Spring Cloud Function – 2024

Spring Cloud Function is a project with the following high-level goals:

* Promote the implementation of business logic via functions.
* Decouple the development lifecycle of business logic from any specific runtime target so that the same code can run as a web endpoint, a stream processor, or a task.
* Support a uniform programming model across serverless providers, as well as the ability to run standalone (locally or in a PaaS).
* Enable Spring Boot features (auto-configuration, dependency injection, metrics) on serverless providers.

It abstracts away all of the transport details and infrastructure, allowing the developer to keep all the familiar tools and processes, and focus firmly on business logic.

**How to create Spring Cloud Function Project in Eclipse**

**Project Structure**

A screenshot of a computer

Description automatically generated

**Pom.xml relevant portion**

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.3.6</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<java.version>21</java.version>

<spring-cloud.version>2023.0.3</spring-cloud.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-function-context</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-function-webflux</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

<!--<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-function-web</artifactId>

</dependency>-->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

**application.properties**

spring.cloud.function.scan.packages=com.ddlab.rnd.function 🡸 Define all functions in that package

#http://localhost:8080/myfn/myCheckFunction

# Below is the context path

spring.cloud.function.web.path=/myfn 🡸 Context path for Spring cloud application

**Main Class**

@EnableAutoConfiguration

@SpringBootApplication

**public** **class** SpringFunctionApp {

// http://localhost:8080/uppercase/hallo

@Bean

**public** Function<String, String> uppercase() {

**return** value -> value.toUpperCase();

}

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringFunctionApp.**class**, args);

}

}

**Class defining some functions**

@Component

**public** **class** SomeClass {

//GET http://localhost:8080/enrich

// It provide only header foo as key bar as value

@Bean

**public** Function<Message<?>, Message<?>> enrich() {

**return** message -> MessageBuilder.*fromMessage*(message).setHeader("foo", "bar").build();

}

// GET http://localhost:8080/getSome

@Bean

**public** Supplier<Message<String>> getSome() {

**return** () -> MessageBuilder.*withPayload*("Hello Mr").build();

}

//GET http://localhost:8080/consumeSome

@Bean

**public** Consumer<Message<String>> consumeSome() {

**return** msg -> {

System.***out***.println("Some Randome Message: "+msg);

System.***out***.println("Message Payload---->"+msg.getPayload());

};

}

}

**Inside the package com.ddlab.rnd.function**

// GET http://localhost:8080/charCounter/abcd

**public** **class** CharCounter **implements** Function<String, Integer> {

@Override

**public** Integer apply(String word) {

**return** word.length();

}

}

//GET http://localhost:8080/greeter/John

**public** **class** Greeter **implements** Function<String, String> {

**public** String apply(String name) {

**return** "Hello " + name;

}

}

/\*\*

\* POST or PUT http://localhost:8080/myCheckFunction

\* Below is the message body

\* {

"id": 123,

"name": "John Abraham"

}

\*/

**public** **class** MyCheckFunction **implements** Function<Employee,Employee> {

@Override

**public** Employee apply(Employee emp) {

System.***out***.println("Employee received : "+emp);

Employee ee = **new** Employee();

ee.setId(11);

ee.setName("DD");

**return** ee;

}

}

Here Employee is a POJO annotated with Lombok @Data.

Notes

You can also use both PUT and Post call http://localhost:8080/myCheckFunction

A typical Application Context may include beans that are valid java functions, but not intended to be candidates to be registered with FunctionCatalog. You can use following properties.

**spring.cloud.function.ineligible-definitions=foo,bar**

**Funtion or Supplier can be imperative(normal way) or reactive(Reactive type Flux or Mono)**

--spring.cloud.function.definition=foo;bar This will only export function foo and function bar regardless how many functions are available in catalog (e.g., localhost:8080/foo).

--spring.cloud.function.definition=foo|bar;baz This will only export function composition foo|bar and function baz regardless how many functions are available in catalog (e.g., localhost:8080/foo,bar).

CRUD REST with Spring Cloud Function By now it should be clear that functions are exported as REST endpoints and can be invoked using various HTTP methods. In other words a single function could be triggered via GET, POST, PUT etc.

However, it is not always desirable and certainly does not fit the CRUD concept. And while SCF does not support and has no intention of supporting all the features of Spring web stack, the framework does provide support for CRUD mappings where a single function could be mapped to a particular HTTP method(s). It is done via spring.cloud.function.http. property.

For example,

spring.cloud.function.http.GET=uppercase;reverse;foo|bar spring.cloud.function.http.POST=reverse spring.cloud.function.http.DELETE=deleteById