Installing by Using Docker Compose

Spring Cloud Data Flow provides a Docker Compose file to let you quickly bring up Spring Cloud Data Flow, Skipper, MariaDB, and Apache Kafka. The additional customization guides help to extend the basic configuration, showing how to switch the binder to RabbitMQ, use a different database, enable monitoring, and more.

Also, when doing development of custom applications, you need to enable the Docker containers that run the Data Flow and the Skipper servers to see your local file system. The Accessing the Host File System chapter shows how to do that.

You should upgrade to the latest `docker` and `docker-compose` versions. This guide is tested with Docker Engine: `20.10.21` and docker-compose: `v2.12.2`.

Configure your Docker daemon with at least `8 GB` of memory. On Windows or Mac, you can use the Docker Desktop's `Preferences/Resource/Advanced` menu to set the amount of memory.

Downloading the Docker Compose Files

You will need to download `docker-compose.yml`, `docker-compose-<broker>.yml`, `docker-compose-<database>.yml` where `<broker>` is one of rabbitmq or kafka and `<database>` is one of postgres, mariadb or mysql.

Linux / OSX   Windows (Cmd)

```
wget -O docker-compose.yml https://raw.githubusercontent.com/spring-cloud/spri
wget -O docker-compose-<broker>.yml https://raw.githubusercontent.com/spring-c
wget -O docker-compose-<database>.yml https://raw.githubusercontent.com/spring
```

The Docker Compose Customization guides provide additional files that you can combine with the basic docker-compose.yml to extend or alter its configuration.

Starting Docker Compose

From within the directory where `docker-compose.yml` and other files are downloaded, run:

Linux / OSX   Windows (Cmd)   Windows (PowerShell)

```
export DATAFLOW_VERSION=2.10.1
export SKIPPER_VERSION=2.9.1
docker-compose -f docker-compose.yml -f docker-compose-<broker>.yml -f docker-
```

By default, Docker Compose uses locally available images. Run `docker-compose pull` prior to `docker-compose up` to ensure the latest image versions are downloaded.

Once the emitting of log messages on the command prompt stops, open the Spring Cloud Data Flow Dashboard at http://localhost:9393/dashboard or use the Shell as explained later.

You can use the following environment variables to configure the `docker-compose.yml`:

Variable name	Default value	Description
`DATAFLOW_VERSION`	`2.10.1`	Data Flow Server version to install. Example: `2.4.0.RELEASE` or `2.10.1` for the latest version.
`SKIPPER_VERSION`	`2.9.1`	Skipper Server version to install. Example: `2.3.0.RELEASE` or

Variable name	Default value	Description
		`2.9.1` for the latest Skipper version.
`STREAM_APPS_URI`	https://dataflow.spring.io/kafka-maven-latest (or https://dataflow.spring.io/kafka-docker-latest for DooD)	Pre-registered Stream applications. Find here the available Stream Application Starters links.
`TASK_APPS_URI`	https://dataflow.spring.io/task-maven-latest (or https://dataflow.spring.io/task-docker-latest for DooD)	Pre-registered Task applications. You can find the available Task Application Starters links here.
`HOST_MOUNT_PATH`	.	Defines the host machine folder path on the mount. See Accessing the Host File System for further details.
`DOCKER_MOUNT_PATH`	`/home/cnb/scdf`	Defines the target (in-container) path on which to mount the host folder. See Accessing the Host File System for further details.

The docker-compose.yml configurations expose the following container ports to the host machine:

Host ports	Container ports	Description
9393	9393	The port on which the Data Flow server listens. You can use it to reach the Dashboard at http://localhost:9393/dashboard or the REST API at http://localhost:9393

Host ports	Container ports	Description
7577	7577	The port that the Skipper server listens on. You can use it to reach the Skipper REST API at http://localhost:7577/api
20000-20105	20000-20105	Skipper and Local Deployer are configured to use this port range for all deployed stream applications. That means you can reach the application's actuator endpoints from your host machine. You can use the `server.port` deployment property to override those ports.

You can use the exposed application ports (`20000-20105`) in your stream applications to expose certain ports to the host machine. For example, the `http --server.port=20015 | log` stream definition would let you use `curl` and POST HTTP messages to the `http` source directly from your host machine on the `20015` port.

Stopping Spring Cloud Data Flow

1. Press `Ctrl+C` to shut down the docker-compose process.
2. Run the following command to clean the used Docker containers:

```
docker-compose down
```

If errors occur due to old or hanging containers, clean all containers:

Linux / OSX / Windows (PowerShell)   Windows (Cmd)

```
docker stop $(docker ps -a -q)
docker rm $(docker ps -a -q)
```

Using the Shell

For convenience and as an alternative to the Spring Cloud Data Flow Dashboard, you can use the Spring Cloud Data Flow Shell. The shell supports tab completion for commands and application configuration properties.

To download the Spring Cloud Data Flow Shell application, run the following command:

wget   curl

```
wget -O spring-cloud-dataflow-shell-2.10.1.jar https://repo.maven.apache.org/m
```

Launch the shell:

```
java -jar spring-cloud-dataflow-shell-2.10.1.jar
```

Accessing the Host File System

If you develop custom applications on your local machine, you need to register them with Spring Cloud Data Flow. Since Data Flow server runs inside a Docker container, you need to configure this container to access to your local file system to resolve the applications registration references. To deploy those custom applications, the Skipper Server also needs to access them from within its own Docker container.

By default, `docker-compose.yml` mounts the local host folder (the folder where the docker-compose process is started) to a `/home/cnb/scdf` folder inside both the `dataflow-server` and the `skipper` containers.

> It is vital that the Data Flow and the Skipper containers use **exactly the same** mount points. This allows application registration references in Data Flow to be resolved and deployed in Skipper by using the same references.

The `HOST_MOUNT_PATH` and `DOCKER_MOUNT_PATH` environment variables (see the configuration table) let you customize the default host and container paths.

For example, if the `my-app-1.0.0.RELEASE.jar` is stored in the `/tmp/myapps/` folder on the host machine (`C:\Users\User\MyApps` on Windows), you can make it accessible to the `dataflow-server` and `skipper` containers by setting the `HOST_MOUNT_PATH`:

Linux / OSX   Windows (Cmd)   Windows (PowerShell)

```
export HOST_MOUNT_PATH=/tmp/myapps
```

Then follow the starting docker-compose instructions to start the cluster.

See the compose-file reference for further configuration details.

Once the host folder is mounted, you can register the app starters (from `/home/cnb/scdf`) by using either the Data Flow Shell or the Dashboard. To do so, use the `file://` URI schema. The following example shows how to do so:

```
app register --type source --name my-app --uri file://home/cnb/scdf/my-app-1.0.0.R
```

You can use the optional `--metadata-uri` parameter if a metadata jar is available in the `/home/cnb/scdf` folder for the same application.

You can also pre-register the apps directly, by modifying the `app-import-stream` and `app-import-task` configurations in the `docker-compose.yml` file. For every pre-registered app starer, add an additional `wget` statement to the `app-import-stream` block configuration, as the following example shows:

```
le:/home/cnb/apps/my-app.jar&metadata-uri=file:/home/cnb/apps/my-app-metadata.jar"
```

See the Data Flow REST API for further details.

Maven Local Repository Mounting

You can develop applications and install them in the local Maven repository (using `mvn install`) while the Data Flow server is running and have immediate access to the newly built applications.

To do so, you must mount the host’s local maven repository to the `dataflow-server` and `skipper` containers using a volume called `/home/cnb/.m2/`. The Maven Local repository location defaults to `~/.m2` for Linux and OSX and to `C:\Users\{your-username}\.m2` for Windows.

We can leverage the `HOST_MOUNT_PATH` and `DOCKER_MOUNT_PATH` variables to configure mount volumes, as follows:

Linux / OSX   Windows (Cmd)   Windows (PowerShell)

```
export HOST_MOUNT_PATH=~/.m2
export DOCKER_MOUNT_PATH=/root/.m2/
```

Then follow the starting docker-compose instructions to start the cluster.

Now you can use the `maven://` URI schema and Maven coordinates to resolve jars installed in the host’s maven repository, as follows:

```
app register --type processor --name pose-estimation --uri maven://org.springframe
```

This approach lets you use applications that are built and installed on the host machine (for example, by using `mvn clean install`) directly with the Spring Cloud Data Flow server.

Monitoring

The basic Data Flow docker-compose configuration does not enable the monitoring functionality for Stream and Task applications. Follow the Monitoring with Prometheus and Grafana or Monitoring with InfluxDB and Grafana customization guides to learn how to enable and configure the monitoring for Spring Cloud Data Flow.

To learn more about the monitoring experience in Spring Cloud Data Flow with Prometheus and InfluxDB, see the Stream Monitoring feature guide.

Running Java 17 Applications

Currently Spring Cloud Data Flow defaults to Java 8 when running applications. If you w
to run Java 17 applications set the `BP_JVM_VERSION` to `-jdk17` as shown below:

```
export BP_JVM_VERSION=-jdk17
```

Debugging

The Debug Stream Applications guide shows how to enable remote debugging for Stream Applications deployed by Data Flow.

The Debug Data Flow Server guide shows how to extend the docker-compose configuration to enable remote Data Flow Server debugging with your IDE (such as IntelliJ or Eclipse).

The Debug Skipper Server guide shows how to extend the docker-compose configuration to enable remote Skipper Server debugging with your IDE (such as IntelliJ or Eclipse).

Docker Stream & Task applications

Basic docker-compose installation supports only uber-jar Stream and Task applications. As the Docker specification does not support container nesting, the Data Flow and Skipper servers are not able to run Docker applications from within their own Docker containers.

The docker-compose-dood.yml extension uses the `Docker-out-of-Docker (DooD)` approach to let Skipper and Data Flow deploy Stream and Task Docker apps.

In this approach, containers created from within the Data Flow and the Skipper containers are sibling containers (spawned by the Docker daemon in the Host). There is no Docker daemon inside the server's containers and, thus, no container nesting.

The `docker-compose-dood.yml` extends `docker-compose.yml` by installing the Docker CLI to the Data Flow and Skipper servers containers and mounting the server's Docker sockets to the Host's socket:

Linux / OSX   Windows

```
export COMPOSE_PROJECT_NAME=scdf
docker-compose -f ./docker-compose.yml -f ./docker-compose-dood.yml up
```

- `COMPOSE_PROJECT_NAME` sets the docker-compose project name. It is later used for naming the network passed to the apps containers.

- You can use `STREAM_APPS_URI` and `TASK_APPS_URI` to register Docker-based Stream and Task apps.

If docker-compose exits before the data pipelines are stopped, you should manually clean the containers:
`docker stop $(docker ps -a -q); docker rm $(docker ps -a -q)`

Set the `DOCKER_DELETE_CONTAINER_ON_EXIT` environment variable to `false` to retain the stopped docker containers so that you can check their logs:
`docker logs <container id>`

Edit this page on GitHub

Documentation
Local Machine

Documentation
Docker Compose Customization