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

What

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

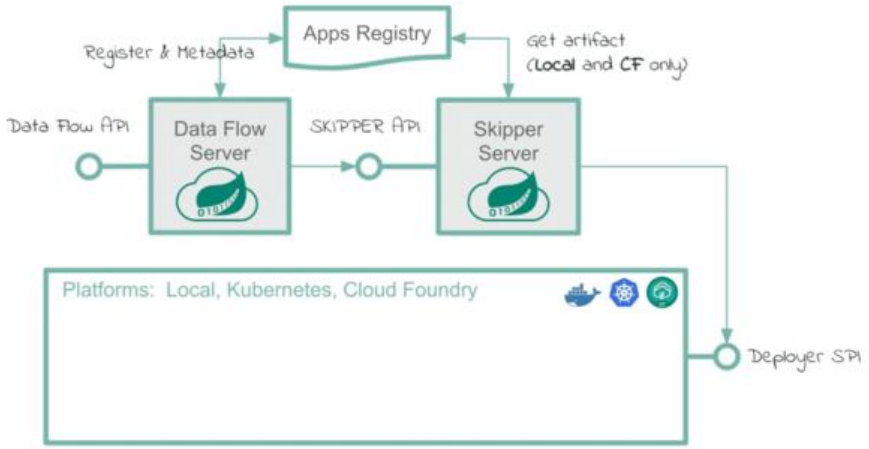
<!-- more -->

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。
- 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。

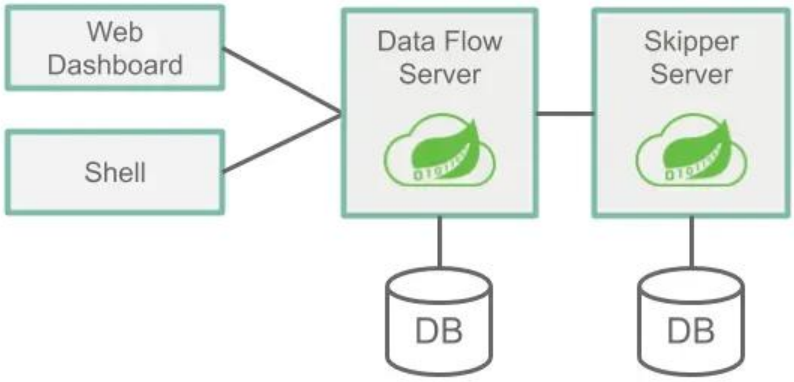


Concept

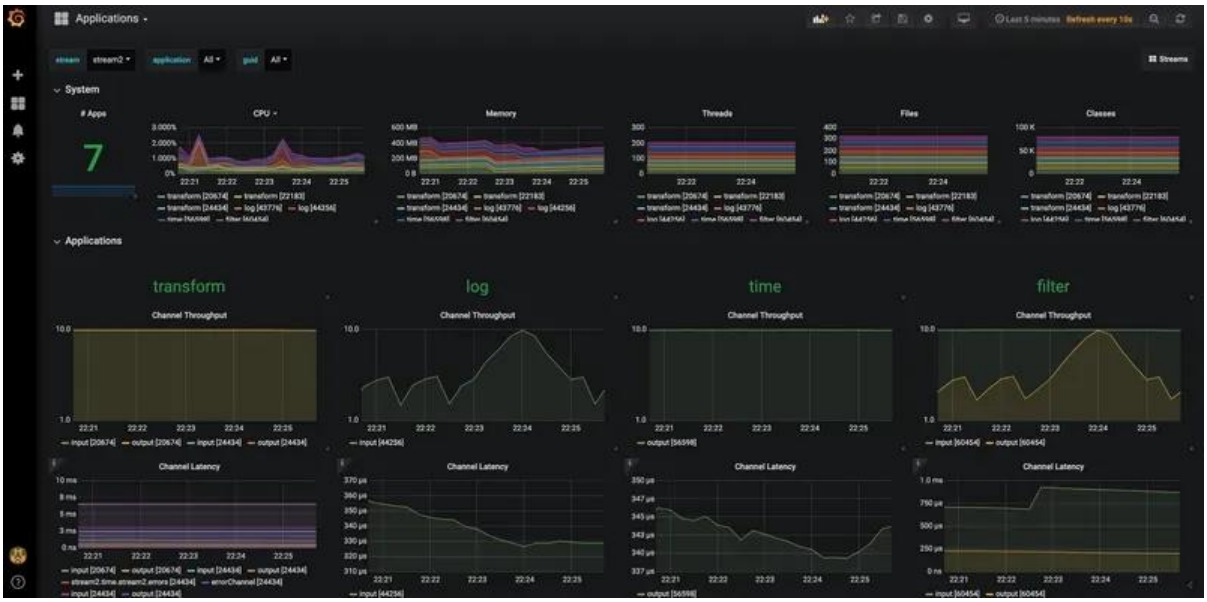
- **Application**: 指向某一个**jar**或**docker**的Stream或Batch程序
- **Task**: 通过**DSL**组装多个Application，定义处理任务
- **Stream**: 通过**DSL**组装多个Application，按照source->processor->sink定义数据处理流
- **Job**: 每个Task的执行记录，包括入参及结果记录

组件

原文链接：Spring Cloud Data Flow 入门



- **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

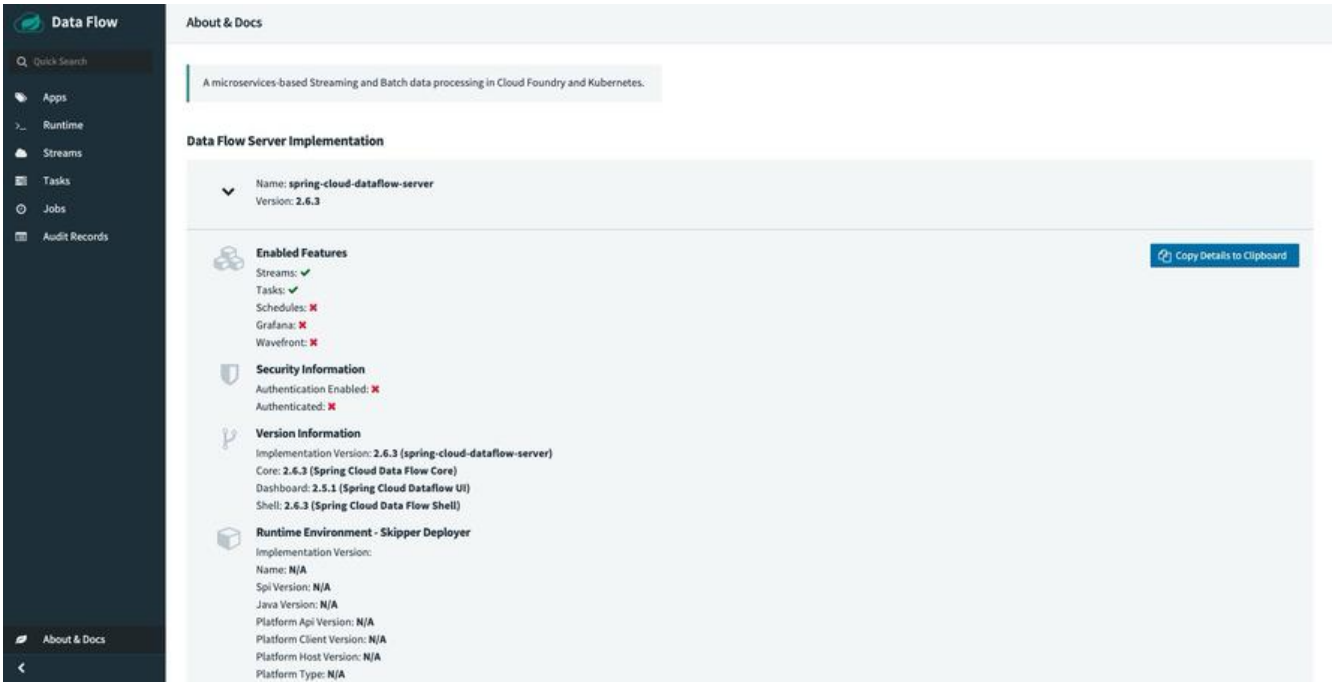
上面的概念和组件有点复杂？我们先快速启动一个应用，通过界面看一下如何使用

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
```

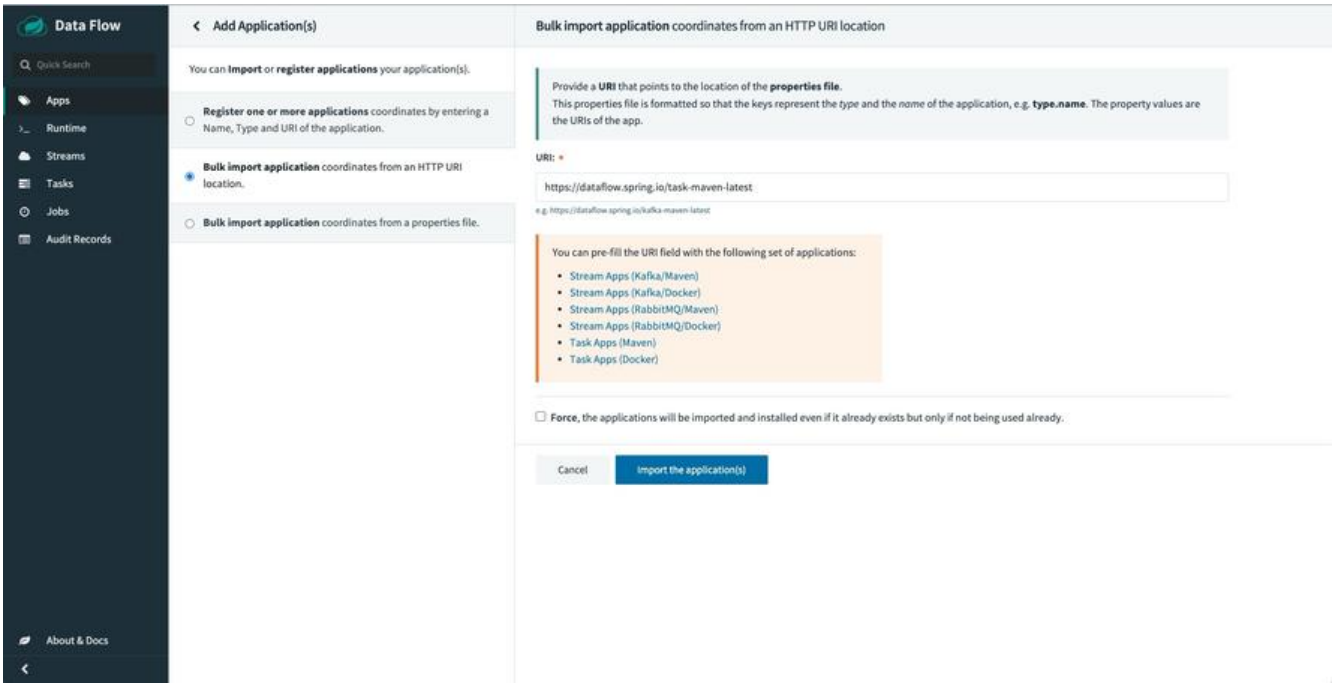
通过 http://localhost:9393/dashboard/#/about 访问管理界面



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**

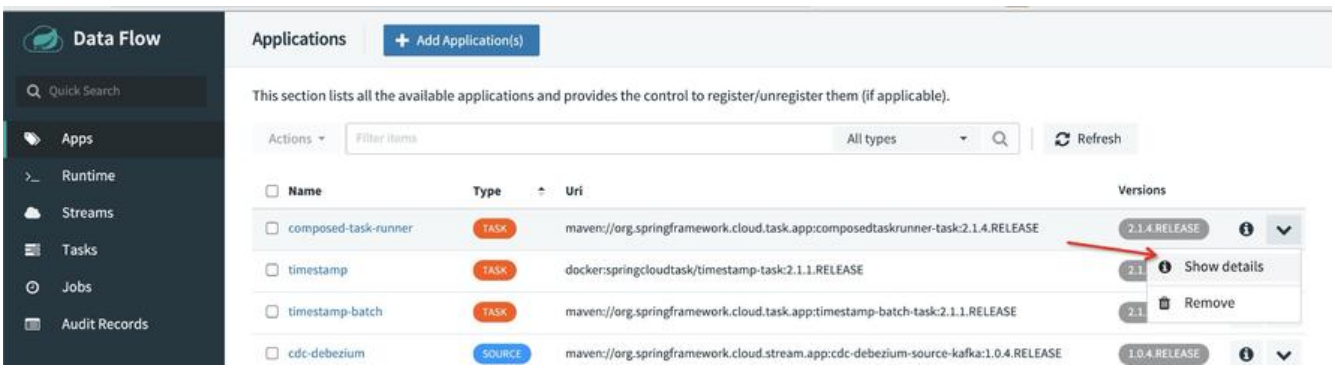


选用第二种(import from an HTTP URI location)，并选择官方**Task Apps Maven**示例，本质上是官方拉取下列配置，在 第三种(import from properties) 中输入下列信息也是同样的效果。

```
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:composedtaskrunner-task:2.1.4.RELEASE
task.composed-task-runner.metadata=maven://org.springframework.cloud.task.app:composedtaskrunner-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
```

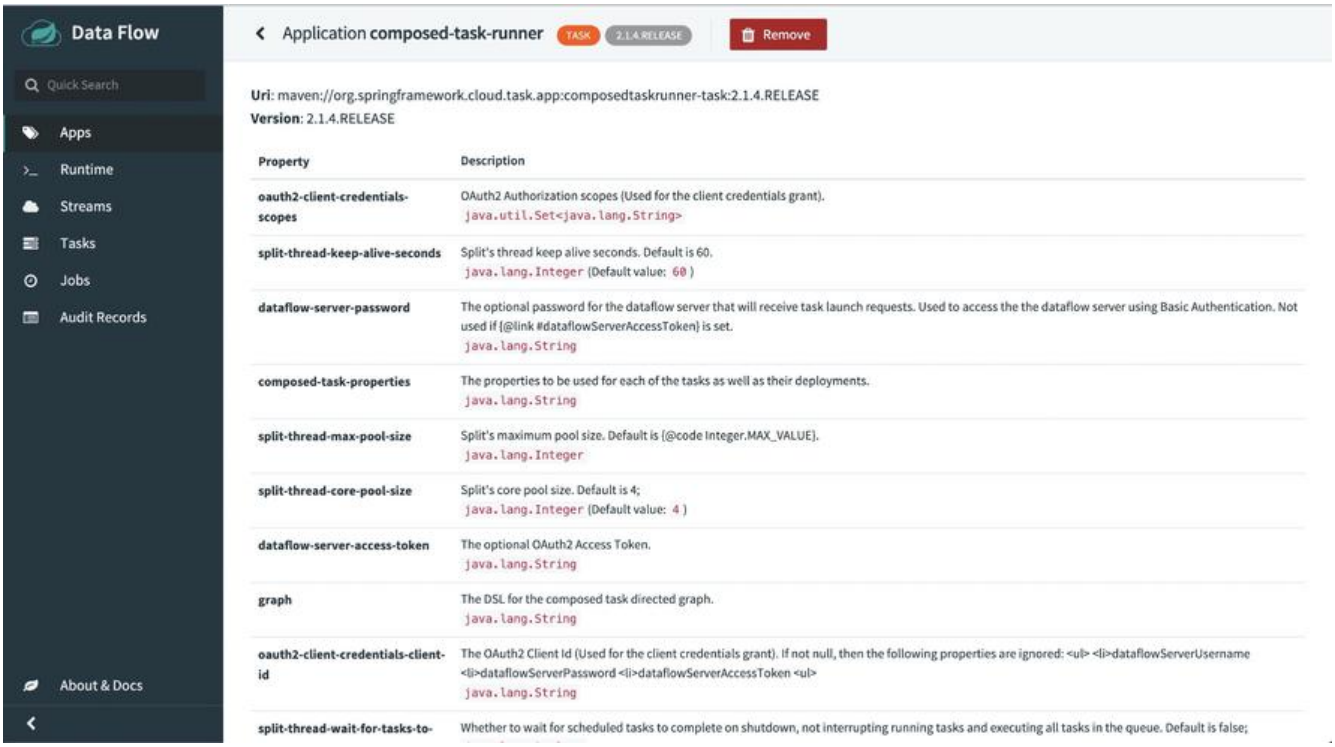
每个 application需要指定应用地址和 metadata信息地址。metadata会在使用过程中明确参数输入单。如果未指定 metadata地址，会尝试通过应用进行提取。

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

原文链接：Spring Cloud Data Flow 入门



注意: 这里有一个坑。如果你按照官方示例 https://github.com/spring-cloud/spring-cloud-tas/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:

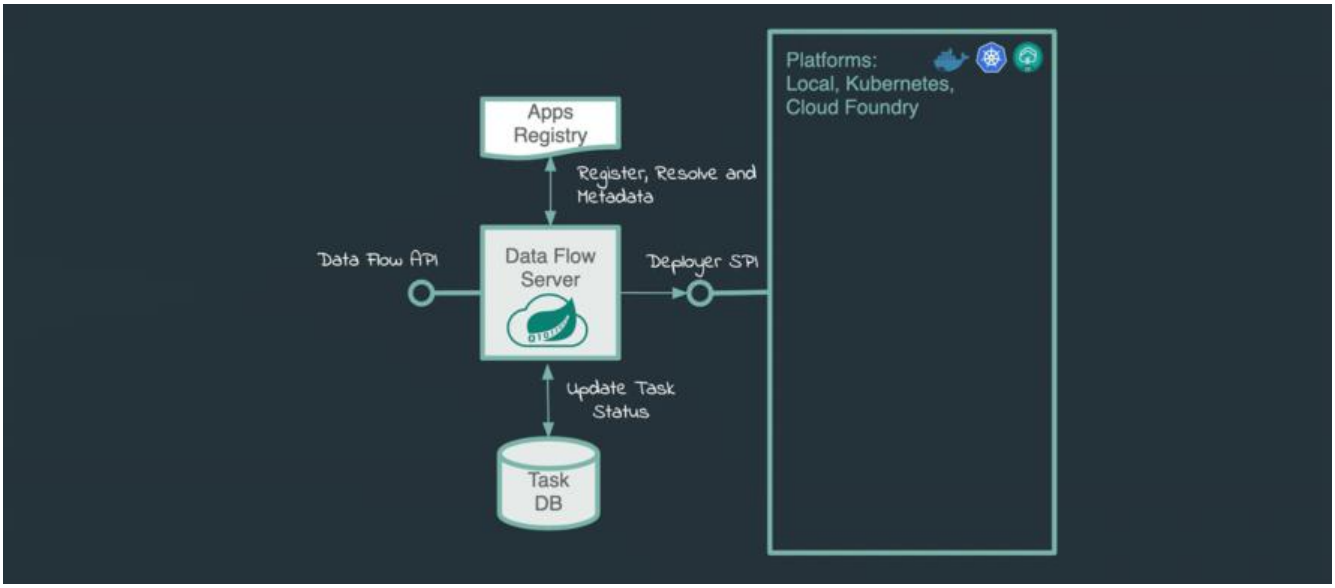
```
private static Resource[] visibleConfigurationMetadataResources(ClassLoader classLoader) throws IOException {
    ResourcePatternResolver resourcePatternResolver = new PathMatchingResourcePatternResolver(classLoader);
    Resource[] configurationResources = resourcePatternResolver.getResources(VISIBLE_PROPERTIES);

    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";
```

原文链接：Spring Cloud Data Flow 入门

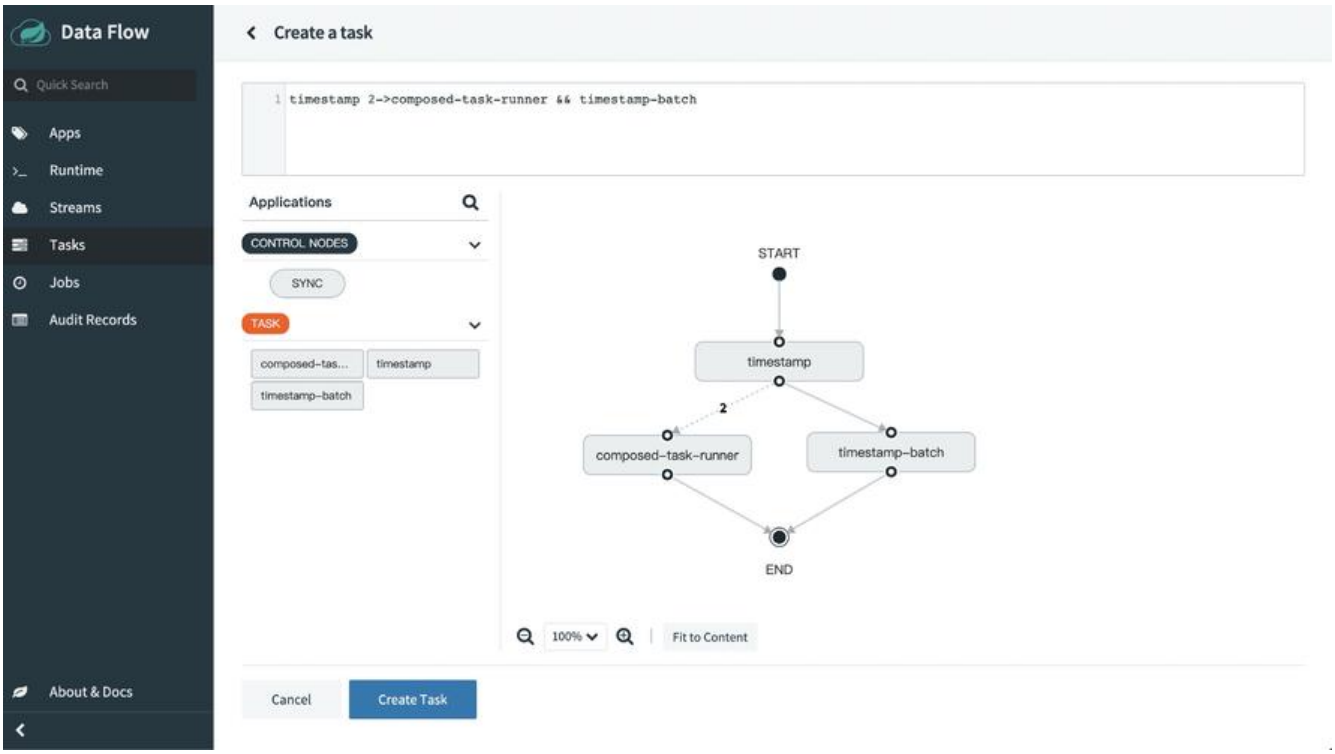
另一个坑是，如果是 maven地址，会先从**localRepository(.m2)**获取 jar文件解析 metadata。如果你设置的是 docker，每次都会通过网络进行 docker鉴权并下载，无法直接判断本地是否存在，以速度会比 maven慢很多。

Task



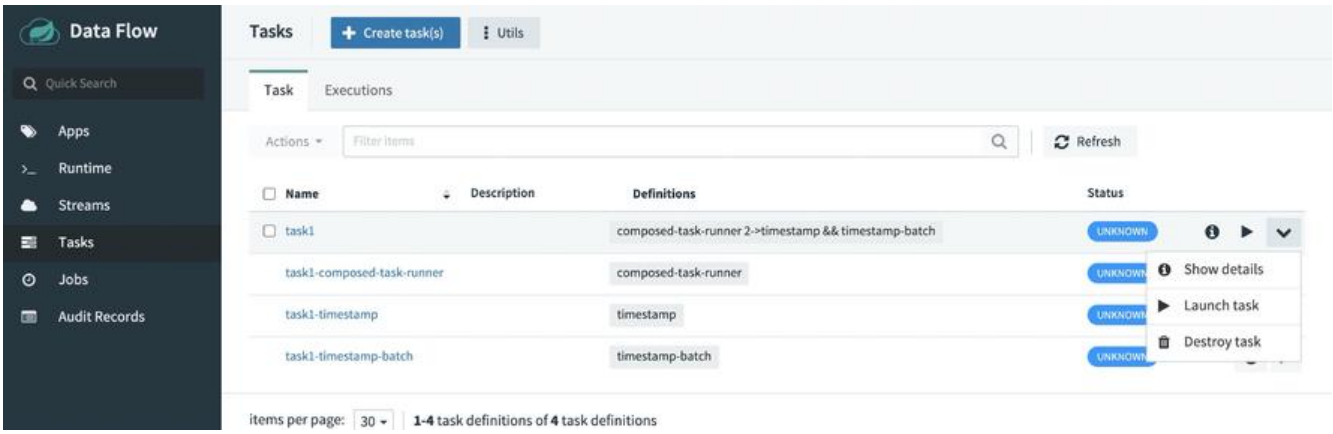
创建 Task

可以通过页面或者DSL定义Task

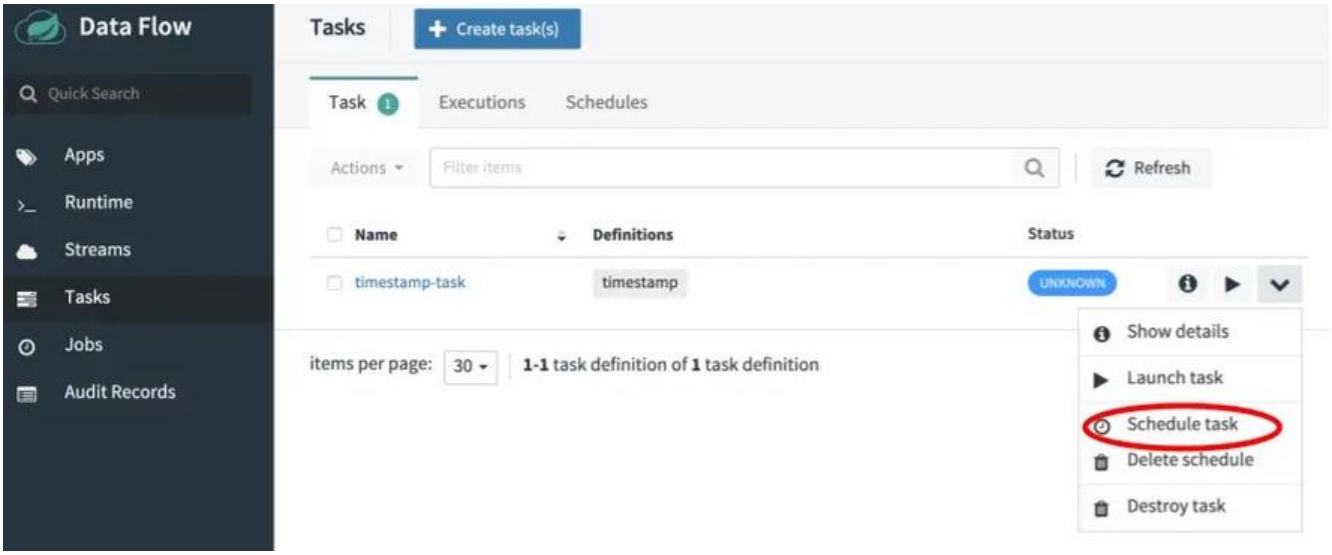


创建好的 Task会存在于 Tasks列表中，可以在列表页查看 Task详情或者执行任务(生成 job)

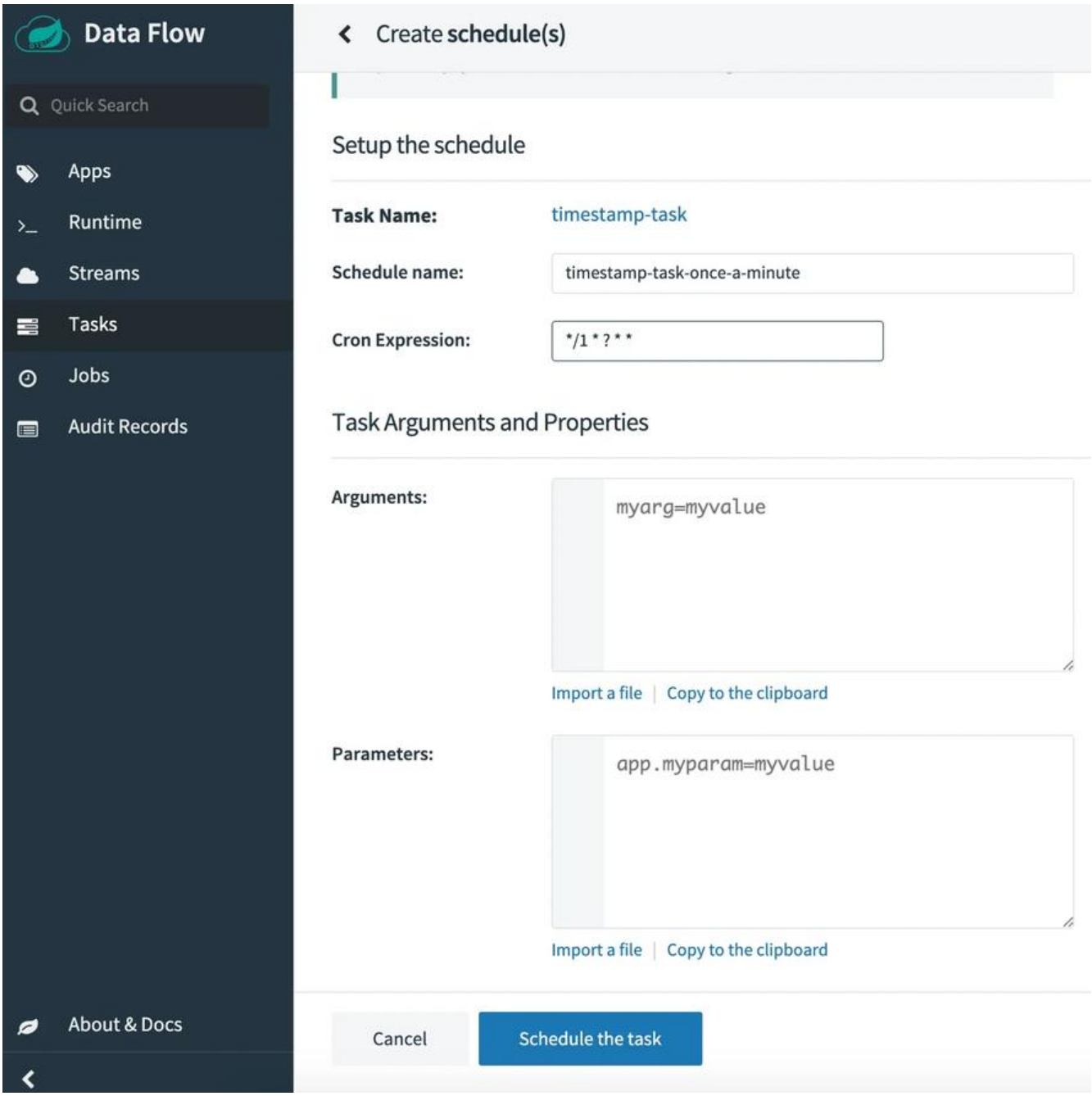
原文链接：Spring Cloud Data Flow 入门



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



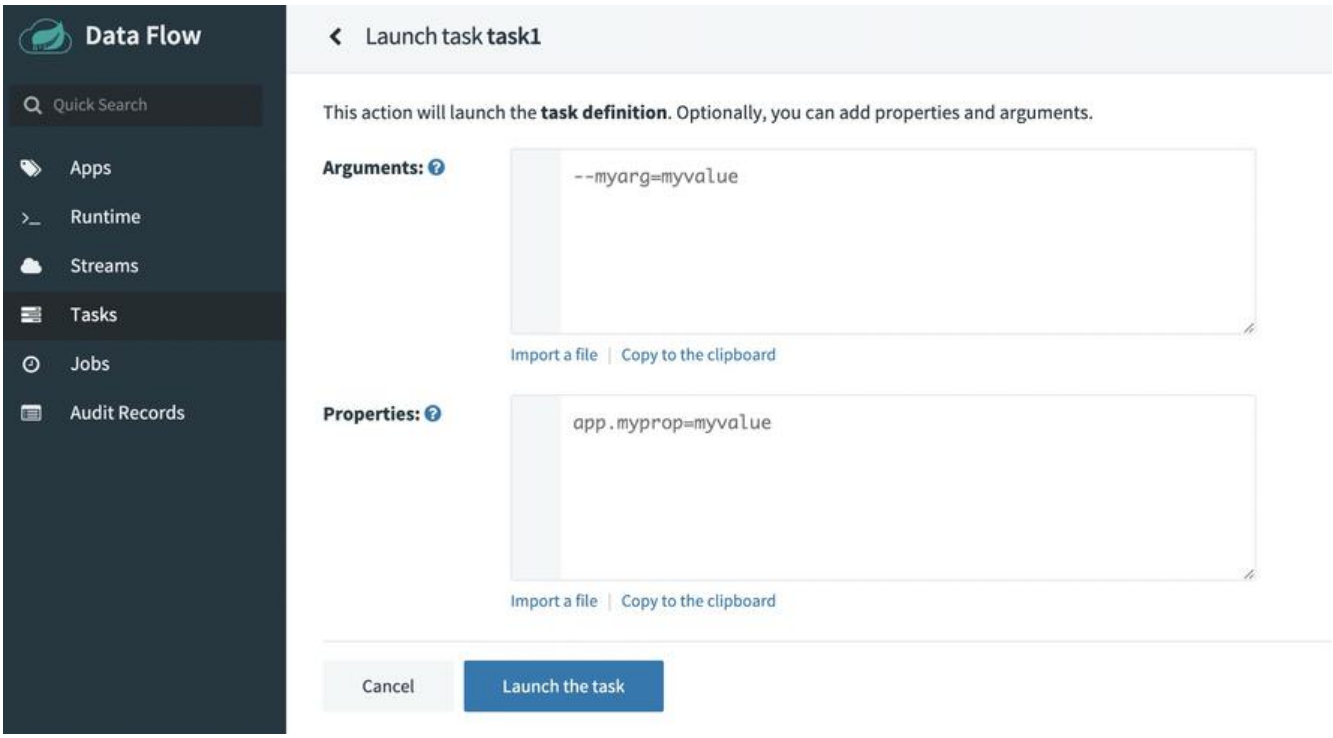
原文链接：Spring Cloud Data Flow 入门



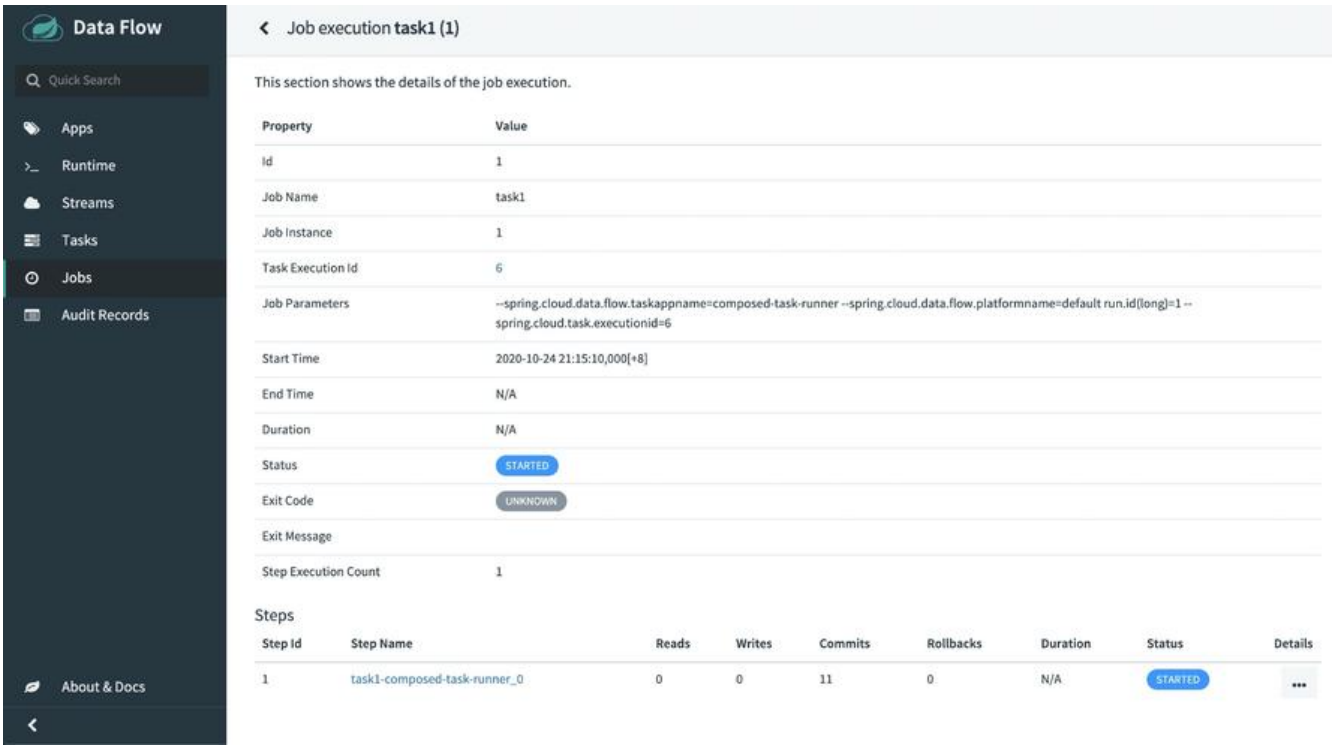
运行 Task

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

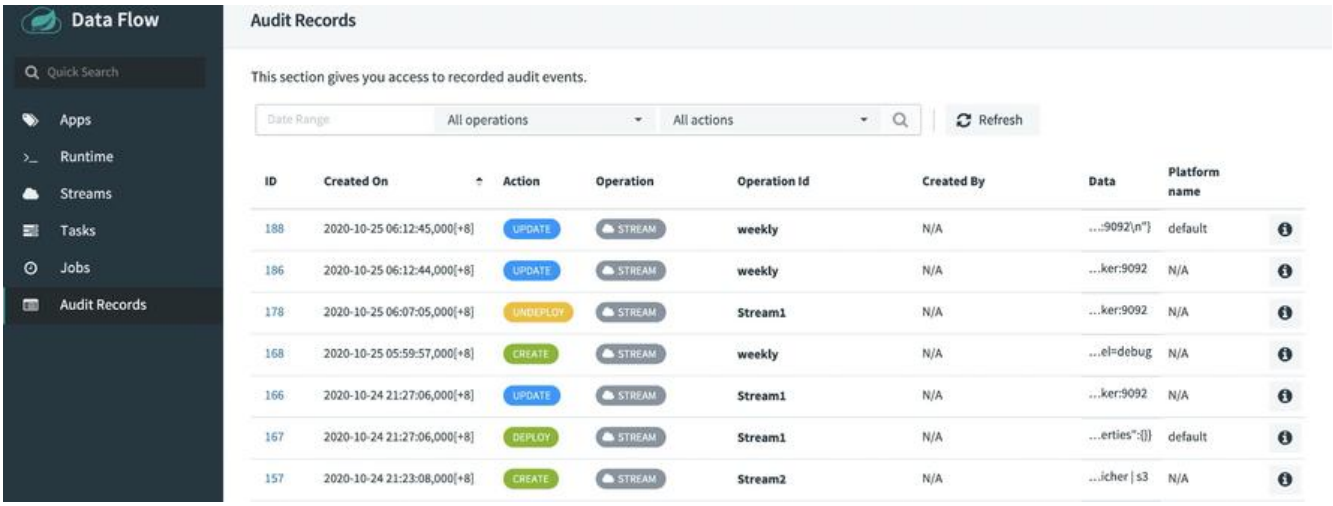


每次执行会生成一个 Job记录，可以在 Task或者 Job列表中查看执行状态、日志等信息

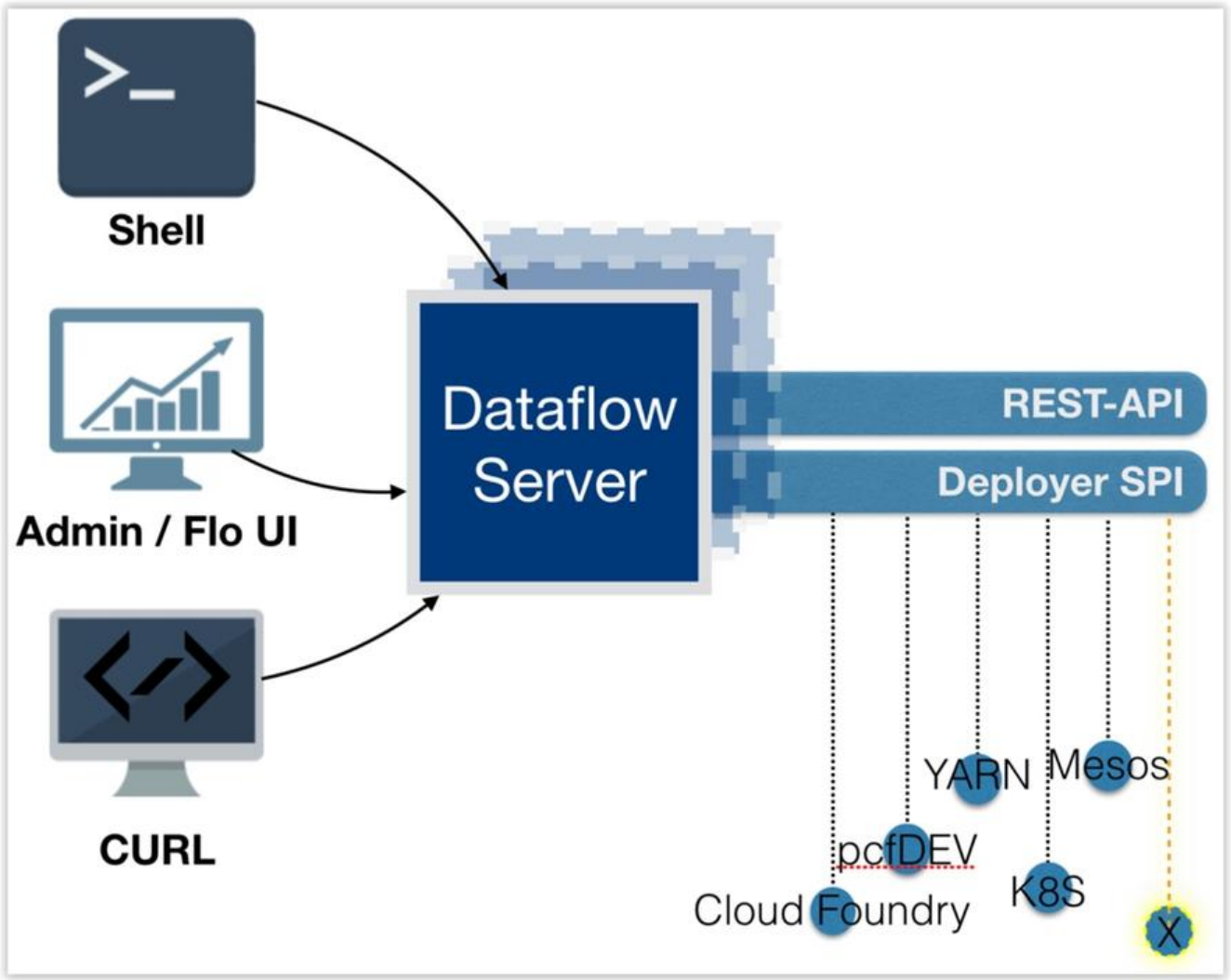


审计留痕

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。



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:
```

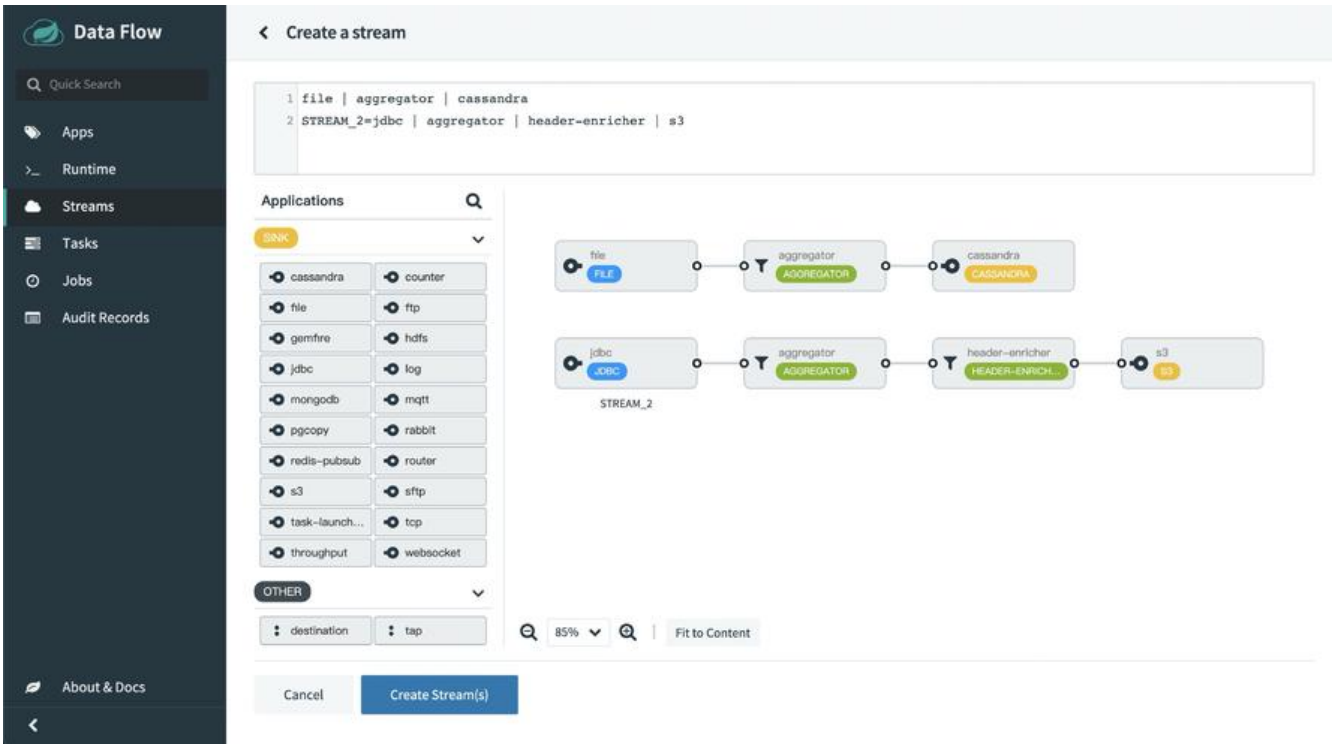
原文链接：Spring Cloud Data Flow 入门

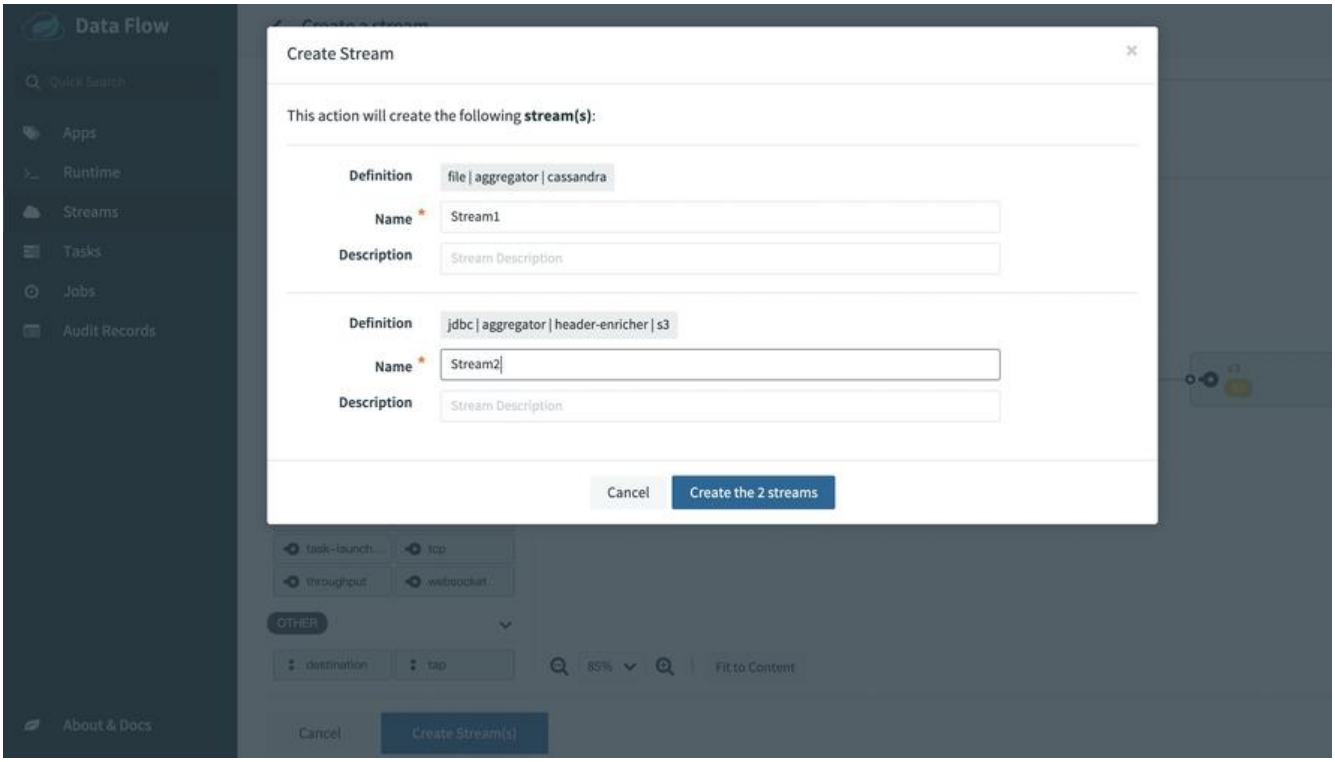
```
    image: mysql:5.7.25
    container_name: dataflow-mysql
    environment:
      MYSQL_DATABASE: dataflow
      MYSQL_USER: root
      MYSQL_ROOT_PASSWORD: rootpw
    expose:
      - 3306

  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=${STREAM_APPS_URI:-ht
ps://dataflow.spring.io/kafka-maven-latest&force=true}';
        echo 'Stream apps imported'
        wget -qO- 'http://dataflow-server:9393/apps' --post-data='uri=${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**



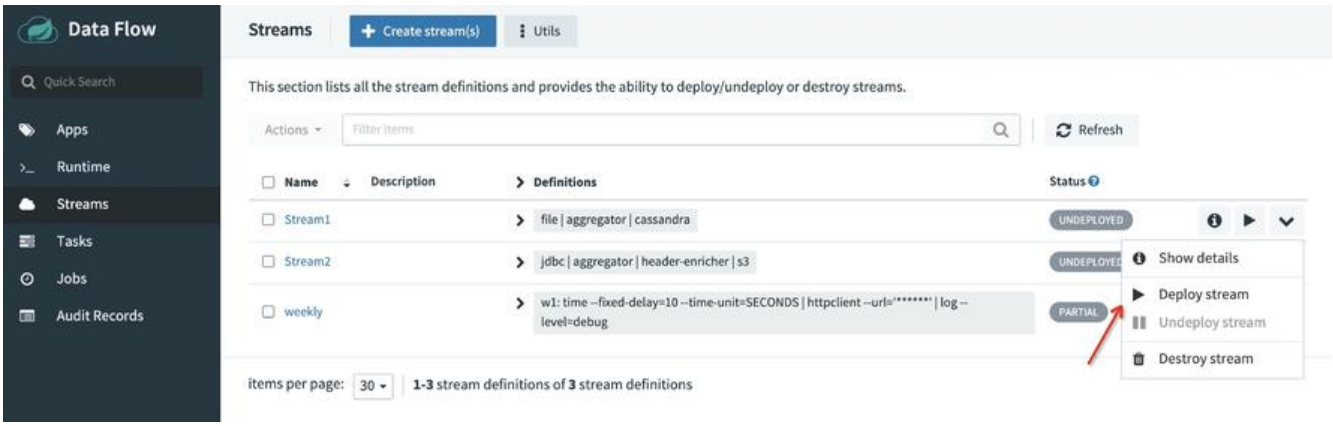


发布 Stream

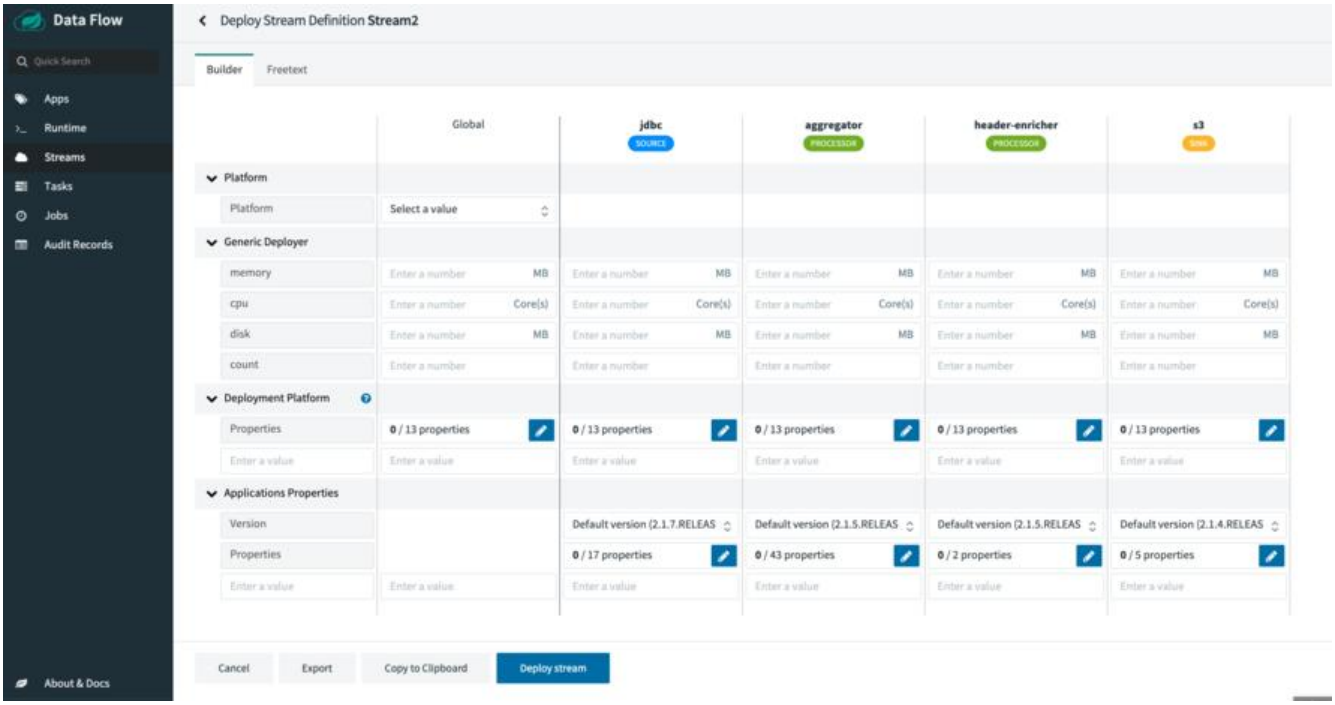
Stream 创建成功后，需要通过 skipper-server发布至**Runtime**，基于消息队列进行驱动并执行数据理

注意: 如果要使用流，需要找一个支持流的持久化，如 Redis

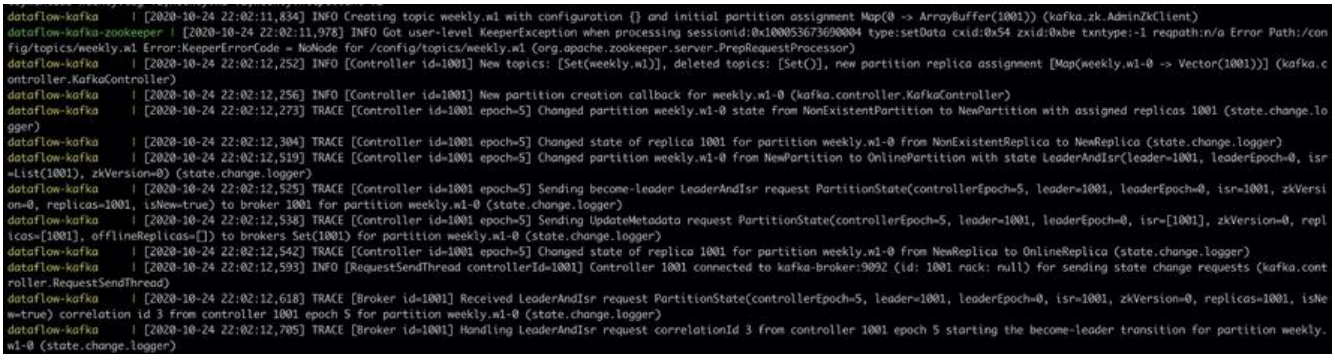
在列表页可以查看 Stream详情，或者发布 Stream。



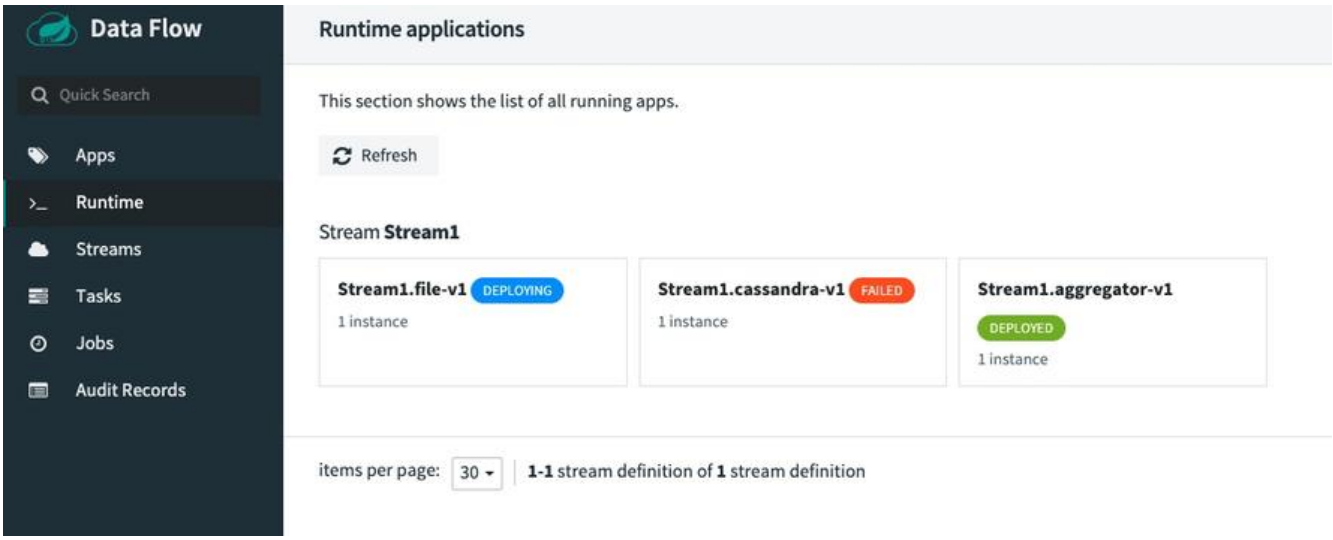
发布时需要指定每个**Application**的**properties**，并且指定资源限制。如果是发布至k8s环境，会根集群配置进行分配。

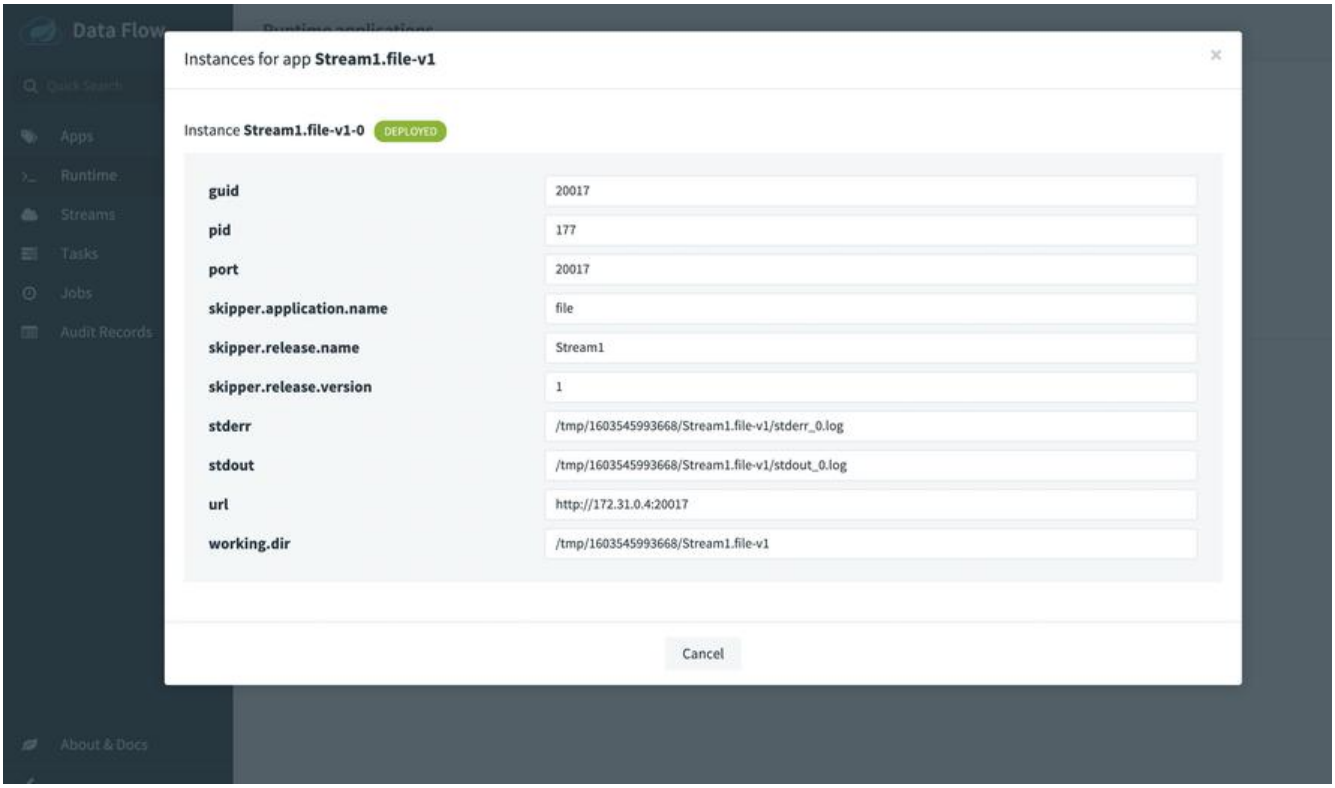


细节: 会按照**label**在 kafka中创建对应的**topic**



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**

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、tream等)

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