- Installing Spring Cloud Data Flow
- Creating Streams and Tasks
- Using the Dashboard and Flo
- Creating Composed Tasks
- Monitoring and updating pipelines