

1. Scheduler is Source to push the TaskLaunchRequest Payload which is pushed to AQMP

**Spring.cloud.stream.bindings.output.destination** defines the exchange which

Produces the payload.

1. **EnableTaskLauncher** TaskLauncherSink is the listener to AQMP, Launches a task upon the receipt of a valid TaskLaunchRequest.

**Spring.cloud.stream.bindings.input.destination** defines the intake queue.

**spring.cloud.stream.bindings.input.group**

**Spring.cloud.stream.bindings.input.consumer.concurrency,** defines the number of

Concurrent consumers of TaskLauncherSink.

A `task-launcher` sink expects a message containing a [TaskLaunchRequest] object in

its payload. From the `TaskLaunchRequest` object, the `task-launcher` obtains the URI

of the artifact to be launched, as well as the environment properties, command line

arguments, deployment properties, and application name to be used by the task.

1. @EnableTask to define the short lived individual microservice. Spring Cloud Task will contain the list of jobs which need to be performed by the task which are managed via JobFactory.

A task executes a process on demand. In the case of Spring Cloud Task, a task is a

application that is annotated with `@EnableTask`.

A user launches a task that performs a certain process, and, once complete, the task

ends. Unlike a stream where a stream definition can have at most one deployment a

single task definition can be launched multiple times simultaneously.

**The Lifecycle of a Task:**

. <<spring-cloud-dataflow-create-task-apps>>

. <<spring-cloud-dataflow-register-task-apps>>

. <<spring-cloud-dataflow-create-task-definition>>

. <<spring-cloud-dataflow-task-launch>>

. <<spring-cloud-dataflow-task-review-executions>>

. <<spring-cloud-dataflow-task-definition-destroying>>

**Task Database Configuration:**

CAUTION: When launching a task application, be sure that the database driver that is

being used by Spring Cloud Data Flow is also a dependency on the task application.

**Simple Task Launch**

When launching a task from Spring Cloud Data Flow, Data Flow adds its datasource

properties (`spring.datasource.url`, `spring.datasource.driverClassName`,

`spring.datasource.username`, `spring.datasource.password`)

to the app properties of the task being launched. Thus a task application

will record its task execution information to the Spring Cloud Data Flow repository.

**Reviewing Task Executions**

Once the task is launched, the state of the task is stored in a relational DB. The state

includes:

\* Task Name

\* Start Time

\* End Time

\* Exit Code

\* Exit Message

\* Last Updated Time

\* Parameters

**Subscribing to Task/Batch Events**

You can also tap into various task and batch events when the task is launched.

If the task is enabled to generate task or batch events (with the additional dependencies

`spring-cloud-task-stream` and, in the case of Kafka as the binder,

`spring-cloud-stream-binder-kafka`), those events are published during the task lifecycle.

By default, the destination names for those published events on the broker (Rabbit

Kafka, and others) are the event names themselves (for instance: `task-events`,

`job-execution-events`, and so on).

**Exit Statuses**

The following list shows how the Exit Status is set for each step (task) contained in the

composed task following each step execution:

\* If the `TaskExecution` has an `ExitMessage`, that is used as the `ExitStatus`.

\* If no `ExitMessage` is present and the `ExitCode` is set to zero, then the `ExitStatus`

for the step is `COMPLETED`.

\* If no `ExitMessage` is present and the `ExitCode` is set to any non-zero number, the

`ExitStatus` for the step is `FAILED`.

A user launches a task that performs a certain process, and, once complete, the task ends. Unlike a stream where a stream definition can have at most one deployment a single task definition can be launched multiple times simultaneously.

1. Spring cloud dataflow server UI display the Task and Jobs which are executed.

Download spring-cloud-dataflow-server-local-1.3.1.RELEASE.jar or any latest

Version of jar.

Run the command, to display the Spring cloud dataflow server UI:

**java -jar spring-cloud-dataflow-server-local-1.3.1.RELEASE.jar**

**--spring.datasource.url="jdbc:mysql://localhost/springcloud" --spring.datasource.username="root" --spring.datasource.password="root" --spring.datasource.driver-class-name="org.mariadb.jdbc.Driver"**

1. spring.cloud.deployer.memory for specifying "container" memory in megabytes - 1024 or 1024m are both acceptable formats
2. spring.cloud.deployer.cpu for specifying number of cpu cores.
3. **TaskConfigurer** is used to customize the way components of Spring Cloud Task are configured. By default, the DefaultTaskConfigurer provides logical defaults (Map based in memory components useful for development if no DataSource is provided and JDBC based components if there is a DataSource available.

Create database **springTask** which will stores the information regarding the Spring

Cloud task.

Create another database **test** which will stores the information regarding the application.

**Concerns:**

1. .
2. .
3. .