



链滴

Spring Cloud Data Flow 入门

作者: [crick77](#)

原文链接: <https://ld246.com/article/1603679687858>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

What

Microservice based Streaming and Batch data processing for Cloud Foundry and Kubernetes

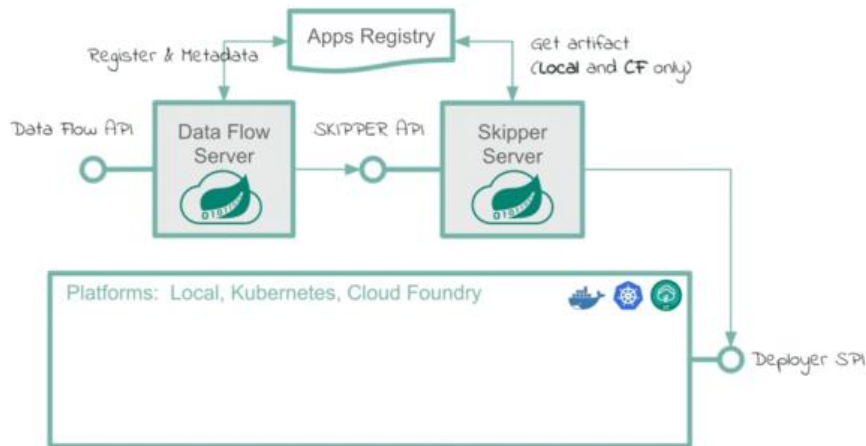
<!-- more -->

简单解释一下 **Streaming**和 **Batch**处理的区别

- **Streaming**: 应用始终处于启动状态，有数据就处理。通过消息传递中间件消耗或生成无限量的数据。
- **Batch(Task)**: 需要处理数据时启动应用，处理完成后关闭应用，只处理有限量的数据。

思考一个问题。如何动态启动一个未启动的Spring项目？

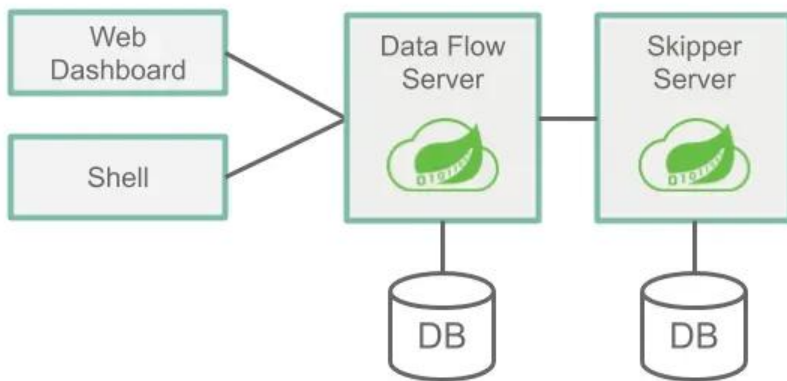
答案是配置了一个 **maven jar**包地址，或者是一个 **docker image**地址，在任务触发时，下载对应的序(镜像)并启动。



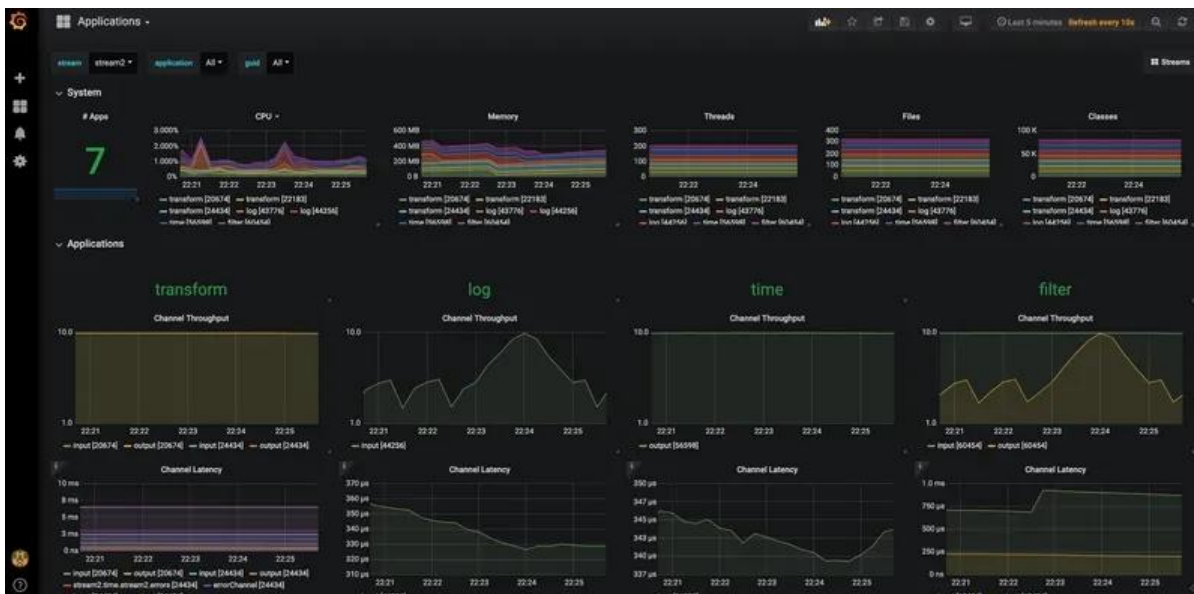
Concept

- **Application**: 指向某一个jar或docker的Stream或Batch程序
- **Task**: 通过DSL组装多个Application，定义处理任务
- **Stream**: 通过DSL组装多个Application，按照source->processor->sink定义数据处理流
- **Job**: 每个Task的执行记录，包括入参与结果记录

组件



- **Data Flow Server:** 定义、校验并执行Stream和Batch，监听应用状态，记录执行记录。
 - 基于 DSL 定义Stream和Batch
 - 通过 jar包 或者docker镜像 注册应用
 - 执行 Stream和Batch并记录执行记录，监听应用状态
- **Skipper Server:** 负责流处理发布
 - 部署 Streams到一或多个平台
 - 基于蓝绿更新策略升级或回滚 Stream
 - 持久化 Stream的描述信息，包括历史版本
- **Database:**Data Flow Server和Skipper Server依赖关系数据库。
 - 默认通过内嵌的 H2数据库启动，支持多种关系型数据库
 - 在服务启动时自动创建表结构
 - Task可以通过配置共享外部数据库作为持久化
- **Messaging Middleware:** 用于消息驱动模型的消息中间件。
 - 支持 Kafka、RabbitMQ
 - Stream必须通过消息中间件驱动
- **Monitor:** 监控系统运行指标。
 - 通过 Prometheus或InfluxDB存储
 - 通过 Grafana展示



使用

上面的概念和组件有点复杂？我们先快速启动一个应用，通过界面看一下如何使用

安装

可以通过一句docker命令启动

`docker run --rm -it -p 9393:9393 springcloud/spring-cloud-dataflow-server:2.6.3`

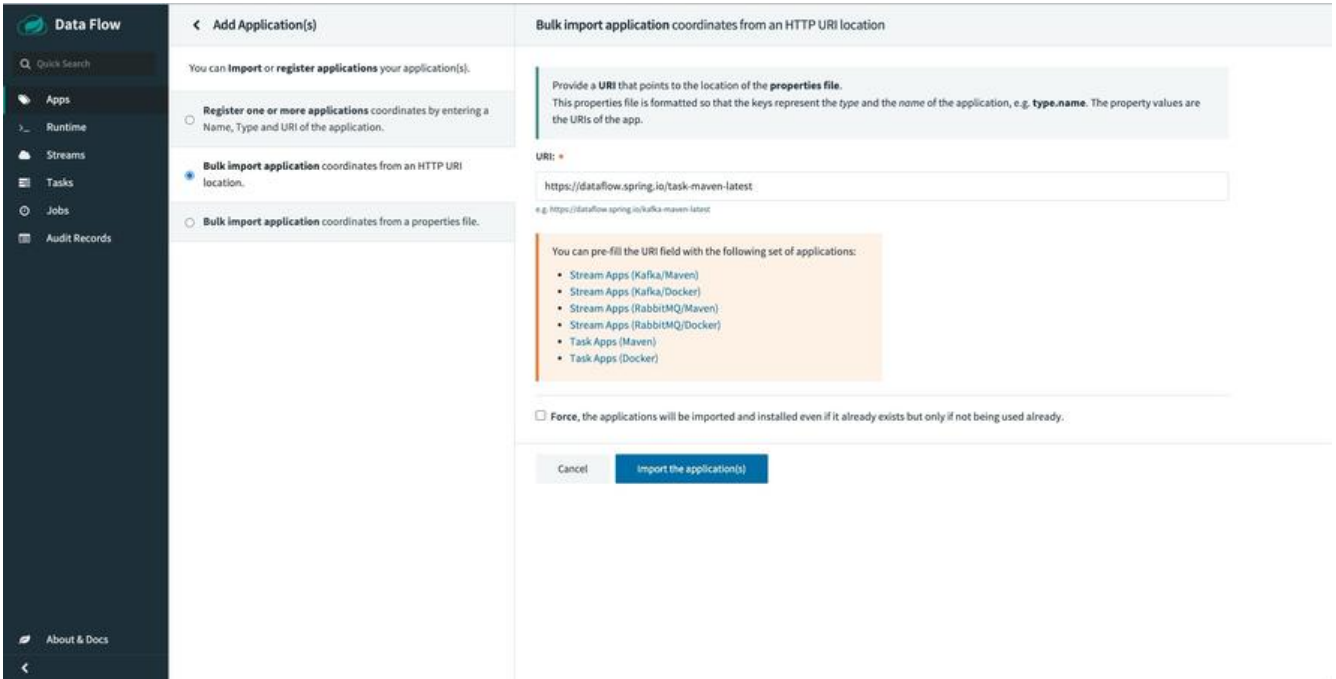
通过 <http://localhost:9393/dashboard/#/about> 访问管理界面

The screenshot shows the 'About & Docs' page of the Spring Cloud Data Flow dashboard. The page title is 'Data Flow' and the subtitle is 'About & Docs'. The main content area is titled 'Data Flow Server Implementation' and provides details about the server version and features. The server name is 'spring-cloud-dataflow-server' and the version is '2.6.3'. The 'Enabled Features' section lists: Streams (checked), Tasks (checked), Schedules (unchecked), Grafana (unchecked), and Wavefront (unchecked). The 'Security Information' section shows: Authentication Enabled (unchecked) and Authenticated (unchecked). The 'Version Information' section lists: Implementation Version: 2.6.3 (Spring Cloud Dataflow-server), Core: 2.6.3 (Spring Cloud Data Flow Core), Dashboard: 2.5.1 (Spring Cloud Dataflow UI), and Shell: 2.6.3 (Spring Cloud Data Flow Shell). The 'Runtime Environment - Skipper Deployer' section lists: Implementation Version: N/A, Name: N/A, Spi Version: N/A, Java Version: N/A, Platform Api Version: N/A, Platform Client Version: N/A, Platform Host Version: N/A, and Platform Type: N/A. A 'Copy Details to Clipboard' button is visible in the top right corner of the content area.

注: Schedules未开启的原因是单机版的安装方式不支持定时任务。

创建 Application

提供了三种方式，但是要注意，不支持直接上传jar包或者发现运行中的应用，只能指定maven仓库地址(或者docker地址)进行拉取

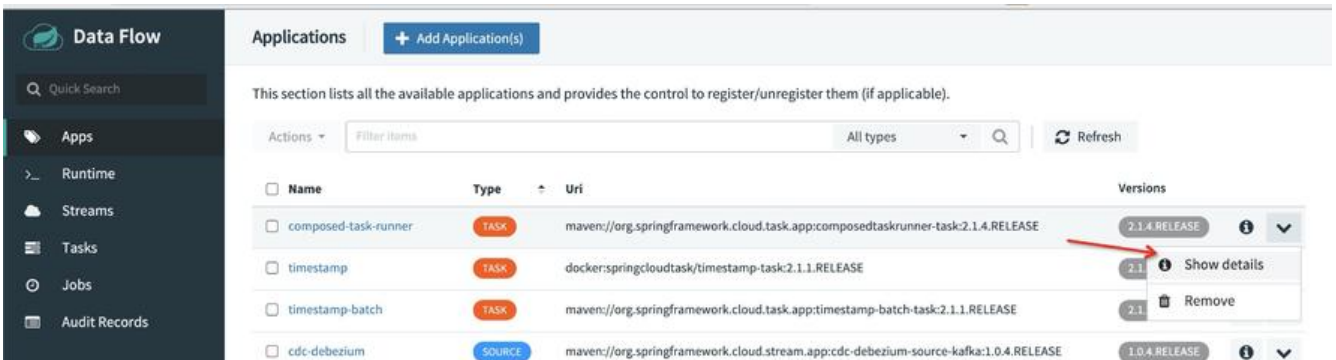


选用第二种(import from an HTTP URI location), 并选择官方**Task Apps Maven**示例, 本质上是官方拉取下列配置, 在 第三种(import from properties) 中输入下列信息也是同样的效果。

```
task.timestamp=maven://org.springframework.cloud.task.app:timestamp-task:2.1.1.RELEASE
task.timestamp.metadata=maven://org.springframework.cloud.task.app:timestamp-task:jar:metadata:2.1.1.RELEASE
task.composed-task-runner=maven://org.springframework.cloud.task.app:composdtaskrunner-task:2.1.4.RELEASE
task.composed-task-runner.metadata=maven://org.springframework.cloud.task.app:composdtaskrunner-task:jar:metadata:2.1.4.RELEASE
task.timestamp-batch=maven://org.springframework.cloud.task.app:timestamp-batch-task:2.1.1.RELEASE
task.timestamp-batch.metadata=maven://org.springframework.cloud.task.app:timestamp-batch-task:jar:metadata:2.1.1.RELEASE
```

每个 **application**需要指定应用地址和 **metadata**信息地址。**metadata**会在使用过程中明确参数输入单。如果未指定 **metadata**地址, 会尝试通过应用进行提取。

导入的应用可以在应用列表中查看, 可以通过**show details**查看应用的 **metadata**



metadata通过KV形式展示

The screenshot shows the configuration page for 'Application composed-task-runner' in the Spring Cloud Data Flow console. The page displays the URI 'maven://org.springframework.cloud.task.app:composetaskrunner-task:2.1.4.RELEASE' and the version '2.1.4.RELEASE'. A table lists various properties and their descriptions:

Property	Description
oauth2-client-credentials-scopes	OAuth2 Authorization scopes (Used for the client credentials grant). java.util.Set<java.lang.String>
split-thread-keep-alive-seconds	Split's thread keep alive seconds. Default is 60. java.lang.Integer (Default value: 60)
dataflow-server-password	The optional password for the dataflow server that will receive task launch requests. Used to access the the dataflow server using Basic Authentication. Not used if {@link #dataflowServerAccessToken} is set. java.lang.String
composed-task-properties	The properties to be used for each of the tasks as well as their deployments. java.lang.String
split-thread-max-pool-size	Split's maximum pool size. Default is {@code Integer.MAX_VALUE}. java.lang.Integer
split-thread-core-pool-size	Split's core pool size. Default is 4; java.lang.Integer (Default value: 4)
dataflow-server-access-token	The optional OAuth2 Access Token. java.lang.String
graph	The DSL for the composed task directed graph. java.lang.String
oauth2-client-credentials-client-id	The OAuth2 Client Id (Used for the client credentials grant). If not null, then the following properties are ignored: dataflowServerUsernamedataflowServerPassworddataflowServerAccessToken java.lang.String
split-thread-wait-for-tasks-to-	Whether to wait for scheduled tasks to complete on shutdown, not interrupting running tasks and executing all tasks in the queue. Default is false;

****注意: 这里有一个坑。 ****如果你按照官方示例 <https://github.com/spring-cloud/spring-cloud-task/tree/master/spring-cloud-task-samples/timestamp> 新建了一个 Spring Cloud Task, 会发现没法查看 metadata 信息, 原因是因为官方项目中缺少配置文件

classpath*/META-INF/dataflow-configuration-metadata.properties

内容为 configuration-properties.classes 执行配置类

configuration-properties.classes=wang.yuheng.SyncPrestoProperties

Spring Cloud Data Flow 对 metadata 解析过程如下:

```
private static Resource[] visibleConfigurationMetadataResources(ClassLoader classLoader) throws IOException {
    ResourcePatternResolver resourcePatternResolver = new PathMatchingResourcePatternResolver(classLoader);
    Resource[] configurationResources = resourcePatternResolver.getResources(VISIBLE_PROPERTIES);

    Resource[] deprecatedSpringConfigurationResources = resourcePatternResolver
        .getResources(DEPRECATED_SPRING_CONFIGURATION_PROPERTIES);
    if (deprecatedSpringConfigurationResources.length > 0) {
        logger.warn("The use of " + DEPRECATED_SPRING_CONFIGURATION_PROPERTIES + " is a deprecated. Please use "
            + VISIBLE_PROPERTIES + " instead.");
    }

    Resource[] deprecatedDataflowConfigurationResources = resourcePatternResolver
        .getResources(DEPRECATED_DATAFLOW_CONFIGURATION_PROPERTIES);
    if (deprecatedDataflowConfigurationResources.length > 0) {
        logger.warn("The use of " + DEPRECATED_DATAFLOW_CONFIGURATION_PROPERTIES
            + " is a deprecated. Please use " + VISIBLE_PROPERTIES + " instead.");
    }

    return concatArrays(configurationResources, deprecatedSpringConfigurationResources,
        deprecatedDataflowConfigurationResources);
}

// this is superseded by name prefixed with dataflow and will get removed in future
private static final String DEPRECATED_SPRING_CONFIGURATION_PROPERTIES = "classpath*/META-INF/spring-configuration-metadata-whitelist.properties";

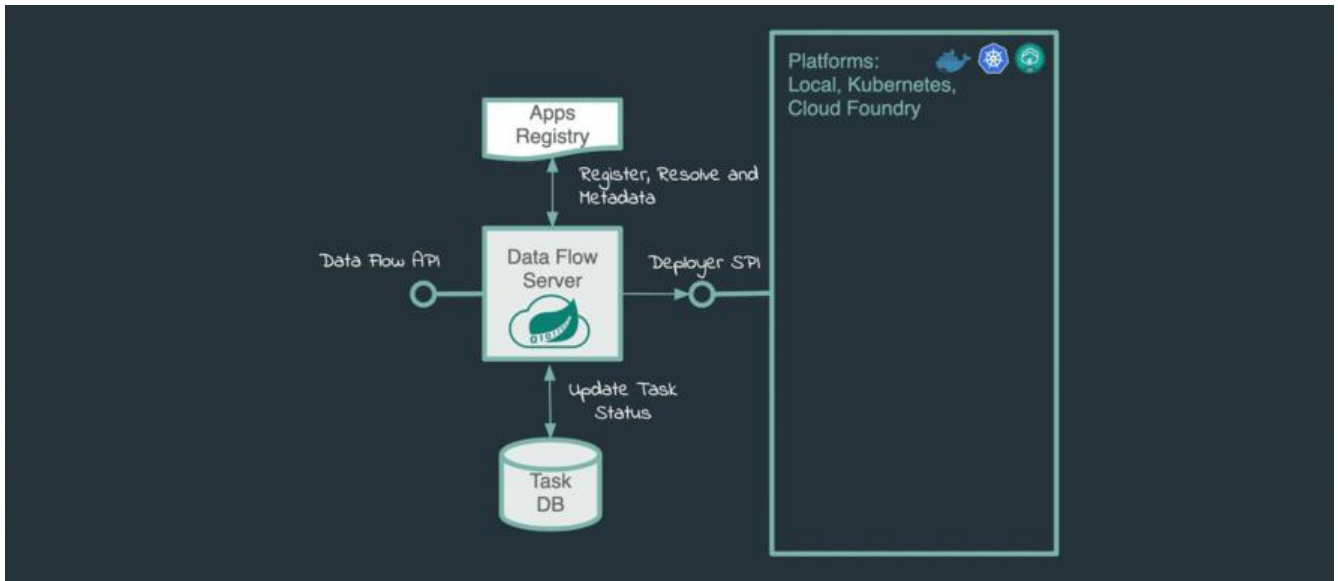
// this is superseded by VISIBLE_PROPERTIES
private static final String DEPRECATED_DATAFLOW_CONFIGURATION_PROPERTIES = "classpath*/META-INF/dataflow-configuration-metadata-whitelist.properties";

private static final String VISIBLE_PROPERTIES = "classpath*/META-INF/dataflow-configuration-metadata.properties";

private static final String CONFIGURATION_PROPERTIES_CLASSES = "configuration-properties.classes";
```

另一个坑是，如果是 **maven** 地址，会先从 ****localRepository(.m2)**** 获取 **jar** 文件解析 **metadata**。如果你设置的是 **docker**，每次都会通过网络进行 **docker** 鉴权并下载，无法直接判断本地是否存在，以速度会比 **maven** 慢很多。

Task

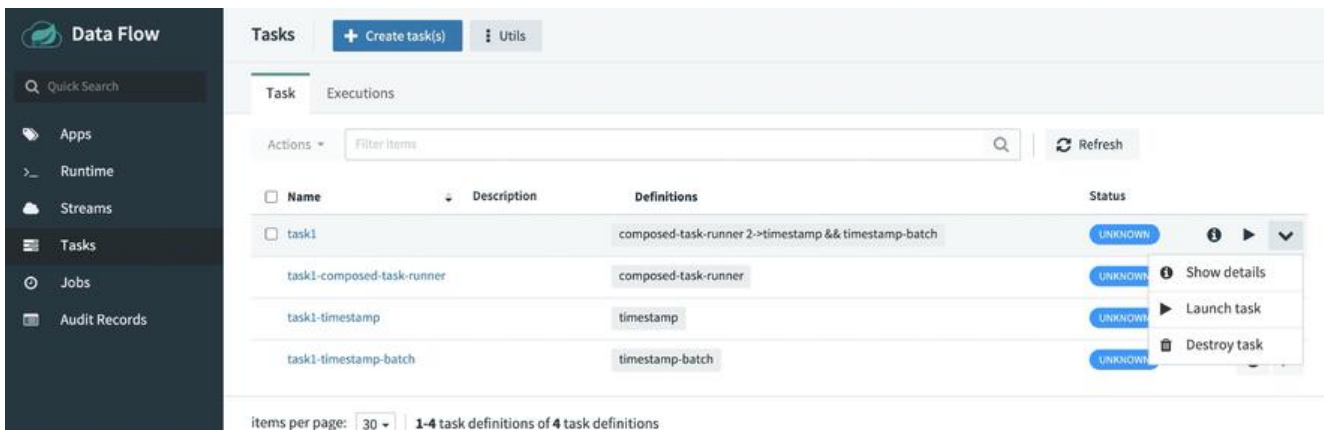


创建 Task

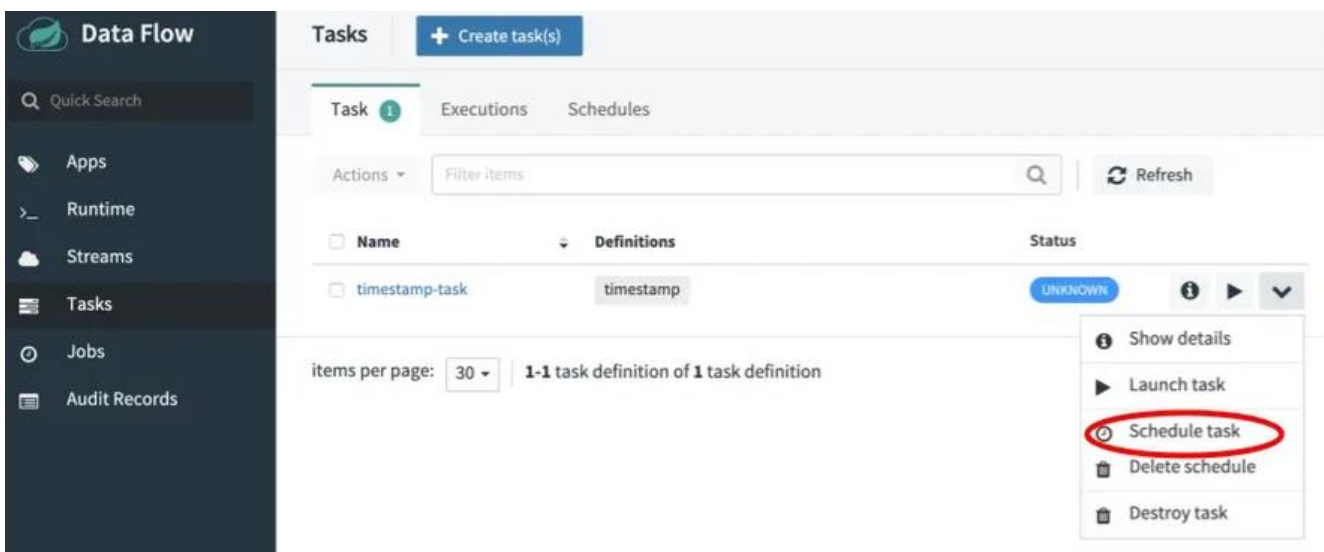
可以通过页面或者DSL定义Task

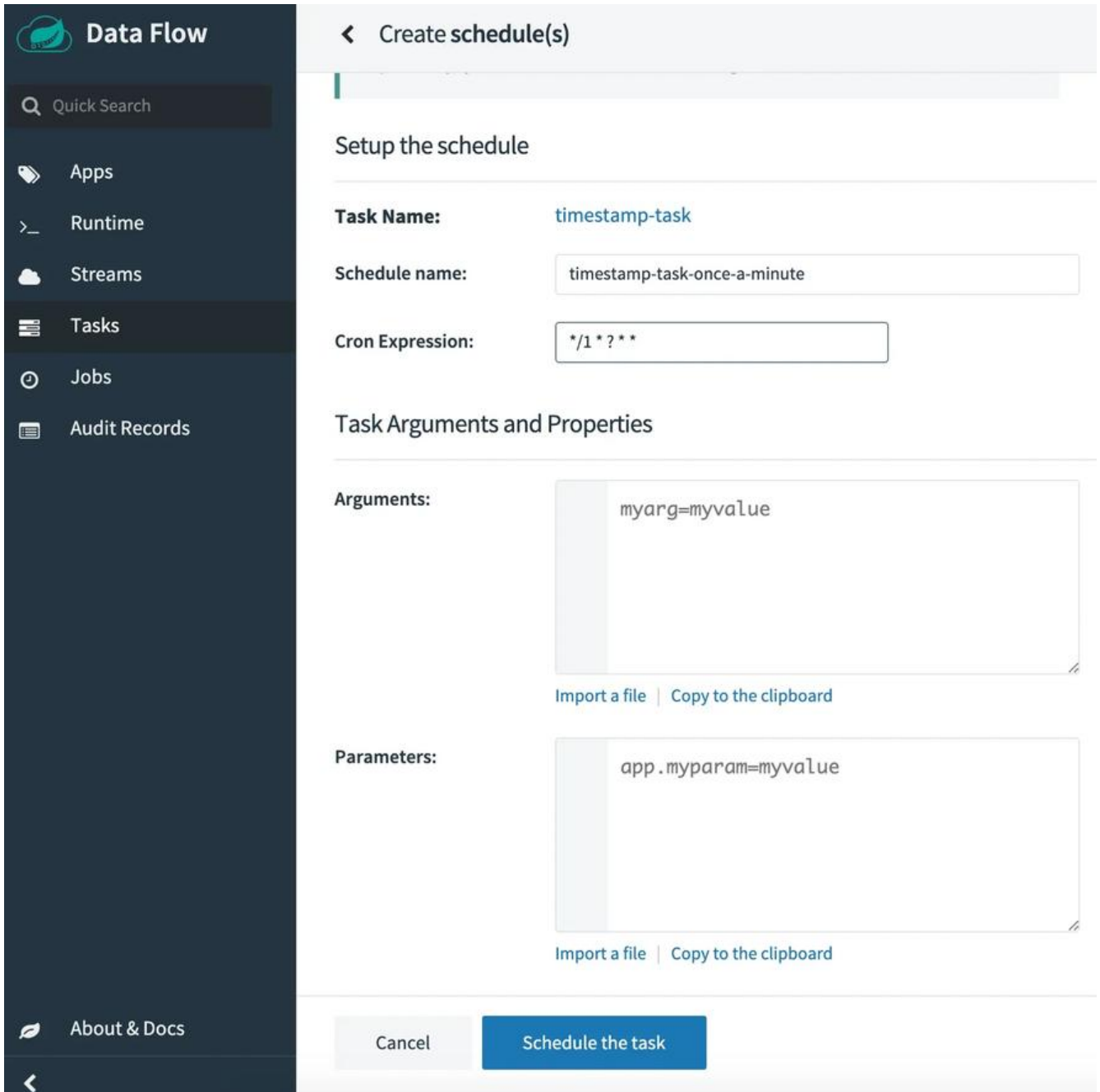
The screenshot shows the 'Create a task' interface in the Data Flow console. The top bar displays the DSL: `1 timestamp 2->composed-task-runner && timestamp-batch`. Below this, the 'Applications' section is expanded to show 'TASK' nodes, including 'composed-tas...', 'timestamp', and 'timestamp-batch'. The main area displays a visual task graph starting with 'START', followed by a 'timestamp' node, which then branches into 'composed-task-runner' and 'timestamp-batch' nodes, both of which converge at an 'END' node. The 'composed-task-runner' node has a '2' next to it, indicating a parallel execution of two instances. The interface includes a search bar, a zoom level of 100%, and a 'Fit to Content' button. At the bottom, there are 'Cancel' and 'Create Task' buttons.

创建好的 **Task** 会存在于 **Tasks** 列表中，可以在列表页查看 **Task** 详情或者执行任务(生成 **job**)



如果通过Cloud方式进行部署，可以指定定时任务，相比普通任务，可以通过 **cron expression** 设置时

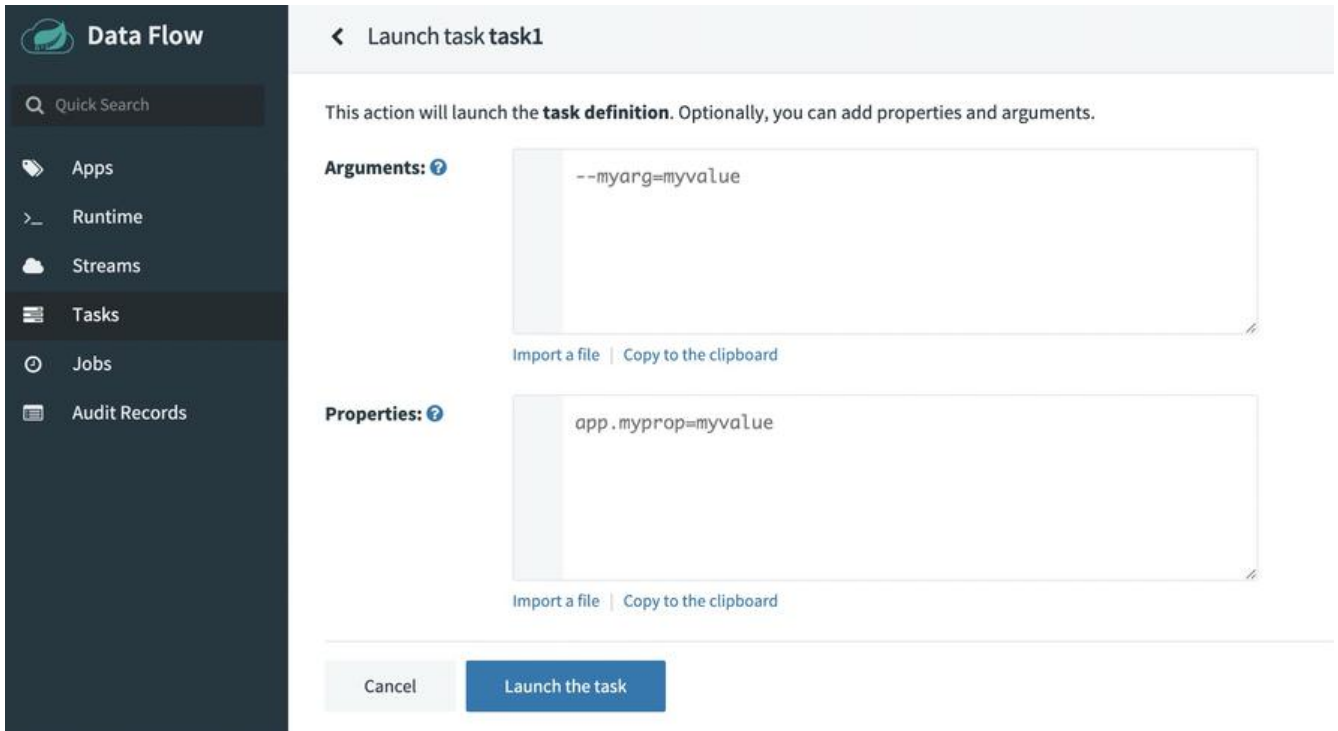




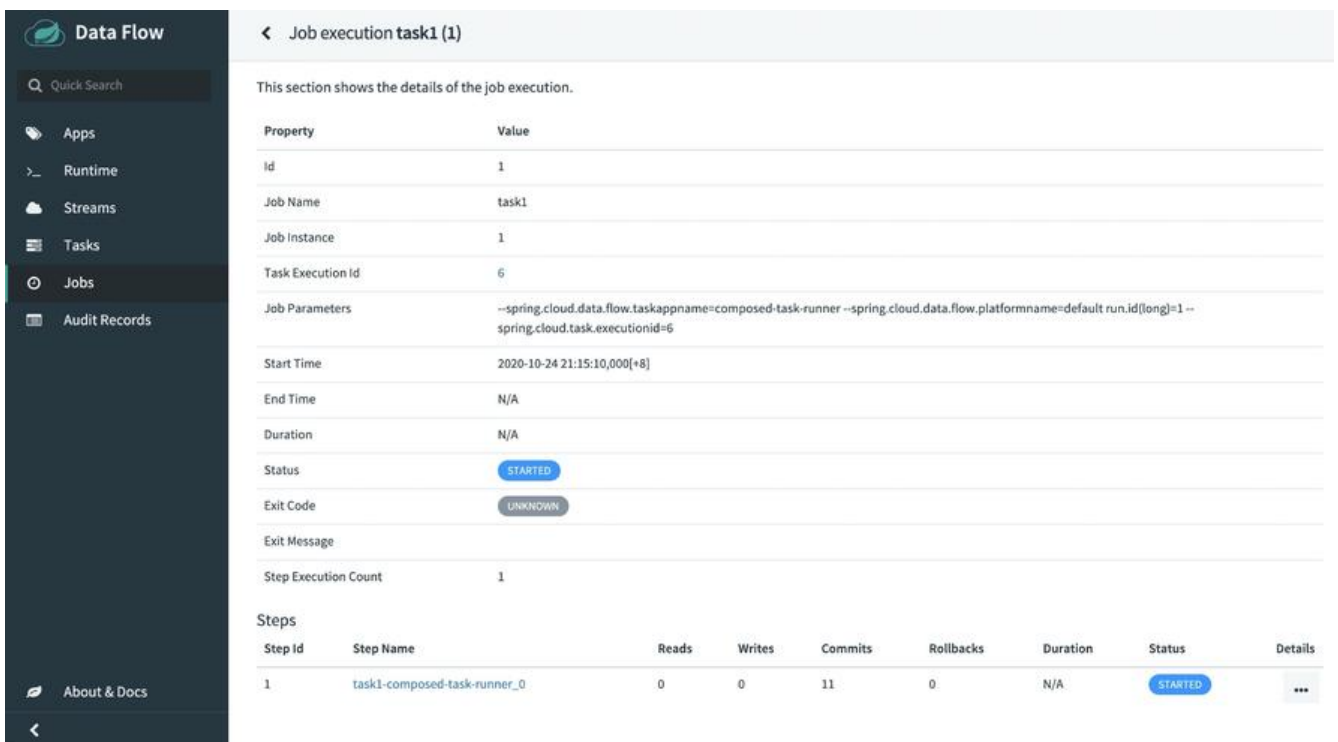
运行 Task

可以在**Task**列表中选择要执行的**Task**，并通过**KV List**的方式指定入参及properties。明确一下这2概念

- properties: 配置参数，比如数据库配置
- Arguments: 方法入参，即 `String[] args`



每次执行会生成一个 Job记录，可以在 Task或者 Job列表中查看执行状态、日志等信息

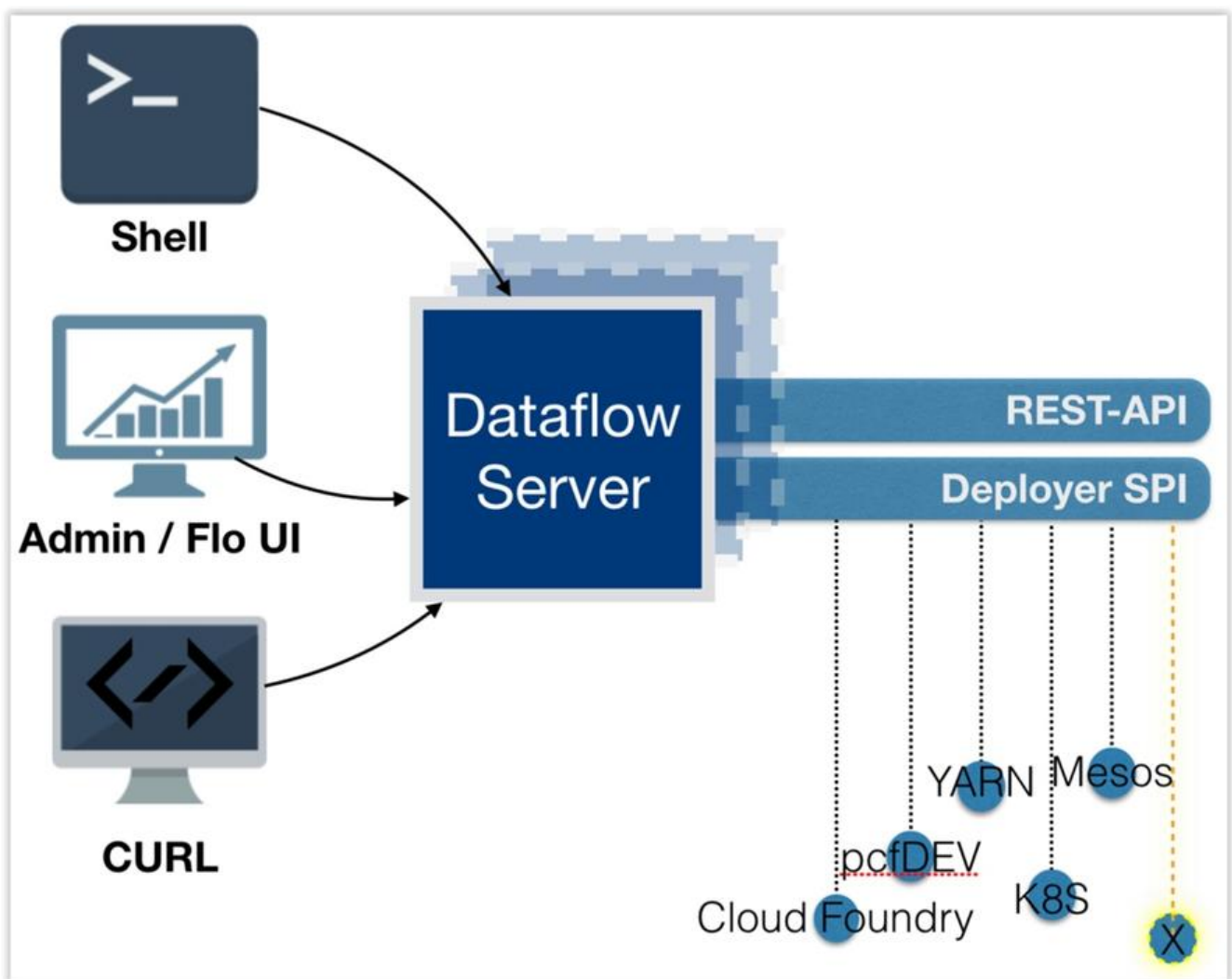


审计留痕

Audit Records页面会看到触发数据及执行记录等信息，包括不同方式(Restful、Shell、WebUI)触的行为。开启登录验证后可以查看操作人信息。

ID	Created On	Action	Operation	Operation Id	Created By	Data	Platform name
188	2020-10-25 06:12:45,000[+8]	UPDATE	STREAM	weekly	N/A	...9092[n"]	default
186	2020-10-25 06:12:44,000[+8]	UPDATE	STREAM	weekly	N/A	...ker:9092	N/A
178	2020-10-25 06:07:05,000[+8]	UNDEPLOY	STREAM	Stream1	N/A	...ker:9092	N/A
168	2020-10-25 05:59:57,000[+8]	CREATE	STREAM	weekly	N/A	...el=debug	N/A
166	2020-10-24 21:27:06,000[+8]	UPDATE	STREAM	Stream1	N/A	...ker:9092	N/A
167	2020-10-24 21:27:06,000[+8]	DEPLOY	STREAM	Stream1	N/A	...erties"-:[])	default
157	2020-10-24 21:23:08,000[+8]	CREATE	STREAM	Stream2	N/A	...licher s3	N/A

Stream



使用 Stream 需要依赖 skipper-server 和消息中间件，通过 docker-compose 启动

```
version: '3'
```

```
services:
  mysql:
```

image: mysql:5.7.25
container_name: dataflow-mysql
environment:
 MYSQL_DATABASE: dataflow
 MYSQL_USER: root
 MYSQL_ROOT_PASSWORD: rootpw
expose:
 - 3306

kafka-broker:
image: confluentinc/cp-kafka:5.3.1
container_name: dataflow-kafka
expose:
 - "9092"
environment:
 - KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://kafka-broker:9092
 - KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181
 - KAFKA_ADVERTISED_HOST_NAME=kafka-broker
 - KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR=1
depends_on:
 - zookeeper

zookeeper:
image: confluentinc/cp-zookeeper:5.3.1
container_name: dataflow-kafka-zookeeper
expose:
 - "2181"
environment:
 - ZOOKEEPER_CLIENT_PORT=2181

dataflow-server:
image: springcloud/spring-cloud-dataflow-server:2.6.3
container_name: dataflow-server
ports:
 - "9393:9393"
environment:
 - spring.cloud.dataflow.applicationProperties.stream.spring.cloud.stream.kafka.binder.bro
ers=PLAINTEXT://kafka-broker:9092
 - spring.cloud.dataflow.applicationProperties.stream.spring.cloud.stream.kafka.streams.bi
der.brokers=PLAINTEXT://kafka-broker:9092
 - spring.cloud.dataflow.applicationProperties.stream.spring.cloud.stream.kafka.binder.zkN
des=zookeeper:2181
 - spring.cloud.dataflow.applicationProperties.stream.spring.cloud.stream.kafka.streams.bi
der.zkNodes=zookeeper:2181
 - spring.cloud.skipper.client.serverUri=http://skipper-server:7577/api
 - SPRING_DATASOURCE_URL=jdbc:mysql://mysql:3306/dataflow
 - SPRING_DATASOURCE_USERNAME=root
 - SPRING_DATASOURCE_PASSWORD=rootpw
 - SPRING_DATASOURCE_DRIVER_CLASS_NAME=org.mariadb.jdbc.Driver
depends_on:
 - kafka-broker
entrypoint: "./wait-for-it.sh mysql:3306 -- java -jar /maven/spring-cloud-dataflow-server.jar
volumes:

```
- ${HOST_MOUNT_PATH:-.}:${DOCKER_MOUNT_PATH:-/root/scdf}
```

app-import:

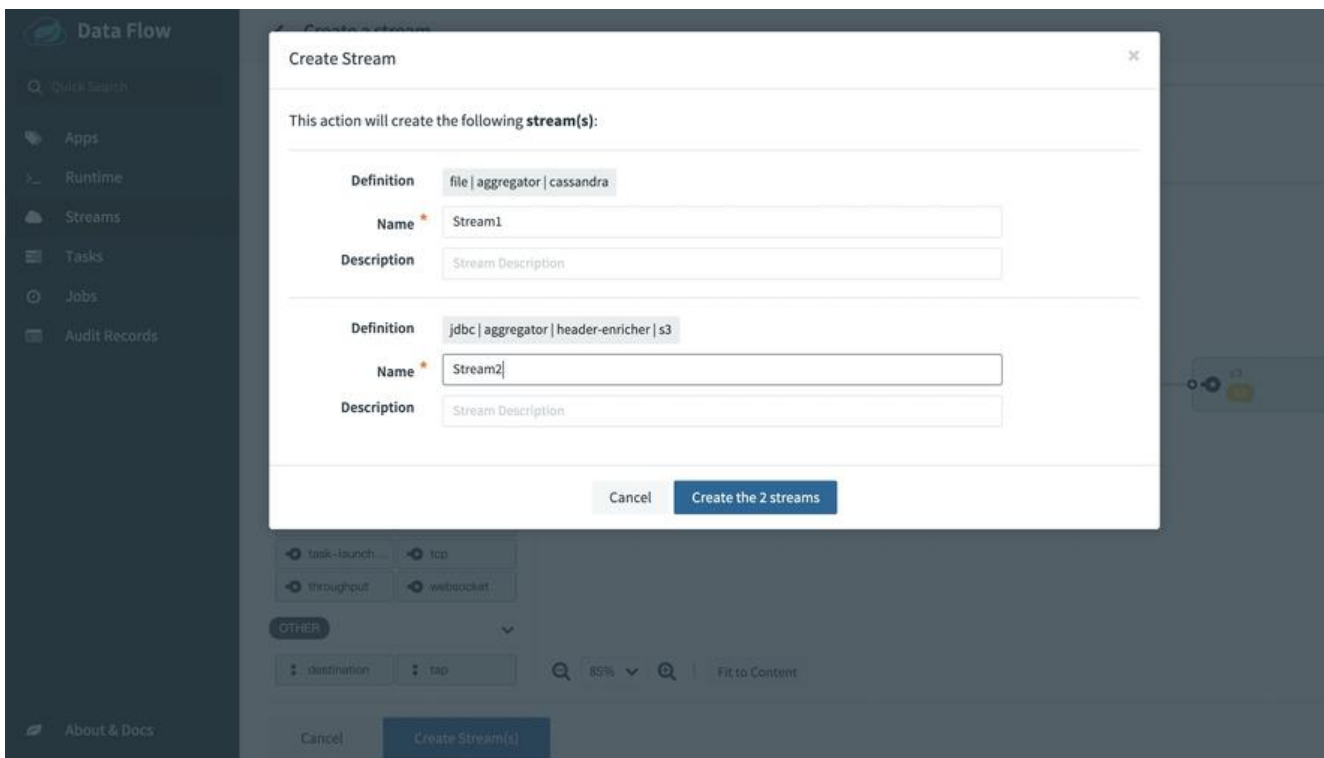
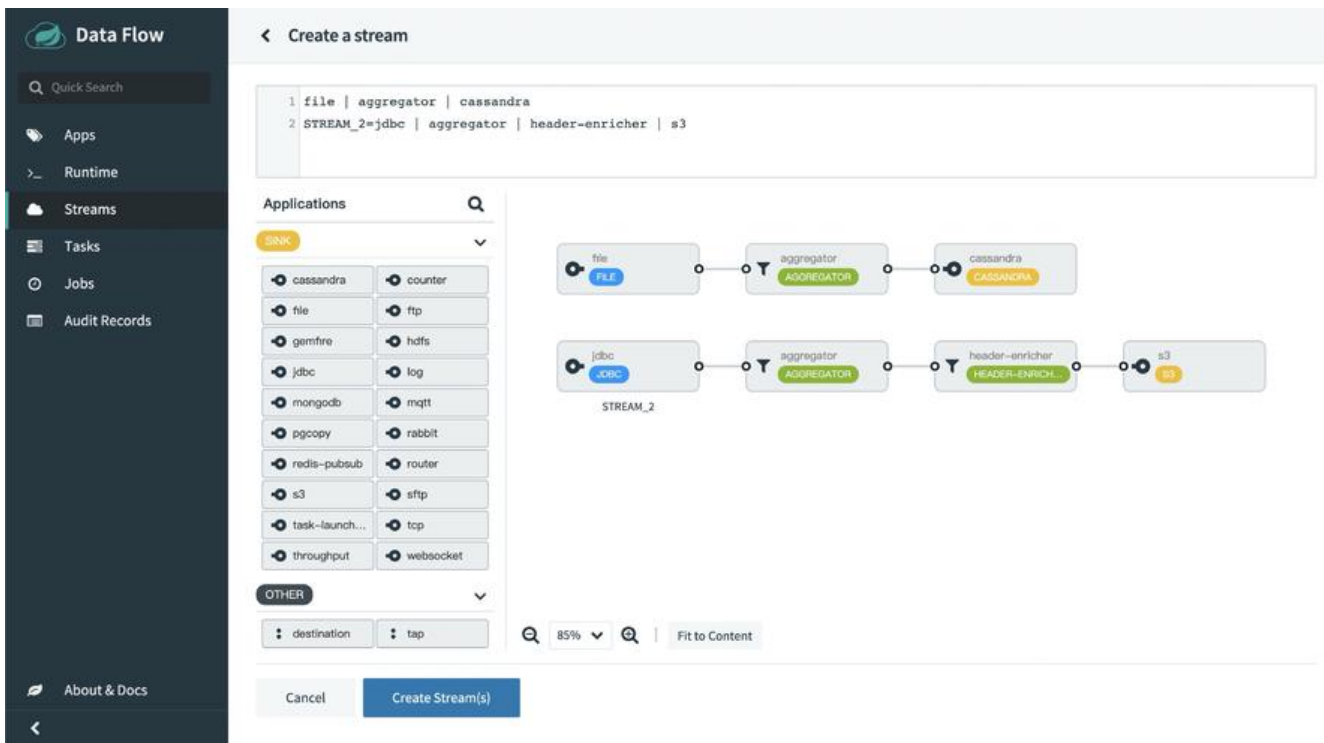
```
image: springcloud/openjdk:2.0.0.RELEASE
container_name: dataflow-app-import
depends_on:
  - dataflow-server
command: >
  /bin/sh -c "
    ./wait-for-it.sh -t 180 dataflow-server:9393;
    wget -qO- 'http://dataflow-server:9393/apps' --post-data='uri=${STREAM_APPS_URI:-https://dataflow.spring.io/kafka-maven-latest&force=true}';
    echo 'Stream apps imported'
    wget -qO- 'http://dataflow-server:9393/apps' --post-data='uri=${TASK_APPS_URI:-https://dataflow.spring.io/task-maven-latest&force=true}';
    echo 'Task apps imported'"
```

skipper-server:

```
image: springcloud/spring-cloud-skipper-server:2.5.2
container_name: skipper
ports:
  - "7577:7577"
  - "20000-20105:20000-20105"
environment:
  - SPRING_CLOUD_SKIPPER_SERVER_PLATFORM_LOCAL_ACCOUNTS_DEFAULT_PORTRAN
E_LOW=20000
  - SPRING_CLOUD_SKIPPER_SERVER_PLATFORM_LOCAL_ACCOUNTS_DEFAULT_PORTRAN
E_HIGH=20100
  - SPRING_DATASOURCE_URL=jdbc:mysql://mysql:3306/dataflow
  - SPRING_DATASOURCE_USERNAME=root
  - SPRING_DATASOURCE_PASSWORD=rootpw
  - SPRING_DATASOURCE_DRIVER_CLASS_NAME=org.mariadb.jdbc.Driver
entrypoint: "./wait-for-it.sh mysql:3306 -- java -jar /maven/spring-cloud-skipper-server.jar"
volumes:
  - ${HOST_MOUNT_PATH:-.}:${DOCKER_MOUNT_PATH:-/root/scdf}
```

创建 Stream

同样可以通过页面或者DSL按照 **source->processor->sink** 定义Stream。每个 **source->processor-sink** 是一个Stream，可以同时创建多个独立的Stream

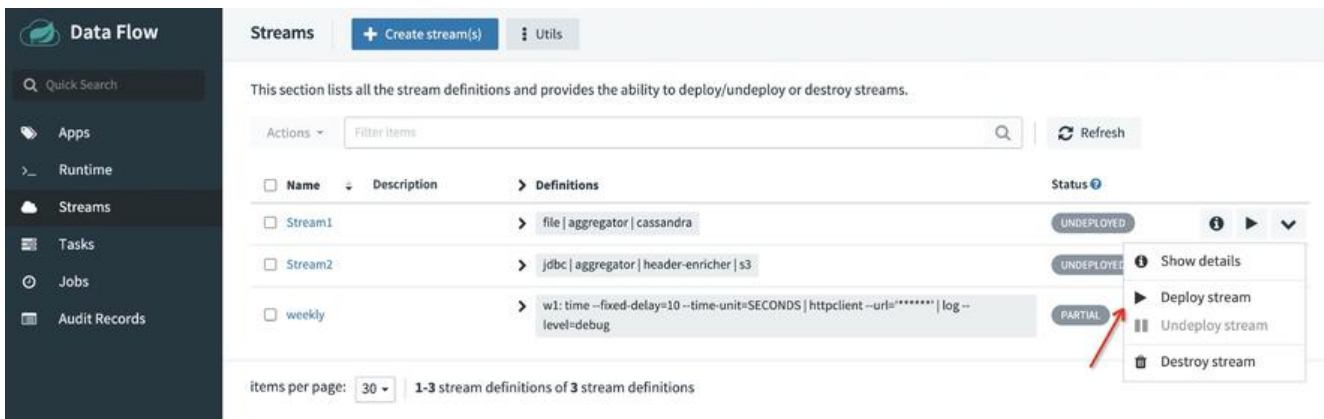


发布 Stream

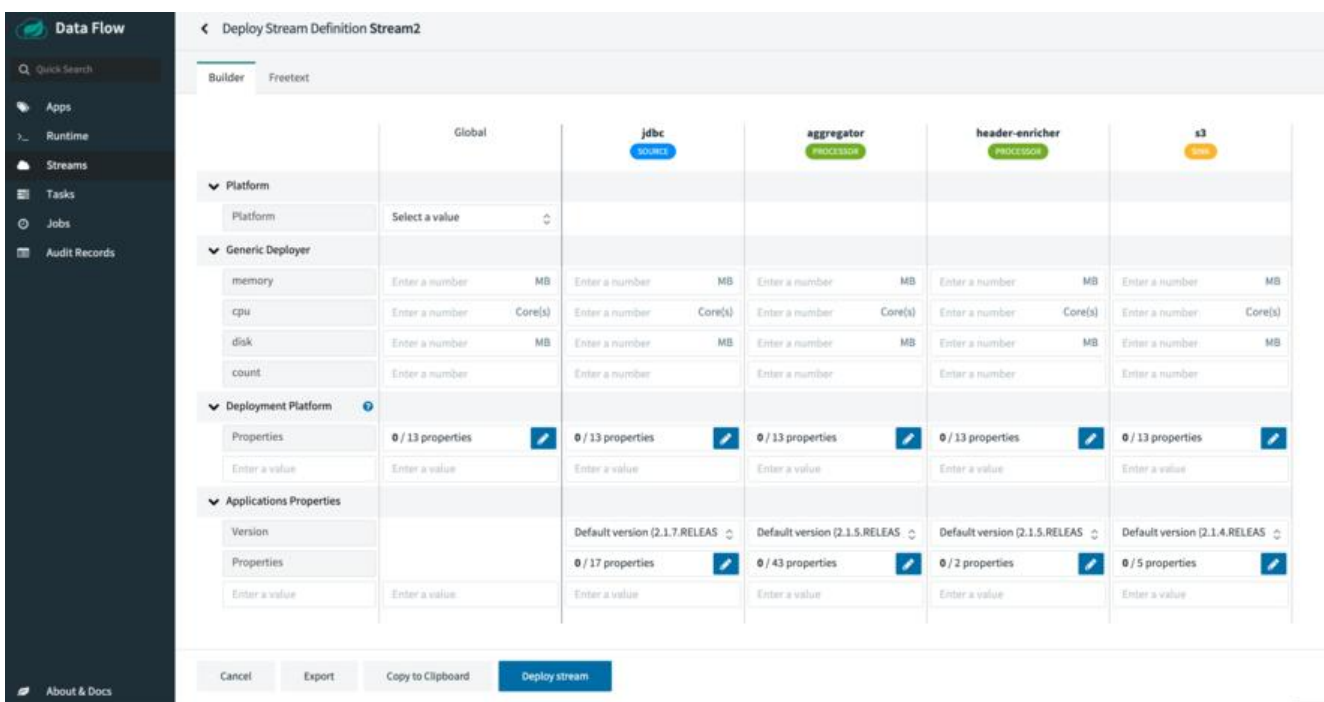
Stream 创建成功后，需要通过 **skipper-server**发布至**Runtime**，基于消息队列进行驱动并执行数据处理

注意: 如果要使用流，需要找一个支持流的持久化，如 **Redis**

在列表页可以查看 **Stream**详情，或者发布 **Stream**。



发布时需要指定每个Application的properties，并且指定资源限制。如果是发布至k8s环境，会根据集群配置进行分配。



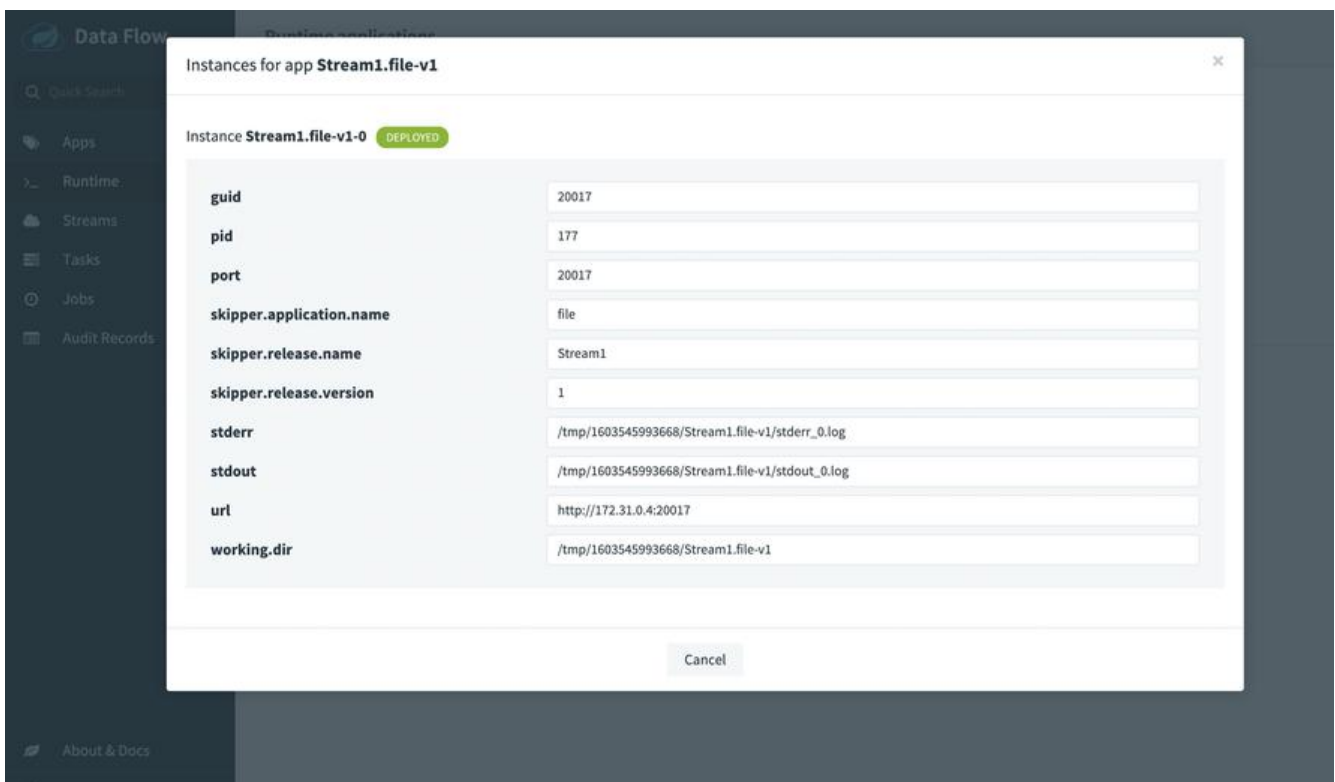
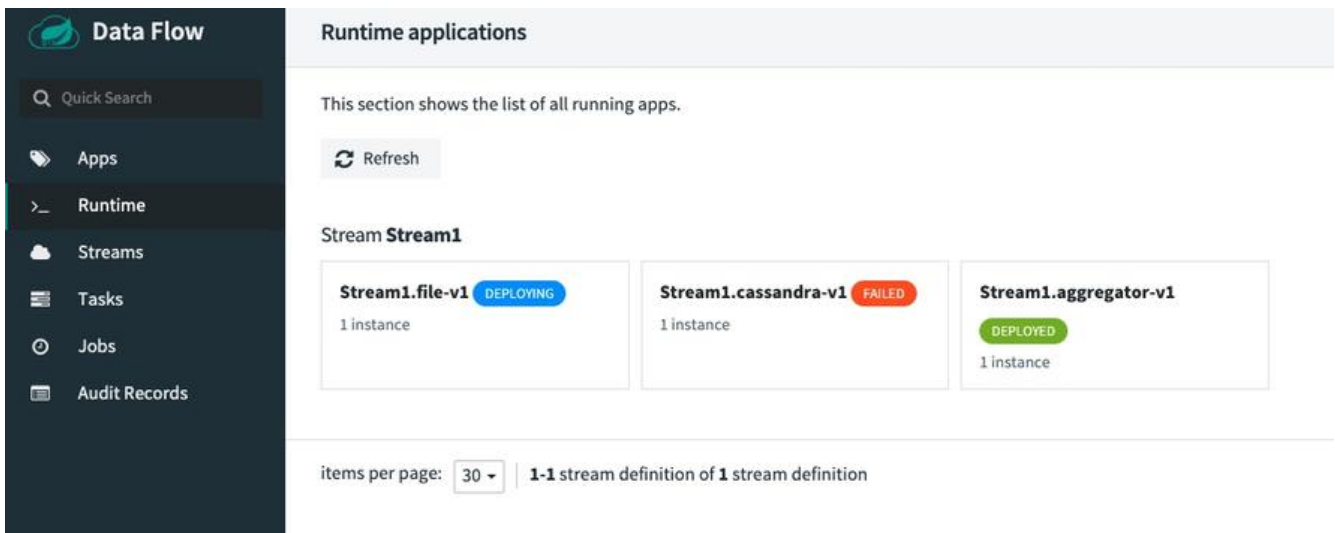
细节: 会按照label在 kafka中创建对应的topic

```

dataflow-kafka | [2020-10-24 22:02:11,834] INFO Creating topic weekly.w1 with configuration {} and initial partition assignment Map(0 -> ArrayBuffer(1001)) (kafka.zk.AdminZkClient)
dataflow-kafka-zookeeper | [2020-10-24 22:02:11,978] INFO Got user-level KeeperException when processing sessionId:0x100053673690004 type:setData cxid:0x54 zxid:0xbe txntype:-1 reqpath:n/a Error Path:/con
fig/topics/weekly.w1 Error:KeeperErrorCode = NoNode for /config/topics/weekly.w1 (org.apache.zookeeper.server.PrepareRequestProcessor)
dataflow-kafka | [2020-10-24 22:02:12,252] INFO [Controller id=1001] New topics: [Set(weekly.w1)], deleted topics: [Set()], new partition replica assignment [Map(weekly.w1-0 -> Vector(1001))] (kafka.c
ontroller.KafkaController)
dataflow-kafka | [2020-10-24 22:02:12,256] INFO [Controller id=1001] New partition creation callback for weekly.w1-0 (kafka.controller.KafkaController)
dataflow-kafka | [2020-10-24 22:02:12,273] TRACE [Controller id=1001 epoch=5] Changed partition weekly.w1-0 state from NonExistentPartition to NewPartition with assigned replicas 1001 (state.change.l
ogger)
dataflow-kafka | [2020-10-24 22:02:12,304] TRACE [Controller id=1001 epoch=5] Changed state of replica 1001 for partition weekly.w1-0 from NonExistentReplica to NewReplica (state.change.logger)
dataflow-kafka | [2020-10-24 22:02:12,519] TRACE [Controller id=1001 epoch=5] Changed partition weekly.w1-0 from NewPartition to OnlinePartition with state LeaderAndIsr(leader=1001, leaderEpoch=0, isr
s=[1001], zkVersion=0) (state.change.logger)
dataflow-kafka | [2020-10-24 22:02:12,525] TRACE [Controller id=1001 epoch=5] Sending become-leader LeaderAndIsr request PartitionState(controllerEpoch=5, leader=1001, leaderEpoch=0, isr=1001, zkVersi
on=0, replicas=1001, isNew=true) to broker 1001 for partition weekly.w1-0 (state.change.logger)
dataflow-kafka | [2020-10-24 22:02:12,538] TRACE [Controller id=1001 epoch=5] Sending UpdateMetadata request PartitionState(controllerEpoch=5, leader=1001, leaderEpoch=0, isr=[1001], zkVersion=0, repli
cas=[1001], offlineReplicas=[]) to brokers Set(1001) for partition weekly.w1-0 (state.change.logger)
dataflow-kafka | [2020-10-24 22:02:12,542] TRACE [Controller id=1001 epoch=5] Changed state of replica 1001 for partition weekly.w1-0 from NewReplica to OnlineReplica (state.change.logger)
dataflow-kafka | [2020-10-24 22:02:12,593] INFO [RequestSendThread controllerId=1001] Controller 1001 connected to kafka-broker:9092 (id: 1001 rack: null) for sending state change requests (kafka.cont
roller.RequestSendThread)
dataflow-kafka | [2020-10-24 22:02:12,618] TRACE [Broker id=1001] Received LeaderAndIsr request PartitionState(controllerEpoch=5, leader=1001, leaderEpoch=0, isr=1001, zkVersion=0, replicas=1001, isNe
w=true) correlation id 3 from controller 1001 epoch 5 for partition weekly.w1-0 (state.change.logger)
dataflow-kafka | [2020-10-24 22:02:12,705] TRACE [Broker id=1001] Handling LeaderAndIsr request correlationId 3 from controller 1001 epoch 5 starting the become-leader transition for partition weekly.
w1-0 (state.change.logger)

```

发布后可以在Runtime查看对应的 Stream状态及详情



通过docker-compose启动，所以会在 skipper-server 这台机器上运行相关Jar包作为消息Consumer

```

root@09161eda08dc:~# ps -ef |grep java
root      1      0   3 19:03 ?        00:05:59 java -jar /maven/spring-cloud-skipper-server.jar
root     284      1  10 22:00 ?        00:02:12 /opt/openjdk/bin/java -jar /root/.m2/repository/org/springframework/cloud/stream/app/httpclient-processor-kafka/2.1.5.RELEASE/httpclient-processor-kafka-2.1.5.RELEASE.jar
root     635      0  22:21 pts/0    00:00:00 grep --color=auto java

```

Shell

上述操作均可通过 命令行 or Restful进行调用，并且配置都可以导出为对应的配置文件(DSL、Task、Stream等)


```

dataflow>app list

```

app	source	processor	sink	task
s3		object-detection	sftp	composed-task-runner
time		counter	redis-pubsub	timestamp-batch
tcp		groovy-transform	mqtt	timestamp
ftp		image-recognition	mongodb	
jdbc		scriptable-transform	websocket	
http		tcp-client	task-launcher-dataflow	
mail		twitter-sentiment	cassandra	
rabbit		pose-estimation	ftp	
sftp-dataflow		splitter	tcp	
load-generator		tensorflow	counter	
trigger		prml	file	
js		tasklaunchrequest-transform	router	
file		httpClient	rabbit	
cdc-debezium		header-enricher	throughput	
triggertask		python-jython	log	
gemfire-cq		transform	s3	
mongodb		bridge	jdbc	
mqtt		groovy-filter	pgcopy	
tcp-client		aggregator	gemfire	
gemfire		grpc	hdfs	
syslog		filter		
sftp		python-http		
loggregator				
twitterstream				

```

dataflow>stream
stream all
stream platform-list
dataflow>stream list

```

Stream Name	Description	Stream Definition	Status
Stream1		file aggregator cassandra	The app or group is known to the system, but is not currently deployed
Stream2		jdbc aggregator header-enricher s3	The app or group is known to the system, but is not currently deployed
weekly		nl: time --fixed-delay=10 --time-unit=SECONDS httpClient --url='*****' log --level=debug	In the case of multiple apps, some have successfully deployed, while others have not

```

dataflow::

```

基于此就可配合当前使用的 **CICD**完成 **devops**。

想象一个场景，在你提交代码后，自动发布至某个环境等待运行。

1. push code
2. maven package
3. deploy maven repository
4. create application
5. create task
6. run task

观点

现实骨感，未来部分期待，如果你正在 All in Spring Cloud。

整体基于**Spring Cloud**，但是又拥抱**docker**。如果只是**docker**就可以去掉很多限制，就像是**k8s**和**pring Cloud** 组件本身就存在诸多重复

优点

1. 基于Spring微服务，无切换成本，可独立开发、测试
2. 完整的闭环，提供了从服务定制、管理、运行、监控全生命周期解决方案
3. 拖拽式UI操作界面，配合DSL，配置简单，页面看起来很现代(你知道我在讽刺谁)

中立

1. 未提供特定的计算引擎集群，类似 Flink、Spark 等
2. 不能覆盖工作流场景
3. 稳定性，目测现阶段上生产可以很快成为 **contributors**
4. 资源占用(看到有吐槽，但是未测试，不发表评论)

缺点

1. 仅基于Spring微服务，比如一行命令 or 一句sql 必须通过Spring Cloud Task(or Stream) 编写。过Java编写job， 你需要一个高版本的JDK
2. 依赖 **maven repo** (可能提供了http、ftp等其他方式， 但是笔者没找到。。)
3. 如果涉及到大数据处理， 还是要依靠Hadoop中的模块。那么为什么混用， 而不是直接使用全家桶？

资料

- <https://dataflow.spring.io/docs/concepts/tooling/>
- https://www.springcloud.cc/spring-cloud-dataflow.html#_launching_a_task_externally_from_spring_cloud_data_flow