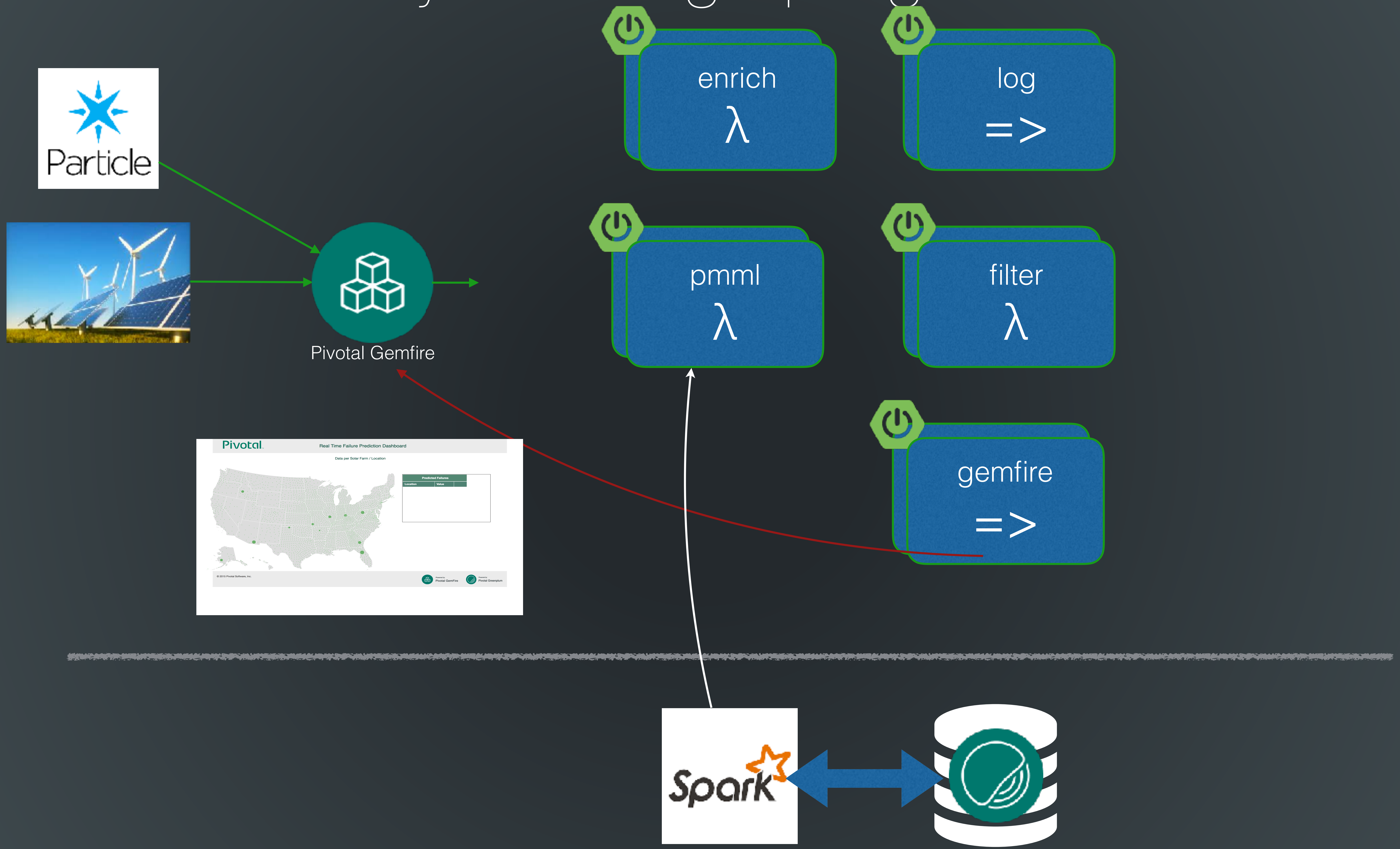
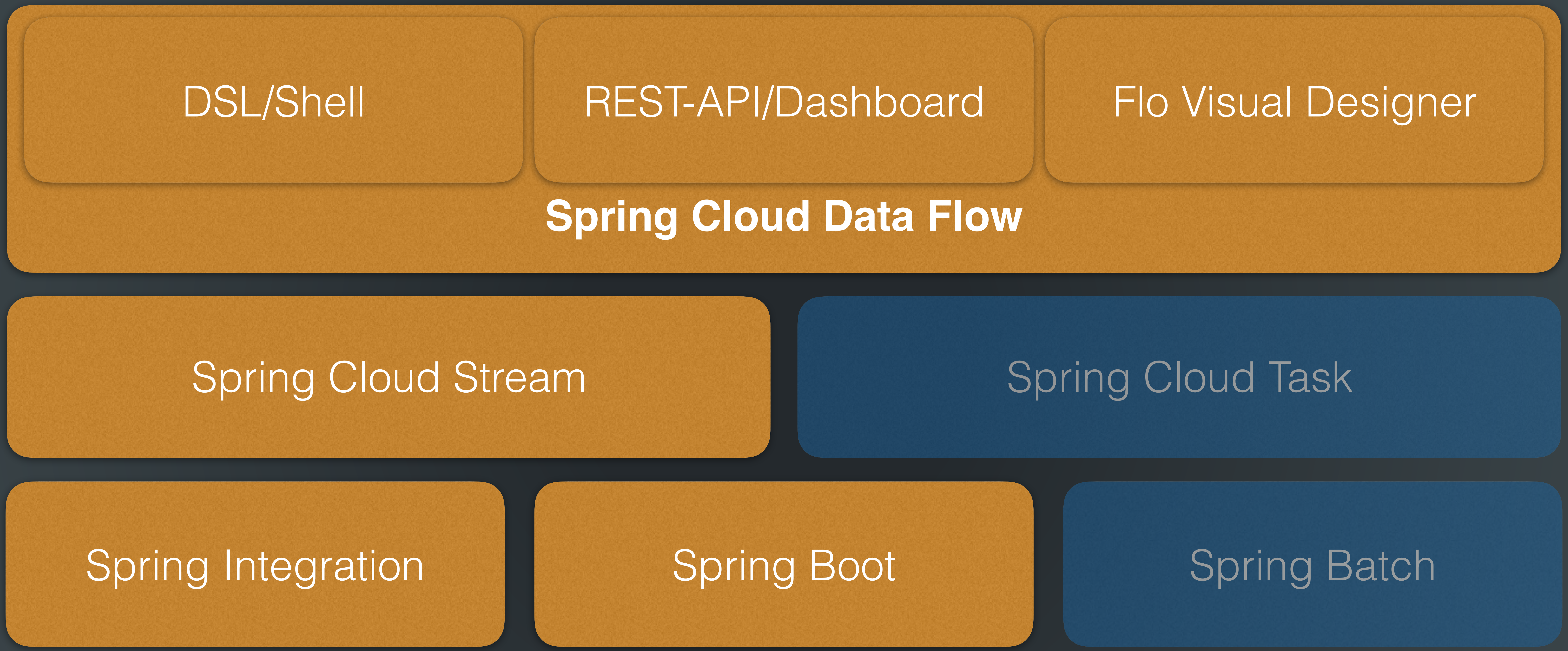
# Stream Processing
# +
# Spring Cloud Data Flow

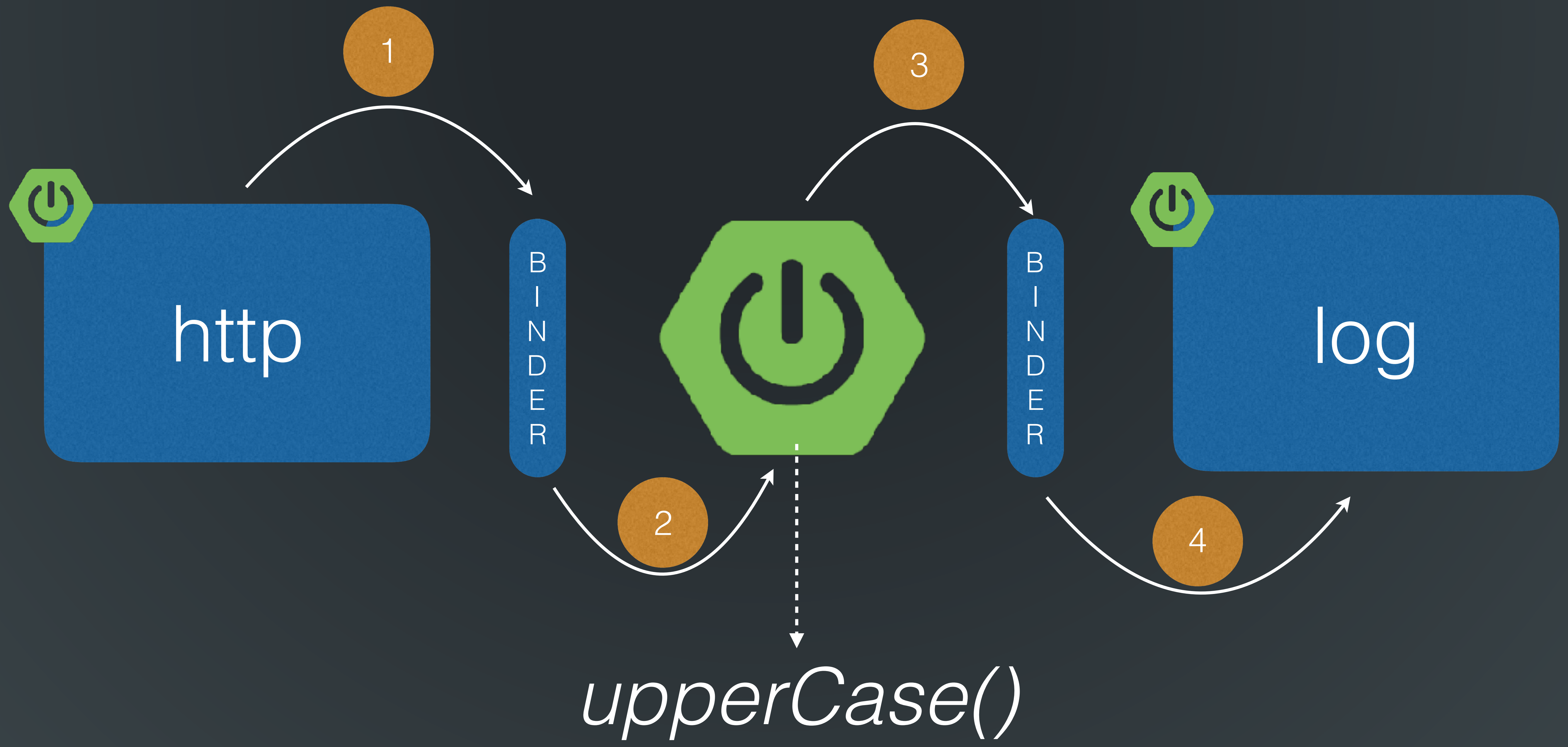
# Predictive Analytics Using Spring Cloud Data Flow




DSL/Shell
REST-API/Dashboard
Flo Visual Designer
Spring Cloud Data Flow
Spring Cloud Stream
Spring Cloud Task
Spring Integration
Spring Boot
Spring Batch

## Stream Processing in Spring Cloud Data Flow

```
dataflow:>stream create foo --definition "http --port=9001 | uppercase | log" --deploy
```

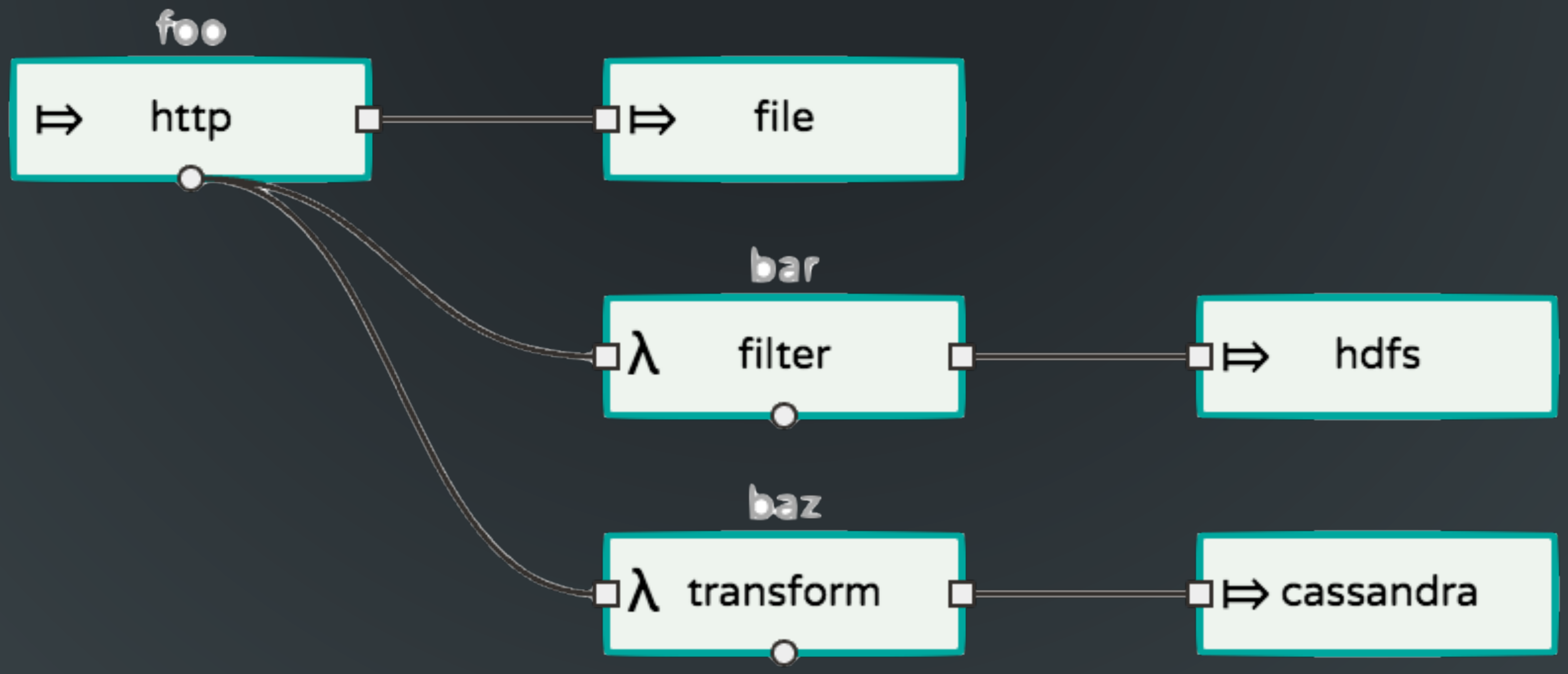


# Wiretaps in Streams

```
dataflow:>stream create foo --definition "http --port=9001 | file”
```

```
dataflow:>stream create bar --definition “:foo.http > filter | hdfs”
```

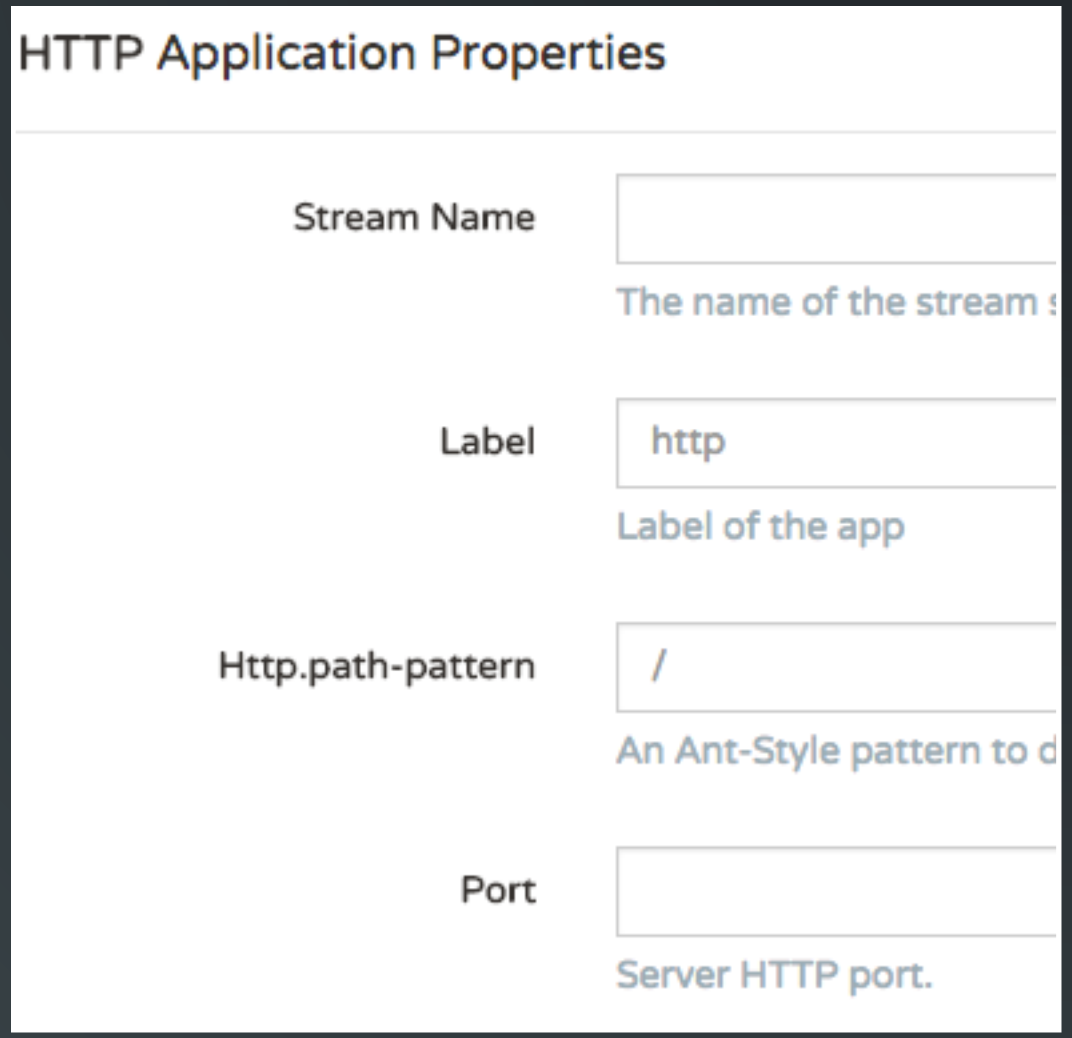
```
dataflow:>stream create baz --definition “:foo.http > transform | cassandra”
```



# Application Properties

```
dataflow:>app info source:http
```

```
║     Option Name      │     Description      │       Default        │         Type         ║
╠══════════════════════╪══════════════════════╪══════════════════════╪══════════════════════╣
║http.path-pattern     │An Ant-Style pattern to│/                    │java.lang.String      ║
║                      │determine which http requests│               │                      ║
║                      │will be captured.     │                      │                      ║
║server.port           │Server HTTP port.     │<none>                │java.lang.Integer     ║
```

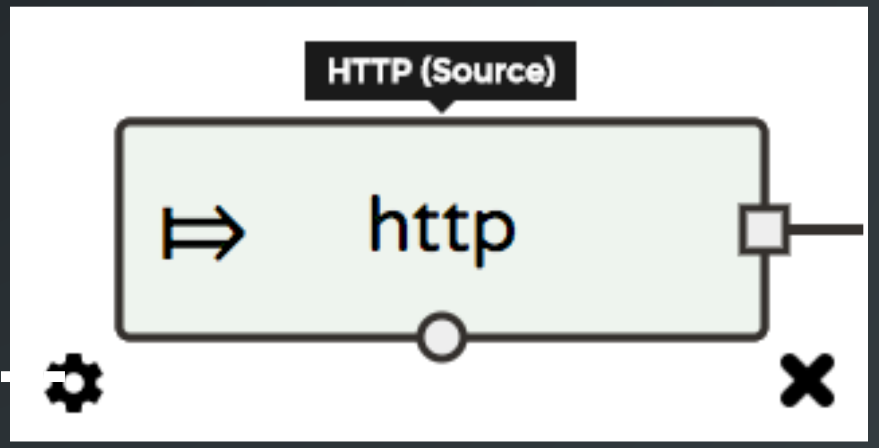

HTTP Application Properties

Stream Name
The name of the stream s

Label
http
Label of the app

Http.path-pattern
/
An Ant-Style pattern to d

Port
Server HTTP port.



# Deployment Properties

*Simple Stream*

```
dataflow:>stream create foo --definition “http --port=9001 | log”
```

*Scale-out Deployment*

```
dataflow:>stream deploy foo --properties “app.log.count=2”
```

*Partitioned Stream*

```
dataflow:>stream deploy foo --properties “app.http.producer.partitionKeyExpression=payload”
```

*Stream Binding Overrides*

```
dataflow:>stream deploy foo --properties
”app.http.spring.cloud.stream.bindings.output.destination=bar,
 app.log.spring.cloud.stream.bindings.input.destination=bar”
```

# Properties via Reference File

```
app.http.producer.partitionKeyExpression=payload
app.http.spring.cloud.stream.bindings.output.destination=bar
app.log.spring.cloud.stream.bindings.input.destination=bar
app.log.count=2
```

*Properties File*

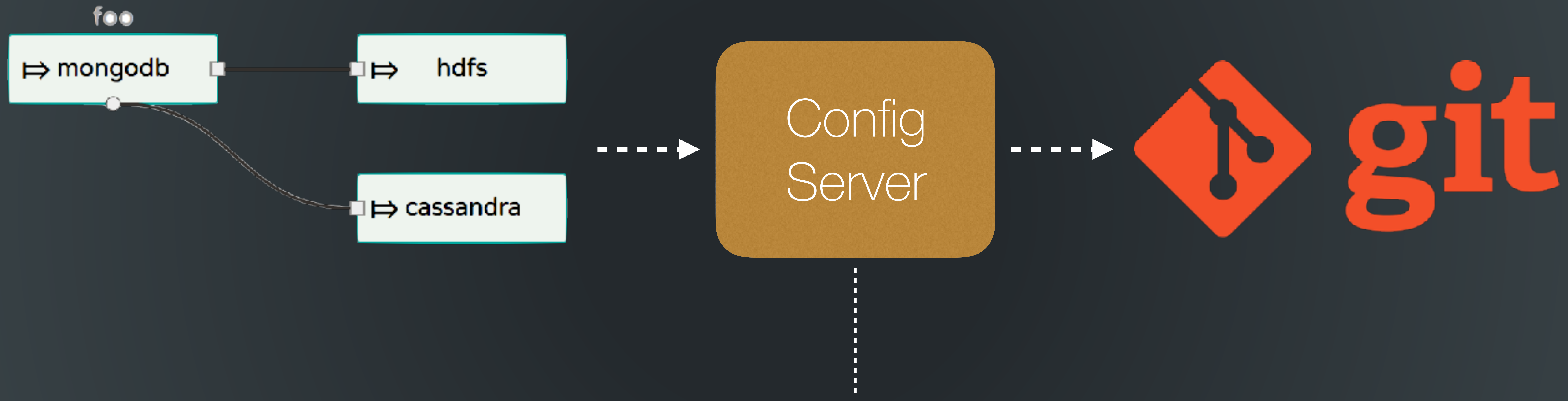
```
dataflow:>stream deploy foo --propertiesFile myprops.properties
```

*YAML File*

```
dataflow:>stream deploy foo --propertiesFile myprops.yml
```

```
app:
  http:
    producer:
      partitionKeyExpression: payload
    spring:
      cloud:
        stream:
          bindings:
            output:
              destination: bar
....
....
....
```

# Properties via Reference File



```
app.mongodb.producer.partitionKeyExpression=payload
app.mongodb.spring.cloud.stream.bindings.output.destination=bar
app.hdfs.spring.cloud.stream.bindings.input.destination=bar
app.cassandra.spring.cloud.stream.bindings.input.destination=bar
app.cassandra.count=2
```

# Explicit Broker Destinations

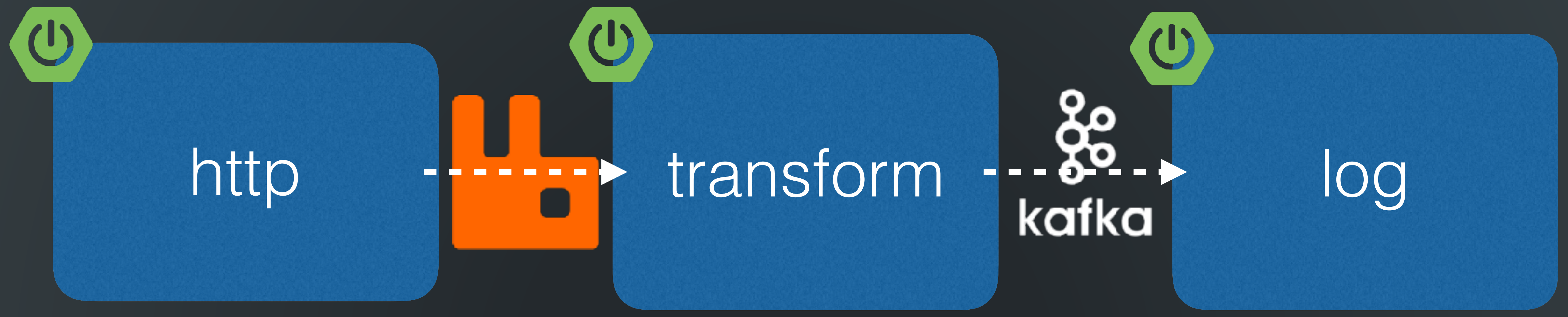
*Destination as a Source*

```
dataflow:>stream create foo --definition “:myFancySourceDestination > log”
```

*Destination as a Sink*

```
dataflow:>stream create foo --definition “http > :myFancySourceDestination”
```

# Advanced Broker Interactions



*Stream Definition*

```
dataflow:>stream create foo --definition “http | transform --expression=payload.toUpperCase() | log”
```

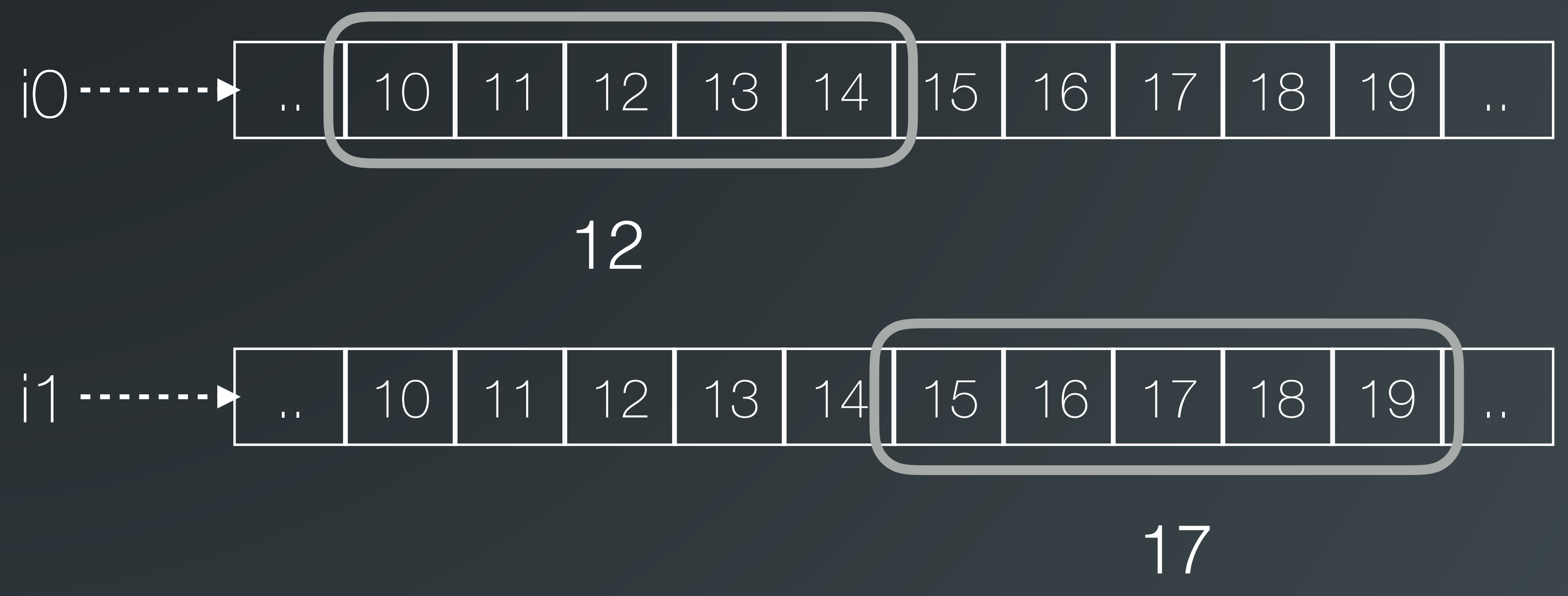
*Stream Deployment*

```
dataflow:>stream deployment foo --properties
"app.http.spring.cloud.stream.bindings.output.binder=rabbitBinder,
 app.transform.spring.cloud.stream.bindings.input.binder=rabbitBinder,
 app.transform.spring.cloud.stream.bindings.output.binder=kafkaBinder,
 app.log.spring.cloud.stream.bindings.input.binder=kafkaBinder"
```

# Reactive Stream Processing

```java
@SpringBootApplication
@EnableBinding(Processor.class)
public class SensorAverageApplication {

     @StreamListener
     @Output(Processor.OUTPUT)
     public Flux<AverageData> calculateAverage(@Input(Processor.INPUT) Flux<ReceivedSensorData> data) {
          return data.window(Duration.ofSeconds(20), Duration.ofSeconds(10))
                    .flatMap(window -> window.groupBy(sensorData -> sensorData.getId())
                    .flatMap(group -> calculateAverage(group)));
     }
...
...

}
```

```java
@EnableRxJavaProcessor
public class RxJavaTransformer {

     @Bean
     public RxJavaProcessor<String,String> processor() {
          return inputStream -> inputStream.map(data -> {
               return data;
          }).buffer(5).map(data -> String.valueOf(avg(data)));
     }
...
...

}
```


i0
..
10
11
12
13
14
15
16
17
18
19
..
12
i1
..
10
11
12
13
14
15
16
17
18
19
..
17