

Spring Cloud Data Flow for Pivotal Cloud Foundry® Documentation

Version 1.0.1

Published: 15 March 2019

Spring Cloud® Data Flow for PCF

Note: Spring Cloud Data Flow for PCF v1.0.x is no longer supported. The support period for v1.0.x has expired. To stay up to date with the latest software and security updates, upgrade to a supported version.

About Spring Cloud Data Flow

Spring Cloud Data Flow is an open-source toolkit for data integration and real-time data processing (see the [project home page](#)). The Spring Cloud Data Flow (SCDF) server deploys pipelines composed of [Spring Cloud Stream](#) or [Spring Cloud Task](#) applications. A domain-specific language (DSL) makes it easy to describe the applications in a pipeline and how they are connected.

The SCDF server exposes a REST API for composing and deploying data pipelines and a dashboard with a graphical pipeline editor. It also includes a shell for working with the API from the command line.

About Spring Cloud Data Flow for PCF

Spring Cloud Data Flow for Pivotal Cloud Foundry (PCF) automates the deployment of SCDF and its dependent services so that developers can use Apps Manager to deploy their own SCDF instances. They can then access their Data Flow server, either via the SCDF dashboard UI or using their PCF credentials with the SCDF shell.

Key Features

- Spring Cloud Data Flow for PCF includes the following key features:
- Addition of the Spring Cloud Data Flow server to the Marketplace as a managed service
 - Automatic integration with dependent PCF services:
 - MySQL for PCF for apps, pipelines and task history
 - RabbitMQ for PCF for event messaging
 - Redis for PCF for capturing analytics data
 - Integration of Data Flow server with PCF's UAA security model

Product Snapshot

The following table provides version and version-support information about Spring Cloud Data Flow for PCF.

Element	Details
Version	v1.0.1
Release date	March 9, 2018
Software component version	v1.3.0.RELEASE
Compatible Ops Manager version(s)	v1.10.x, v1.11.x, v1.12.x and v2.0.x
Compatible Elastic Runtime version(s)	v1.10.x, v1.11.x, v1.12.x and v2.0.x
IaaS support	All supported by PCF

Requirements

- Spring Cloud Data Flow for PCF has the following requirements:
- [Java Cloud Foundry buildpack](#) version 3.8 or later (see the [Prerequisites](#) section of [Installation](#) for more information)
 - [MySQL for PCF v2](#), or an alternative relational database service
 - [RabbitMQ for PCF](#), or an alternative RabbitMQ or Kafka service
 - [Redis for PCF](#), or an alternative Redis service
 - [Cloud Foundry Command Line Interface](#) (cf CLI)

Optional

To use the Spring Cloud Data Flow shell interface with Spring Cloud Data Flow for PCF service instances, install the following cf CLI plugins:

- [Spring Cloud Data Flow for PCF cf CLI plugin](#). To install the plugin, run the following command:

```
$ cf install-plugin -r CF-Community "spring-cloud-dataflow-for-pcf"
```

- [Java Runtime Environment](#) (JRE). Required to run the Data Flow shell. You can download the JRE from the [Java website](#).
- [Service Instance Logging cf CLI plugin](#). To install the plugin, run the following command:

```
$ cf install-plugin -r CF-Community "Service Instance Logging"
```

Please provide any bugs, feature requests, or questions to the [Pivotal Cloud Foundry Feedback](#) list.

Spring Cloud Data Flow Release Notes

Release notes for Spring Cloud Data Flow for PCF

v1.0.1

Release Date: March 9, 2018

Features included in this release:

- Operators can now configure the default relational database service name and plan for the Data Flow server and Skipper applications when creating Spring Cloud Data Flow service instances.

Issues resolved in this release:

- Removed unintentional hard dependency on MySQL for PCF v1 tile.
- Fixed race condition that could cause service instance creation to fail when mapping the Skipper application route.
- Fixed issue when deleting service instances that use asynchronously-deleted backing data services.

v1.0.0

Release Date: February 13, 2018

Features included in this release:

- A Spring Cloud Data Flow service offering is added to the Pivotal Cloud Foundry Apps Manager® Services Marketplace.
- The Spring Cloud Data Flow service is fully integrated with PCF's security model, using UAA for authentication and access control.
- Creation of a Spring Cloud Data Flow service instance results in creation of the following Spring Boot applications in a backing org and space:
 - Spring Cloud Data Flow server
 - Spring Cloud Data Flow metrics collector
 - Spring Cloud Skipper package manager
- Creation of a Spring Cloud Data Flow service instance results in creation of four backing data service instances made available to the Spring Cloud Data Flow service instance's backing Data Flow server, metrics collection, and Skipper package management applications, as well as to applications deployed via the Spring Cloud Data Flow service instance:
 - Two relational database services (MySQL for PCF is the optional default)
 - A messaging service (RabbitMQ for PCF is the optional default)
 - An analytics service (Redis for PCF service instance is the optional default)

Alternative backing data services can be specified by providing either a service offering name and plan or configuration for a user-provided service instance.

- The Spring Cloud Data Flow shell can be used to interact locally with a Spring Cloud Data Flow service instance. For more information about the shell, see the [OSS Spring Cloud Data Flow documentation](#).
- A Service Instances Dashboard is available from Apps Manager and lists all provisioned Spring Cloud Data Flow service instances, with links to view backing server application logs for each service instance as well as controls for launching streams and tasks. For more information about the dashboard, see the [OSS Spring Cloud Data Flow documentation](#).

Known issues in this release:

- The service broker fails to delete a Spring Cloud Data Flow service instance that uses an asynchronous backing service. An attempt to delete such a service instance may result in a message similar to the following:

```
status: delete failed
message: CF-AsyncRequired(10001): Service instance relational-service: This service
        plan requires client support for asynchronous
        service operations.
```

- There is an unintentional hard dependency on [MySQL for PCF v1](#). The `deploy-all` errand will fail with `400` or `500` HTTP status code response depending on the services available in the PCF Marketplace catalog. A fix is scheduled to be released in the `v1.0.1` release of SCDF for PCF.
- The `run-tests` errand requires access to the Internet. If your PCF environment does not allow errand VMs to access the Internet, this errand will fail. If the `run-tests` errand fails, this indicates that the service broker has been installed and made available across PCF orgs. In this case, Pivotal recommends disabling the `run-tests` errand and manually verifying the broker's functionality by creating a service instance.

Installing and Configuring Spring Cloud® Data Flow for PCF

Follow the below steps to install Spring Cloud Data Flow for PCF.

Prerequisites

Spring Cloud Data Flow for PCF is built using Spring Boot 1.5, which requires version 3.8 or later of the Java Cloud Foundry buildpack. The default Java buildpack—the buildpack at the lowest position of all Java buildpacks—on your PCF installation must therefore be at version 3.8 or later.

You can use the Cloud Foundry Command Line Interface tool (cf CLI) to see the version of the Java buildpack that is currently installed.

```
$ cf buildpacks
Getting buildpacks...

buildpack      position  enabled  locked  filename
java_buildpack_offline  1      true    false   java-buildpack-offline-v3.8.1.zip
ruby_buildpack      2      true    false   ruby_buildpack-cached-v1.6.19.zip
nodejs_buildpack    3      true    false   nodejs_buildpack-cached-v1.5.15.zip
go_buildpack        4      true    false   go_buildpack-cached-v1.7.10.zip
```

If the default Java buildpack is older than version 3.8, you can download a newer version from [Pivotal Network](#) and update Pivotal Cloud Foundry by following the instructions in the [Managing Custom Buildpacks](#) topic. To ensure that the newer buildpack is the default Java buildpack, you may delete or disable the older buildpack or make sure that the newer buildpack is in a lower position.

Installation Steps

1. Download the product file from [Pivotal Network](#).
2. Navigate to the Ops Manager Installation Dashboard and click **Import a Product** to upload the product file.
3. Under the **Import a Product** button, click + next to the version number of Spring Cloud® Data Flow for PCF. This adds the tile to your staging area.
4. Click the newly added **Spring Cloud Data Flow** tile. In the **Settings** tab, click **Assign AZs and Networks**.

Select the availability zones for the tile to use. In the **Network** section, select the PAS (or ERT) network.

5. In the **Settings** tab, click **Service Broker**.

Enter the relational database service name and plan name for the Spring Cloud Data Flow tile to use for storing its service broker's service instance data. If you have the MySQL for PCF v2 tile installed and it has an active **db-small** plan, you may leave these fields unmodified.

Note: If you are not using the MySQL for PCF v2 default for the service broker's database, you must provide information for another relational database service from your PCF marketplace (see the [PCF documentation](#)).

Spring Cloud Data Flow for PCF supports MySQL for PCF v1. If you wish to use it, set this service name to **p-mysql** and set the plan name to an active MySQL for PCF v1 plan.

6. In the **Settings** tab, click **Data Flow Server**.

- Assign AZs and Networks
- Service Broker
- Data Flow Server**
- Skipper
- Service Access
- Errands
- Resource Config

Default configuration settings for Spring Cloud Data Flow servers

Service name to use for default Data Flow server relational database *

Service plan to use for default Data Flow server relational database *

Save

Configure the relational database service name and plan used by the Data Flow server application deployed for each Data Flow service instance. If you have the MySQL for PCF v2 tile installed and it has an active `db-small` plan, you can leave these fields unmodified.

7. Still in the **Settings** tab, click **Skipper**.

- Assign AZs and Networks
- Service Broker
- Data Flow Server
- Skipper**
- Service Access
- Errands
- Resource Config

Default configuration settings for Spring Cloud Skipper

Service name to use for default Skipper relational database *

Service plan to use for default Skipper relational database *

Save

Configure the relational database service name and plan used by the Skipper application deployed for each Data Flow service instance. If you have the MySQL for PCF v2 tile installed and it has an active `db-small` plan, you can leave these fields unmodified.

8. Still in the **Settings** tab, click **Errands**.

- Assign AZs and Networks
- Service Broker
- Service Access
- Errands**
- Resource Config
- Stemcell

Errands

Errands are scripts that run at designated points during an installation.

Post-Deploy Errands

create-uaa-client

deploy-all

run-tests

Pre-Delete Errands

delete-all

Save

Each of the four lifecycle errands can be set to run conditionally (**When Changed**), always (**On**), or not at all (**Off**). Pivotal recommends that all Spring Cloud Data Flow lifecycle errands be set to always run (**On**).

Important: The “run-tests” errand requires access to the Internet. If your Pivotal Cloud Foundry deployment does not allow external network access, set this errand to **Off**.

9. Return to the Ops Manager Installation Dashboard and click **Apply Changes** to install the Spring Cloud® Data Flow tile.

Tile Configuration

See below for information about Spring Cloud® Data Flow’s deployment model and other information which may be useful in administering Data Flow service instances or deployed applications.

Note: Ops Manager administrators can use Role-Based Access Control (RBAC) to manage which operators can make deployment changes, view credentials, and manage user roles in Ops Manager. Therefore, your role permissions might not allow you to perform every procedure in this operator guide. For more information about roles in Ops Manager, see [Understand Roles in Ops Manager](#).

Tile Configuration Options

The Spring Cloud Data Flow tile includes settings for various options. You can configure these by visiting the Installation Dashboard of Pivotal Cloud Foundry® Operations Manager and clicking the Spring Cloud Data Flow tile.

Configure Broker Database Service and Plan

By default, the Spring Cloud Data Flow product uses the MySQL for PCF v2 product and its default `db-small` service plan to provision a database service instance for the Data Flow service broker. If you are using the MySQL for PCF v2 product and it does not have an active `db-small` plan or if you wish to use an alternative service, you must configure the Spring Cloud Data Flow product for the service and service plan you wish to use. You can configure an alternative relational database service and service plan in the **Service Broker** pane of the Spring Cloud Data Flow settings.

Assign AZs and Networks

Service Broker

Data Flow Server

Skipper

Service Access

Errands

Resource Config

Configuration settings for the Spring Cloud Data Flow for PCF service broker.

Service broker database service name *

p.mysql

The database service name to use for storing service broker data. DO NOT change when updating tile.

Service broker database service plan name *

db-small

Save

The broker’s database service should be configured only during the Spring Cloud Data Flow tile installation process. If you have already completed the tile installation process, do not alter that setting (you may still configure the broker’s database service plan after the tile installation).

warning: Configuring the service broker’s dependent relational database service after the tile installation has finished can result in orphaned Data Flow service instances or in multiple data sets and corruption of the broker database’s data.

See below for more information about the results of configuring the service broker’s relational database.

If you	and change	then
install the tile	the service name or plan	the tile uses the specified service and plan. You may also use the default values of <code>p.mysql</code> and <code>db-small</code> .
update the tile	the service plan	the tile’s relational database service instance is changed to use the specified plan. No data is lost.
update the tile	the service name	the service name change is ignored. If you have also changed the service plan, this may leave the broker in an inoperative state. To resolve this issue, revert your changes.

Configure Data Flow Server Database Service and Plan

By default, the Spring Cloud Data Flow product uses the MySQL for PCF v2 product and its default `db-small` service plan to provision a database service instance for the Data Flow server application deployed for each Data Flow service instance. If you are using the MySQL for PCF v2 product and it does not have an active `db-small` plan or if you wish to use an alternative service, you must configure the service and service plan you wish to use for the Data Flow server. You can configure an alternative relational database service and service plan in the **Data Flow Server** pane of the Spring Cloud Data Flow settings.

Assign AZs and Networks

Service Broker

Data Flow Server

Skipper

Service Access

Errands

Resource Config

Default configuration settings for Spring Cloud Data Flow servers

Service name to use for default Data Flow server relational database *

p.mysql

Service plan to use for default Data Flow server relational database *

db-dev

Save

Configure Skipper Database Service and Plan

By default, the Spring Cloud Data Flow product uses the MySQL for PCF v2 product and its default `db-small` service plan to provision a database service instance for the Skipper application deployed for each Data Flow service instance. If you are using the MySQL for PCF v2 product and it does not have an active `db-small` plan or if you wish to use an alternative service, you must configure the service and service plan you wish to use for Skipper. You can configure an alternative relational database service and service plan in the **Skipper** pane of the Spring Cloud Data Flow settings.

Assign AZs and Networks

Service Broker

Data Flow Server

Skipper

Service Access

Errands

Resource Config

Default configuration settings for Spring Cloud Skipper

Service name to use for default Skipper relational database *

p.mysql

Service plan to use for default Skipper relational database *

db-dev

Save

Enable or Disable Global Marketplace Access

By default, the Spring Cloud Data Flow product enables access to its service offering, called `p-dataflow`, across all orgs in the PCF deployment as part of the tile installation process. If you wish to manually grant service access to specific organizations, you can configure the default global access in the **Service Access** pane of the Spring Cloud Data Flow settings.

PCF Ops Manager

pivotalcf

Installation Dashboard

Spring Cloud Data Flow

Settings

Status

Credentials

Logs

Assign AZs and Networks

Service Broker

Service Access

Errands

Resource Config

Stemcell

Determine availability of services

Enable global access to plans of service p_dataflow

Save

To disable the default of service access enabled for all orgs, clear the **Enable global access to plans of service `p_dataflow`** checkbox. Click **Save**, return to the Installation Dashboard, and apply your changes. You can now enable or disable access to the Data Flow service offering for specific orgs.

The Service Broker and Instances

See below for information about Spring Cloud® Data Flow's deployment model and other information which may be useful in administering Data Flow service instances or deployed applications.

Note: Ops Manager administrators can use Role-Based Access Control (RBAC) to manage which operators can make deployment changes, view credentials, and manage user roles in Ops Manager. Therefore, your role permissions might not allow you to perform every procedure in this operator guide. For more information about roles in Ops Manager, see [Understand Roles in Ops Manager](#).

The Service Broker

Spring Cloud Data Flow provides a Spring Cloud Data Flow server as a [Managed Service](#) on [Pivotal Cloud Foundry](#) (PCF). It uses Cloud Foundry's [Service Broker API](#) to manage this service. See below for information about Spring Cloud Data Flow's broker implementation.

The Spring Cloud Data Flow service broker's functionality is contained in the following Spring Boot application instance, which is deployed in the "system" organization to the "p-dataflow" space.

- **p-dataflow-[version]:** Implements the Service Broker API to act on provision, deprovision, bind, and unbind requests.

The broker relies on the [MySQL for Pivotal Cloud Foundry v2](#) product for the following service instance.

- **p_dataflow-p-mysql:** A MySQL database used as a backing store for the service broker.

Note: You can configure an alternate relational database service for the broker to use. See the [Configure Broker Database Service and Plan](#) section of the [Tile Configuration](#) topic.

Service Broker Upgrades

The Spring Cloud Data Flow product upgrade process checks before redeploying the service broker to see whether the product's version has changed. If the version has not changed, the upgrade process will continue without redeploying the service broker.

The service broker application is deployed using a [blue-green deployment strategy](#). During an upgrade of the service broker, the broker will continue processing requests to provision, deprovision, bind, and unbind service instances, without downtime.

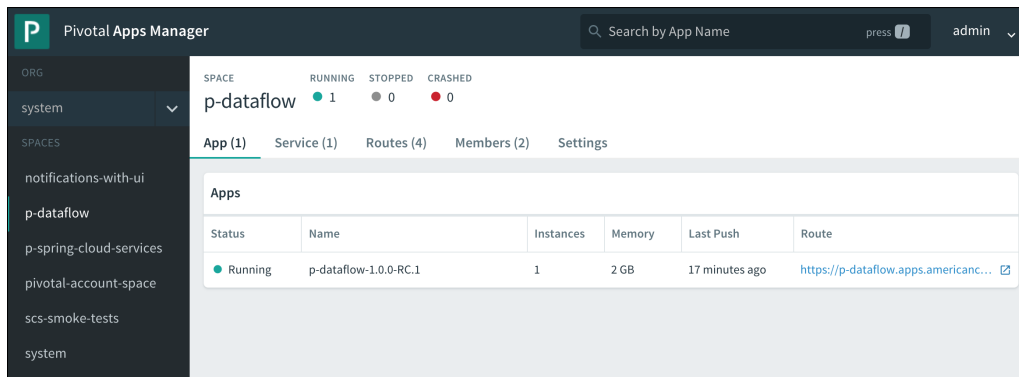
Access Via Apps Manager

To view the broker application in Pivotal Cloud Foundry® Apps Manager, log into Apps Manager as an admin user and select the "system" org.

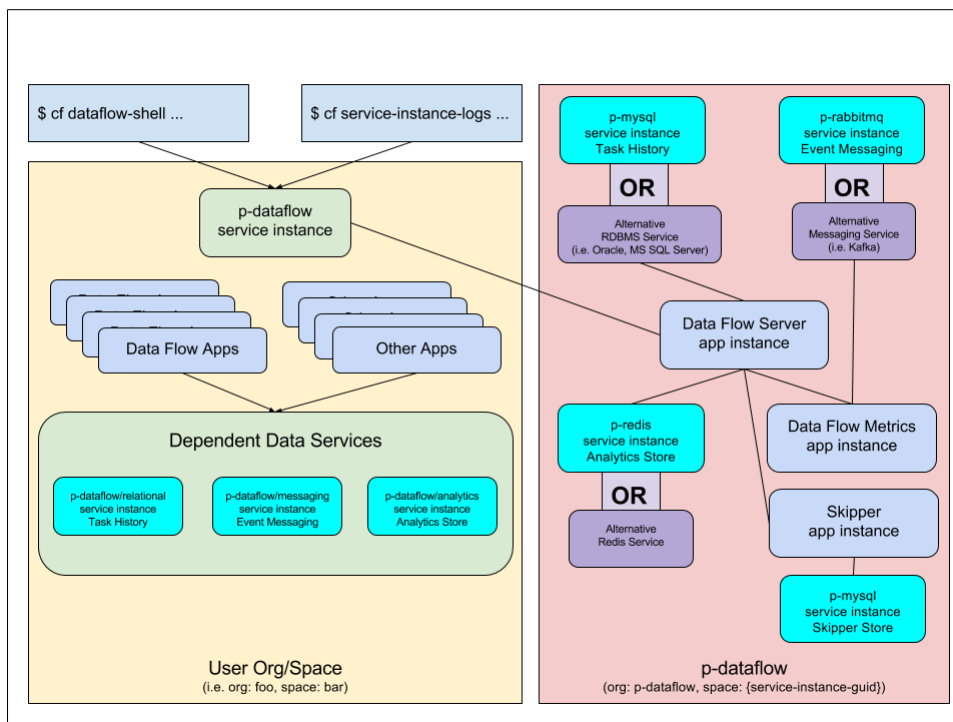
The screenshot displays the Pivotal Apps Manager interface. On the left is a dark sidebar with a navigation menu. The main content area shows the 'system' organization selected, with a quota bar indicating 9.38 GB / 100 GB (9%). Below the organization header, there are tabs for 'Spaces (6)', 'Domains (2)', 'Members (6)', and 'Settings'. The 'Spaces (6)' tab is active, showing a grid of space cards. Each card represents a space and contains a table of applications (APPS) and services (SERVICES) with their status (green, grey, or red dots) and counts. The spaces shown are 'notifications-with-ui', 'p-dataflow', 'pivotal-account-space', and 'scs-smoke-tests'.

Space	APPS	SERVICES	Quota
notifications-with-ui	1 (Green), 0 (Grey), 0 (Red)	0	0% of Org Quota
p-dataflow	1 (Green), 1 (Grey), 0 (Red)	1	0% of Org Quota
pivotal-account-space	2 (Green), 1 (Grey), 0 (Red)	0	1% of Org Quota
scs-smoke-tests	0 (Green), 0 (Grey), 0 (Red)	0	0% of Org Quota

The application is deployed in the “p-dataflow” space.



Service Instance Architecture



For each Spring Cloud Data Flow service instance created, the service broker provisions the following resources, all within the space from which the service instance was created (“the user space”) unless noted otherwise.

- A new space within the “p-dataflow” org, named using the service instance GUID and containing:
 - A Data Flow server application.
 - A Data Flow metrics application.
 - A Skipper package management application.
 - A relational database service, bound to the Data Flow server application.

Note: This relational database service is a “p.mysql” service instance by default. You can configure an alternate relational database when you create the service instance.

- A relational database service, bound to the Skipper package management application.
- A messaging data service, bound to the Data Flow metrics application.

Note: This messaging data service is a “p.rabbitmq” service instance by default. You can configure an alternate messaging service when you create the service instance.

- A Redis database service, bound to the Data Flow server application.
- The following resources are created in the originating user space where the service instance command was targeted at:
 - A “p-dataflow” service instance.
 - A “p-dataflow” relational database service instance (providing access to the relational database service created in the service instance’s space within the “p-dataflow” org).
 - A “p-dataflow” messaging service instance (providing access to the messaging data service created in the service instance’s space within the “p-dataflow” org).
 - A “p-dataflow” analytics service instance (providing access to the analytics service created in the service instance’s space within the “p-dataflow” org).

org).

Capacity Requirements

Below are the usage requirements of the Spring Cloud Data Flow service broker.

Application	Memory Allocated	Disk Allocation
Service Broker	2 GB	1 GB

The service broker is bound to a relational database service instance, which stores data relating to the broker's service instances. The relational database service to use is configurable, as described in [The Service Broker](#) section above.

Below are the usage requirements of the Data Flow server and metrics applications that back each Spring Cloud Data Flow service instance.

Backing Application	Memory Allocation / App Instance	Disk Allocation / App Instance
Data Flow Server	2 GB	2 GB
Data Flow Metrics	1 GB	1 GB
Skipper	1 GB	2 GB

Spring Cloud Data Flow service instances are also backed by instances of other PCF services. These are either services from PCF data service products or custom services provided to a Data Flow service instance at create time. These data services include a relational database service, a messaging service, and an analytics database service instance for each Data Flow service instance created. The Skipper backing application uses a MySQL database service for its backing store.

Managing Service Instances

See below for information about managing Data Flow service instances using the Cloud Foundry Command Line Interface tool (cf CLI). You can also manage Data Flow service instances using Pivotal Cloud Foundry® Apps Manager.

Creating an Instance

Important: If you are using the [Redis for PCF](#) product for the Spring Cloud Data Flow analytics store, you cannot create more Data Flow service instances than the setting of the Redis product's Service Instance Limit (the default Service Instance Limit for Redis for PCF is 5). See the [Shared-VM Plan](#) section of the [Installing and Upgrading Redis for PCF](#) topic in the [Redis for PCF documentation](#) for information about configuring this limit.

Begin by targeting the correct org and space.

```
$ cf target -o myorg -s development
api endpoint: https://api.system.wise.com
api version: 2.75.0
user: user
org: myorg
space: development
```

You can view plan details for the Data Flow product using `cf marketplace -s`.

```
$ cf marketplace
Getting services from marketplace in org myorg / space development as user...
OK

service      plans  description
p-dataflow   standard Deploys Spring Cloud Data Flow servers to orchestrate data pipelines
p-dataflow-mysql proxy Proxies to the Spring Cloud Data Flow MySQL service instance
p-dataflow-rabbitmq proxy Proxies to the Spring Cloud Data Flow RabbitMQ service instance
p-dataflow-redis proxy Proxies to the Spring Cloud Data Flow Redis service instance

TIP: Use 'cf marketplace -s SERVICE' to view descriptions of individual plans of a given service.

$ cf marketplace -s p-dataflow
Getting service plan information for service p-dataflow as user...
OK

service plan  description  free or paid
standard      Standard Plan free
```

Each Data Flow service instance uses three dependent data services. Defaults for these services can be configured in the tile settings, and these defaults can be overridden for each individual service instance at create time.

Note: The service offerings with the plan `proxy` are proxy services used by Spring Cloud Data Flow for PCF service instances. The Spring Cloud Data Flow service broker creates and deletes instances of these services automatically along with each Spring Cloud Data Flow service instance. Do not manually create or delete instances of these services.

General parameters used to configure dependent data services for a Data Flow service instance are listed below.

Parameter	Function
<code>relational-data-service.name</code>	The name of the service to use for a relational database that stores Spring Cloud Data Flow metadata and task history.
<code>relational-data-service.plan</code>	The name of the service plan to use for the relational database service.
<code>messaging-data-service.name</code>	The name of the service to use for a RabbitMQ or Kafka server that facilitates event messaging.
<code>messaging-data-service.plan</code>	The name of the service plan to use for the RabbitMQ or Kafka service.
<code>analytics-data-service.name</code>	The name of the service to use for a Redis server that stores analytics.
<code>analytics-data-service.plan</code>	The name of the service plan to use for the Redis server.
<code>skipper-relational.name</code>	The name of the service to use for a relational database used by the Skipper application.
<code>skipper-relational.plan</code>	The name of the service plan to use for a relational database used by the Skipper application.

Each Data Flow service instance can optionally specify Maven configuration properties. For the complete list of properties that can be specified, see the [“Maven” section in the OSS Spring Cloud Data Flow documentation](#).

Maven configuration properties can be set for each Data Flow service instance using parameters given to `cf create-service`. To set the

`maven.remote-repositories.repo1.url` property, you might use a command such as the following:

```
$ cf create-service p-dataflow standard data-flow -c '{"maven.remote-repositories.repo1.url": "https://repo.spring.io/libs-snapshot"}'
```

Create the service instance using `cf create-service`. To create a Data Flow service that uses a Redis Cloud service available from your PCF marketplace and

sets the Maven `maven.remote-repositories.repo1.url` property to `https://repo.spring.io/release`, you might run:

```
$ cf create-service p-dataflow standard data-flow -c '{ "analytics-data-service": { "name": "rediscloud", "plan": "30mb" }, "maven.remote-repositories.repo1.url": "https://repo.spring.io/libs-snaps" }'
Creating service instance data-flow in org myorg / space development as user...
OK

Create in progress. Use 'cf services' or 'cf service data-flow' to check operation status.
```

As the command output suggests, you can use the `cf services` or `cf service` commands to check the status of the service instance. When the service instance is ready, the `cf service` command will give a status of `create succeeded`:

```
$ cf service data-flow

Service instance: data-flow
Service: p-dataflow
Bound apps:
Tags:
Plan: standard
Description: Deploys Spring Cloud Data Flow servers to orchestrate data pipelines
Documentation url: http://cloud.spring.io/spring-cloud-dataflow/
Dashboard: https://p-dataflow.apps.wise.com/instances/f09e5c77-c526-4f49-86d6-721c6b8e2fd9/dashboard

Last Operation
Status: create succeeded
Message: Created
Started: 2017-07-20T18:24:14Z
Updated: 2017-07-20T18:26:17Z
```

Deleting an Instance

Deleting a Data Flow service instance will result in deletion of all of its dependent service instances.

Begin by targeting the correct org and space.

```
$ cf target -o myorg -s development
api endpoint: https://api.system.wise.com
api version: 2.75.0
user: user
org: myorg
space: development
```

You can view all service instances in the space using `cf services`.

```
$ cf services
Getting services in org myorg / space development as user...
OK

name          service      plan  bound apps  last operation
data-flow     p-dataflow  standard  create succeeded
mysql-b3e76c87-c5ae-47e4-a83e-5fab2fc4f11  p-dataflow-mysql  proxy  create succeeded
rabbitmq-b3e76c87-c5ae-47e4-a83e-5fab2fc4f11  p-dataflow-rabbitmq  proxy  create succeeded
redis-b3e76c87-c5ae-47e4-a83e-5fab2fc4f11  p-dataflow-redis  proxy  create succeeded
```

Delete the Data Flow service instance using `cf delete-service`. When prompted, enter `y` to confirm the deletion.

```
$ cf delete-service data-flow

Really delete the service data-flow?>y
Deleting service data-flow in org myorg / space development as user...
OK

Delete in progress. Use 'cf services' or 'cf service data-flow' to check operation status.
```

The dependent service instances for the Data Flow server service instance are deleted first, and then the Data Flow server service instance itself is deleted.

As the output from the `cf delete-service` command suggests, you can use the `cf services` or `cf service` commands to check the status of the service instance.

When the Data Flow service instance and its dependent service instances have been deleted, the `cf services` command will no longer list the service instance:

```
$ cf services
Getting services in org bklein / space dev as bklein...
OK

No services found
```

Using User-Provided Service Instances

By default, a Spring Cloud Data Flow service instance uses MySQL for PCF for its backing relational database service, RabbitMQ for PCF for its backing messaging service, and Redis for PCF for its backing analytics service. When creating a Spring Cloud Data Flow service instance, you can configure it to use [Cloud Foundry user-provided services](#) for its dependent data services instead. See below for information about configuring a Data Flow service instance to use an alternative dependent data service.

Dependent Service Parameters

You can specify that a Data Flow service instance should use a user-provided service instance by passing configuration parameters to the `cf create-service` command in JSON using the `-c` flag. The parameters used to configure alternative dependent data services for a Data Flow service instance are listed below.

Parameter	Function
<code>relational-data-service.user-provided</code>	A JSON object containing connection information for the relational database service used to store task history.
<code>messaging-data-service.user-provided</code>	A JSON object containing connection information for the messaging service used for event messaging.
<code>analytics-data-service.user-provided</code>	A JSON object containing connection information for the Redis service used for an analytics store.

See the following sections for information about using these parameters.

User-Provided Relational Database Service

You can use the `relational-data-service` parameter to supply the configuration for a user-provided relational database service. This parameter contains a JSON object `user-provided`, with fields for each of the connection values needed to provision a Cloud Foundry user-provided relational database service instance.

The relational database service can be the default MySQL for PCF v2 service or any other relational database service. It must provide a relational database.

An example of fields contained in the `user-provided` object for a relational database service is shown below.

```
{
  "uri": "mysql://kempercabb:fake-password@123.234.456.879:3306/vigil",
  "username": "kempercabb",
  "password": "fake-password",
  "dbname": "vigil",
  "host": "123.234.456.879",
  "port": 3306,
  "tags": ["mysql"]
}
```

To create a Spring Cloud Data Flow service instance using these connection values for the service instance's relational database service instance:

```
$ cf create-service p-dataflow standard data-flow -c '{"relational-data-service": {"user-provided": {"uri": "mysql://kempercabb:fake-password@123.234.456.879:3306/vigil", "username": "kempercabb", "password": "fake-password", "dbname": "vigil", "host": "123.234.456.879", "port": 3306, "tags": ["mysql"]}}}'
```

User-Provided Messaging Service

You can use the `messaging-data-service` parameter to supply the configuration for a user-provided messaging service. This parameter contains a JSON object `user-provided`, with fields for each of the connection values needed to provision a Cloud Foundry user-provided messaging service instance.

The messaging data service can be the default RabbitMQ for PCF service or another messaging service. It must provide either a RabbitMQ server or a Kafka server.

An example of fields contained in the `user-provided` object for a messaging service is shown below.

```
{
  "dashboard_url": "https://api.cloudamqp.com/console/a-GUID-would-go-here/details",
  "username": "buckstorm",
  "vhost": "buckstorm",
  "password": "fake-password",
  "ssl": false,
  "hostname": "stonehill.rmqs.cloudamqp.com",
  "uri": "amqp://buckstorm:fake-password@stonehill.rmqs.cloudamqp.com/buckstorm",
  "http_api_uri": "http://buckstorm:fake-password@stonehill.rmqs.cloudamqp.com:1883/api",
  "tags": ["rabbitmq"]
}
```

To create a Spring Cloud Data Flow service instance using these connection values for the service instance's messaging service instance:

```
$ cf create-service p-dataflow standard dataflow -c '{"messaging-data-service": {"user-provided": {"dashboard_url": "https://api.cloudamqp.com/console/a-GUID-would-go-here/details", "username": "buckstorm", "vhost": "buckstorm", "password": "fake-password", "ssl": false, "hostname": "stonehill.rmqs.cloudamqp.com", "uri": "amqp://buckstorm:fake-password@stonehill.rmqs.cloudamqp.com/buckstorm", "http_api_uri": "http://buckstorm:fake-password@stonehill.rmqs.cloudamqp.com:1883/api", "tags": ["rabbitmq"]}}}'
```

User-Provided Analytics Service

You can use the `analytics-data-service` parameter to supply the configuration for a user-provided analytics service. This parameter contains a JSON object `user-provided`, with fields for each of the connection values needed to provision a Cloud Foundry user-provided Redis service instance.

The analytics service can be the default Redis for PCF service or any other Redis service. It must provide a Redis database.

An example of fields contained in the `user-provided` object for an analytics service is shown below.

```
{
  "host": "garrels.gce.cloud.redislabs.com",
  "uri": "redis://fake-password@garrels.gce.cloud.redislabs.com:11781/boars-head",
  "port": 11781,
  "dbname": "boars-head",
  "password": "fake-password",
  "tags": ["redis"]
}
```

To create a Spring Cloud Data Flow service instance using these connection values for the service instance's analytics service instance:

```
$ cf create-service p-dataflow standard dataflow -c {"analytics-data-service": {"user-provided": {"host":"garrels.gce.cloud.redislabs.com","uri":"redis://fake-password@garrels.gce.cloud.redisla
```

Viewing Service Instance Logs

Spring Cloud Data Flow for PCF provides access to the logs generated by each Data Flow server service instance, including logs for each of the three backing applications (Data Flow server application, metrics application, and Skipper application) for each instance. You can view these logs either using the Service Instance Logs cf CLI plugin or by visiting the dashboard of the Spring Cloud Data Flow service broker.

Using the cf CLI Plugin

After installing the [Service Instance Logs cf CLI plugin](#) (see the instructions in the [Installing](#) section of the plugin's [README](#)), you can use the `service-logs` command to tail logs or dump recent logs for a service instance.

To tail logs for a Data Flow service instance, run `cf service-logs SERVICE_NAME`, where `SERVICE_NAME` is the name of the service instance:

```
$ cf service-logs data-flow
```

To dump recent logs for the instance, use the `--recent` flag:

```
$ cf service-logs --recent data-flow
```

If your Pivotal Cloud Foundry deployment uses a self-signed certificate, you must use the `--skip-ssl-validation` flag to disable the default validation of the platform's SSL certificate:

```
$ cf service-logs --skip-ssl-validation data-flow
```


Using the Service Broker Dashboard

Note: To access the service broker dashboard, you must be a Space Developer in the broker application's space (this is typically the `system` org and `p-dataflow` space).

Visit the Spring Cloud Data Flow service broker's dashboard. You can access it at the following URL, where `apps.wise.com` is the application domain of your PCF deployment:

```
https://p-dataflow.apps.wise.com/
```

The dashboard shows the name, org, and space of each service instance, as well as a link to view logs for the instance.




Service Broker Dashboard

user ▾

Service Instances

Name ▾	Org	Space	Version	Logs
data-flow	myorg	development	1.3.0.RELEASE	Logs
data-flow-2	otherorg	dev	1.3.0.RELEASE	Logs

Click the **Logs** link to view logs for a particular service instance's backing application.



Service Broker Dashboard

user ▾


Logs

data-flow

2017-09-25T15:41:29.757-05:00 [RTR/0] [OUT] dataflow-f8fc3843-48b4-4a5b-bb14-fd12bcaab769.orchid.springapps.io - [2017-09-25T20:41:29.750+00:00] "GET /dashboard/scripts/stream/app.js HTTP/1.1" 200 0 1397 "https://dataflow-f8fc3843-48b4-4a5b-bb14-fd12bcaab769.orchid.springapps.io/dashboard/index.html" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.90 Safari/537.36" "10.194.38.254:34576" "10.194.38.143:60372" x_forwarded_for:"10.35.59.42, 10.194.38.254" x_forwarded_proto:"https" vcap_request_id:"6fd09dfd-51e4-4e70-4f1e-698c6ae7a295" response_time:0.006585593 app_id:"e12a36d5-bbb4-45c1-8604-95a2897a7b11" app_index:"0" x_b3_traceid:"af551dff173514cf" x_b3_spanid:"af551dff173514cf" x_b3_parentsspanid:"-"

2017-09-25T15:41:29.820-05:00 [RTR/0] [OUT] dataflow-f8fc3843-48b4-4a5b-bb14-fd12bcaab769.orchid.springapps.io - [2017-09-25T20:41:29.814+00:00] "GET /dashboard/scripts/runtime/app.js HTTP/1.1" 200 0 1251 "https://dataflow-f8fc3843-48b4-4a5b-bb14-fd12bcaab769.orchid.springapps.io/dashboard/index.html" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.90 Safari/537.36" "10.194.38.254:34568" "10.194.38.143:60372" x_forwarded_for:"10.35.59.42, 10.194.38.254" x_forwarded_proto:"https" vcap_request_id:"2b5df5c6-27ac-4f51-57ac-9f4e3cbcc1a4" response_time:0.006458999 app_id:"e12a36d5-bbb4-45c1-8604-95a2897a7b11" app_index:"0" x_b3_traceid:"f0d2b5f63195fdca"

You can stream current logs for the instance by clicking the ► button.



Service Broker Dashboard

user ▾

Logs

data-flow

```

2017-09-25T15:41:29.757-05:00 [RTR/0] [OUT] dataflow-f8fc3843-48b4-4a5b-bb14-fd12bcaab769.orchid.springapps.io - [2017-09-25T20:41:29.750+0000] "GET /dashboard/scripts/stream/app.js HTTP/1.1" 200 0 1397 "https://dataflow-f8fc3843-48b4-4a5b-bb14-fd12bcaab769.orchid.springapps.io/dashboard/index.html" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.90 Safari/537.36" "10.194.38.254:34576" "10.194.38.143:60372"
x_forwarded_for:"10.35.59.42, 10.194.38.254" x_forwarded_proto:"https" vcap_request_id:"6fd09dfd-51e4-4e70-4f1e-698c6ae7a295"
response_time:0.006585593 app_id:"e12a36d5-bbb4-45c1-8604-95a2897a7b11" app_index:"0" x_b3_traceid:"af551dff173514cf"
x_b3_spanid:"af551dff173514cf" x_b3_parentspanid:"-"/>

```

Reading Aggregated Logs

The logs retrieved by the Service Instance Logs cf CLI plugin aggregate logs from three backing applications: a Spring Cloud Data Flow server application, a metrics application, and a Spring Cloud Skipper application. The following excerpt shows logs after deploying a stream:

```

2018-02-09T11:12:02.45-0600 [RTR/dataflow 0] OUT dataflow-11f71dd3-f902-4e31-b631-fbbbfdb82459.apps.americancanyon.cf-app.com - [2018-02-09T17:12:02.429+0000] "GET /metrics/streams?names=httptest HTTP/1.1" 200 0 14765 "https://dataflow-11f71dd3-f902-4e31-b631-fbbbfdb82459.apps.americancanyon.cf-app.com/metrics/streams" "Apache-HttpClient/4.5.3 (Java/1.8.0_65)"

2018-02-09T11:12:02.45-0600 [RTR/metrics 0] OUT df-metrics-11f71dd3-f902-4e31-b631-fbbbfdb82459.apps.americancanyon.cf-app.com - [2018-02-09T17:12:02.444+0000] "GET /collector/metrics/streams HTTP/1.1" 200 0 22261 "-" "Apache-HttpClient/4.5.3 (Java/1.8.0_65)"

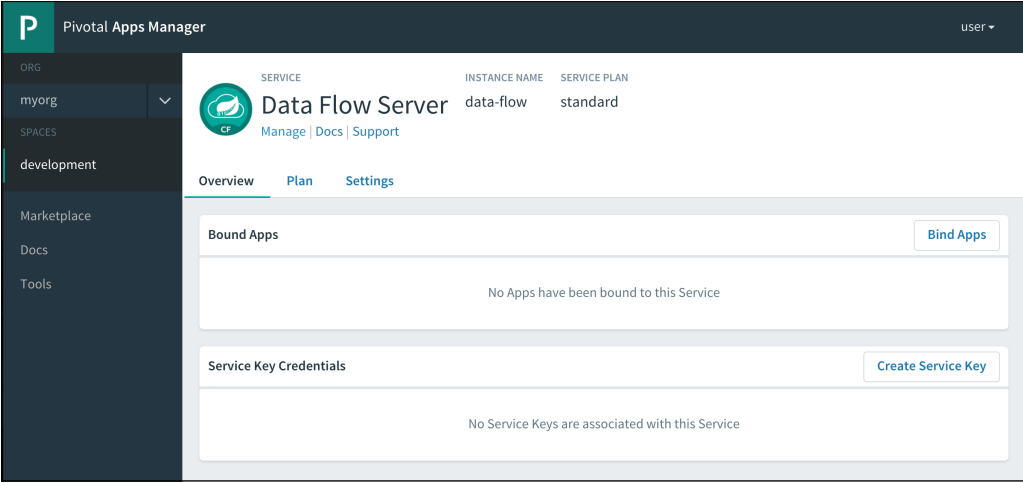
2018-02-09T11:12:02.60-0600 [APP/PROC/WEB/skipper 0] OUT 2018-02-09 17:12:02.603 INFO 15 --- [ry-client-nio-3] o.s.c.d.s.c.CloudFoundryAppDeployer : Successfully computed status [deployed] for httptest-http-v1
2018-02-09T11:12:02.94-0600 [APP/PROC/WEB/skipper 0] OUT 2018-02-09 17:12:02.940 INFO 15 --- [ry-client-nio-3] o.s.c.d.s.c.CloudFoundryAppDeployer : Successfully computed status [deployed] for astream-log-v4
2018-02-09T11:12:03.06-0600 [APP/PROC/WEB/skipper 0] OUT 2018-02-09 17:12:03.068 INFO 15 --- [ry-client-nio-1] o.s.c.d.s.c.CloudFoundryAppDeployer : Successfully computed status [deployed] for astream-time-v4

```

The Data Flow server application's logs are identified as belonging to the `dataflow` application. The metrics application similarly is called `metrics`. The Spring Cloud Skipper application is called `skipper`.

Using the Dashboard

To find the dashboard, navigate in Pivotal Cloud Foundry® Apps Manager to the Data Flow service instance’s space, click the listing for the service instance, and then click **Manage**.



If you are using version 6.8.0 or later of the Cloud Foundry Command Line Interface tool (cf CLI), you can also use `cf service SERVICE_NAME`, where `SERVICE_NAME` is the name of the Data Flow service instance:

```
$ cf service data-flow

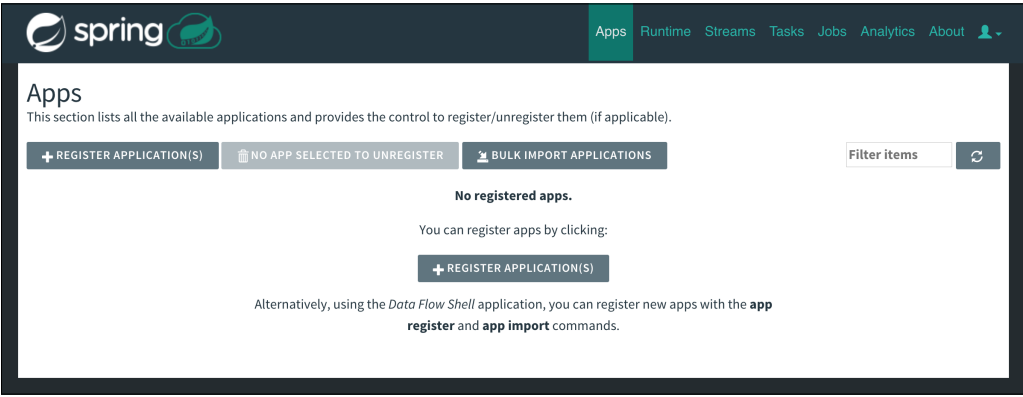
Service instance: data-flow
Service: p-dataflow
Bound apps:
Tags:
Plan: standard
Description: Deploys Spring Cloud Data Flow servers to orchestrate data pipelines
Documentation url: http://cloud.spring.io/spring-cloud-dataflow/
Dashboard: https://p-dataflow.apps.wise.com/instances/f09e5c77-e526-4f49-86d6-721c6b8e2fd9/dashboard

Last Operation
Status: create succeeded
Message: Created
Started: 2017-07-20T18:24:14Z
Updated: 2017-07-20T18:26:17Z
```

Visit the URL given for “Dashboard”.

Dashboard Information

The dashboard provides an overview of registered applications, stream and task definitions, and batch jobs. It also provides controls for deploying streams, launching tasks, and restarting batch job executions.



For complete information about the dashboard and its provided functionality, see the [OSS Spring Cloud Data Flow project's documentation](#).

Using the Shell

The open-source [Spring Cloud Data Flow](#) project provides a [shell](#), which can be used to interact locally with a Data Flow service instance deployed on Pivotal Cloud Foundry (PCF).

You can use the shell with Spring Cloud Data Flow for PCF service instances in either of two ways:

- The [Spring Cloud Data Flow for PCF Cloud Foundry CLI plugin](#) (recommended)
- The open-source shell binary, with manually-configured command-line options

The Spring Cloud Data Flow for PCF Cloud Foundry CLI plugin was created to ease the use of the Spring Cloud Data Flow shell with Spring Cloud Data Flow service instances on PCF.

 Note: To run the Spring Cloud Data Flow shell, you must have a Java Runtime Environment (JRE) installed. You can download the JRE from the [Java website](#) .

Note: Before connecting to a Data Flow service instance with the Spring Cloud Data Flow shell, be sure to log in to the PCF deployment using the Cloud Foundry Command Line Interface tool (cf CLI) and target the org and space of the Data Flow service instance. The Data Flow shell uses the cf CLI to authenticate to PCF.

Using the Cloud Foundry CLI Plugin

After installing the plugin, you can use it to attach the Data Flow shell to a Spring Cloud Data Flow for PCF service instance. Given an existing service instance named `data-flow`, the following command will download the appropriate shell version and attach it to the service instance:

[illegible]

For more information about the Spring Cloud Data Flow shell, see its [documentation](#).

Using the Shell Manually

If you would like to download and configure the Data Flow shell manually, [download the shell JAR file from the Spring Releases Maven repository](#).
Spring Cloud Data Flow for PCF can be used with version 1.3.0.RELEASE or later of the Spring Cloud Data Flow shell.

To target a Data Flow service instance's server with the shell, you must obtain the server's URL. Run `cf service SERVICE_NAME`, where `SERVICE_NAME` is the name of the service instance:

```
$ cf service data-flow

Service instance: data-flow
Service: p-dataflow
Bound apps:
Tags:
Plan: standard
Description: Deploys Spring Cloud Data Flow servers to orchestrate data pipelines
Documentation url: http://cloud.spring.io/spring-cloud-dataflow/
Dashboard: https://p-dataflow.apps.wise.com/instances/f09e5c77-e526-4f49-86d6-721c6b8e2fd9/dashboard
...
```

Visit the URL given for “Dashboard” and authenticate using your PCF credentials. When redirected to the service instance’s dashboard, copy the dashboard’s domain name (the URL minus the path following the domain).

For example, given the following URL:

<https://dataflow-f09e5c77-e526-4f49-86d6-721c6b8e2fd9.apps.wise.com/dashboard/index.html>

Copy the following:

<https://dataflow-f09e5c77-e526-4f49-86d6-721c6b8e2fd9.apps.wise.com>

From the directory containing the shell JAR file, run the shell from the command line using a command as shown below.

```
$ java -jar JAR_NAME SKIP_VALIDATION --dataflow.uri=SERVER_URL --dataflow.credentials-provider-command="cf oauth-token" --dataflow.mode=skipper
```

In this command, replace the following placeholders as shown below.

- `JAR_NAME` with the name of the downloaded Spring Cloud Data Flow shell JAR file
- `SKIP_VALIDATION` with the flag and value `--dataflow.skip-ssl-validation=true` if your PCF installation is using self-signed SSL certificates; otherwise remove
- `SERVER_URL` with the Data Flow service instance's server URL (the domain copied above)

The complete command may look something like the following (newlines added for readability):

```
$ java -jar spring-cloud-dataflow-shell-1.3.0.RELEASE.jar
--dataflow.skip-ssl-validation=true
--dataflow.uri=https://dataflow-f09e5c77-e526-4f49-86d6-721c6b8e2fd9.apps.wise.com
--dataflow.credentials-provider-command="cf oauth-token"
--dataflow.mode=skipper
```

When you run the `java -jar` command, the shell will initialize and give a `dataflow:>` prompt.

```
Welcome to the Spring Cloud Data Flow shell. For assistance hit TAB or type "help".
dataflow:>
```

For a list of available shell commands, run `help` from within the shell. For information about the flags which can be provided to the shell on startup, see the [OSS Spring Cloud Data Flow project's documentation](#).