

# Quarkus - Funqy

Quarkus Funqy is part of Quarkus's serverless strategy and aims to provide a portable Java API to write functions deployable to various FaaS environments like AWS Lambda, Azure Functions, Knative, and Knative Events (Cloud Events). It is also usable as a standalone service.

Another goal of Funqy is to create an RPC framework that is as small and as optimized as possible for the Quarkus runtime. This means sacrificing a little bit on flexibility to provide a runtime that has little to no overhead. Funqy should never become more complicated than you see in this initial doc.

## Funqy Basics

The Funqy API is simple. Annotate a method with `@Funq`. This method may only have one optional input parameter and may or may not return a response.

```
import io.quarkus.funqy.Funq;

public class GreetingFunction {
    @Funq
    public String greet(String name) {
        return "Hello " + name;
    }
}
```

Java classes can also be used as input and output and must follow the Java bean convention and have a default constructor. The Java type that is declared as the parameter or return type is the type that will be expected by the Funqy runtime. Funqy does type introspection at build time to speed up boot time, so any derived types will not be noticed by the Funqy marshalling layer at runtime.

Here's an example of using a POJO as input and output types.

```

public class GreetingFunction {
    public static class Friend {
        String name;

        public String getName() { return name; }
        public void setName(String name) { this.name = name; }
    }

    public static class Greeting {
        String msg;

        public Greeting() {}
        public Greeting(String msg) { this.msg = msg; }

        public String getMessage() { return msg; }
        public void setMessage(String msg) { this.msg = msg; }
    }

    @Funq
    public Greeting greet(Friend friend) {
        return new Greeting("Hello " + friend.getName());
    }
}

```

## Async Reactive Types

Funqy supports the [Smallrye Mutiny Uni](#) reactive type as a return type. The only requirement is that the [Uni](#) must fill out the generic type.

```

import io.quarkus.funqy.Funq;
import io.smallrye.mutiny.Uni;

public class GreetingFunction {

    @Funq
    public Uni<Greeting> reactiveGreeting(String name) {
        ...
    }
}

```

## Function Names

The function name defaults to the method name and is case sensitive. If you want your function referenced by a different name, parameterize the [@Funq](#) annotation as follows:

```
import io.quarkus.funqy.Funq;

public class GreetingFunction {

    @Funq("HelloWorld")
    public String greet(String name) {
        return "Hello " + name;
    }
}
```

## Funqy DI

Each Funqy Java class is a Quarkus Arc component and supports dependency injection through CDI or Spring DI. Spring DI requires including the `quarkus-spring-di` dependency in your build.

The default object lifecycle for a Funqy class is `@Dependent`.

```
import io.quarkus.funqy.Funq;

import javax.inject.Inject;
import javax.enterprise.context.ApplicationScoped;

@ApplicationScoped
public class GreetingFunction {

    @Inject
    GreetingService service;

    @Funq
    public Greeting greet(Friend friend) {
        Greeting greeting = new Greeting();
        greeting.setMessage(service.greet(friend.getName()));
        return greeting;
    }
}
```

## Context injection

You can inject contextual information that is specific to the Funqy runtime or specific to the Funqy binding you are using (lambda, azure, cloud events, etc.).



We do not recommend injecting contextual information specific to a runtime. Keep your functions portable.

Contextual information is injected via the `@Context` annotation which can be used on a function

parameter or a class field.

```
import io.quarkus.funqy.Funq;
import io.quarkus.funqy.Context;

public class GreetingFunction {

    @Funq
    public Greeting greet(Friend friend, @Context AwsContext ctx) {
        Greeting greeting = new Greeting();
        greeting.setMessage(service.greet(friend.getName()));
        return greeting;
    }
}
```