DEPENDENCY INJECTION IN SERVERLESS

Introduction:

Despite functions encapsulating only minimal parts of the business logic (at least, it should be so), FaaS applications can still have large sizes and a big codebase due to the composition of many functions. Their complexity derives from the dependencies between functions and external resources like libraries, DBs or other cloud services, as well as from the relationships between the functions themselves. This complexity can lead to non-maintainable and non-scalable systems. Other issues can present during testing phase: if our functions are strictly coupled with their dependencies, it will be hard to test the system in different environments such as non-production ones. In fact, if we perform tests in a local environment, it is highly probable that we can't access the same cloud resources. So, we want to easily switch from production's dependencies to mock dependencies without having to change the code more than necessary.

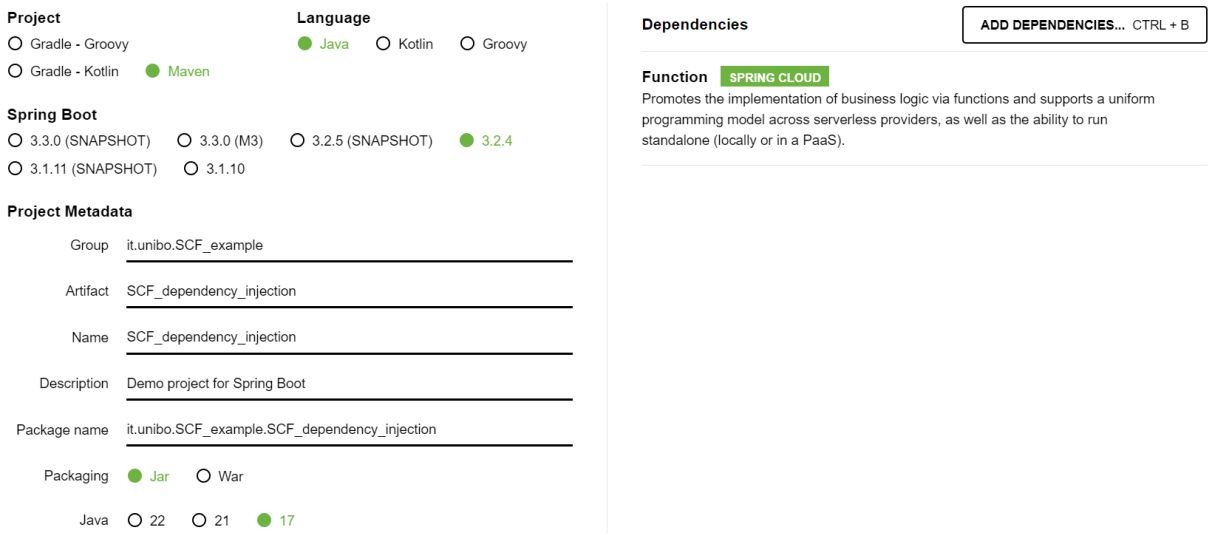
In such a scenario, dependency injection can play a fundamental role to obtain a more modular architecture and loose coupled functions. In this way, our applications will be more portable and independent from the specific implementation of the services they need to use. In the following chapters, we will present two open-source frameworks that provide two different ways to implement dependency injection pattern in Function as a Service using Java: Spring Cloud Function and Dagger 2. In the last part, we will examine other ways to implement dependency injection "manually" via modification of Knative and OpenFunction components.

Spring Cloud Function:

It is a framework that promotes the implementation of business logic as Java Functions. So, not only it provides a container that enables dependency injection, but it can also be used to develop, build and deploy serverless applications. It offers a set of annotations that easily allow the development of Functions.

We present a practical example to understand what can be done using Spring Cloud Function (full code at https://github.com/FabioGentili99/project-work-on-Distributed-Systems-M/tree/main/SCF_dependency_injection/SCF_dependency_injection). It shows a simple application composed by a single function that uses a class to compose the response message:

1) Go to https://start.spring.io/ and setup a project like in the picture below.



2) Create a class ResponseBuilder that is responsible for building the response message. By tagging it with @Service, we are saying that it is a resource that can be injected in other classes by Spring container:

```java
@Service
public class ResponseBuilder {

    private StringBuilder sb;
    private final String greeting = "hello ";

    public ResponseBuilder() { this.sb = new StringBuilder(); }

    public String getResponse(String value){
        return sb.append(greeting).append(value).toString();
    }
}
```

3) Modify the main class by adding our Function tagged with @Bean and the field representing the injectable service tagged with @Autowire.

```java
@SpringBootApplication
public class ScfDependencyInjectionApplication {

    @Autowired
    private ResponseBuilder rb;

    public static void main(String[] args) {
        SpringApplication.run(ScfDependencyInjectionApplication.class, args);
    }

    @Bean
    public Function<String, String> greetings(){
        return value -> rb.getResponse(value);
    }
}
```

4) By running the application, Spring container will search for methods and field tagged with @Autowired and it will inject the corresponding dependencies. Furthermore, our

Function will be accessible at http://localhost:8080/greetings. So, we can trigger it by running the following command from terminal:

```
curl localhost:8080/greetings -H "Content-Type: text/plain" -d "world"
```

The response message will be: `hello world`

As we can see, Spring Cloud Function allows us to write code for serverless applications that use dependency injection as if they were traditional web applications.

Dagger 2:

Usually, dependency injection is executed at run-time by using reflection mechanism. This allows our applications to be more dynamic and to start running before the dependencies are resolved, but it adds overhead during execution.

Dagger 2 is a fully static, compile-time, open-source dependency injection framework for both Java and Android. It implements dependency injection by using code generation: the annotations are translated into code during compile-time. Dagger functionalities are based on four annotations:

- @Inject: it is used to annotate classes and field that need dependency injection. So, Dagger will construct those instances.
- @Module: it is used to annotate classes whose methods provide dependencies. In this way, Dagger will know where to find the dependencies in order to satisfy them when constructing class instances.
- @Provide: inside modules we define methods containing this annotation wich tells Dagger how we want to construct and provide those dependencies.
- @Component: it is used to annotate injectors.

References:

https://javascript.plainenglish.io/dependency-injection-for-serverless-applications-359bd3f43b4ahttps://javascript.plainenglish.io/dependency-injection-for-serverless-applications-359bd3f43b4a

https://anilktalla.medium.com/di-for-aws-java-lambda-48a83338d806

https://medium.com/@dasi.rajesh08/serverless-functions-with-spring-cloud-function-97e88c940b33

https://frogermcs.github.io/dependency-injection-with-dagger-2-the-api/