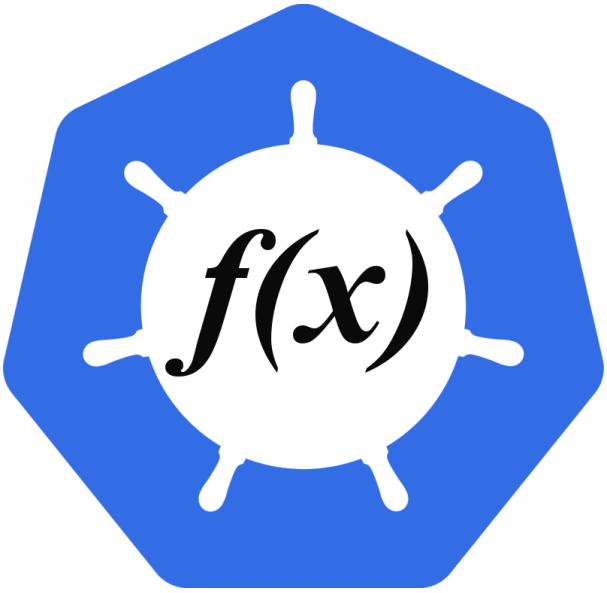
Funktion

Funktion

Chapter 1. Introduction

`Funktion` is an open source event driven lambda style programming model designed for Kubernetes.



Funktion supports over 200 of different event sources and connectors including most network protocols, transports, databases, messaging systems, social networks, cloud services and SaaS offerings.

In a sense funktion is a serverless approach to event driven microservices as you focus on just writing simple *functions* in whatever programming language you prefer, then `funktion` and Kubernetes takes care of the rest. Its not that there's no servers; its more that you as the funktion developer don't have to worry about managing them!

Chapter 2. Installing Funktion

To use `funktion` you will need a kubernetes or openshift cluster.

If you are on your laptop a quick way to get a kubernetes cluster is by installing and starting minikube and then installing kubectl and putting it on your `PATH` environment variable.

You will also need to download the funktion binary for your platform and add it to your `PATH` environment variable

Chapter 3. Getting Started

Type the following commands.

To make it easier to see what kubernetes resources are being created you may wish to create a new namespace for this experiment first:

```
kubectl create namespace funky
kubectl config set-context `kubectl config current-context` --namespace=funky
```

Now we'll install the runtimes and a couple of connectors

```
funktion install timer twitter
```

Now lets run the `funktion operator` to watch for funktion resources and create the necessary kubernetes `Deployment` and `Services`.

```
funktion operate
```

Open another terminal then type:

```
kubectl apply -f https://raw.githubusercontent.com/fabric8io/funktion-
operator/master/examples/subscription1.yml
```

You should now have created a subscription flow. You can view the subscription via

```
funktion get subscription
```

To view the output of the subscription you can use the following (assuming you've enabled tab completion for kubectl

```
kubectl logs -f subscription1-[TAB]
```

If you don't have tab completion you can specify the exact pod name, or you can use this command to find it and use it:

```
kubectl logs -f `kubectl get pod -oname -lfunktion.fabric8.io/kind=Subscription`
```

To delete the subscription:

```
funktion delete subscription subscription1
```

Now lets create a function:

```
kubectl apply -f https://raw.githubusercontent.com/fabric8io/funktion-
operator/master/examples/function1.yml
```

If you are running the fabric8 console then you will have the link:[exposecontroller] microservice running and will be able to invoke it via running one of these commands:

```
minikube service function1 -n funky
gofabric8 service function1 -n funky
```

Or clicking on the `funktion1` service in the fabric8 console in the `Services` tab for the `funky` namespace.

Chapter 4. Using the CLI

You can get help on the available commands via:

```
funktion
```

4.1. Browsing resources

To list all the resources of different kind via:

```
funktion get connector
funktion get subscription
funktion get function
funktion get runtime
```

or to save typing you can use:

```
funktion get c
funktion get s
funktion get f
funktion get r
```

4.2. Deleting resources

You can delete a Connector or Subscription via:

```
funktion delete connector foo
funktion delete subscription bar
funktion delete function whatnot
funktion delete runtime nodejs
```

Or to remove all the Subscriptions or Connectors use `--all`

```
funktion delete subscription --all
```

4.3. Installing Runtimes and Connectors

To install the default function runtimes and connectors into your namespace type the following:

```
funktion install --all-connectors
```

There's over 200 connectors provided out of the box. If you only want to install a number of them

you can specify their names as parameters

```
funktion install amqp kafka timer
```

To just get a feel for what connectors are available without installing them try:

```
funktion install --list-connectors
```

or for short:

```
funktion install -l
```

4.4. Running the Operator

You can run the funktion operator from the command line if you prefer:

```
funktion operate
```

Though ideally we'd run the `funktion application` inside kubernetes; via a helm chart, `kubectl apply` or the `Run⋯` button in the fabric8 developer console

4.5. Subscribing to events

To create a new subscription for a connector try the following:

```
funktion subscribe --from timer://bar?period=5000 --to http://foo/
```

This will generate a new `Subscription` which will result in a new `Deployment` being created and one or more Pods should spin up.

Note that you must be running the `Operator` as described in the section above; its the `Operator` which actually creates a `Deployment` for each `Subscription`.

Also note that the first time you try out a new Connector kind it may take a few moments to download the docker image for this connector - particularly the first time you use a connector.

Once a pod has started for the `Deployment` you can then view the logs of a subscription via `kubectl`

```
kubectl logs -f nameOfSubscription[TAB]
```

Scaling a Subscription

If you want to stop a subscription type:

```
kubectl scale --replicas=0 deployment nameOfSubscription
```

To start it again:

```
kubectl scale --replicas=1 deployment nameOfSubscription
```

Using kubectl directly

You can also create a Subscription using `kubectl` if you prefer:

```
kubectl apply -f https://github.com/fabric8io/funktion-
operator/blob/master/examples/subscription1.yml
```

You can view all the Connectors and Subscriptions via:

```
kubectl get cm
```

Or delete them via

```
kubectl delete cm nameOfConnectorOrSubscription
```

Chapter 5. How it works

The `funktion operator` watches for `Subscription` and `Function` resources.

When a new `Subscription` is created then this operator will spin up a matching `Deployment` which consumes from some `Connector` and typically invokes a function using HTTP.

When a new is created then this operator will spin up a matching `Deployment` for running the `function source code` along with a `Service` to expose the service as a HTTP or HTTPS endpoint.

The following kubernetes resources are used:

5.1. Kubernetes Resources

A `Subscription` is modelled as a Kubernetes `ConfigMap` with the label `kind.funktion.fabric8.io: "Subscription"`. A ConfigMap is used so that the entries inside the `ConfigMap` can be mounted as files inside the `Deployment`. For example this will typically involve storing the `funktion.yml` file or maybe a Spring Boot `application.properties` file inside the `ConfigMap` like this example subscription

A `Connector` is generated for every Camel Component and each connector has an associated `ConfigMap` resource like this example which uses the label `kind.funktion.fabric8.io: "Connector"`. The `Connector` stores the `Deployment` metadata, the `schema.yml` for editing the connectors endpoint URL and the `documentation.adoc` documentation for using the Connector.

So a `Connector` can have `0..N Subscriptions` associated with it. For those who know Apache Camel this is like the relationship between a `Component` having `0..N Endpoints`.

For example we could have a Connector called `kafka` which knows how to produce and consume messages on Apache Kafka with the Connector containing the metadata of how to create a consumer, how to configure the kafka endpoint and the documetnation. Then a Subscription could be created for `kafka://cheese` to subscribe on the `cheese` topic and post messages to `http://foo/`.

Typically a number of `Connector` resources are shipped as a package; such as inside the Red Hat iPaaS or as an app inside fabric8. Though a `Connector` can be created as part of the CD Pipeline by an expert Java developer who takes a Camel component and customizes it for use by `Funktion` or the `iPaaS`.

The collection of `Connector` resources installed in a kubernetes namespace creates the `integration palette` thats seen by users in tools like CLI or web UIs.

Then a `Subscription` can be created at any time by users from a `Connector` with a custom configuration (e.g. choosing a particular queue or topic in a messaging system or a particular table in a database or folder in a file system).

5.2. Debugging

If you ever need to you can debug any `Subscription` as each Subscription matches a `Deployment` of one or more pods. So you can just debug that pod which typically is a regular Spring Boot and camel application.

Otherwise you can debug the pod thats exposing an HTTP endpoint using whatever the native debugger is; e.g. using Java or NodeJS or whatever.