Stream and Batch Processing in the Cloud with Data Microservices

Marius Bogoevici and Mark Fisher, Pivotal

Stream and Batch **Processing** in the Cloud with Data Microservices

- Use Cases
  - Predictive maintenance
  - Fraud detection
  - QoS measurement
  - Log analysis
- High throughput/low latency
  - Growing quantities of data
  - Immediate response is required
- Grouping and ordering of data
  - Partitioning
  - Windowing

Stream and **Batch Processing** in the Cloud with Data Microservices

- Use Cases
  - ETL
  - Account Reconciliation
  - Machine Learning (e.g. model updates)
- Periodic activities
- Finite datasets
- Retry, Skip, Stop, Restart
- Dynamic resource allocation
- Increasing demand for the realm of batch processing use-cases to move to real-time (“aka Stream Processing”)

Stream and Batch Processing **in the Cloud** with Data Microservices

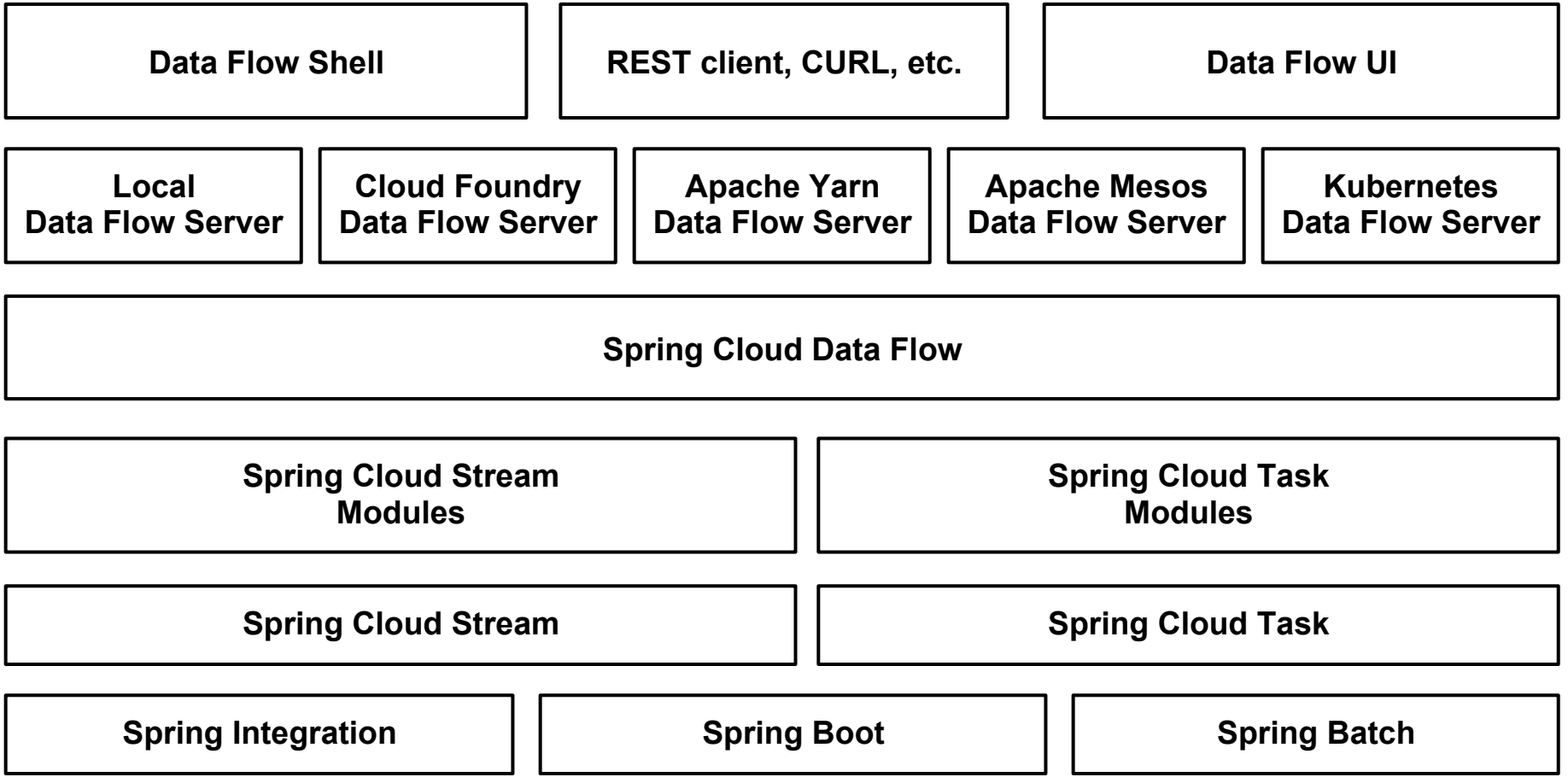
- Huge quantities of data to be analyzed efficiently
- Scaling requirements
  - Massive storage
  - Massive computing power (memory/CPU)
  - Massive scalability, from a few machines to data center level
- Reliance on platform's resource management abilities
  - public and private cloud: AWS
  - cluster managers: Apache YARN, Apache Mesos, Kubernetes
  - full application platforms: Cloud Foundry

Stream and Batch Processing in the Cloud with **Data Microservices**

- Microservice pattern applied to data processing applications
- Typical benefits:
  - scalability, isolation, agility, continuous deployment, operational control
- Tuning process-specific resources
  - Instance count
  - Memory
  - CPU
- Event-driven

Demo

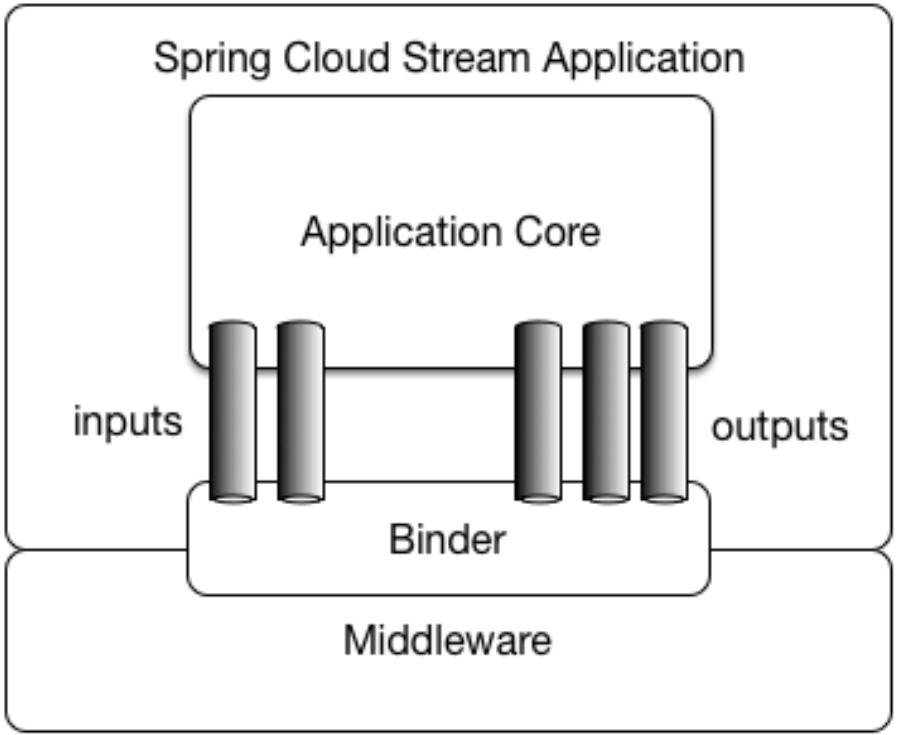
```
dataflow:> stream create demo --definition "http | file"
```


Data Flow Shell
REST client, CURL, etc.
Data Flow UI
Local Data Flow Server
Cloud Foundry Data Flow Server
Apache Yarn Data Flow Server
Apache Mesos Data Flow Server
Kubernetes Data Flow Server
Spring Cloud Data Flow
Spring Cloud Stream Modules
Spring Cloud Task Modules
Spring Cloud Stream
Spring Cloud Task
Spring Integration
Spring Boot
Spring Batch

Spring Cloud Stream

- Event-driven microservice framework
- Built on Spring stack:
  - Spring Boot: full-stack standalone apps, configuration
  - Spring Integration: messaging primitives and enterprise integration patterns
- Simplify access to middleware
- Common abstractions
  - Middleware binding
  - Consumer groups
  - Partitioning
  - Pluggable Binder API

Spring Cloud Stream in a nutshell



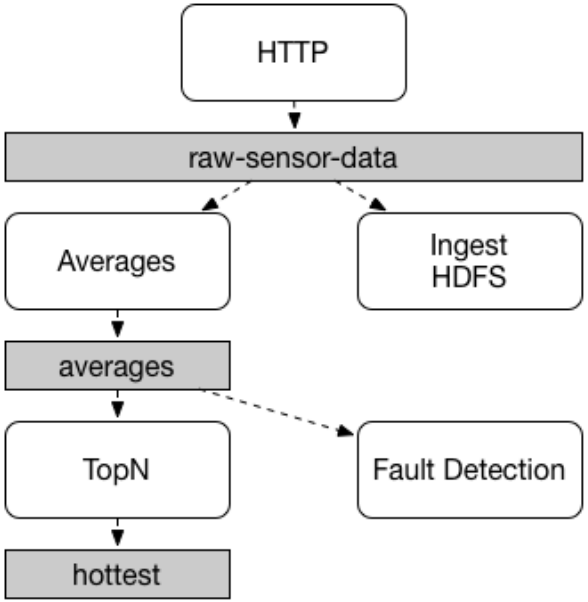
Programming model

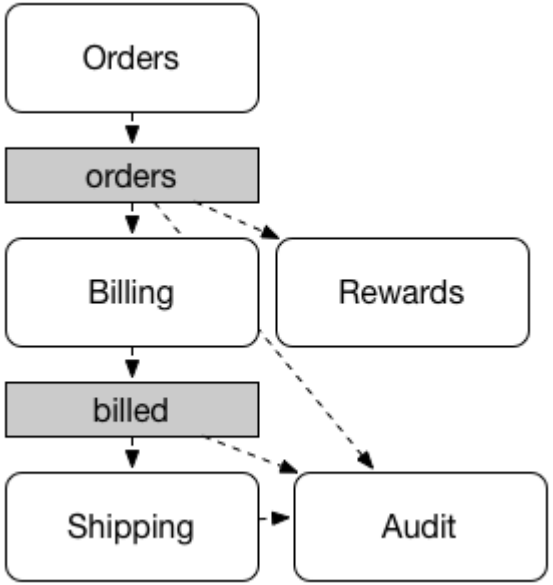
```
package org.springframework.cloud.stream.messaging;

public interface Processor {
	String INPUT = "input";
	String OUTPUT = "output";

	@Input(Processor.INPUT)
		SubscribableChannel input();
		@Output(Processor.OUTPUT)
		MessageChannel output();
}
```

```
@EnableBinding(Processor.class)
public class UpperCase {
		@Transformer(inputChannel = Processor.INPUT, outputChannel=Processor.OUTPUT)
		public String process(String message) {
			return message.toUpperCase();
		}
}
```

Event-driven model, publish subscribe semantics

- Published data broadcast to all subscribers
- Reduce data pipeline complexity
- Fits both data streaming and event-driven microservice use cases



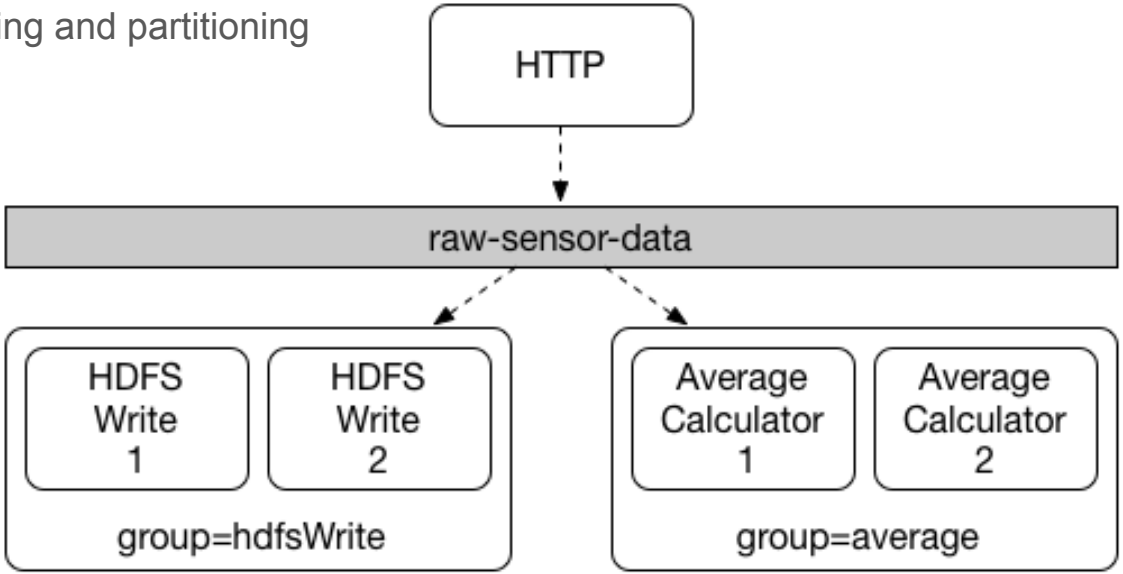


Consumer groups

Borrowed from Kafka, applied across all binders

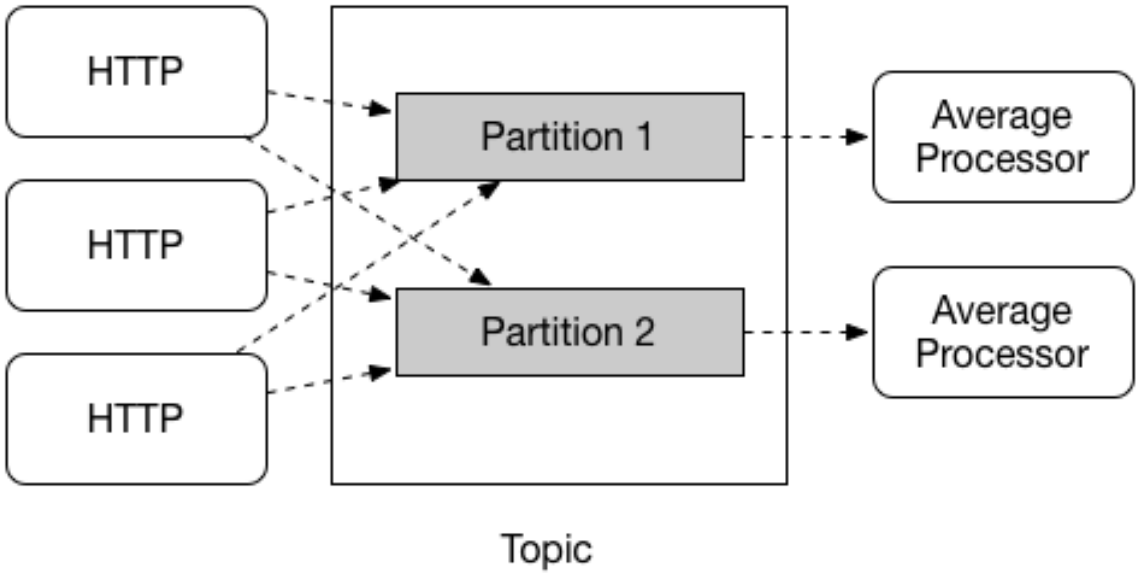
Groups of competing consumers within the pub-sub architecture

Used in scaling and partitioning

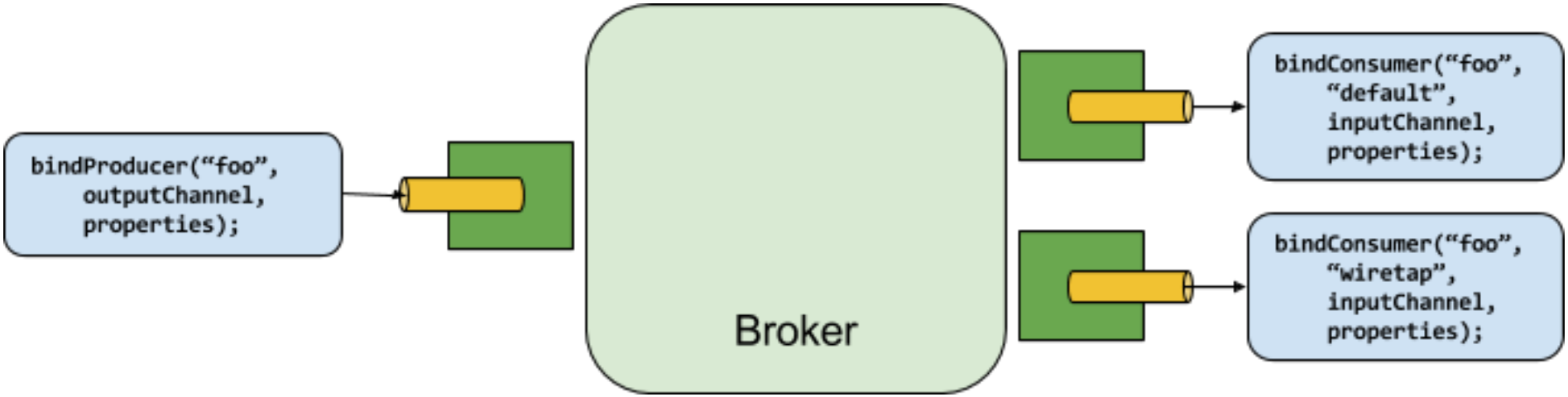


Partitioning

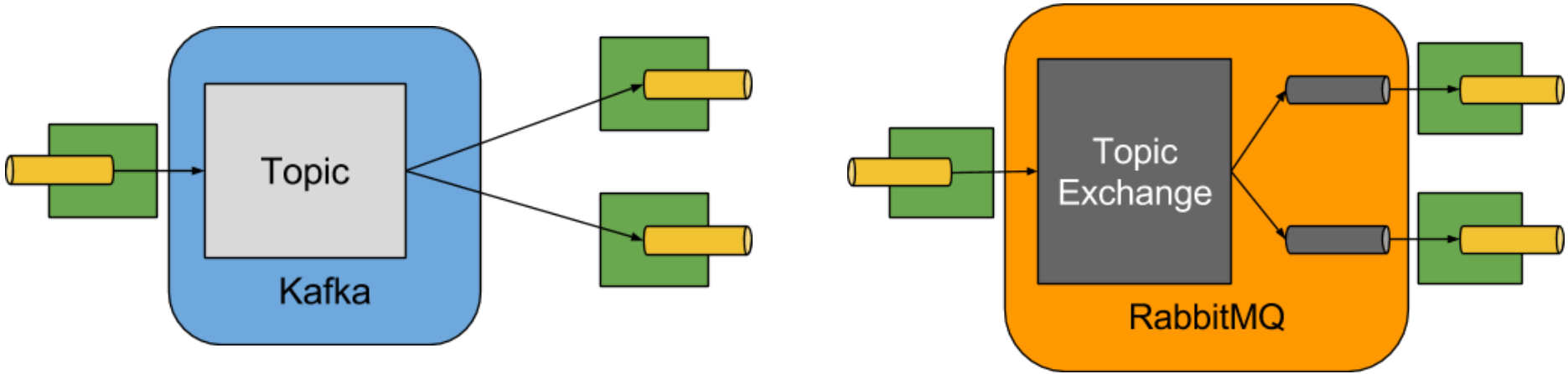
- Required for stateful processing scenarios involving data groups (e.g. average calculation)
- Outputs can specify a data partitioning strategy: SpEL, own implementation
- Inputs can be bound to a specific partition



Binder SPI


bindProducer("foo",
outputChannel,
properties);
Broker
bindConsumer("foo",
"default",
inputChannel,
properties);
bindConsumer("foo",
"wiretap",
inputChannel,
properties);

Binder Implementations



Other implementations: Redis, Gemfire, … your own!

Spring Cloud Task

```
@SpringBootApplication
@EnableTask
public class MyApp {
            @Bean
            public MyTaskApplication myTask() {
                        return new MyTaskApplication();
            }

            public static void main(String[] args) {
                        SpringApplication.run(MyApp.class);
            }

            public static class MyTaskApplication implements
CommandLineRunner {
                        @Override
                        public void run(String... strings) throws Exception
{
                                    System.out.println("Hello World");
                        }
            }
}
```

- task can be deployed, executed and removed on demand
- result of the process persists beyond the life of the task for future reporting

Spring Cloud Data Flow

Orchestration Layer for Streams and Tasks

- DSL
- Repositories for Stream and Task Definitions
- REST API
- Shell
- UI

SPI for Deployment and Lifecycle Management

- Load Balance
- Scale Up/Down

Data Flow Developer Experience

1: Implement Spring Cloud Stream Microservice App:

```
@EnableBinding(Processor.class)
public class UpperCase {
            @Transformer(inputChannel = Processor.INPUT, outputChannel=Processor.OUTPUT)
            public String process(String message) {
                        return message.toUpperCase();
            }
}
```

2: Build and Install:

```
$ mvn clean install
```

3: Register Module with Data Flow:

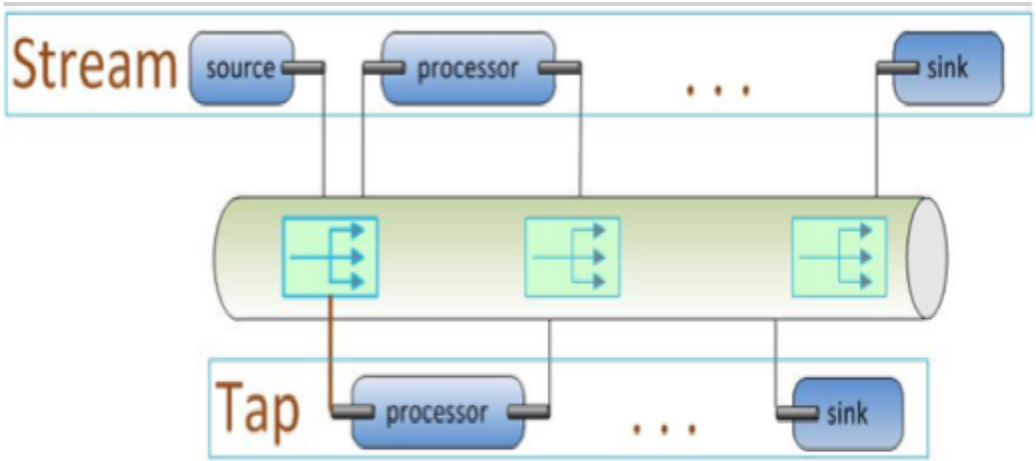
```
dataflow:> module register --name uppercase --type processor --coordinates group:artifact:version
```

4: Define Stream via DSL:

```
dataflow:> stream create demo  --definition
                        "http --server.port=9000 | uppercase | file --directory=/tmp/devnexus"
```

Wire Tap

```
dataflow:> stream create demo  --definition
                        "http --server.port=9000 | uppercase | file --directory=/tmp/devnexus"
```




```
dataflow:> stream create tap  --definition ":demo.http > counter --store=redis"
```

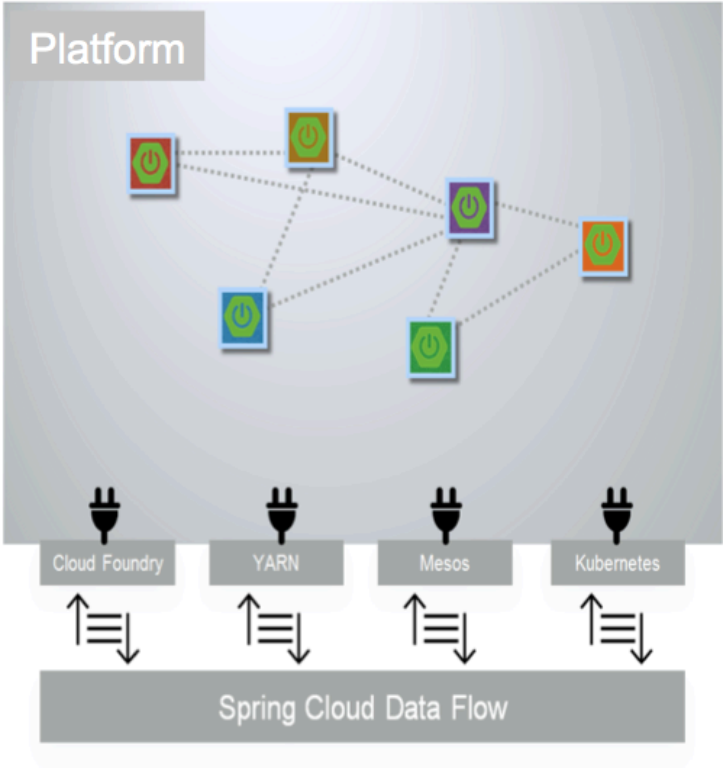
Launching Tasks via Data Flow

```
dataflow:> task create task1 --definition timestamp

dataflow:> task launch task1

dataflow:> task execution list
```

Task Name	ID	Start Time	End Time	Exit Code
task1	1	Tue Feb 16 17:06:53 EST 2016	Tue Feb 16 17:06:53 EST 2016	0

Deployer SPI

- deploy Spring Cloud Stream apps
- deploy Spring Cloud Task apps
- in both cases, pass Spring Boot Configuration Properties in an appropriate way for the target platform
- support for checking status of individual apps as well as app group (e.g. stream)



https://github.com/spring-cloud/spring-cloud-dataflow-admin-cloudfoundry
https://github.com/spring-cloud/spring-cloud-dataflow-admin-yarn
https://github.com/spring-cloud/spring-cloud-dataflow-admin-mesos
https://github.com/spring-cloud/spring-cloud-dataflow-admin-kubernetes

Links

http://cloud.spring.io/spring-cloud-stream

https://github.com/spring-cloud/spring-cloud-stream

http://cloud.spring.io/spring-cloud-task

https://github.com/spring-cloud/spring-cloud-task

http://cloud.spring.io/spring-cloud-dataflow

https://github.com/spring-cloud/spring-cloud-dataflow

Pivotal

Open.
Agile.
Cloud-Ready.