Spring Cloud Data Flow Server for Cloud Foundry

1.0.0.M2

Table of Contents

Part I. Spring Cloud Data Flow for Cloud Foundry

This project provides support for deploying Spring Cloud Stream and Spring Cloud Task apps to Cloud Foundry.

1. Deploying on Cloud Foundry

Spring Cloud Data Flow can be used to deploy modules in a Cloud Foundry environment. When doing so, the server application can either run itself on Cloud Foundry, or on another installation (e.g. a simple laptop).

The required configuration amounts to the same in either case, and is merely related to providing credentials to the Cloud Foundry instance so that the server can spawn applications itself. Any Spring Boot compatible configuration mechanism can be used (passing program arguments, editing configuration files before building the application, using Spring Cloud Config, using environment variables, etc.), although some may prove more practicable than others when running *on* Cloud Foundry.

1.1 Provision a redis service instance on Cloud Foundry.

Use `cf marketplace` to discover which plans are available to you, depending on the details of your Cloud Foundry setup. For example when using Pivotal Web Services:

```
cf create-service rediscloud 30mb redis
```

1.2 Download the Spring Cloud Data Flow Server and Shell apps:

```
wget http://repo.spring.io/milestone/org/springframework/cloud/spring-cloud-dataflow-server-
cloudfoundry/1.0.0.M2/spring-cloud-dataflow-server-cloudfoundry-1.0.0.M2.jar
wget http://repo.spring.io/milestone/org/springframework/cloud/spring-cloud-dataflow-shell/1.0.0.M3/
spring-cloud-dataflow-shell-1.0.0.M3.jar
```

You can either deploy the server application on Cloud Foundry itself or on your local machine. The following two sections explain each way of running the server.

1.3 Deploying the Server app on Cloud Foundry

Push the server application on Cloud Foundry, configure it (see below) and start it.

> **Note**
>
> You must use a unique name for your app; an app with the same name in the same organization will cause your deployment to fail

```
cf push s-c-dataflow-server --no-start -p spring-cloud-dataflow-server-cloudfoundry-1.0.0.M2.jar
cf bind-service s-c-dataflow-server redis
```

> **Note**
>
> If you are pushing to a space with multiple users, for example on PWS, there may already be a route taken for the applicaiton name you have chosen. You can use the options `--random-route` to avoid this when pushing the app.

Now we can configure the app. The following configuration is for Pivotal Web Services. You need to fill in {org}, {space}, {email} and {password} before running these commands.

> **Note**
>
> Only set 'Skip SSL Validation' to true if you're running on a Cloud Foundry instance using self-signed certs (e.g. in development). Do not use for production.

```
cf set-env s-c-dataflow-server SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_API_ENDPOINT https://
api.run.pivotal.io
cf set-env s-c-dataflow-server SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_ORGANIZATION {org}
cf set-env s-c-dataflow-server SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_SPACE {space}
cf set-env s-c-dataflow-server SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_DOMAIN cfapps.io
cf set-env s-c-dataflow-server SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_SERVICES redis
cf set-env s-c-dataflow-server SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_USERNAME {email}
cf set-env s-c-dataflow-server SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_PASSWORD {password}
cf set-env s-c-dataflow-server SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_SKIP_SSL_VALIDATION false
```

We are now ready to start the app.

```
cf start s-c-dataflow-server
```

Alternatively, you can run the Admin application locally on your machine which is described in the next section.

1.4 Running the Server app locally

To run the server application locally, targeting your Cloud Foundry installation, you you need to configure the application either by passing in command line arguments (see below) or setting a number of environment variables.

To use environment variables set the following:

```
export SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_API_ENDPOINT=https://api.run.pivotal.io
export SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_ORGANIZATION={org}
export SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_SPACE={space}
export SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_DOMAIN=cfapps.io
export SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_SERVICES=redis
export SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_USERNAME={email}
export SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_PASSWORD={password}
export SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_SKIP_SSL_VALIDATION=false
```

You need to fill in {org}, {space}, {email} and {password} before running these commands.

> **Note**
>
> Only set 'Skip SSL Validation' to true if you're running on a Cloud Foundry instance using self-signed certs (e.g. in development). Do not use for production.

Now we are ready to start the server application:

```
java -jar spring-cloud-dataflow-server-cloudfoundry-1.0.0.M2.jar [--option1=value1] [--option2=value2]
 [etc.]
```

1.5 Running Spring Cloud Data Flow Shell locally

Run the shell and optionally target the Admin application if not running on the same host (will typically be the case if deployed on Cloud Foundry as explained here)

```
$ java -jar spring-cloud-dataflow-shell-1.0.0.M3.jar
```

```
server-unknown:>dataflow config server http://s-c-dataflow-server.cfapps.io
Successfully targeted http://s-c-dataflow-server.cfapps.io
dataflow:>
```

2. Configuration Reference

The following pieces of configuration must be provided, e.g. by setting variables in the apps environment, or passing variables on the Java invocation:

```
# Default values cited after the equal sign.
# Example values, typical for Pivotal Web Services, cited as a comment

# url of the CF API (used when using cf login -a for example), e.g. https://api.run.pivotal.io
# (for setting env var use SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_API_ENDPOINT)
spring.cloud.deployer.cloudfoundry.apiEndpoint=

# name of the organization that owns the space above, e.g. youruser-org
# (For Setting Env var use SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_ORGANIZATION)
spring.cloud.deployer.cloudfoundry.organization=

# name of the space into which modules will be deployed
# (for setting env var use SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_SPACE)
spring.cloud.deployer.cloudfoundry.space=<same space as server when running on CF, or 'development'>

# the root domain to use when mapping routes, e.g. cfapps.io
# (for setting env var use SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_DOMAIN)
spring.cloud.deployer.cloudfoundry.domain=

# Comma separated set of service instance names to bind to the module.
# Amongst other things, this should include a service that will be used
# for Spring Cloud Stream binding
# (for setting env var use SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_SERVICES)
spring.cloud.deployer.cloudfoundry.services=redis

# username and password of the user to use to create apps (modules)
# (for setting env var use SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_USERNAME and
 SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_PASSWORD)
spring.cloud.deployer.cloudfoundry.username=
spring.cloud.deployer.cloudfoundry.password=

# Whether to allow self-signed certificates during SSL validation
# (for setting env var use SPRING_CLOUD_DEPLOYER_CLOUDFOUNDRY_SKIP_SSL_VALIDATION)
spring.cloud.deployer.cloudfoundry.skipSslValidation=false
```

Part II. Appendices

3. Building

3.1 Basic Compile and Test

To build the source you will need to install JDK 1.8.

The build uses the Maven wrapper so you don't have to install a specific version of Maven. To enable the tests for Redis you should run the server before bulding. See below for more information on how run Redis.

The main build command is

```
$ ./mvnw clean install
```

You can also add '-DskipTests' if you like, to avoid running the tests.

> **Note**
>
> You can also install Maven (>=3.3.3) yourself and run the `mvn` command in place of `./mvnw` in the examples below. If you do that you also might need to add `-P spring` if your local Maven settings do not contain repository declarations for spring pre-release artifacts.

> **Note**
>
> Be aware that you might need to increase the amount of memory available to Maven by setting a `MAVEN_OPTS` environment variable with a value like `-Xmx512m -XX:MaxPermSize=128m`. We try to cover this in the `.mvn` configuration, so if you find you have to do it to make a build succeed, please raise a ticket to get the settings added to source control.

The projects that require middleware generally include a `docker-compose.yml`, so consider using Docker Compose to run the middeware servers in Docker containers. See the README in the scripts demo repository for specific instructions about the common cases of mongo, rabbit and redis.

3.2 Documentation

There is a "full" profile that will generate documentation. You can build just the documentation by executing

```
$ ./mvnw package -DskipTests=true -P full -pl spring-cloud-dataflow-server-cloudfoundry-docs -am
```

3.3 Working with the code

If you don't have an IDE preference we would recommend that you use Spring Tools Suite or Eclipse when working with the code. We use the m2eclipe eclipse plugin for maven support. Other IDEs and tools should also work without issue.

Importing into eclipse with m2eclipse

We recommend the m2eclipe eclipse plugin when working with eclipse. If you don't already have m2eclipse installed it is available from the "eclipse marketplace".

Unfortunately m2e does not yet support Maven 3.3, so once the projects are imported into Eclipse you will also need to tell m2eclipse to use the `.settings.xml` file for the projects. If you do not do this

you may see many different errors related to the POMs in the projects. Open your Eclipse preferences, expand the Maven preferences, and select User Settings. In the User Settings field click Browse and navigate to the Spring Cloud project you imported selecting the `.settings.xml` file in that project. Click Apply and then OK to save the preference changes.

> **Note**
>
> Alternatively you can copy the repository settings from `.settings.xml` into your own `~/.m2/settings.xml`.

Importing into eclipse without m2eclipse

If you prefer not to use m2eclipse you can generate eclipse project metadata using the following command:

```
$ ./mvnw eclipse:eclipse
```

The generated eclipse projects can be imported by selecting `import existing projects` from the `file` menu.