Spring Cloud Data Flow Task Processing Lab

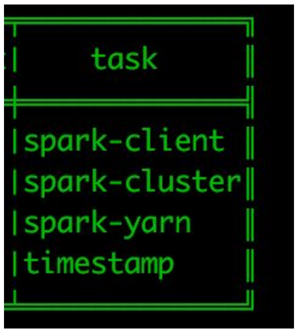
Setting up the environment

1. Install RabbitMQ using the instructions in the InstallRabbitMQ.pdf file**.**
2. Now let's setup the lab environment:
    a. Using git
        i. From a your local terminal or command prompt change directory to a clean working directory.
        ii. Now execute: `git clone https://github.com/cppwfs/DNDataflow.git`
        iii. Now `cd DNDataflow`
    b. Using Thumbdrive
        i. Copy the **DNDataflow** directory from the thumbdrive to a location on your laptop hard drive
        ii. Now from a terminal or command prompt **cd** to the **DNDataflow** directory you just created on your hard drive.

Creating your first task

- If you haven't already completed all the installation steps from \`lab3\`, please proceed there to set up Spring Cloud Data Flow "Server" and "Shell" applications

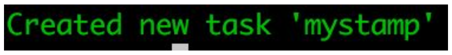
1. First let's register a basic suite of tasks by importing their registrations using the Spring Cloud Data Flow Shell with the following command command:
    a. `app import --uri http://bit.ly/1-0-1-GA-task-applications-maven`
2. Now from the shell we can check to see if those task apps have been registered
    a. `app list`
    b. The following list should appear:



```
╔═════════════╗
|    task     ║
╠═════════════╣
|spark-client ║
|spark-cluster║
|spark-yarn   ║
|timestamp    ║
╚═════════════╝
```

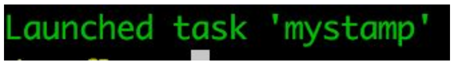
   c. In particular we will be using the timestamp app. This app just prints a timestamp to the console then terminates.

3. Now let's create a task definition using the timestamp task. This will be done using the task create command:
   a. `task create --name mystamp --definition "timestamp"`
   b. The following message should then be displayed:



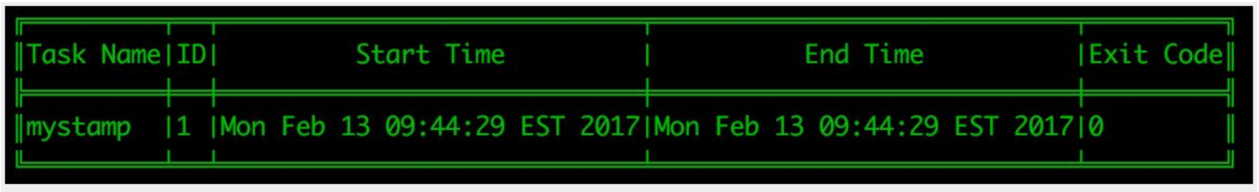
```
Created new task 'mystamp'
```

4. Unlike a stream command we do not deploy a Long Running Process but rather a short lived process that will perform its "task" and then terminates. To do this we want to launch this task and this is done using the task launch command:
   a. `task launch mystamp`
   b. The following message should be displayed:



```
Launched task 'mystamp'
```

5. To verify that the task was executed successfully we can use our Spring Cloud Data Flow Shell to execute the following:
   a. `task execution list`



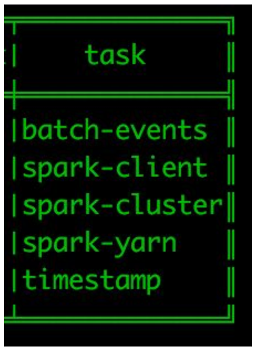
```
╔═════════╤══╤════════════════════════════╤════════════════════════════╤═════════╗
║Task Name│ID│         Start Time         │          End Time          │Exit Code║
╠═════════╪══╪════════════════════════════╪════════════════════════════╪═════════╣
║mystamp  │1 │Mon Feb 13 09:44:29 EST 2017│Mon Feb 13 09:44:29 EST 2017│0        ║
╚═════════╧══╧════════════════════════════╧════════════════════════════╧═════════╝
```

   b. We see that mystamp returned an exit code of 0 which means that it successfully ran.

Creating your first batch-task

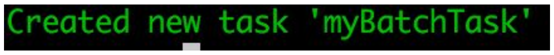
1. First let's register a Spring Batch-Task Application (basically a batch app that uses @EnableTask) using the Spring Cloud Data Flow Shell with the following command command:
   a. `app register --name batch-events --type task --uri file:///<FOLDER>/DNDataflow/labs/jars/batch-events.jar`
2. Now from the shell we can check to see if those task apps have been registered

    a. `app list`
    b. The following list should appear:



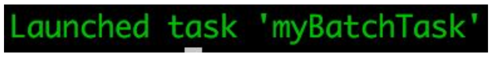
```
|    task      ║
|batch-events  ║
|spark-client  ║
|spark-cluster ║
|spark-yarn    ║
|timestamp     ║
```

3. Now let's create a task definition using the batch-events task. This will be done using the task create command:
    a. `task create --name myBatchTask --definition "batch-events"`
    b. The following message should then be displayed:



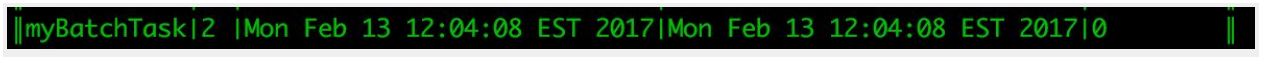
```
Created new task 'myBatchTask'
```

4. Now that its registered let's launch this task using the task launch command:
    a. `task launch myBatchTask`
    b. The following message should be displayed:



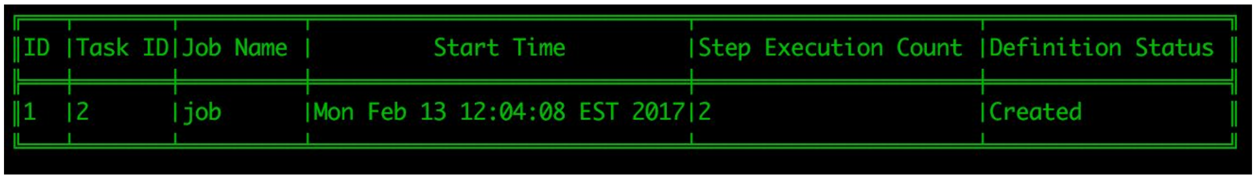
```
Launched task 'myBatchTask'
```

5. To verify that the task was executed successfully we can use our Spring Cloud Data Flow Shell to execute the following:
    a. `task execution list`



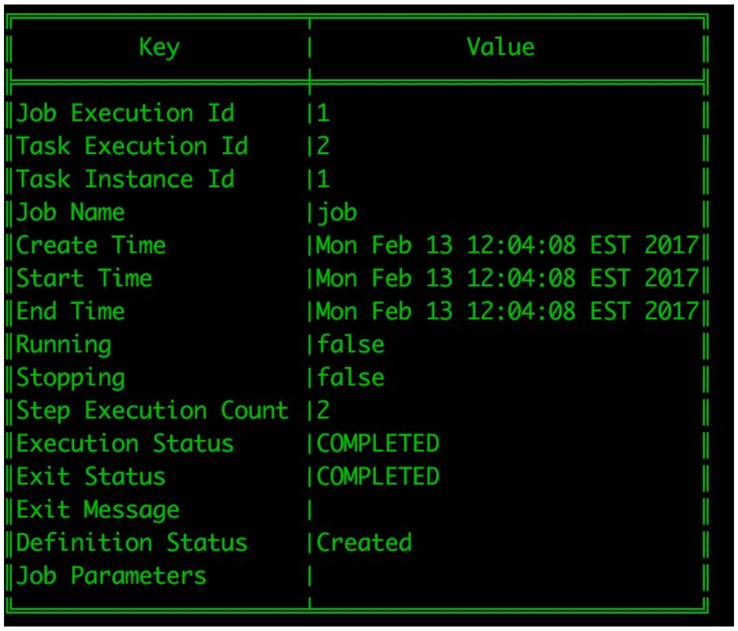
```
║myBatchTask|2 |Mon Feb 13 12:04:08 EST 2017|Mon Feb 13 12:04:08 EST 2017|0          ║
```

    b. We see that myTaskBatch returned an exit code of 0 which means that it successfully ran.
6. But since it was a Spring Batch Job we can verify that the batch job ran successfully using the Spring Cloud Data Flow Shell
    a. First we want to get the Job Execution ID. This is done by executing the job execution list as shown below:
        i. `job execution list`
    b. You should see something like the following as a result:



```
║ID |Task ID|Job Name |         Start Time          |Step Execution Count |Definition Status ║
║1  |2      |job      |Mon Feb 13 12:04:08 EST 2017|2                    |Created           ║
```

    c. Now we can get the job execution id from the column "ID" in this case it's 1.
    d. Now to get the full status of the job execution we can execute a job display command to get the details of the job execution as shown below:
        i. `job execution display --id 1`
    e. You should see something like the following as a result:



```
╔══════════════════════╤══════════════════════════════╗
║         Key          │            Value             ║
╠══════════════════════╪══════════════════════════════╣
║Job Execution Id      │1                             ║
║Task Execution Id     │2                             ║
║Task Instance Id      │1                             ║
║Job Name              │job                           ║
║Create Time           │Mon Feb 13 12:04:08 EST 2017║
║Start Time            │Mon Feb 13 12:04:08 EST 2017║
║End Time              │Mon Feb 13 12:04:08 EST 2017║
║Running               │false                         ║
║Stopping              │false                         ║
║Step Execution Count  │2                             ║
║Execution Status      │COMPLETED                     ║
║Exit Status           │COMPLETED                     ║
║Exit Message          │                              ║
║Definition Status     │Created                       ║
║Job Parameters        │                              ║
╚══════════════════════╧══════════════════════════════╝
```

f. As we see above that the Execution Status is Complete. This means that the Job succeeded.

You can also review the results using the UI by going to http://localhost:9393/dashboard