OpenFaaS First Function

In this example, we are going to deploy Python3 functions to our OpenFaaS.

Before we start, make sure all is working and you followed the guides before on how to set up K3s on Raspberry Pi 4, create TLS private registry and install faas-cli/OpenFaaS.

Read the official documentation

Always, the best and first thing to do is to read the official OpenFaaS documentation.

- https://docs.openfaas.com/

Let's start

Notice

Keep in mind, we are going this on control01 which is my control node for my Kubernetes cluster and where we set up Docker. Normally, you should not do this: Official recommendation is to build on your workstation and push via OpenFaas gateway, like in the documentation.

Create a new directory where you are going to work; I called mine `openfaas_scripts`. We need to download templates into this directory. Faas-cli is reading a skeleton of the function from this directory when creating a new function.

```
cd openfaas_scripts
faas-cli template pull
```

Now you should have new folder called `template`.

Template store

We are going to use Python3 because that's what I'm the most familiar with, but the beauty of OpenFaaS is that you are not limited to one language. You can list supported templates like this:

```
ubuntu@control01:~/openfaas_scripts$ faas-cli template store list

NAME                     SOURCE             DESCRIPTION
csharp                   openfaas           Classic C# template
dockerfile               openfaas           Classic Dockerfile template
go                       openfaas           Classic Golang template
java8                    openfaas           Java 8 template
java11                   openfaas           Java 11 template
java11-vert-x            openfaas           Java 11 Vert.x template
node12                   openfaas           HTTP-based Node 12 template
node                     openfaas           Classic NodeJS 8 template
php7                     openfaas           Classic PHP 7 template
python                   openfaas           Classic Python 2.7 template
python3                  openfaas           Classic Python 3.6 template
python3-dlrs             intel              Deep Learning Reference Stack v0.4 for ML workloads
ruby                     openfaas           Classic Ruby 2.5 template
ruby-http                openfaas           Ruby 2.4 HTTP template
python27-flask           openfaas           Python 2.7 Flask template
python3-flask            openfaas           Python 3.7 Flask template
python3-flask-debian     openfaas           Python 3.7 Flask template based on Debian
python3-http             openfaas           Python 3.7 with Flask and HTTP
python3-http-debian      openfaas           Python 3.7 with Flask and HTTP based on Debian
golang-http              openfaas           Golang HTTP template
golang-middleware        openfaas           Golang Middleware template
python3-debian           openfaas           Python 3 Debian template
powershell-template      openfaas-incubator Powershell Core Ubuntu:16.04 template
powershell-http-template openfaas-incubator Powershell Core HTTP Ubuntu:16.04 template
rust                     booyaa             Rust template
crystal                  tpei               Crystal template
csharp-httprequest       distantcam         C# HTTP template
csharp-kestrel           burtonr            C# Kestrel HTTP template
vertx-native             pmlopes            Eclipse Vert.x native image template
swift                    affix              Swift 4.2 Template
lua53                    affix              Lua 5.3 Template
vala                     affix              Vala Template
vala-http                affix              Non-Forking Vala Template
quarkus-native           pmlopes            Quarkus.io native image template
perl-alpine              tmiklas            Perl language template based on Alpine image
crystal-http             koffeinfrei        Crystal HTTP template
rust-http                openfaas-incubator Rust HTTP template
bash-streaming           openfaas-incubator Bash Streaming template
cobol                    devries            COBOL Template
```

Creating new function

Creating new functions in OpenFaaS is super easy.

Run:

```
faas-cli new --lang python3 mailme
```

`--lang` specifies the template name from above, and the next parameter is the name of our function, in my case it's `mailme`.

Now you should have new folder called `mailme`, and in it:

```
root@control01:/home/ubuntu/openfaas_scripts/mailme# ll
total 12
drwx------ 2 ubuntu ubuntu 4096 Feb  2 17:27 ./
drwxrwxr-x 5 ubuntu ubuntu 4096 Feb  2 16:19 ../
-rw-rw-r-- 1 ubuntu ubuntu    0 Jan 31 13:47 __init__.py
-rw-rw-r-- 1 ubuntu ubuntu 2568 Feb  2 17:27 handler.py
-rw-rw-r-- 1 ubuntu ubuntu    0 Jan 31 13:47 requirements.txt
```

and also `mailme.yml` outside of it.

This is super simple:

- ***handler.py*** - This is where your function will live.
- ***requirements.txt*** - If you require any additional modules to be present that are not in Python3 by default, here you can add the names and they will get automagically installed (it works as any requirements.txt in standalone python).

What will our function do?

I wanted something simple that would also include some core stuff that OpenFaaS supports, especially `secrets` and passing `variables` to script from outside. Our function will send mail. Simple as that 🙂.

It's good to define some input parameters ahead of starting so that we do not change stuff in the middle of coding. Our function will accept 3 parameters in yaml format.

- ***api-key*** - a secret api-key that will live in `Kubernetes secrets`, and will be presented to your function as a readable file (so it's not hard coded into code), and can be shared with other functions if you want. Read more about `Kubernetes secrets`: kubernetes.io
- ***msg*** - What message to send.
- ***to*** - To what email to send the message.

Input parameters in yaml are:

```
{
"api-key": "gr35p4inyyr4e9",
"msg": "test msg",
"to": "my@gmail.com"
}
```

Secrets

Secrets are the preferred way to pass sensitive information to your function. They need to be created ahead of the deployment of the function. We are going to put our api-key and password for the smtp server we are going to use to send mail.

```
kubectl create secret generic api-key --from-literal api-key="gr35p4inyyr4e9" --namespace openfaas-fn
kubectl create secret generic email-pass --from-literal email-pass="smtp_email_password_goes_here" --namespace openfaas-fn
```

In case you want to list your secret names use:

```
root@control01:/home/ubuntu/openfaas_scripts# kubectl get secret -n openfaas-fn
NAME                  TYPE                                  DATA   AGE
api-key               Opaque                                1      3d22h
default-token-jr2bh   kubernetes.io/service-account-token   3      11d
email-pass            Opaque                                1      3d22h
```

To delete secret:

```
kubectl delete secret -n openfaas-fn <secret_name>
```

To read the secret:

```
# You might need to install jq
root@control01:/home/ubuntu/openfaas_scripts# kubectl get secret api-key -n openfaas-fn -o json | jq '.data '
{
  "api-key": "Z3IzNXA0aW55eXI0ZTk="
}
echo "Z3IzNXA0aW55eXI0ZTk=" | base64 --decode
gr35p4inyyr4e9
```

Secrets in OpenFaaS

To get to the secret inside your OpenFaaS function you first need to create it, then define it in your function yaml file.

My whole `mailme.yml` looks like this (I will get to the other parameters later on):

```
version: 1.0
provider:
  name: openfaas
  gateway: http://openfaas.cube.local:8080
functions:
  mailme:
    lang: python3
    handler: ./mailme
    image: registry.cube.local:5000/mailme:latest

    environment:
      smtp_server: smtp.websupport.sk
      smtp_login: admin@rpi4cluster.com
      sender: admin@rpi4cluster.com

    secrets:
      - api-key
      - email-pass
```

You can see there is a section called `secrets`; here, we specify the names of secrets we created.

But how the fuck did I get to them inside my function? I'm glad you asked. They will be presented as a file; it's up to you to read and use it.

They will always be in format:

```
'/var/openfaas/secrets/' + secret_name
```

That's secrets, but what about not so secret stuff I don't want to bake directly into code and be able to change without recompiling everything? Well, environmental variables are what you want.

Environment variables in OpenFaas

These are variables defined outside your code; something you might like to change without going through the process of rebuilding your whole function. There are for non-sensitive data, perhaps configuration.

In this case, you can see above in my `mailme.yml` I defined three such variables:

- ***smtp_server*** - Server we are going to use to send my mail through.
- ***smtp_login*** - Username for that server (remember we have the password in Kubernetes secrets).
- ***sender*** - Who's the sender of this message?

Super simple right?

To get to these in your code, you need to query environmental variables from the OS. This depends very much on language you going to use, for example in Python 3 can do:

```
os.getenv(var_name)
```

That should be it for secrets and environment variables. Let's get to the main function.

OpenFaaS Function

Go to `../mailme/handler.py`. The `handler.py` is your function, and it's pre populated with:

```
def handle(req):
    """handle a request to the function
    Args:
        req (str): request body
    """

    return req
```

This function `handle(input)` is called when your function is invoked by OpenFaaS. We are going to extend it with my mailing functionality.

Notice

Read the comments in code to get whats going on.

```
# import libraries we going to use
# no shebang is needed at the start
# all libs I'm importing are native to python so I did not put anything in requirements.txt
import os
import json
import smtplib
from smtplib import SMTPException
from smtplib import SMTPDataError
from email.mime.text import MIMEText
from email.utils import formataddr



def get_secret(secret_name):
    '''
    - Returns secret value if exists, if not return False
    - Secret needs to be create before the function is build
    - Secret needs to be defined in functions yaml file
    '''
    try:
        with open('/var/openfaas/secrets/' + secret_name) as secret:
            secret_key = secret.readline().rstrip()
            return secret_key
    except FileNotFoundError:
        return False

def get_variable(var_name):
    '''
    - Returns environment variable value if exists, if not return False
    - Variable needs to be defined in functions yaml file
    '''
    return os.getenv(var_name, False)


def api_key_check(provided_key):
    '''
    - Check if provided api key is valid
    '''
    if get_secret('api-key') == provided_key:
        return True
    else:
        return False


def key_present(json, key):
    '''
    - Return true if Key exist in json
    '''
    try:
        _x = json[key]
    except KeyError:
        return False
    return True


def handle(req):
    """handle a request to the function
    Args:
        req (str): request body
    """

    # If there is value passed to function
    if req:
        # check if its json formated by trying to load it
        try:
            json_req = json.loads(req)
        except ValueError as e:
            return "Bad Request", 400
    else:
        return "Bad Request", 400

    # Before anything check if api key from secret match api key provided
    # Might me good to implement this so there is no random spamming of function
    if key_present(json_req, 'api-key'):
        key = json_req["api-key"]
        if api_key_check(key) is False:
            return "Unauthorized", 401
    else:
        return "Unauthorized", 401

    # Cool if we are here api key was authorized
    # Let check if in posted body are keys that we need ( msg, to)
    if key_present(json_req, 'msg'):
        msg_text = json_req["msg"]
    else:
        return "Bad Request", 400

    if key_present(json_req, 'to'):
        to = json_req["to"]
    else:
        return "Bad Request", 400

    # So we have values for message, to whom to send it, lets get sender
    sender = get_variable('sender')

    # Lets try to build message body and send it out.
    try:
        msg = MIMEText(msg_text)
        msg['From'] = formataddr(('Author', get_variable('sender')))
        msg['To'] = formataddr(('Recipient', to))
        msg['Subject'] = 'OpenFaas Mailer'

        mail_server = smtplib.SMTP_SSL(get_variable('smtp_server'))
        mail_server.login(get_variable('smtp_login'), get_secret('email-pass'))
        mail_server.sendmail(sender, to, msg.as_string())
        mail_server.quit
        return "request accepted", 202
    except SMTPException:
        return "Failed to send email", 500
    except SMTPDataError:
        return "Failed to send email", 500
```

As you can see, the function is quite simple, and contains no sensitive data.

Build and Push OpenFaaS function.

Next go where the `mailme.yml` is, and if you followed my guide from before (and fixed buildx to know where your repository is 🙂) we should have no problem with the following:

Notice

This is specific for arm64. For more information about building a pushing OpenFaaS functions look here: docs.openfaas.com

```
faas-cli publish -f mailme.yml --platforms linux/arm64
```

This should build the function docker image and push it to your repository. Ending with something like this:

```
#25 pushing layers
#25 pushing layers 4.7s done
#25 pushing manifest for registry.cube.local:5000/mailme:latest
#25 pushing manifest for registry.cube.local:5000/mailme:latest 0.1s done
#25 DONE 18.1s
Image: registry.cube.local:5000/mailme:latest built.
[0] < Building mailme done in 57.37s.
[0] Worker done.

Total build time: 57.37s
```

Fantastic! The last steps are ahead. Deploy this sucker to OpenFaaS on your Kubernetes server!

```
faas-cli deploy -f mailme.yml
```

An example of a successful deployment:

```
buntu@control01:~/openfaas_scripts$ faas-cli deploy -f mailme.yml
Deploying: mailme.
WARNING! Communication is not secure, please consider using HTTPS. Letsencrypt.org offers free SSL/TLS certificates.

Deployed. 202 Accepted.
URL: http://openfaas.cube.local:8080/function/mailme.openfaas-fn
```

Check it in Kubernetes:

```
ubuntu@control01:~/openfaas_scripts$ sudo kubectl get pod -n openfaas-fn
NAME                          READY   STATUS    RESTARTS   AGE
mailme-7b94f89946-7fxwz       1/1     Running   0          70s
nodeinfo-6c4754548b-6zxkq     1/1     Running   2          10d
```

Two functions are deployed here, but only the mailme-7b94f89946-7fxwz is related to what we are doing. It's status is Running, which is great! If there is some error there, run the following command to get some feeling as to why it's not working.

```
sudo kubectl describe pod mailme-7b94f89946-7fxwz -n openfaas-fn
```

Invoking OpenFaaS function

You have a couple of options.

faas-cli

```
echo '{ "api-key": "gr35p4inyyr4e9", "msg": "test msg", "to": "vladoportos@gmail.com" }' | faas-cli invoke mailme
```

And mail was delivered, you just have to trust me on this 🙂.

curl

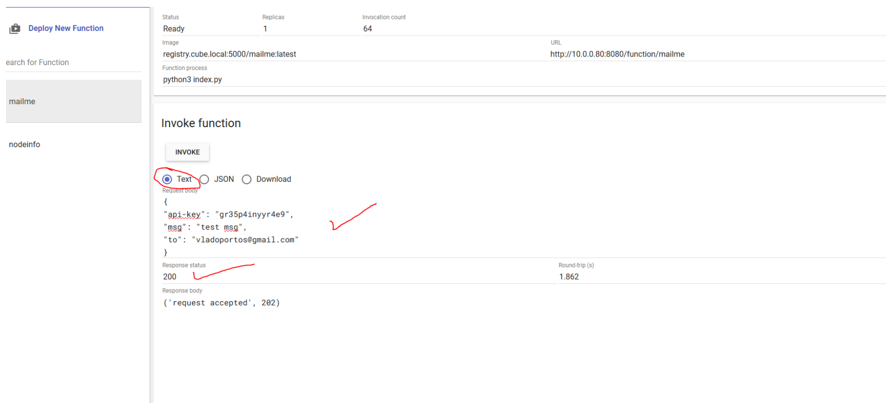
```
curl openfaas.cube.local:8080/function/mailme -d '{ "api-key": "gr35p4inyyr4e9", "msg": "test msg", "to": "vladoportos@gmail.com" }'
```

Web UI

Open the web UI of OpenFaaS, refer to the section about installing OpenFaaS to know how I got it. Docker-Registry TLS

Important

The three toggles on top * ***Text*** * ***JSON*** * ***Download*** What output are you expecting from your function? Not input, so don't switch to json if your function is not returning json.



Status: Ready
Replicas: 1
Invocation count: 64
Image: registry.cube.local:5000/mailme:latest
URL: http://10.0.0.80:8080/function/mailme
Function process: python3 index.py

[✓ Text] JSON Download

Request body
```
{
"api-key": "gr35p4inyyr4e9",
"msg": "test msg",
"to": "vladoportos@gmail.com"
}
```

Response status: 200 ✓
Round-trip (s): 1.862

Response body
```
('request accepted', 202)
```

All of them should return:

```
('request accepted', 202)
```

Did you like it, or was it helpful? Get yourself a drink and maybe get me one as well 🙂.

Liked it ? Buy me a drink :)

Last update: May 26, 2022

Comments

What do you think?

3 Responses

Upvote Funny Love Surprised Angry Sad

3 Comments https://rpi4cluster.com Disqus' Privacy Policy Login

Favorite Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Taz Elkikhia • 2 months ago • edited

This stepped bombed out for me

```
faas-cli publish -f mailme.yml --platforms linux/arm64
```

#25 ERROR: failed to do request: Head "https://registry.cube.local:5000/v2/mailme/blobs/sha256:9db4d8db47bd1e5897028f0fa3f6e03d3cfd0d5e73d2247d9a42cc4e21c6a72e": dial tcp: lookup registry.cube.local on 192.168.1.1:53: server misbehaving

• Reply • Share ›

Alex Ellis • 6 months ago

Thanks for writing about OpenFaaS. How can I contact you Vladimir? Do you want to send me an email alex@openfaas.com

• Reply • Share ›