# OpenFaaS

Finally, we are here: installing OpenFaaS on our already running K3s Kubernetes cluster on premise.

## Install

We are going to use **Arkade**; we installed this before to simplify the whole process.

```
# Switch to root, I did everything under root to avoid issues. ( In production you would use custom user, never run stuff under root in the wild.)
arkade install openfaas
```

The above will end with some information that you should note somewhere, like how to get an admin password for OpenFaaS Gateway.

```
PASSWORD=$(kubectl get secret -n openfaas basic-auth -o jsonpath="{.data.basic-auth-password}" | base64 --decode; echo)
# Above will store password in $PASSWORD environmental variable, this will disappear after relog.
echo $PASSWORD
```

But first, I want to have OpenFaaS gateway on its own IP accessible from outside.

## Custom MetalLB service

Deploying OpenFaaS with arcade will create two Gateway services:

- **gateway**
- **gateway-external**

```
root@control01:/home/ubuntu# kubectl get svc -n openfaas
NAME               TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
alertmanager       ClusterIP      10.43.109.178   <none>          9093/TCP         23h
basic-auth-plugin  ClusterIP      10.43.31.210    <none>          8080/TCP         23h
gateway            ClusterIP      10.43.195.169   <none>          8080/TCP         23h
gateway-external   NodePort       10.43.57.200    <none>          8080:31112/TCP   23h
nats               ClusterIP      10.43.76.87     <none>          4222/TCP         23h
prometheus         ClusterIP      10.43.238.226   <none>          9090/TCP         23h
```

These are all fine and dandy, but we did not deploy MetalLB for nothing. So, I created a new yaml config:

```
apiVersion: v1
kind: Service
metadata:
  name: openfaas-service
  namespace: openfaas
spec:
  selector:
    app: gateway
  type: LoadBalancer
  ports:
    - name: openfaas-port
      protocol: TCP
      port: 8080
      targetPort: 8080
  loadBalancerIP: 192.168.0.233
```

What to pay attention to:

- ***name*** - What is our service going to be called?
- ***namespace*** - This needs to be `openfaas`, that's where the OpenFaaS lives.
- ***port*** - Port from outside; we use the default where the gateway runs, so 8080.
- ***targetPort*** - Inside the container, it is 8080.
- ***loadBalancerIP*** - Our desired external IP (remember we set the range for MetalLB, so it's one of these).

To further explain, and I think I mentioned this before, I run the whole Kubernetes cluster in private LAN, separated from my network. So, when I'm logging in, I'm targeting the WAN IP of the switch, and that's it NATed to the control01 node.

Next, I do NAT again, just for port 8080 from WAN to 192.168.0.233 inside; this gives me access to OpenFaaS Gateway from my LAN. All of this is outside the scope of this guide, but not difficult to set up. It's just specific to your switch/router.

Anyway, apply the config and check:

```
root@control01:/home/ubuntu# kubectl apply -f service.yaml
root@control01:/home/ubuntu# kubectl get svc -n openfaas
NAME               TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
alertmanager       ClusterIP      10.43.109.178   <none>          9093/TCP         23h
basic-auth-plugin  ClusterIP      10.43.31.210    <none>          8080/TCP         23h
gateway            ClusterIP      10.43.195.169   <none>          8080/TCP         23h
gateway-external   NodePort       10.43.57.200    <none>          8080:31112/TCP   23h
nats               ClusterIP      10.43.76.87     <none>          4222/TCP         23h
openfaas-service   LoadBalancer   10.43.58.88     192.168.0.233   8080:31547/TCP   8h
prometheus         ClusterIP      10.43.238.226   <none>          9090/TCP         23h
```

***openfaas-service*** is there with external IP 192.168.0.233, nice!

## /etc/hosts

Let's give the IP some nice DNS name.

```
ansible cube -b -m lineinfile -a "path='/etc/hosts' line='192.168.0.233 openfaas openfaas.cube.local'"
```

## Environment

We will set up environment variables so CLI knows where the OpenFaaS url is. This way, you don't have to type it again. Also, where is our local docker registry? If you check back in this guide, I made an entry in /etc/hosts for every server to point docker-registry.local to the Docker registry service IP in our Kubernetes.

```
nano ~/.bash_profile
export OPENFAAS_URL=http://openfaas.local:8080
export OPENFAAS_PREFIX=docker-registry.local:5000
```

## OpenFaaS faas-cli

We need a special OpenFaaS CLI for us to build, push, remove functions and generally interact with OpenFaaS.

Installation is super simple:

```
curl -sL https://cli.openfaas.com | sudo sh
```

Check:

```
root@control01:/home/ubuntu# faas-cli version
  ___                   _____           ____
 / _ \ _ __   ___ _ __ |  ___|_ _  __ _/ ___|
| | | | '_ \ / _ \ '_ \| |_ / _` |/ _` \___ \
| |_| | |_) |  __/ | | |  _| (_| | (_| |___) |
 \___/| .__/ \___|_| |_|_|  \__,_|\__,_|____/
      |_|

CLI:
 commit:  598336a0cad38a79d5466e6a3a9aebab4fc61ba9
 version: 0.12.21
```

Now, you need to log in to gateway; if you did not restart your connection the password is still in your variable. If you did, just re-run the command.

```
root@control01:/home/ubuntu# echo -n $PASSWORD | faas-cli login --username admin --password-stdin
Calling the OpenFaaS server to validate the credentials...
WARNING! Communication is not secure, please consider using HTTPS. Letsencrypt.org offers free SSL/TLS certificates.
credentials saved for admin http://openfaas.local:8080
```

Ignore the https warning: we do not use https, since there is no domain that points back to our server. We run this on premise anyway 😊.

This is it! OpenFaaS is installed and running. You can confirm just in case, with:

```
root@control01:/home/ubuntu# kubectl get pods -n openfaas
NAME                                 READY   STATUS    RESTARTS   AGE
alertmanager-7b849dbf96-js76z        1/1     Running   0          23h
basic-auth-plugin-6dc5ddd97f-v6vg7   1/1     Running   0          23h
gateway-8d768fb57-4rqnr              2/2     Running   0          23h
nats-7fdd76d465-6677w                1/1     Running   0          23h
prometheus-54c655c4fd-89g9k          1/1     Running   0          23h
queue-worker-64b75867c8-kn7br        1/1     Running   0          23h
```

In the next chapter, we will deploy some python functions as examples.



---

Last update: October 20, 2021

## Comments