

**Spring Data Flow Server**

**Proof of concept in Pivotal Cloud Foundry**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** |
| **Name** | Vasanthan Ekambaranathan |  |  |
| **Role** | Architect |  |  |
| **Signature** |  |  |  |

**Version 1.1**

Contents

[**1.** **Spring Data Flow** 3](#_Toc490045521)

[**1.1** **Stream** 3](#_Toc490045522)

[**1.2** **Task/Batch** 3](#_Toc490045523)

[**2.** **Spring Data Flow deploy on Cloud Foundry** 4](#_Toc490045524)

[**2.1** **Spring Data Flow Server Deployment** 4](#_Toc490045526)

[**2.2** **Application Sample Deployments for Stream & Task** 8](#_Toc490045527)

[**2.2.1** **Stream Deployment** 8](#_Toc490045528)

[**2.2.2** **Batch Deployment** 10](#_Toc490045529)

[**2.2.3** **Scheduling Batch Job Deployment** 11](#_Toc490045530)

[**3.** **Spring Data Flow Vs Spring Batch** 14](#_Toc490045531)

[**3** **Change Log** 15](#_Toc490045532)

# **Spring Data Flow**

Spring Cloud Data Flow is a cloud native data framework that unifies stream and batch processing for data micro services, across the cloud or on premise. It allows developers to create, orchestrate and refactor data pipelines with a single programming model for common use cases like data ingest, real time analytics, and data import/export.

## **Stream**

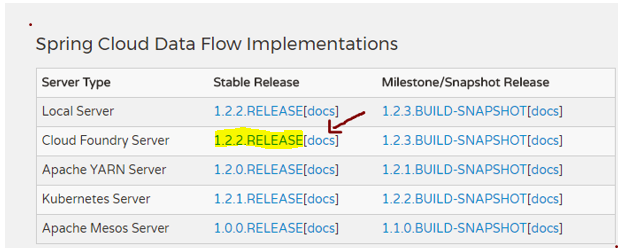
* + Stream - Long Lived Stream applications where an unbounded amount of data is consumed or produced via messaging middleware.

## **Task/Batch**

* + Task/batch Jobs - Short Lived Task applications that process a finite set of data which can terminate once task is completed.

To learn more detail about spring data flow refer to <http://cloud.spring.io/spring-cloud-dataflow/>

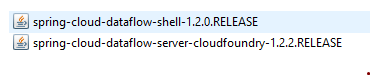
You can find the many implementations of Spring Data Flow server in the above link, where you can download the below mentioned Snapshots or build Release.



As this deployment is going to be in Pivotal Cloud Foundry, download the Cloud Foundry spring data flow server implementation jar file, which is highlighted as a yellow color in the above image.

And also find the complete documentation and steps of how it will be deployed in the cloud foundry environment, which is highlighted as a red arrow in the above image.

For this proof of concept, the following version of the shell and SDF Server jar files are used for this installation.



<http://repo.spring.io/release/org/springframework/cloud/spring-cloud-dataflow-shell/1.2.2.RELEASE/spring-cloud-dataflow-shell-1.2.2.RELEASE.jar>

If Spring data flow server can be hosted on Cloud Foundry server/different host rather than local environment, run the downloaded the shell jar and target to the hosted SDF Server in the cloud foundry.

# **Spring Data Flow deploy on Cloud Foundry**

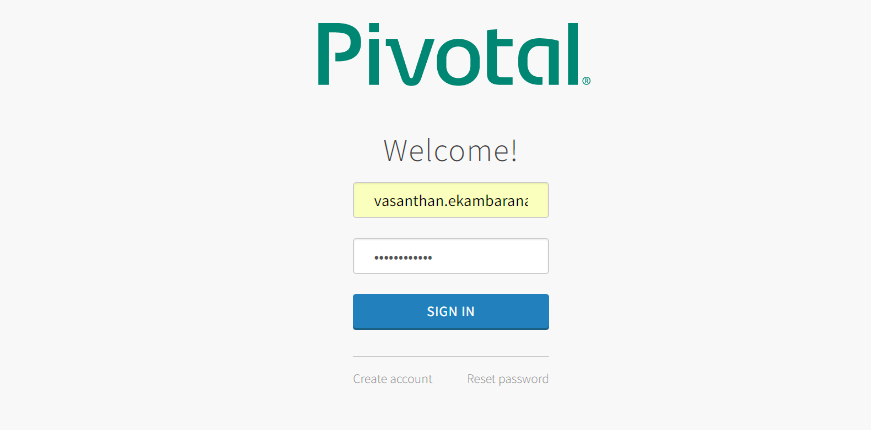
The following steps need to be adhered to deploy in the cloud Foundry application.



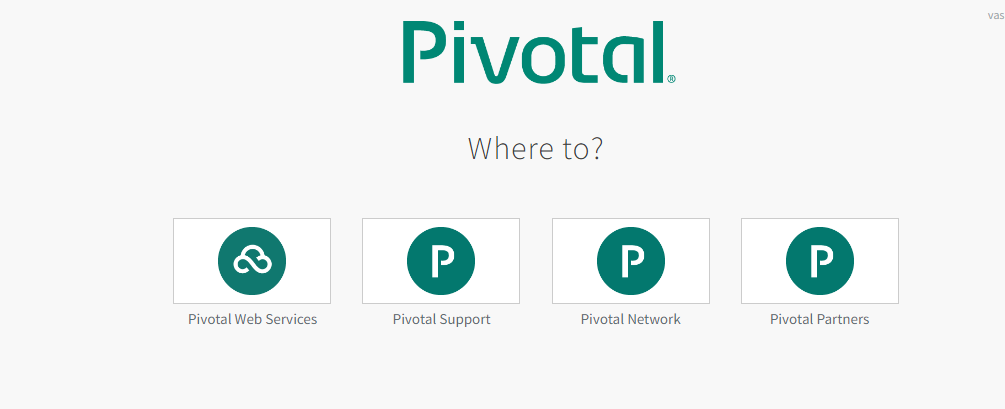
## **Spring Data Flow Server Deployment**

For this example, Pivotal cloud foundry web services portal is being used to deploy the Spring Data Flow server.

1. Login into your PCF account <https://login.run.pivotal.io/login>



1. Once login into the Pivotal account where you can find different sites where you can click the web services portal site. Once you clicked the Pivotal Web Services site, which would automatically be loaded the default organization and spaces for your account.



Now let us start to create MySQL database and RabbitMQ services before deploy the Spring data flow server in the Pivotal Cloud foundry.

1. There are a couple of ways to create the Services either creating from command prompt using CF commands or create the services from online pivotal services.
2. Let us create the services from command prompt using CF commands. Before that logging into your account in command prompt to create the services.

**cf login -a https://api.run.pivotal.io**

1. For this deployment, RabbitMQ and MySQL connections are required in order to bind and store the data for the stream application and Cloud Tasks/ batch jobs.
2. Execute the below commands to create the MySQL Connection.

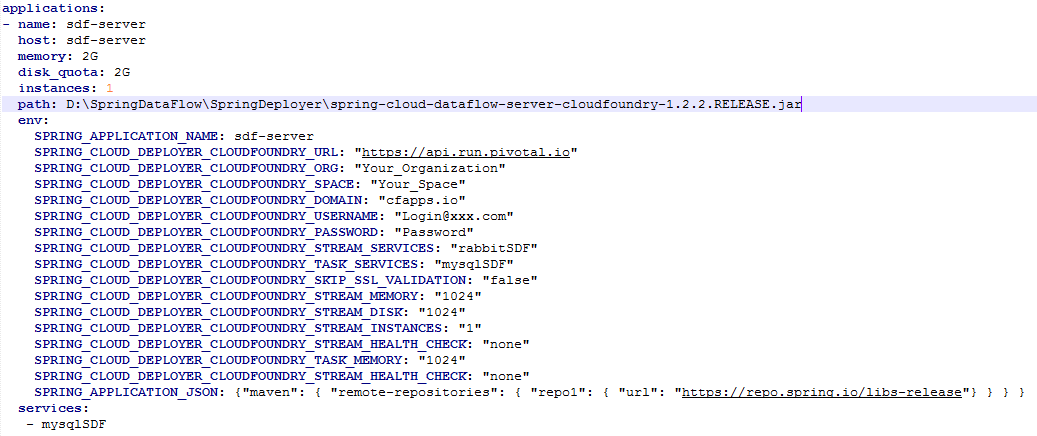
**cf create-service cleardb spark mysqlSDF**

**cf bind-service sdf-server mysqlSDF**

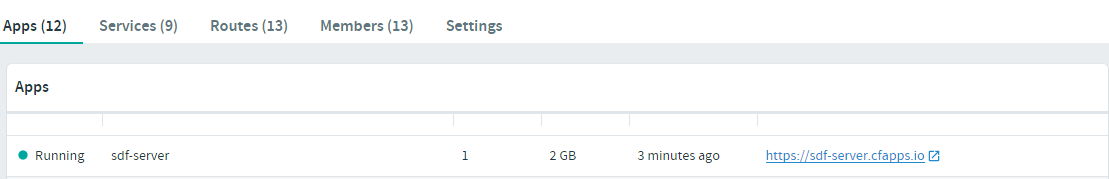
1. Execute the below commands to create the RabbitMQ connection.

**cf create-service cloudamqp lemur rabbitSDF**

1. Deploy the SDF Server into Cloud Foundry environment. The following manifest.yml file entries should be located in your folder where you can issue the CF push command.



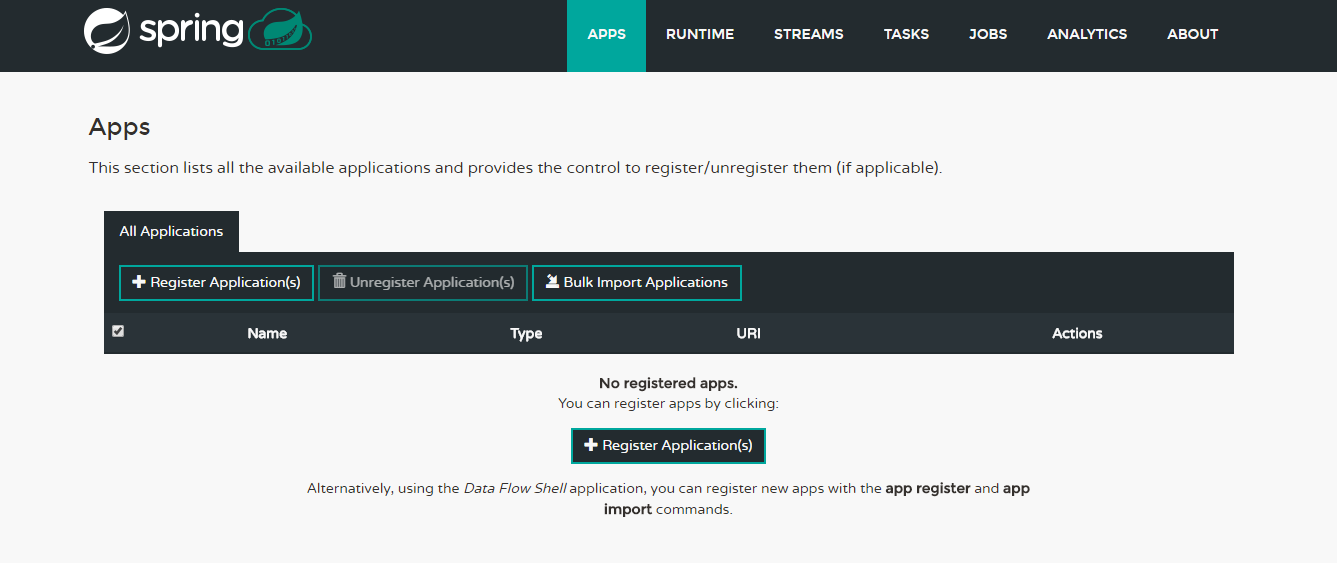
1. Once you deployed the SDF Server, required database and MQ services would automatically bind to your application.



1. Once you deployed the server, you can see the dashboard of Spring data flow server, which could be accessible through browser URL.

For example, <https://sdf-server.cfapps.io/dashboard>

The following screen will get displayed if you browse the dashboard, where you could see the App, streams, Tasks and the batch jobs menu.



1. Let us deploy the simple Stream application which consists of Source, processor and Sink. There are around three different application needs to be deployed in order to configure the stream. The processor might be an optional. The source and Sink must be available to run the stream application.

For this use case, the source would read the data from the files and send the data to the MQ queue that would be listened by Processor and Sink. If the processor doesn’t exist, Sink will get to listen the message and do the operation like data storage or send the data to downstream applications.

1. Before deploy any application in the Spring Data flow Server, the stream or batch applications should be stored in any of the repositories like maven or GitHub or any private repository. For our sample implementations, stored the applications in the cognizant home grown product of “***fileUpload***” API which is being deployed in the cognizant cloud foundry.

For this example, the following URL is used to upload the applications.

[**http://fileapp-dont-del.cognizantone.org/fileupload/**](http://fileapp-dont-del.cognizantone.org/fileupload/)

1. In order to deploy and issue the commands through Shell executable jar, run the shell and target the SDF Server application, which is deployed in Cloud Foundry

**java -jar spring-cloud-dataflow-shell-1.2.0.RELEASE.jar**

**dataflow config server** [**http://sdf-server.cfapps.io**](http://sdf-server.cfapps.io)

## **Application Sample Deployments for Stream & Task**

### **Stream Deployment**

1. Register the source, processor and Sink application in the Spring Data flow, while registering the application, the URI should be deployed/uploaded repository location.

**Source:**

**app register --name** **time-source** **--type source** **--uri** [**http://fileapp-dont-**](http://fileapp-dont-)**del.cognizantone.org/fileDownload/59565cf142d9d0001dfe5719.jar**

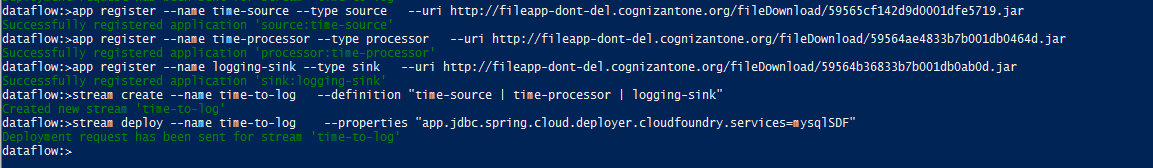
**Processor:**

**app register --name** **time-processor** **--type processor --uri** [**http://fileapp-dont-del.cognizantone.org/fileDownload/59564ae4833b7b001db0464d.jar**](http://fileapp-dont-del.cognizantone.org/fileDownload/59564ae4833b7b001db0464d.jar)

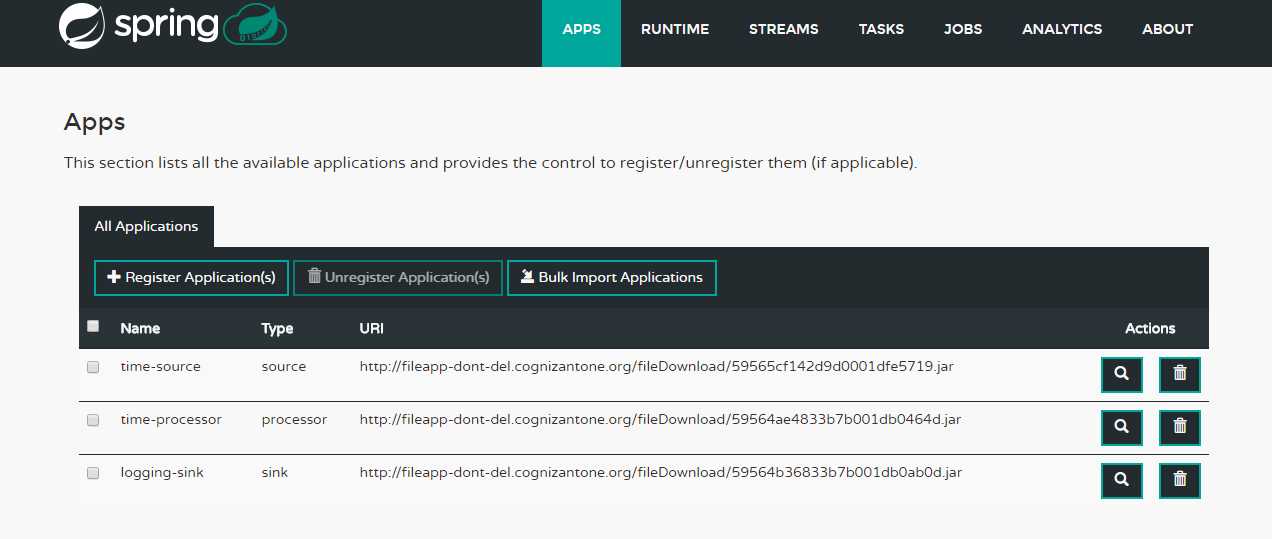
**Sink:**

**app register --name** **logging-sink** **--type sink --uri** [**http://fileapp-dont-del.cognizantone.org/fileDownload/59564b36833b7b001db0ab0d.jar**](http://fileapp-dont-del.cognizantone.org/fileDownload/59564b36833b7b001db0ab0d.jar)

1. The below image shows that the executed commands and response message.



1. The registered application can be seen in the spring dashboard as well.

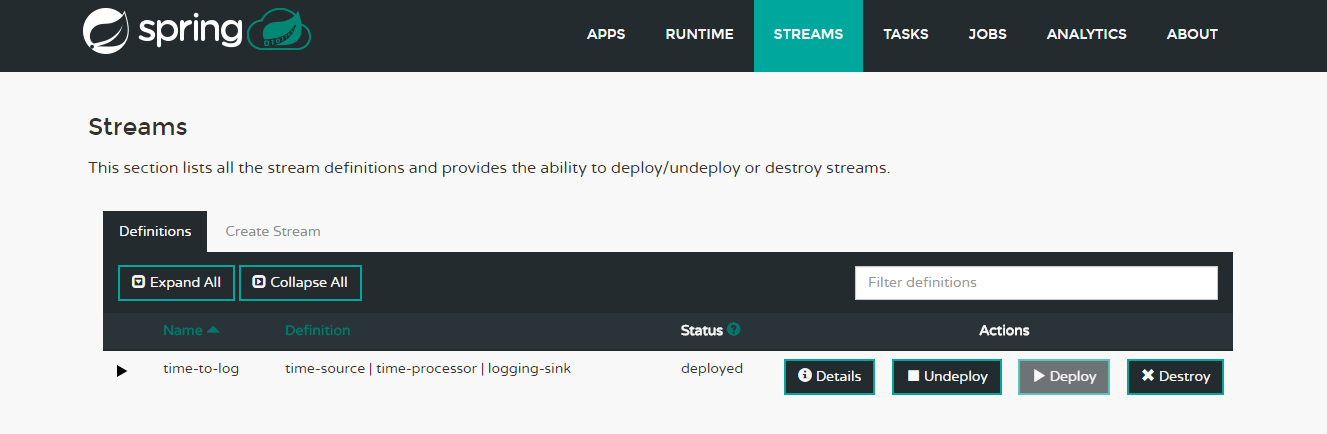


1. To create stream for above deployed the applications.

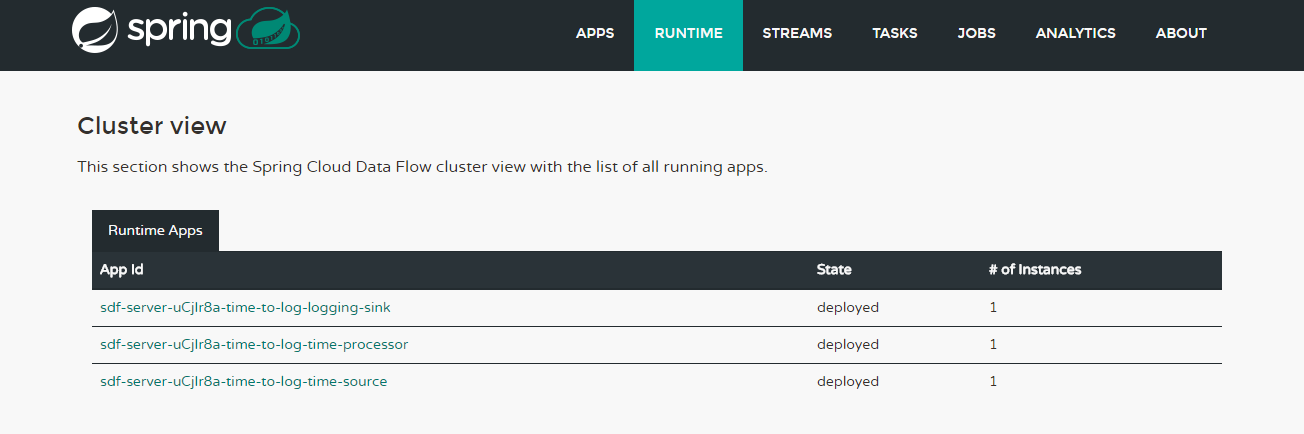
**stream create --name** **time-to-log** **--definition** **"time-source | time-processor | logging-sink"**

1. To deploy the created stream using commands

**stream deploy --name** **time-to-log** -**-properties** **"app.jdbc.spring.cloud.deployer.cloudfoundry.services=mysqlSDF"**

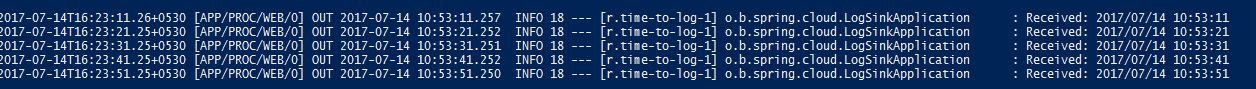


1. And also you can see the run time application Id and status of deployed stream applications.



1. The log details for Sink where you can see the time log details how the data is flowing from source to the processor and to sink

**cf logs sdf-server-uCjIr8a-time-to-log-logging-sink**

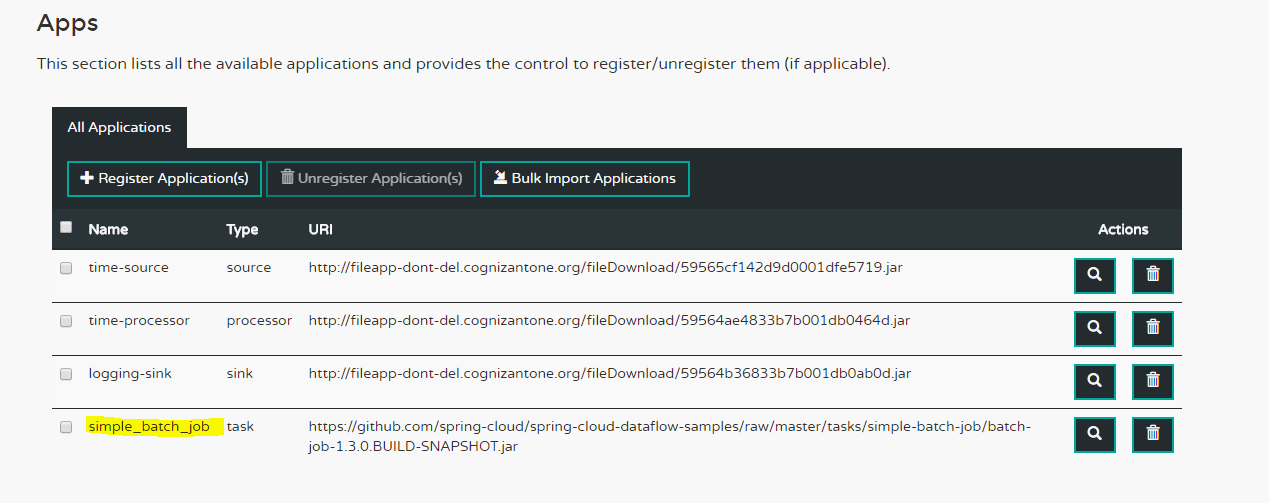


### **Batch Deployment**

1. Similarly, deploy the Batch program in the spring data flow server. This kind of application should be deployed as task to run the batch jobs.

**app register --type task --name** **simple\_batch\_job** **--uri** [**https://github.com/spring-**](https://github.com/spring-)**cloud/spring-cloud-dataflow-samples/raw/master/tasks/simple-batch-job/batch-job-1.3.0.BUILD-SNAPSHOT.jar**

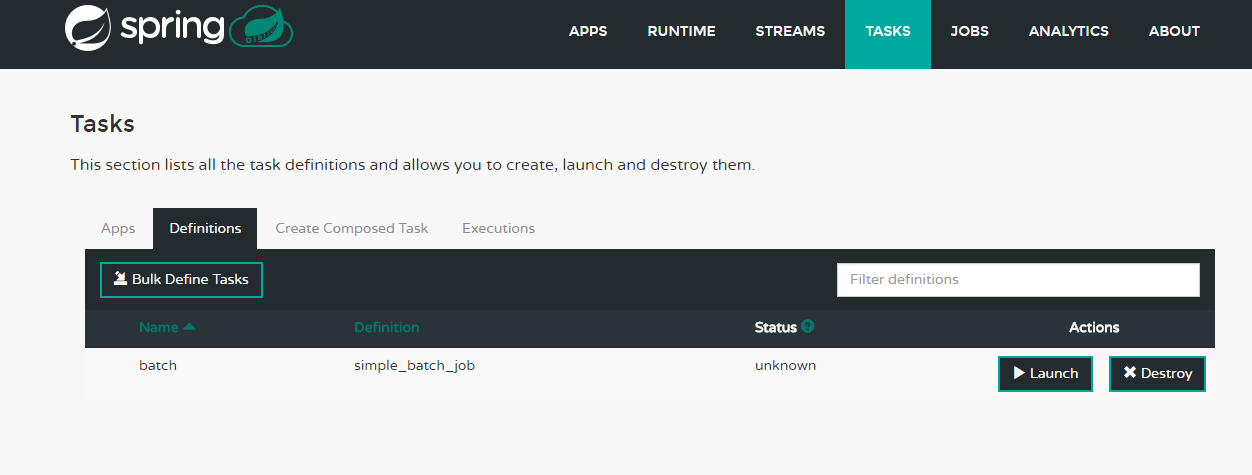
1. The registered application can be seen in the spring dashboard, which is highlighted as a yellow color in the below image.



1. The following command is to create a task of deployed Simple Batch Job application

**task create** **batch** **--definition "simple\_batch\_job"**

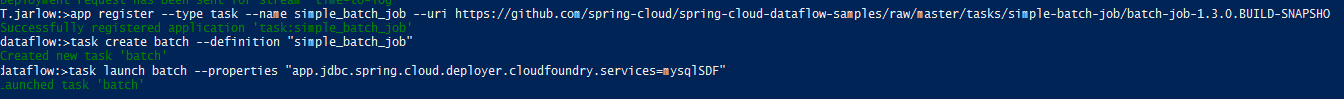
The below image shows the created task entries in the dashboard page.



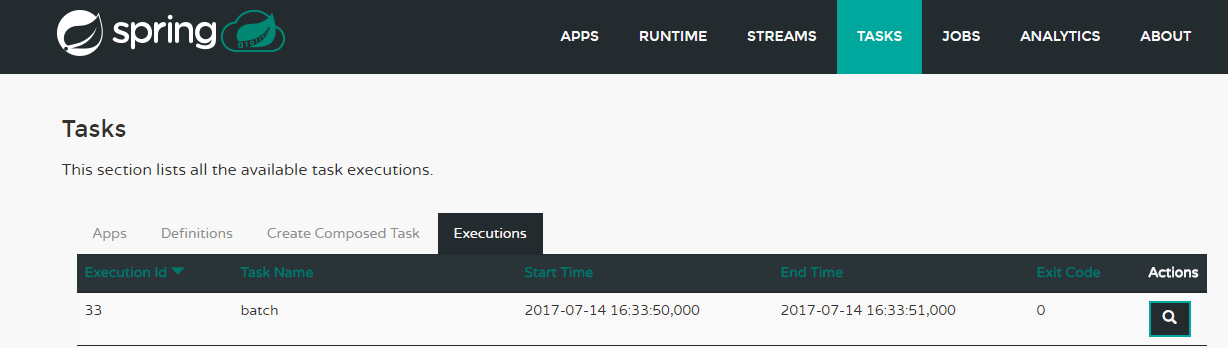
1. Once created the task, launch the task in order to execute the jobs.

**task launch** **batch** **--properties "app.jdbc.spring.cloud.deployer.cloudfoundry.services=mysqlSDF"**

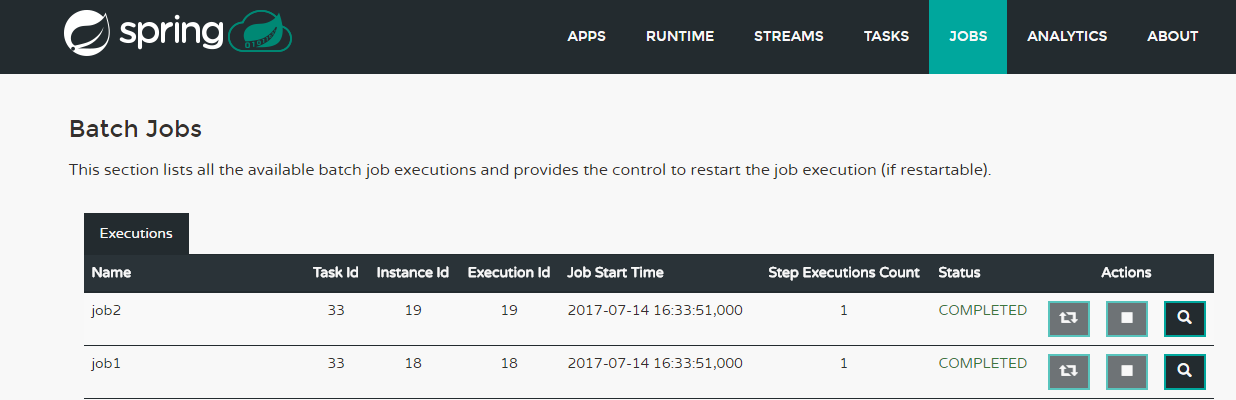
1. The executed commands and response message can be seen in the below image.



1. Once launched the task, you can see the executed the job timestamp and details on the dashboard. See the below screenshot for more details.



1. Two different batch jobs ran with result of Sample batch job execution. The Job entries can be seen in the Jobs menu on the Spring data flow dashboard that is captured in the below screen.



### **Scheduling Batch Job Deployment**

By default, the batch /cloud task doesn’t provide the scheduling feature. In order to schedule the batch job in spring data flow, TriggerTask feature is available in the Stream process.

To create Trigger task, the slightly modified approach of which we have configured in the stream process like Source, TriggerTask and Launch rather than the source, processor and sink.

1. Register the source, TriggerTask Job and Sink application in the Spring Data flow, while registering the application, the URI files should be deployed in the repository location.

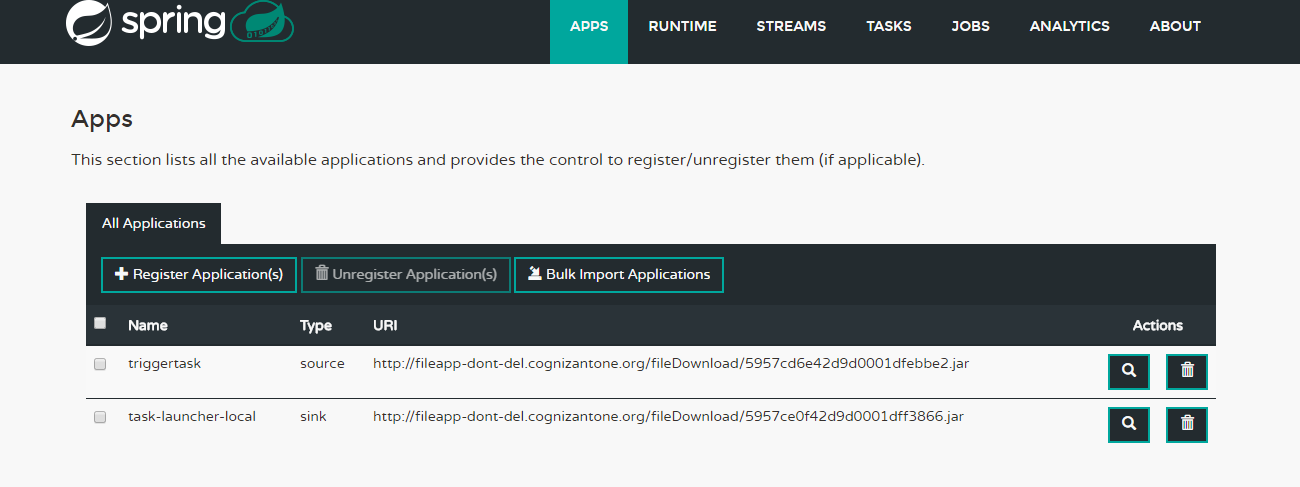
**Source:**

**app register --name** **triggertask --type source** **--uri** **http://fileapp-dont-del.cognizantone.org/fileDownload/5957cd6e42d9d0001dfebbe2.jar**

**Sink:**

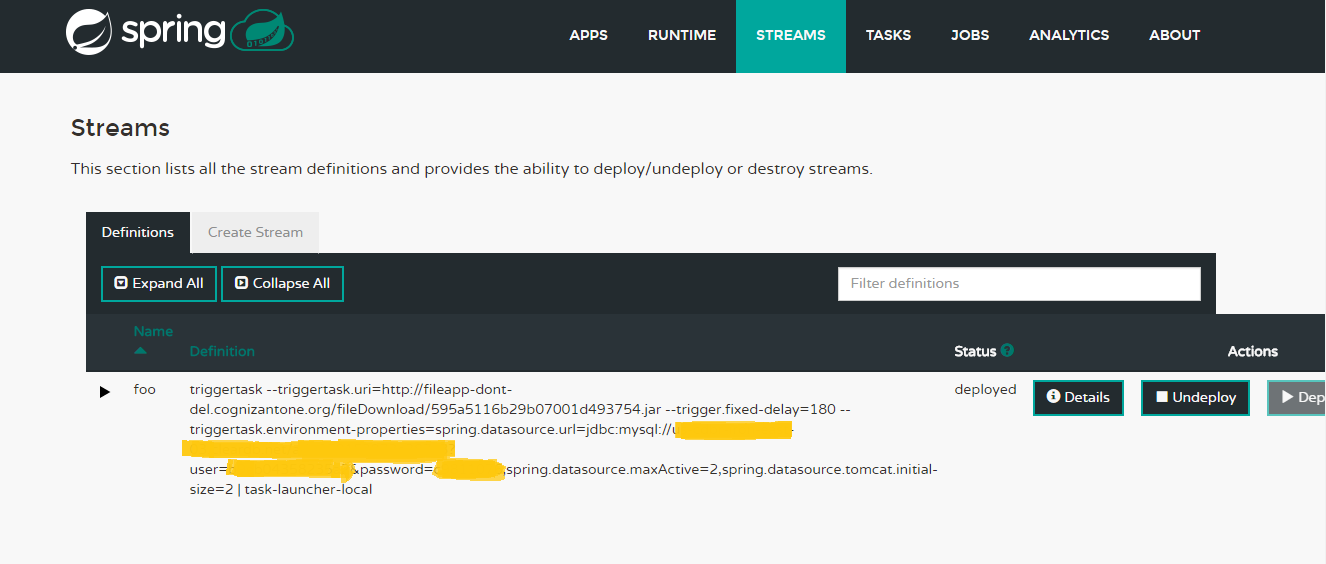
**app register --name** **task-launcher-local --type sink --uri** **http://fileapp-dont-del.cognizantone.org/fileDownload/5957ce0f42d9d0001dff3866.jar**

1. The registered application can be seen in the spring dashboard as well.

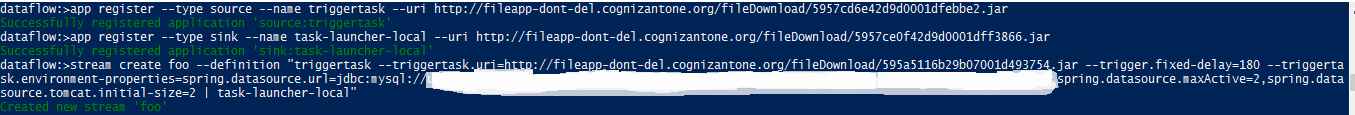


1. Stream creation and job scheduling are configured in the below commands.

**stream create** foo **--definition** **"triggertask** **--triggertask.uri=http://fileapp-dont-del.cognizantone.org/fileDownload/595a5116b29b07001d493754.jar** **--trigger.fixed-delay=180** **--triggertask.environment-properties=spring.datasource.url=jdbc:mysql://xxxxxxx.net/dbname?user=yyyyy&password=Zzzzzz,spring.datasource.maxActive=2,spring.datasource.tomcat.initial-size=2 | task-launcher-local"** **–deploy**



1. The executed commands and response message can be seen in the below image.

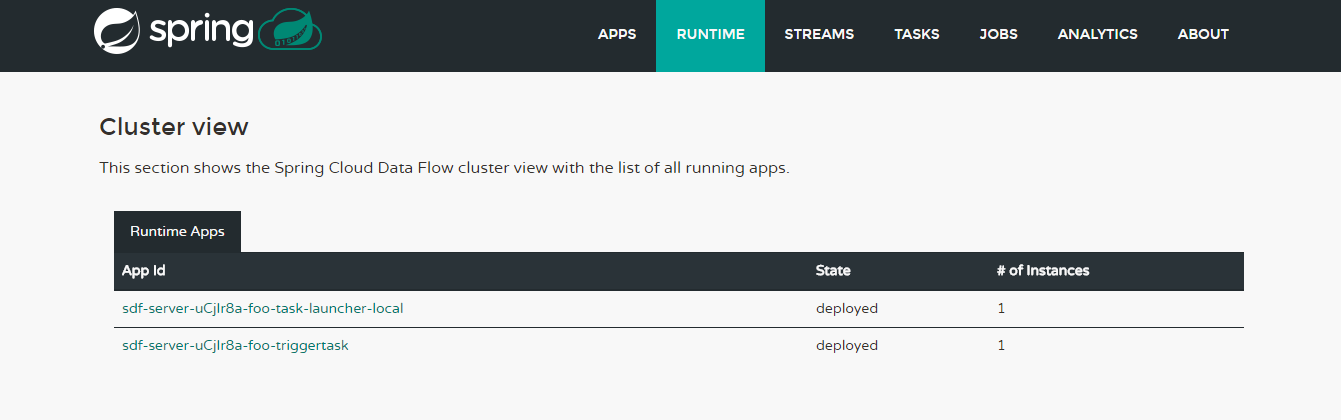


**Note:** MySQL credentials made hidden in the above images in order to avoid any conflict.

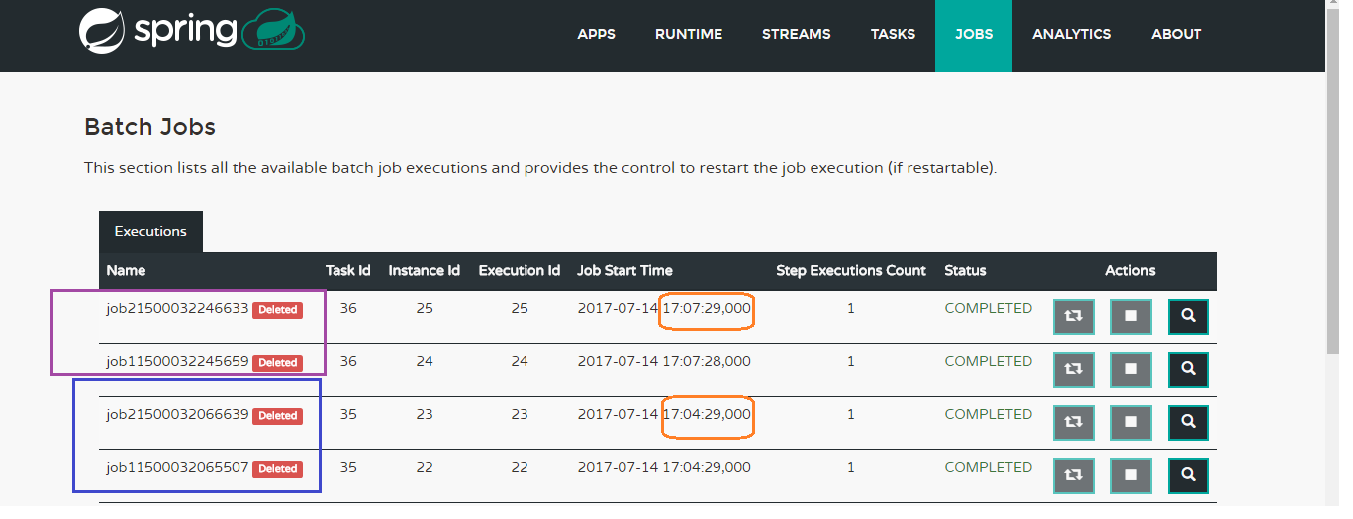
1. This job is scheduled every 180 seconds, which is mentioned in the **trigger.fixed-delay=180** in the above stream creation command.

As you all aware of this, the stream is a long lived process which can handle unbounded data pipelines whereas Cloud task/jobs are short lived process which would destroy and release the container and resources once batch/task process is completed.

1. And also you can see the run time application Id and status of deployed stream applications.

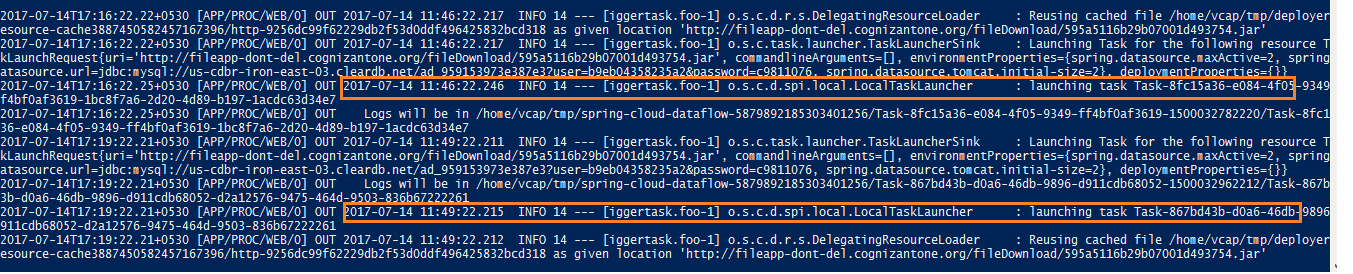


1. Once the job is triggered as scheduled, the executed job status can be seen in the Jobs menu (below screenshot). In this example, the two jobs are executed based on schedule with an interval of 180 seconds.



1. The executed job logs can also be seen in the command prompt. In this example, you can see the two different launch on every 180 seconds in the log files.

**cf logs** **sdf-server-uCjIr8a-foo-task-launcher-local**



# **Spring Data Flow Vs Spring Batch**

* Spring Cloud Data Flow is a Cloud Native composed micro services for ingesting, transforming, storing, and analyzing data.
* Spring Cloud Data Flow enables developers to build both real-time and batch processing applications using model as their Spring Boot RESTful microservices
* Scaling capabilities, canary deployments, dynamic resource allocation, and distributed tracing can be done efficiently in the Spring Data flow than Spring Batch.
* The registered batch job in Spring data flow, you can launch/re-launch it anytime. You could use Spring data flow REST-APIs, Shell or Dashboard to do so.
* Another important feature is interoperability between stream and tasks like Stream trigger the task jobs.

# **Change Log**

Please note that this table needs to be maintained even if a Configuration Management tool is used.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version Number** | **Changes made** | | | |
| V1.0 | *<First version>* | | | |
| V1.1 | *Command Steps and descriptions have changed.* | | | |
| Page no | Changed by | Effective date | Changes effected |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |