1. Spring data Flow

Spring Cloud Data Flow(SCDF) es un conjunto de herramientas para construir y ejecutar en tiempo real procesamiento de datos en pipelines mediante el establecimiento de un flujo de mensajes entre aplicaciones Spring Boot.

Características de Spring Cloud Data Flow:

Aunque esta preparada para los entornos cloud, podemos probarla sin ningún problema en local, para ello podemos probar la integración con las aplicaciones a través de imágenes docker, o haciendo uso de maven para los artefactos.

O podemos desplegar en entornos cloud para lo que facilita su despliegue en Kubernetes o Cloud Foundry.

Dashboard o cliente para mostrarnos una interfaz con la que poder interactúar y verificar y comprobar el flujo.

Nos va a facilitar la conexión con un servidor Oauth2.

Además, gracias a que expone un API Rest, vamos a poder conectar algún sistema de integración continua, como podría ser Jenkins.

En la parte estrictamente de ejecución de procesos batch, Spring Cloud Data Flow es el sustituto del deprecado Spring Batch Admin, de hecho, tiene como modelo de persistencia el mismo repositorio de Spring Batch al que han añadido la parte de persistencia de Spring Cloud Tasks.

Lo más interesante de la arquitectura de Spring Cloud Data Flow es que es cloud agnostic, a día de hoy proporciona un soporte oficial para Kubernetes y Cloud Foundry pero existen proyectos de la comunidad para otras plataformas como HashiCorp Nomad, Red Hat OpenShift y Apache Mesos.

Arquitectura de Spring Cloud Data Flow

La arquitectura de spring cloud data flow se divide en dos, procesos stream (procesos en tiempo real) y procesos batch/tasks (procesos que terminan y acaban en un corto período).

Procesamiento batch: Vamos a definir procesamiento batch como aquel proceso de que tiene una cantidad finita de información. Estas aplicaciones son consideradas efímeras o short-lived en inglés.

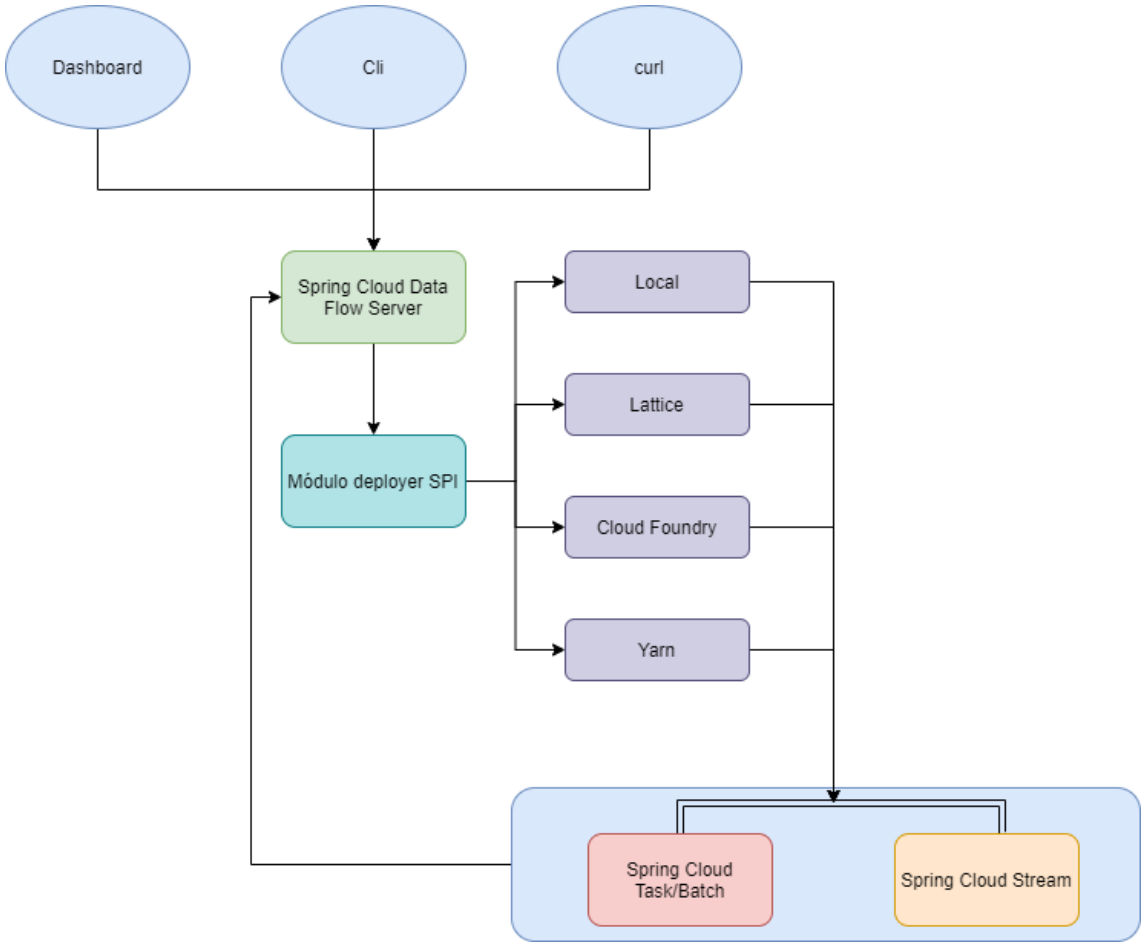
El desarrollo a través de Spring Cloud Task y Spring Cloud Batch, como microservicios efímeros que tienen que devolver un resultado en su ejecución para

su monitorización. Es el propio proceso batch, gracias a recibir en su ejecución la cadena de conexión con la base de datos, quién va dejando trazabilidad de su ejecución en el repositorio común tanto al proceso batch como al propio Spring Data Flow.

Procesamiento en Stream: Vamos a definir o a considerar un stream como una cantidad ilimitada de datos sin interrupción. Procesamiento en Stream con Spring Cloud Data Flow se encuentra implementado como una colección independiente de eventos conectados a través de un broker de mensajería.

Además, a través de Spring Cloud Stream se procesan en tiempo real los streaming de datos, y podemos realizar procesos de ETL en tiempo real basados en la ejecución de tareas dentro de un pipeline donde cada paso es una aplicación independiente dentro del cluster y la comunicación entre ellas se realiza a través de eventos.

En el siguiente diagrama podemos ver un alto nivel de su arquitectura:



Spring Cloud Data Flow

Spring Cloud Data Flow que se despliega como un microservicio más, el cual se descarga de su pagina oficial https://dataflow.spring.io/docs/installation/local/

permitiendo trabajar standalone en local o ejecutarlo con Docker, nos va a proporcionar un API a través del cual podremos acceder con un Dashboard, curls o con un cliente.

Por otro lado tenemos el módulo de deployer SPI, el cual nos permitirá y habilitará diferentes despliegues o cloud o local.

Una vez tenemos todo ejecutado, arrancado y funcionando, los procesos así como su estado serán almacenado en la base de datos del Data Flow.

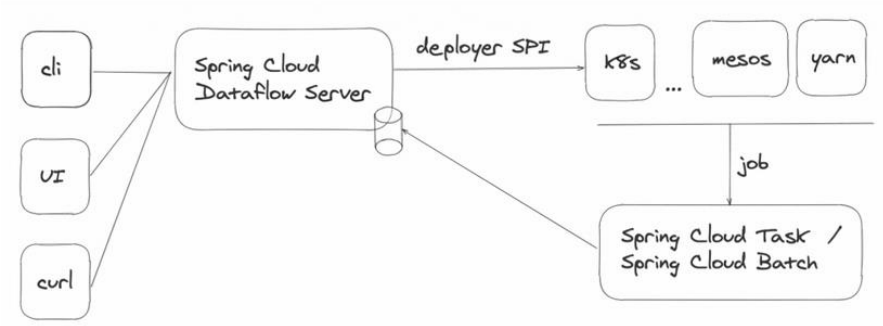
Spring Cloud Data Flow Stream

Cuando hacemos uso de Spring Cloud Stream, vamos a hacer uso de la siguiente arquitectura:

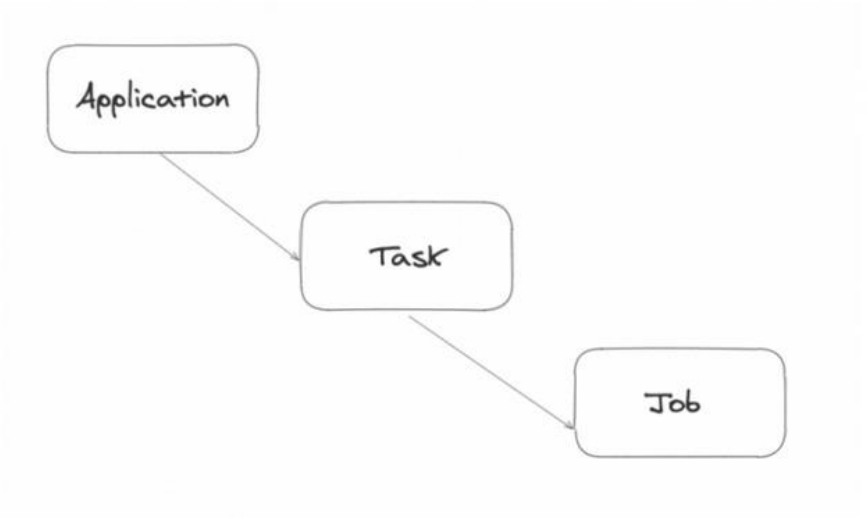
Source: Será el encargado de producir un mensaje.

Proccesor: Es el encargado de realizar un procesamiento en el mensaje.

Sink: Es el encargado de consumir un mensaje



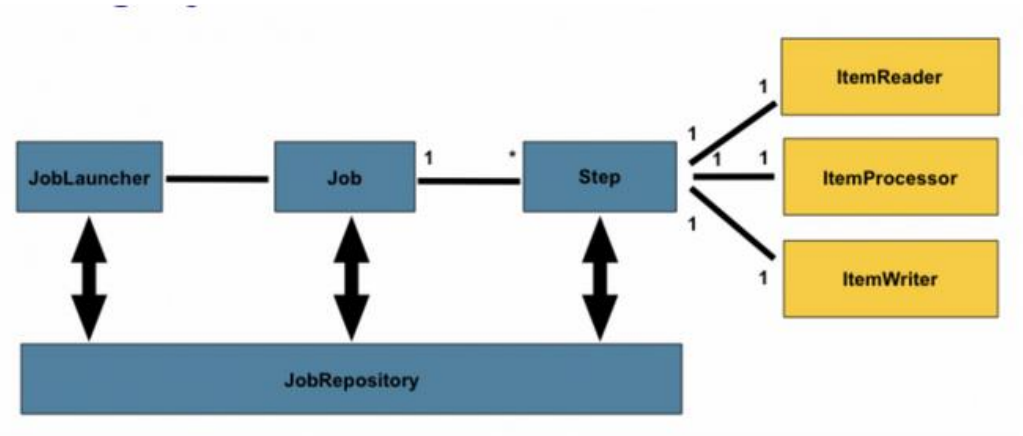
Spring Cloud Data Flow Batch



Aplicaciones que se registran con un número de versión y permiten la gestión del versionado de las mismas, las aplicaciones apuntan a un artefacto que depende del modo de despliegue del Data Flow puede ser incluso un jar aunque en un entorno cloud, lo normal será que referencien a una imagen de docker en un docker registry. Estas aplicaciones son de distintas tipologías, para la ejecución de tareas batch son de tipo task aunque si son susceptibles de formar parte de un pipeline, pueden ser también de tipo source, processor y sink,respondiendo a los pasos de un proceso de ETL,

Esas aplicaciones se pueden añadir a tareas o ejecutarlas como tareas aisladas y las instancias de esas tareas lanzan jobs y los jobs a su vez se dividen en steps.

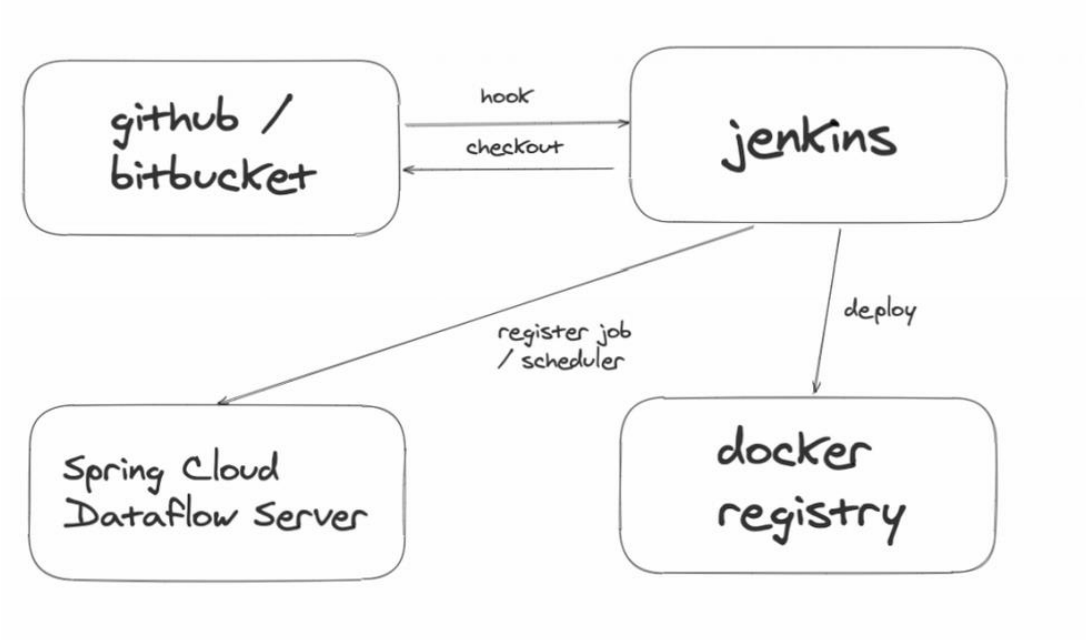
En el siguiente esquema podemos ver cómo está montado el lenguaje de dominio de las tareas batch que se ejecutan. Tienen un lanzador que ejecuta los jobs y estos tienen steps, que van dejando información en la base de datos del Data Flow, común a todos los procesos batch a través de un JobsRepository, los que estéis familiarizados con Spring Data conoceréis la terminología.



En ningún caso, desarrollando un proceso batch necesitaremos conocer internamente estos componentes; solo nos tenemos que preocupar de configurar los jobs y programar los steps que formen parte del proceso batch, en nuestro caso tenemos el caso de un proceso batch de ETL con un reader un processor y un writer; además, además Spring Batch proporciona múltiples implementaciones para leer o escribir un fichero posicional, un csv, un xml…
y para acceder a servicios vía a api rest o web service, como estemos haciendo actualmente en los microservicios.

Especialmente interesante la posibilidad de usar Spring Integration como flujo de entrada de mensajes hacia la ejecución de jobs, podemos configurar, por ejemplo, un flujo de integración teniendo como fuente un directorio remoto de un sftp, de este modo, por cada fichero que encuentre en ese directorio recibiremos un mensaje en modo de fichero y podemos configurar el arranque de un job que tenga como parámetro de entrada dicho fichero. A un flujo de integración se le pueden configurar transformes, de modo que en un solo paso, podemos descomprimir, desencriptar, transformar el mensaje, el fichero, al formato de entrada esperado en el job.

Pipelines CI/CD



Nuestro servidor de integración continua recoge el código del repo y genera una build que construye un artefacto, igual que los microservicios, una imagen de docker del proceso batch que se sube al docker registry y es el propio servidor de integración continua el que registra la nueva versión de la aplicación, la task y, en su caso, la planificación también
automática de cuándo tiene que lanzarse el job, conforme a una expresión del cron; y todo ello se puede hacer haciendo uso del API del Data Flow debidamente securizada.

En el caso de registrar el scheduler automáticamente, es el Data Flow el que habla con el api de k8s para dar de alta el cronjob correspondiente a la image de docker del proceso batch.

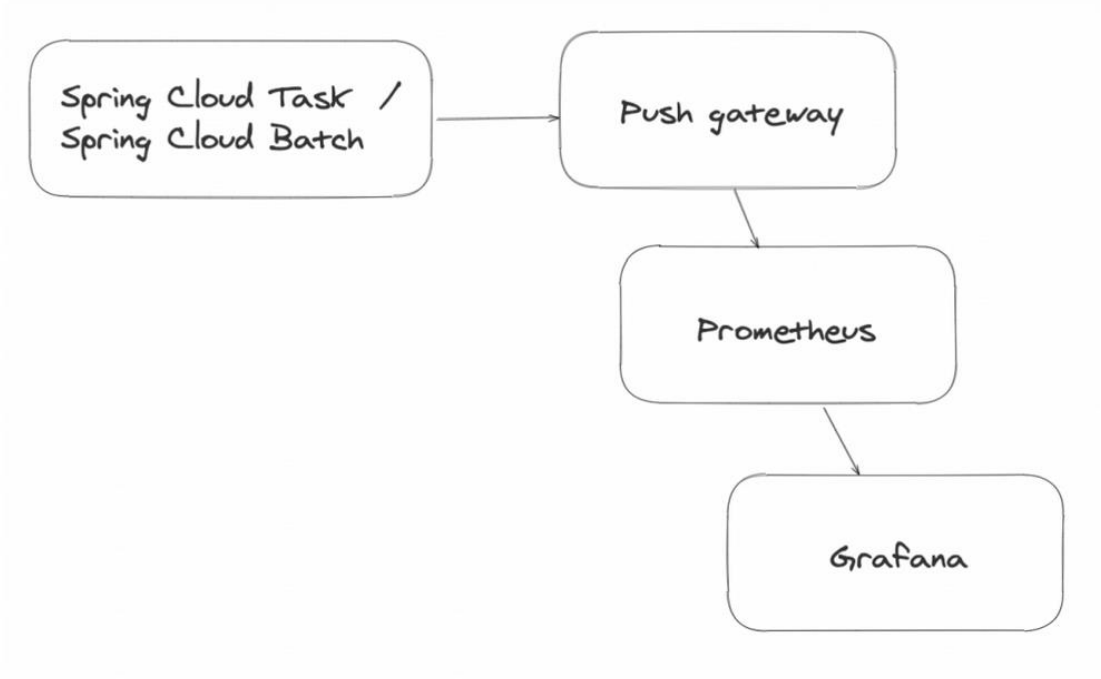
De este modo, por cada build, tendríamos registrada la versión correspondiente en el Data Flow del entorno de integración para su ejecución en las pruebas de UAT o incluso directamente, su calendarización. Y para entornos productivos estará integrado con la promoción entre entornos de los mismos artefactos contra la instancia del Data Flow de los correspondientes entornos.

No podemos plantear el uso de una herramienta como Spring Cloud Data Flow, que nos va a servir de gestor de tareas batch y monitorización de operaciones en un ámbito transaccional, sin pensar en cómo van a integrarse los pipelines de construcción de nuestros procesos batch dentro del proceso de promoción entre entornos de los artefactos.

Monitorización

Así como los microservicios los podemos monitorizar de forma activa con Prometheus siendo procesos de larga duración, los procesos batch al tener una duración indeterminada, al ser procesos efímeros, Spring Cloud Task tiene una integración con una pieza intermedia, que es el **push gateway** que hace de proxy para Prometheus de modo tal, que es el push gateway quien está atendiendo a las peticiones de Prometheus y los los procesos batch quien envían las métricas al gateway.

De este modo, podemos seguir usando las mismas herramientas de monitorización que usamos habitualmente para los microservicios y tener alertas y monitorización basándonos por ejemplo en Grafana.



Implementación Stream proccesing

Previos, indicaciones para instalar Spring Cloud Data Flow servidor

https://dataflow.spring.io/docs/installation/local/

Para este ejemplo se va a realizar la implementación propuesta en contenedores ajustándonos más al mundo devops.

Descargar `docker-compose.yml`, `docker-compose-<broker>.yml`, `docker-compose-<database>.yml`donde `<broker>`es uno de rabbitmq o kafka y `<database>`uno de postgres, mariadb o mysql.

```
wget -O docker-compose.yml https://raw.githubusercontent.com/spring-cloud/spring-cloud-dataflow/main/src/docker-compose/docker-compose.yml;
wget -O docker-compose-<broker>.yml https://raw.githubusercontent.com/spring-cloud/spring-cloud-dataflow/main/src/docker-compose/docker-compose-<broker>.yml;
wget -O docker-compose-<database>.yml https://raw.githubusercontent.com/spring-cloud/spring-cloud-dataflow/main/src/docker-compose/docker-compose-<database>.yml;
```

En la demo Se elige Kafka como bróker y postgress como database para el modelo propio.