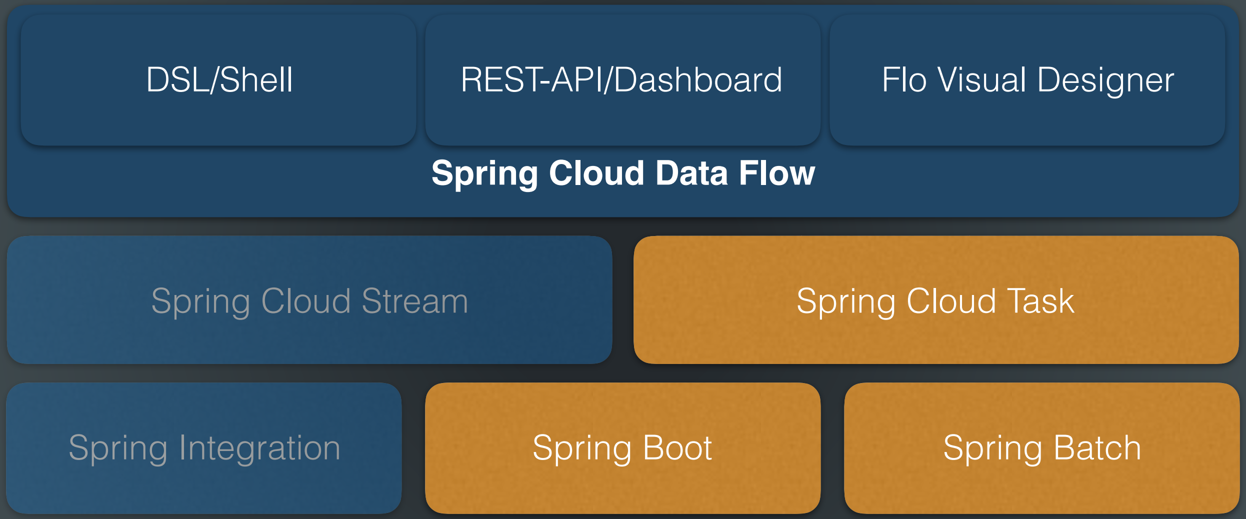
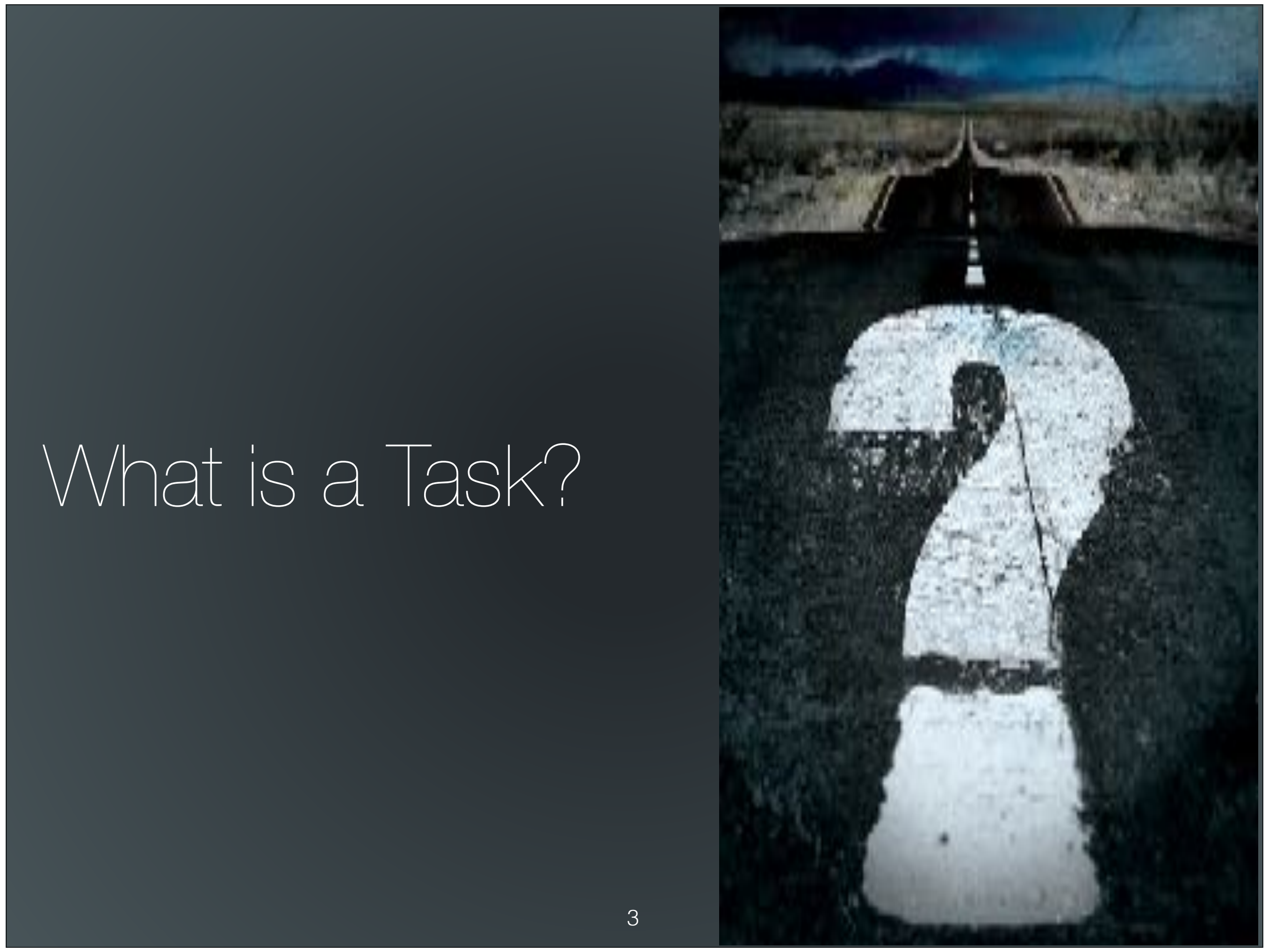
Tasks

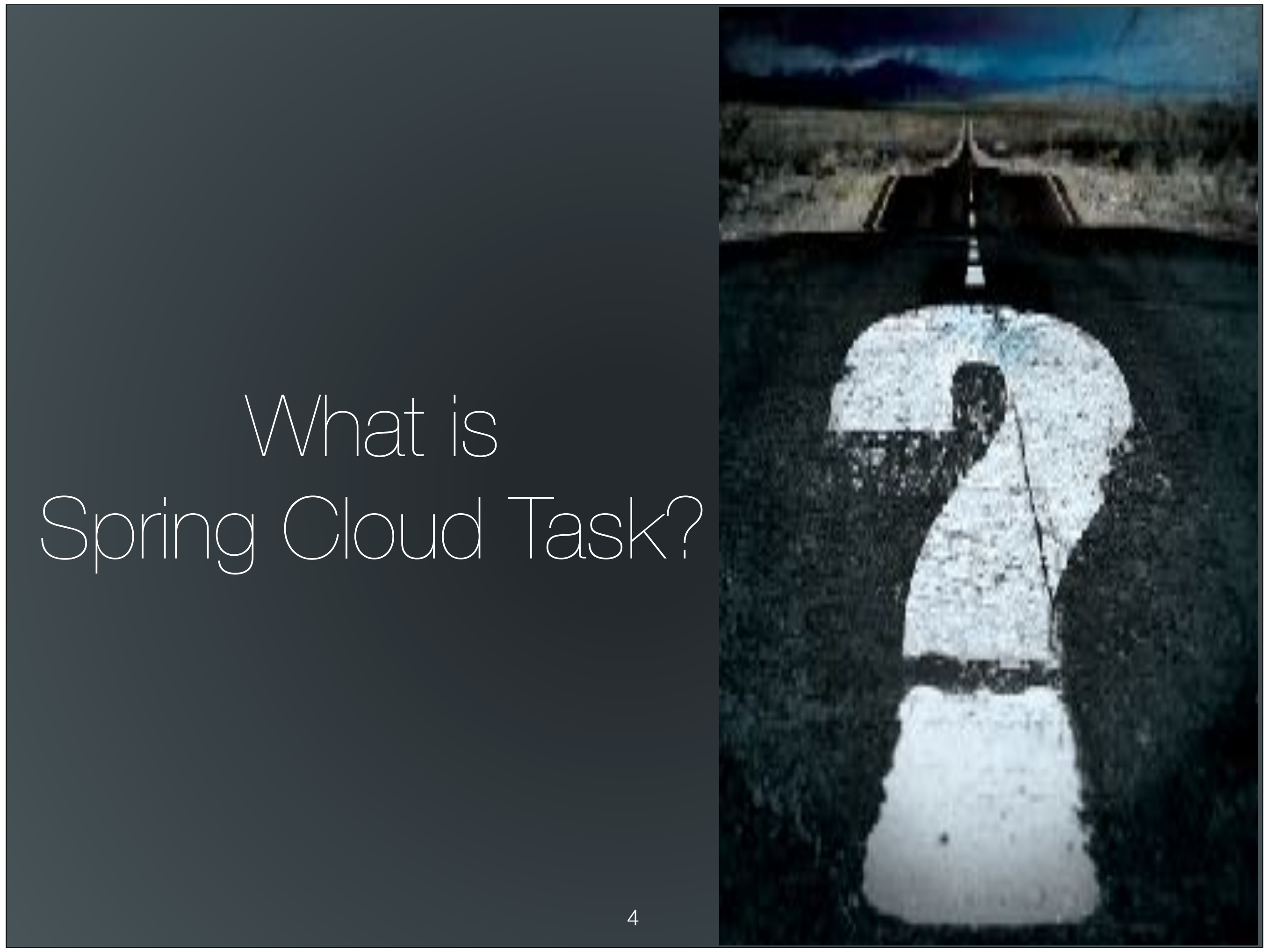

DSL/Shell
REST-API/Dashboard
Flo Visual Designer
Spring Cloud Data Flow
Spring Cloud Stream
Spring Cloud Task
Spring Integration
Spring Boot
Spring Batch



What is a Task?

- In SCSt what we see are LRP or long running processes
    - If they fail what they get restarted
    - And they run till they are shut down
- Tasks are ephemeral
    - Once complete they stop.
    - The framework does not restart them.
- Good when you need perform a certain amount of work then terminate
    - Reduced cost - Only charged for the run time they are up
    - Reduced need for resources.

https://flic.kr/p/8HDJ5B

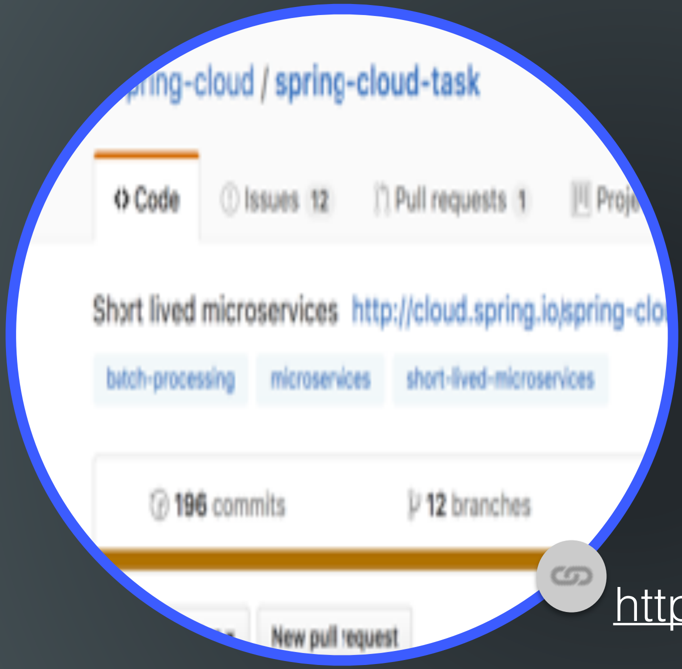


What is Spring Cloud Task?

- Spring Cloud Task is a Ephemeral microservice framework
    - Built on Spring stack:
    - Spring Boot: full-stack standalone apps, configuration
- Provides features required for a task in a cloud environment
- Common abstractions
    - Records the task state information in a datastore
    - Emits Task Event at start and stop of task
        - Job events as well
    - Provides listeners for before, failed, and after events for a task

https://flic.kr/p/8HDJ5B

CODE IS ON GITHUB



Short lived microservices http://cloud.spring.io/spring-clo

https://github.com/spring-cloud/spring-cloud-task

Reference Docs

http://docs.spring.io/spring-cloud-task/docs/1.1.2.RELEASE/reference/htmlsingle/

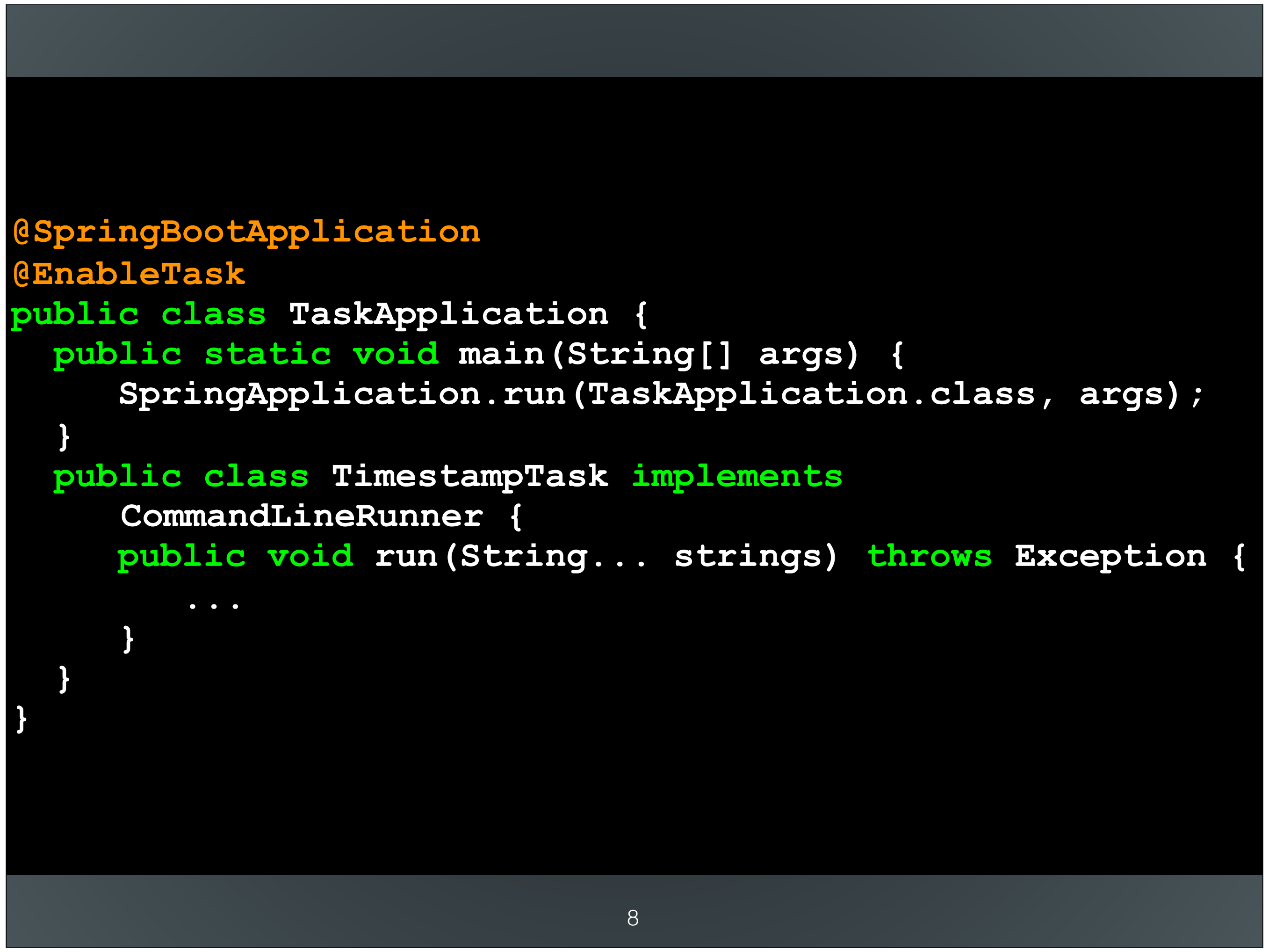
https://flic.kr/p/8MVYfc

Persists Task Execution

Persist Task Execution

- Persists the TaskExecution event to RDBMS data store or in memory at
    - Task Start
    - Task End
- To Enable:
    - spring-cloud-starter-task
    - Add boot-starter-jdbc dependency
    - Add database dependency
- Creates TaskExecution on start
- Updates TaskExecution entry upon completion
    - End Time
    - Exit Code
    - Exit Message
    - Error Message

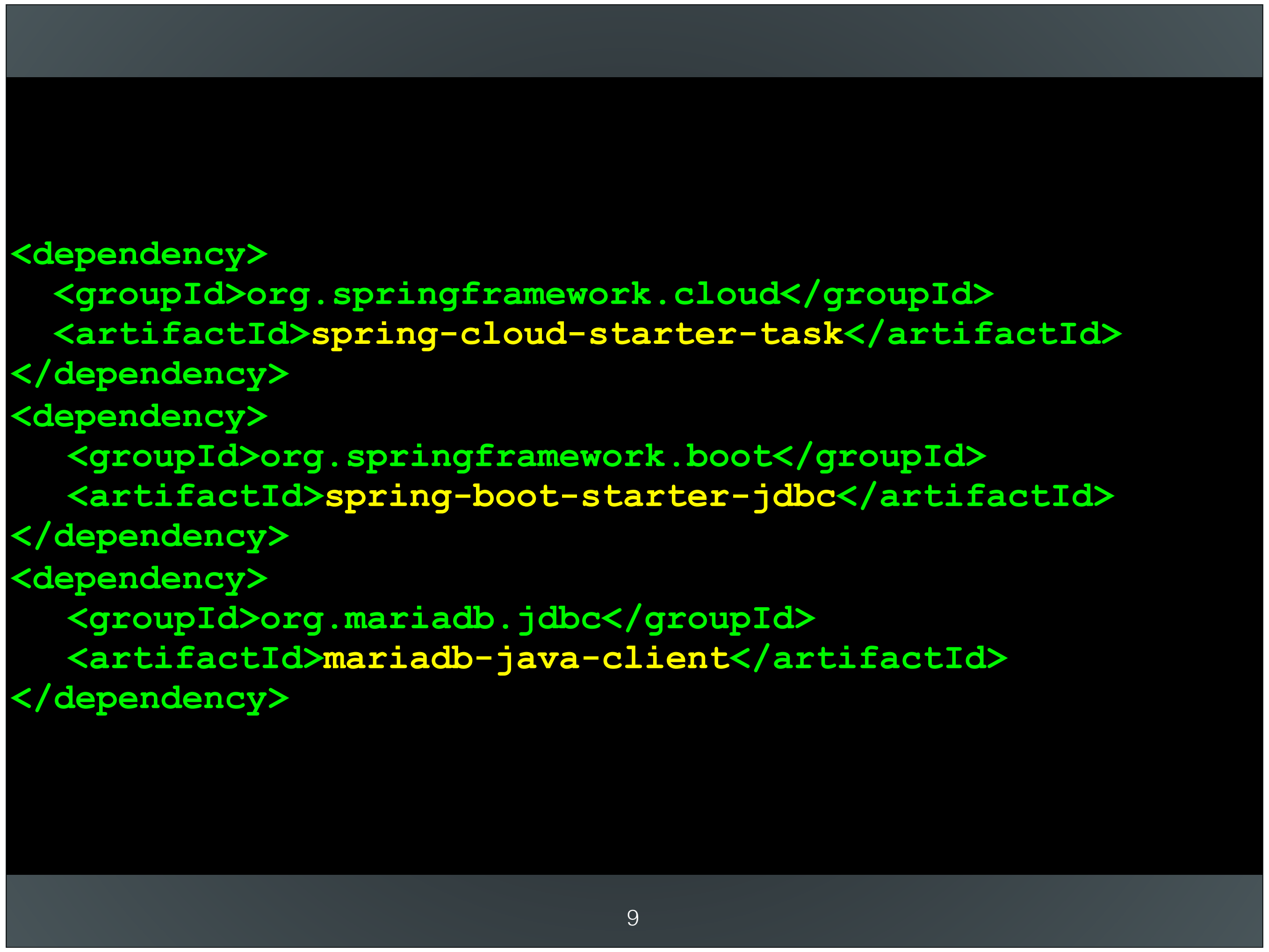
https://flic.kr/p/7DUk5



```
@SpringBootApplication
@EnableTask
public class TaskApplication {
  public static void main(String[] args) {
     SpringApplication.run(TaskApplication.class, args);
  }
  public class TimestampTask implements
     CommandLineRunner {
     public void run(String... strings) throws Exception {
        ...
     }
  }
}
```

So what we see is that it is a core Spring boot application that is using a CommandLineRunner.

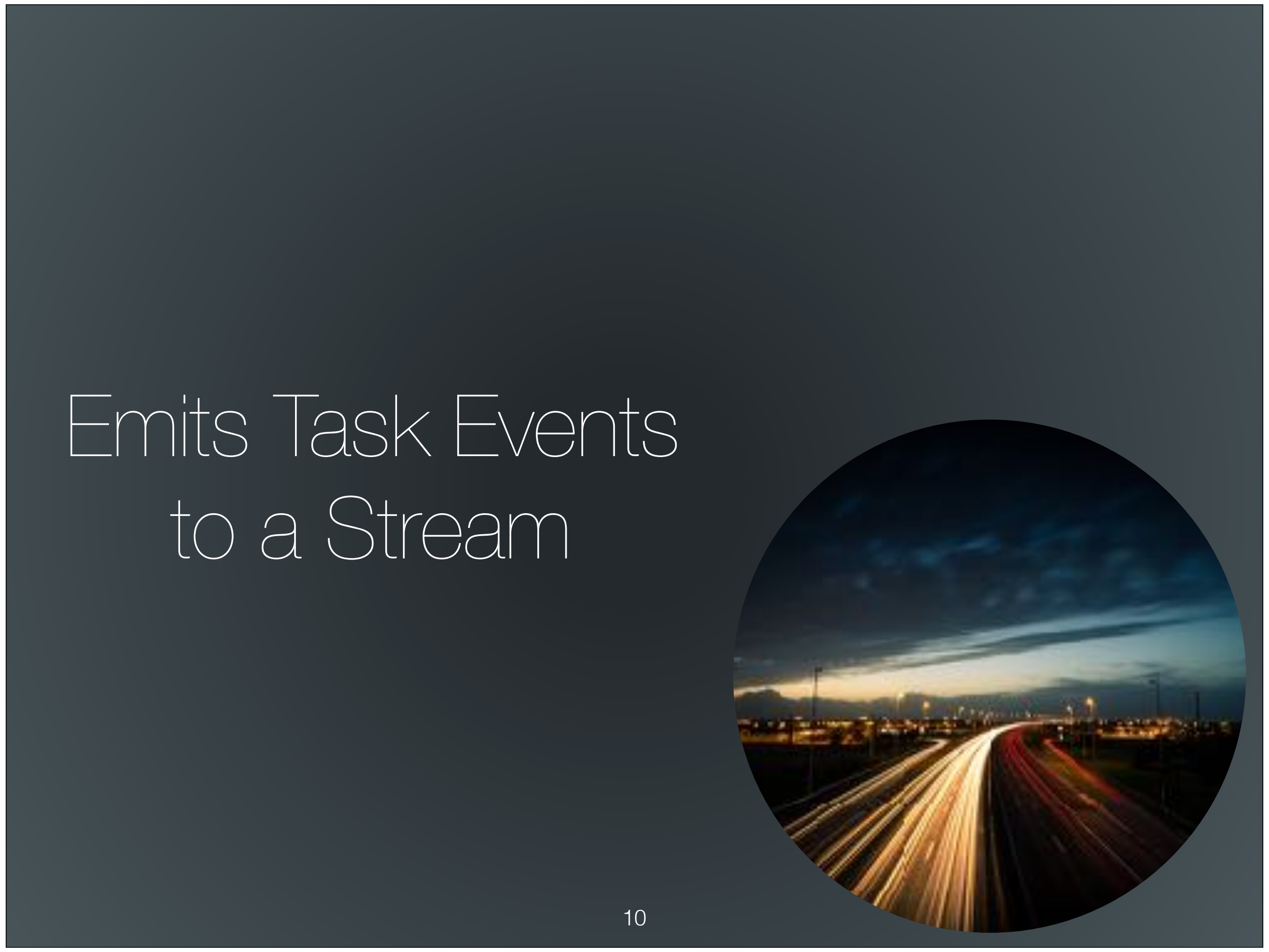
- The primary differentiator is that we have the annotation @EnableTask



```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-task</artifactId>
</dependency>
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<dependency>
   <groupId>org.mariadb.jdbc</groupId>
   <artifactId>mariadb-java-client</artifactId>
</dependency>
```

The dependencies above provide the following:

- spring-cloud-starter-task Auto configures the boot application to be a task
- spring-boot-starter-jdbc Auto configures a datasource for the application and by default task will use it as well.
- mariadb-java-client Establishes what database driver to use for your application. In this example we are using the opensource driver for mysql.

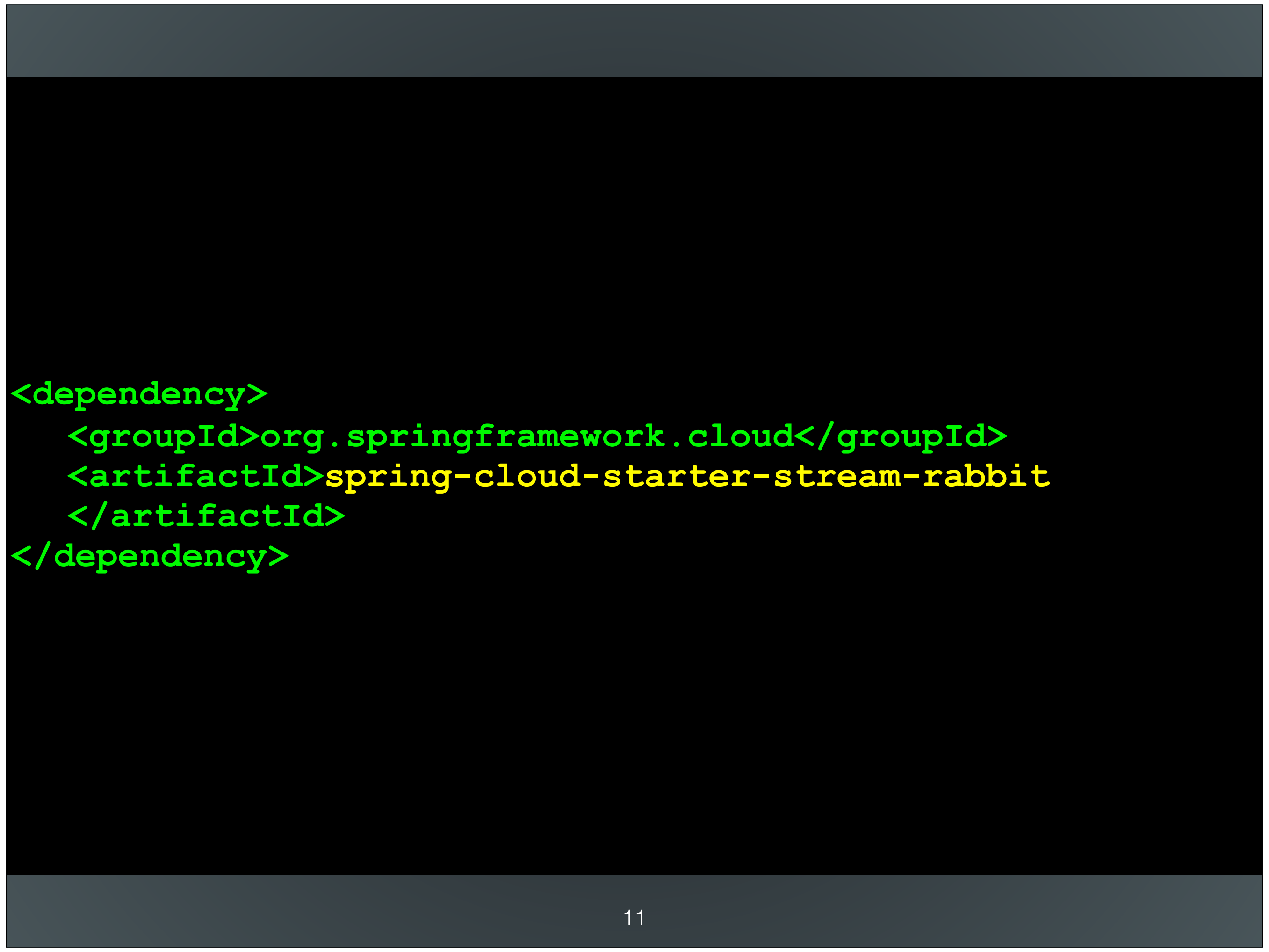


Emits Task Events to a Stream

Task Events

- Emit TaskEvent via Spring Cloud Stream channel
    - Task Start
    - Task End
- Automatically enabled when SCSt dependency is added for example

https://flic.kr/p/oHpnei



```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-stream-rabbit
   </artifactId>
</dependency>
```

As we discussed earlier Spring Cloud Task can emit task events (Task Execution Start and End)
In those cases where we want our task to emit events I can add a starter stream to dependencies. And in that case we can emit the events to Rabbit, Kafka etc.   In the case above it will be to Rabbit.

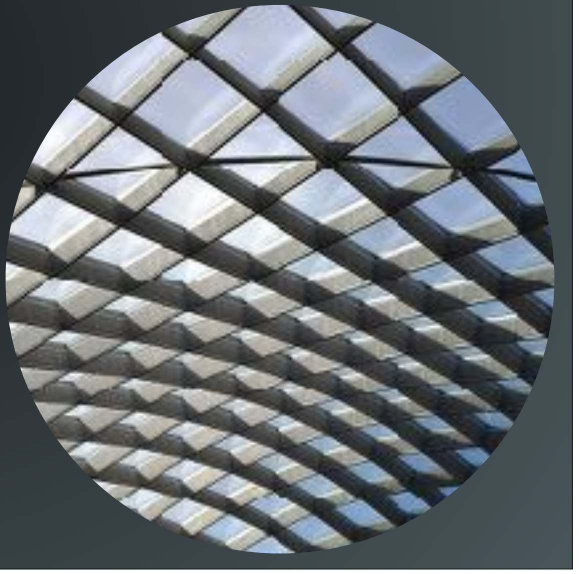
Emits Batch Events to a Stream

Batch Events

- Emit BatchEvents via Spring Cloud Stream channel
    - job-execution-events
    - step-execution-events
    - chunk-events
    - item-read-events
    - item-process-events
    - item-write-events
    - skip-events

https://flic.kr/p/82tvyd

Remote Partitioning



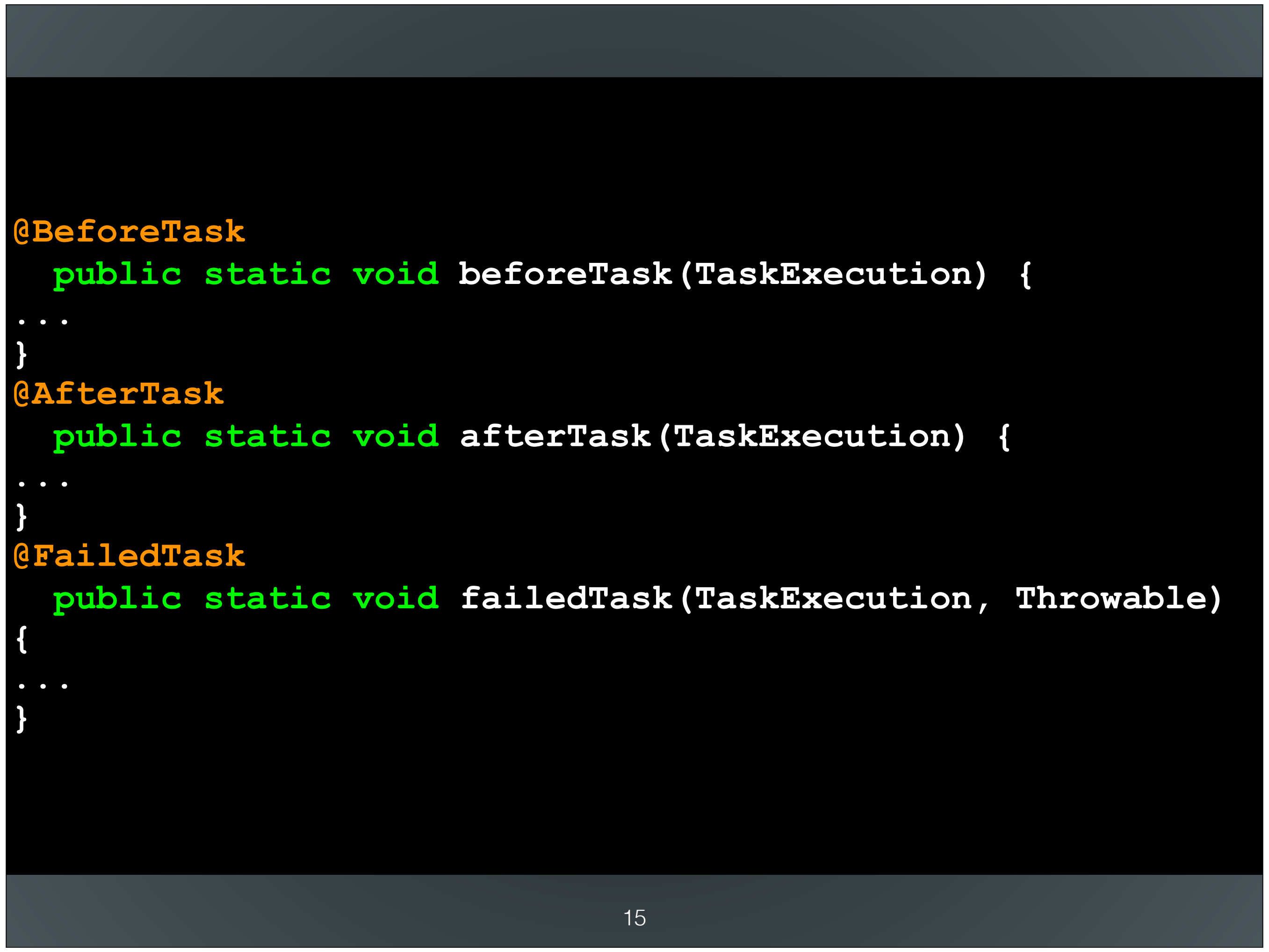
Remote Partitioning

- Spring Batch provides an SPI for partitioning a Step execution and executing it remotely.
- Spring Cloud Task provides the infrastructure to allow a user to execute Batch Partitions on most cloud platforms.
- This is done using Spring Cloud Deployer.
- An example of this can be seen here: https://github.com/spring-cloud/spring-cloud-task/tree/master/spring-cloud-task-samples/partitioned-batch-job

https://flic.kr/p/dHH8ui

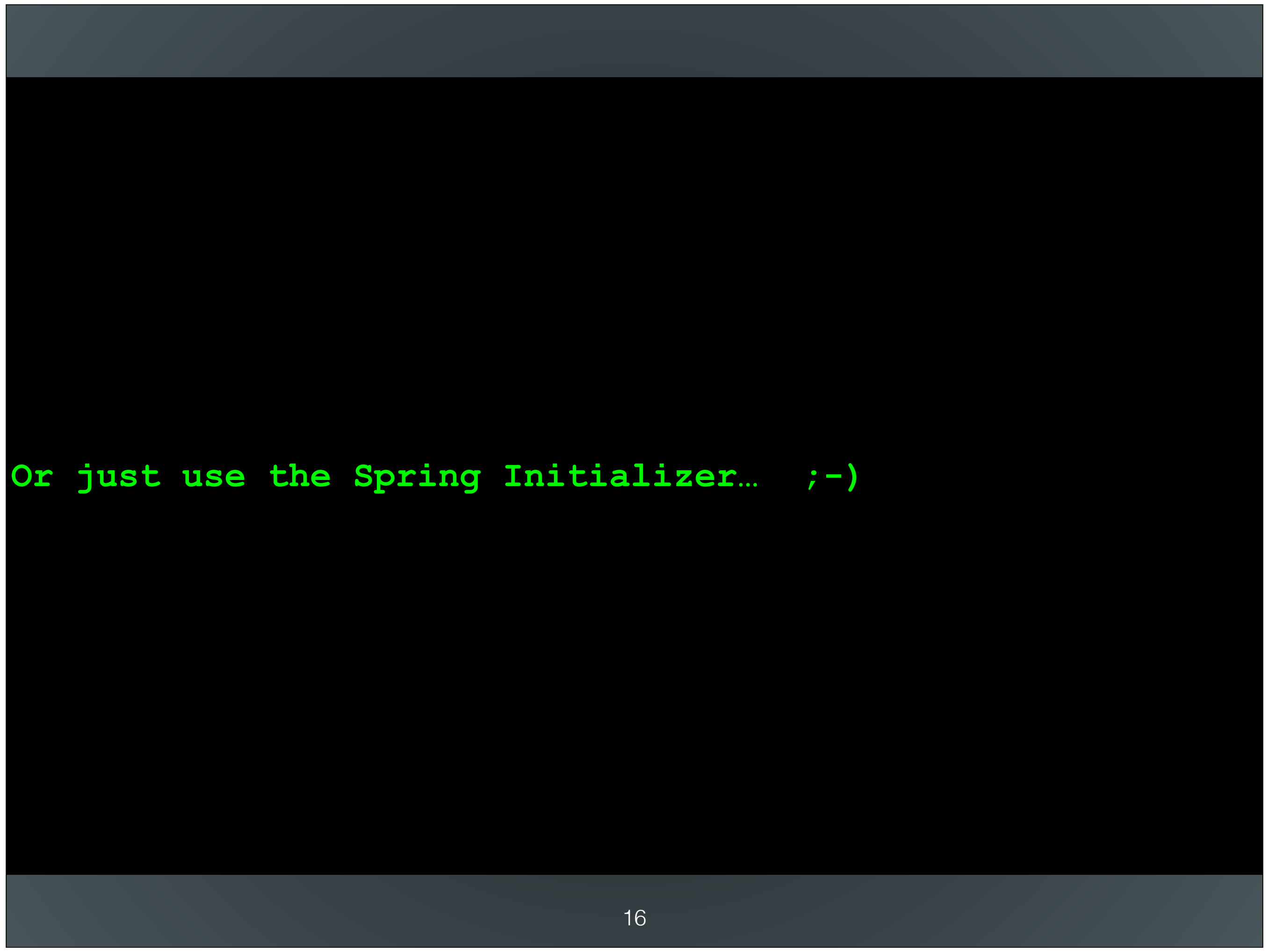
Cool,
but how do you
make it work?

https://flic.kr/p/8stahR



```
@BeforeTask
  public static void beforeTask(TaskExecution) {
...
}
@AfterTask
  public static void afterTask(TaskExecution) {
...
}
@FailedTask
  public static void failedTask(TaskExecution, Throwable)
{
...
}
```

- As we discussed earlier you can register listeners for the following events:
  - beforeTask
  - afterTask
  - failedTask
- This can be done by 2 methods
  - Create a class that implements the TaskExecutionListener
  - or (as shown above) use the @BeforeTask, @AfterTask, or @FailedTask



Or just use the Spring Initializer… ;-)

- Just remember in your tests add the following line below your @SpringBootTest
    - @TestPropertySource(properties = {"spring.cloud.task.closecontext_enable=false"})

Demo Time!

Lab 2 - Spring Cloud Task DNDataFlow/labs/lab2/Lab2-SpringCloudTask.pdf